10/01/2024

AERO 4 - MATHEMATICAL TOOLS FOR DATA SCIENCE (2023/2024)

Ma412_Final_Project

Maxime SPENCER
IPSA

Table of Contents

Introduction

This project delves into clustering algorithms to reveal meaningful patterns within complex datasets. The goal is to gain insights into their capabilities and limitations, providing a comprehensive overview of clustering techniques and their role in uncovering underlying structures within data.

Cluster determination

We will start by searching for the best number of clusters for our data. For that we will use the Silhouette score:

```python
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

data = np.load('data.npy')

cluster_range = range(2, 11)

silhouette_scores = []

for num_clusters in cluster_range:
    kmeans = KMeans(n_clusters=num_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(data)

    silhouette_avg = silhouette_score(data, cluster_labels)
    silhouette_scores.append(silhouette_avg)

    print(f"Silhouette Score for {num_clusters} clusters: {silhouette_avg}")

optimal_num_clusters = cluster_range[np.argmax(silhouette_scores)]
print(f"Optimal number of clusters: {optimal_num_clusters}")
```

We obtain this result:

```
Silhouette Score for 2 clusters: 0.902
Silhouette Score for 3 clusters: 0.645
Silhouette Score for 4 clusters: 0.726
Silhouette Score for 5 clusters: 0.752
Silhouette Score for 6 clusters: 0.773
Silhouette Score for 7 clusters: 0.736
Silhouette Score for 8 clusters: 0.631
Silhouette Score for 9 clusters: 0.508
Silhouette Score for 10 clusters: 0.455
```

Where we see that 2 is the peak. So, we will use 2 clusters during this project.

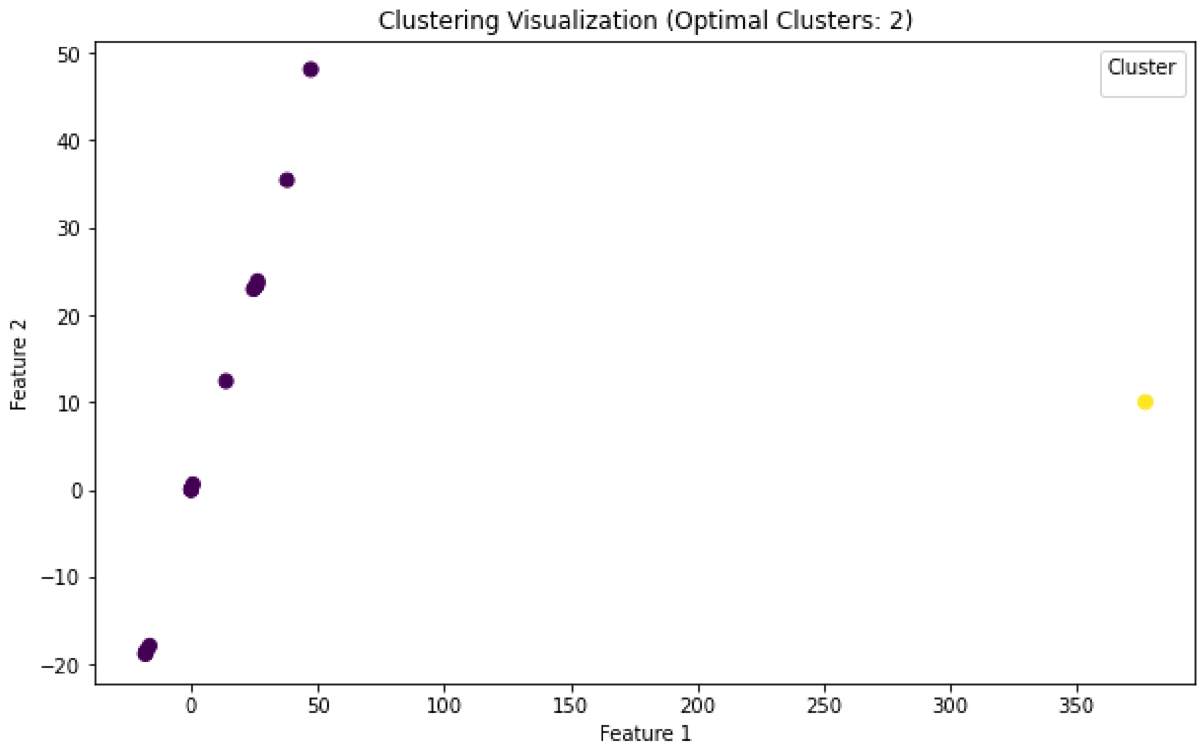
We obtain a Silhouette Score for 10 clusters: 0.45525967142243456.

Which is not a lot but is positive, so our cluster tend to be separate.

We can look at them doing this:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

data = np.load('data.npy')

optimal_num_clusters = 2

kmeans = KMeans(n_clusters=optimal_num_clusters, random_state=42)
cluster_labels = kmeans.fit_predict(data)

data_with_clusters = np.column_stack((data, cluster_labels))

plt.figure(figsize=(10, 6))
plt.scatter(x=data_with_clusters[:, 0], y=data_with_clusters[:, 1], c=cluste
plt.title(f'Clustering Visualization (Optimal Clusters: {optimal_num_cluster
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend(title='Cluster')
plt.show()
```

We obtain:

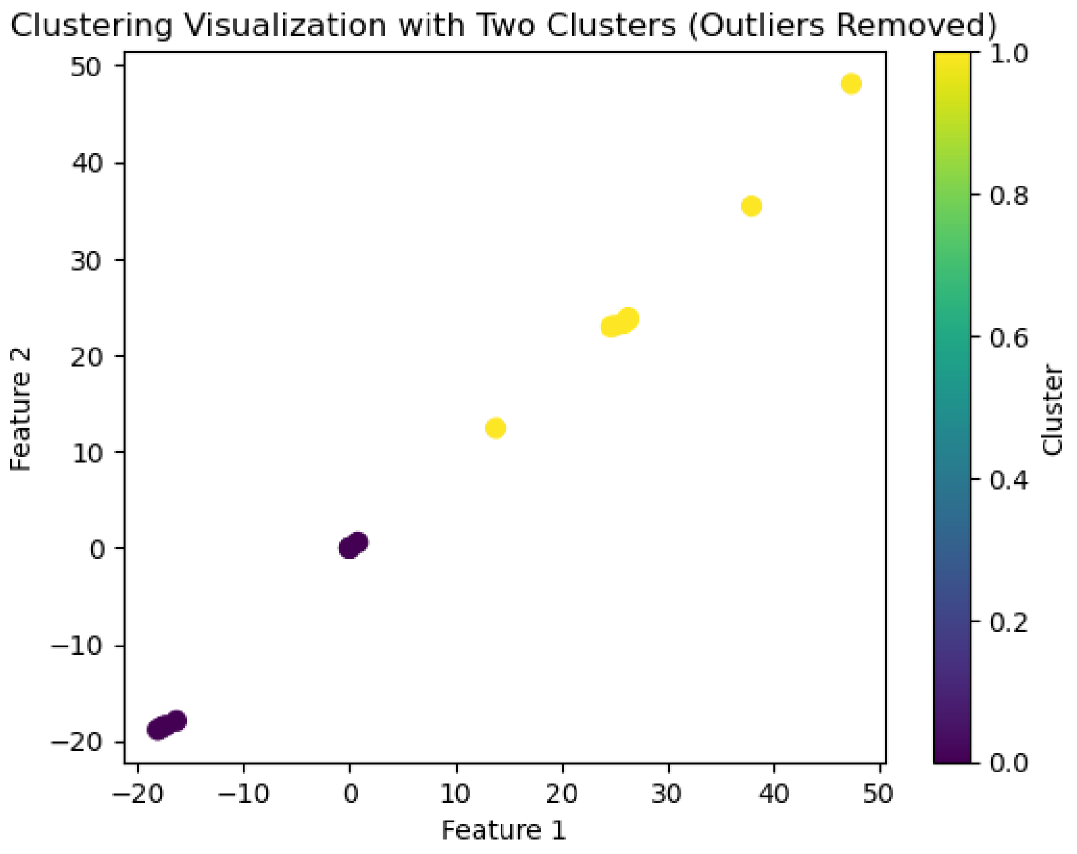


The plot seems good and the cluster 2 seems to be an outlier.

I need to remove the outlier.

```python
import numpy as np
from scipy import stats
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

data = np.load('data.npy')

z_scores = stats.zscore(data, axis=0)

threshold = 3

outlier_indices = np.where(np.abs(z_scores) > threshold)
cleaned_data = np.delete(data, outlier_indices[0], axis=0)

num_clusters = 2
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
cluster_labels = kmeans.fit_predict(cleaned_data)

plt.scatter(cleaned_data[:, 0], cleaned_data[:, 1], c=cluster_labels, cmap='
plt.title('Clustering Visualization with Two Clusters (Outliers Removed)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Cluster')
plt.show()
```

We obtain:



1/Maximum likelihood

With this method we will find the parameters of a linear regression which maximize the data.

The model is defined as:

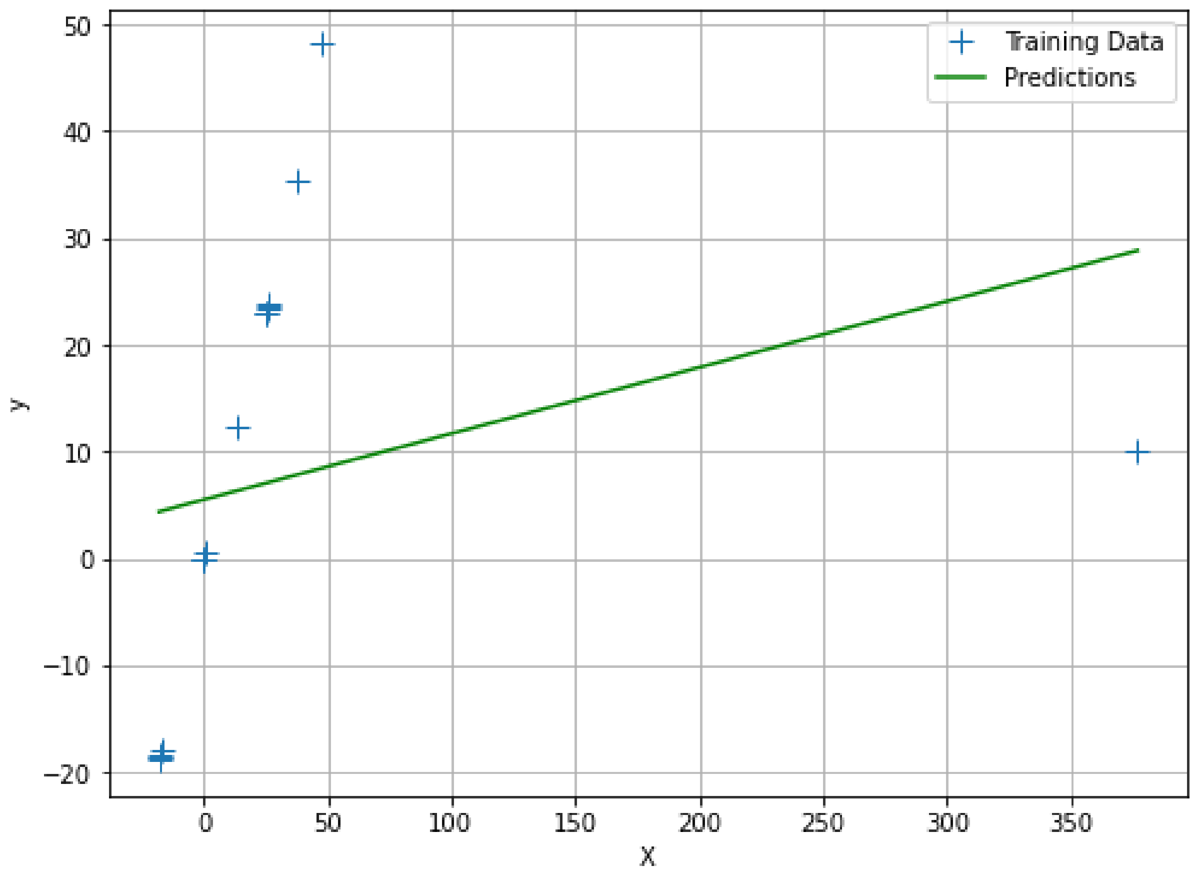
$$y = \theta_0 + \theta_1 . X$$

Where $\theta_0$is the bias term, $\theta_1$is the coefficient and X the input.

We load the dataset 'data.npy' and extract the features (X) and targets (y). Features are augmented by adding a column of ones for bias. The maximum likelihood estimate theta_ml_aug is obtained using the provided function. A test set Xtest is defined, and predictions are made using the trained parameters.

We obtained this :

theta_ml_aug [[5.48402567] [0.06186837]]



The result of theta_ml_aug give us for the first part 5.48402567, the bias term which is the expected values of the target.

[0.06186837] is the coefficient of the feature. It represents the change in the target variable.

The value is positive so there is a positive correlation between the feature and the target variable.

5.48402567 is the point where the prediction crosses the y axis.

2/Ridge regression

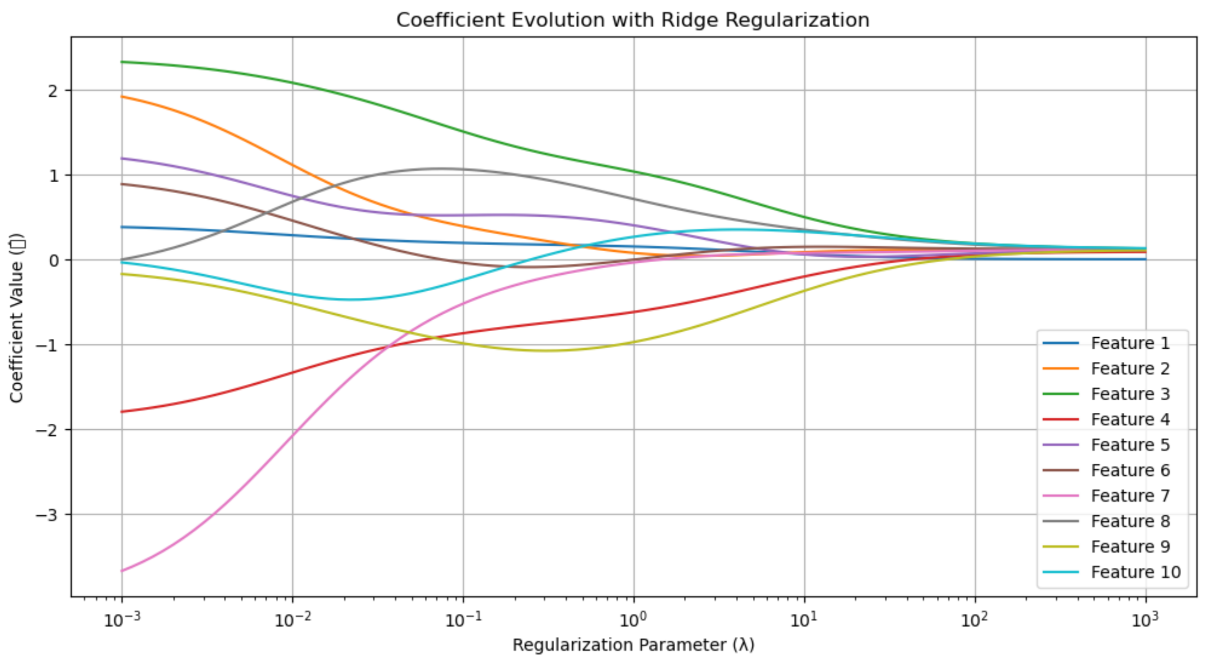
With this method we will predict how our data will evolve, by drawing lines through the data points. This method introduces regularization to the linear regression model to prevent overfitting.

We load the data by defining feature X and variable Y. We spit the data between training and testing sets. We then implement Ridge regression function and RidgeCv. We used them to find the best lambda, and to evaluate the model performance by finding the mean squared error.

Our result are :
Ridge Regression - Intercept (Alpha): 0.008998085603445283
Ridge Regression - Coefficients (Beta): [0.00060901 -0.26219793 0.59498512 0.09201136 0.544247
19 -0.17214915
 -0.0009258   0.15357297 -0.0805883   0.23120064]
We can see that Features 3, 5, 10, and 8 have a positive impact on the predicted target.
Features 2, 6, 9, and 4 have a negative impact on the predicted target.

We have :
Best Lambda Value: 2.1544346900318843
Mean Squared Error (MSE) on Entire Dataset: 0.1586178751435575

The value of lambda is high which means that the model is strongly regulated. The target is heavily penalized.
The MSE is low, so the model as good performances, the lambda is well chosen.



As the regularisation parameter λ increases, the coefficients tend to decrease towards zero.

ven with a high λ, the coefficients are not reduced to zero, but tend towards values very close to zero.

Some coefficients therefore remain important for prediction, although their influence is reduced.

In general, we can conclude that the with the dataset the Ridge regression perform well.

3/Lasso regression

LASSO is a regression analysis method that performs both variable selection and regularization. It adds a penalty term to the absolute values of the coefficients, which encourages sparse solutions.

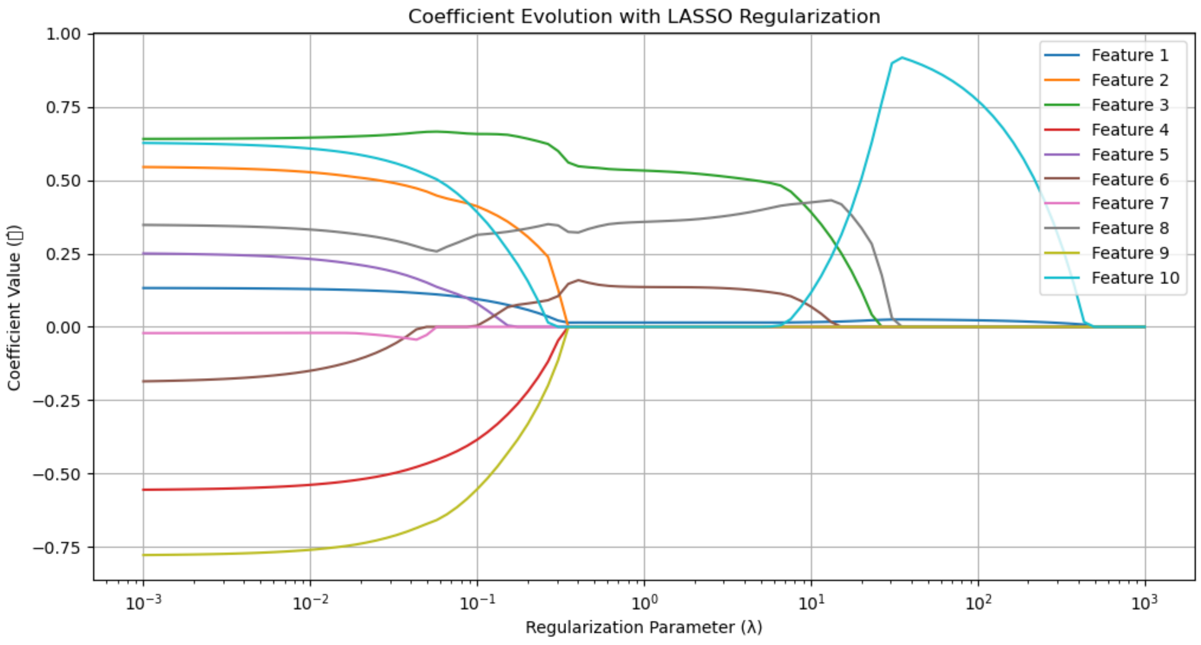
We start by loading the dataset, X is features and y the variable. We split the data into training and testing sets. Then, predictions are made to compute the MSE and to see the evolution of the coefficients. We also print the best lambda.

We have the results:

```
Best Lambda Value: 0.012328467394420659
 Mean Squared Error (MSE) on Entire Dataset: 0.2613285864996973
```

The lambda value is small, so the model is less regulated.

The MSE is low, so the model as good performances.



As the regularisation parameter λ increases, the coefficients tend to decrease towards zero. Even wit h a high λ, the coefficients are not reduced to zero, but tend towards values very close to zero. Some coefficients therefore remain important for prediction, although their influence is reduced.

In general, the LASSO Regression method works reasonably well, providing a balance between complexity and precision in predictions. There is room for improvement, but the model shows promising results.

4/ Elastic net regression

The Elastic Net Regression model combines the strengths of Lasso and Ridge regressions, efficiently handling correlated features and automatically selecting important ones. The model's hyperparameters control the degree of regularization and the balance between Lasso and Ridge penalties.

In this code, we start by initializing the data, X is features, y is variable. Data is trained and tested, then prediction are made and the MSE is calculated from the trained hyperparameters.

The result is:

Mean Squared Error (MSE) on Test Set: 1.115447260638545
The MSE is low, so the model performs well.

We can conclude that the Elastic net regression is a good model to analyse our data.

5/ Comparison of the regression

This code compares the performance of Ridge, Lasso, and Elastic Net Regression methods on a given dataset.

With the code we obtain:

```
Ridge MSE: 0.31440020596625945
Lasso MSE: 1.1349204269434292
Elastic Net MSE: 1.115447260638545
Ridge Regression performs the best.
```

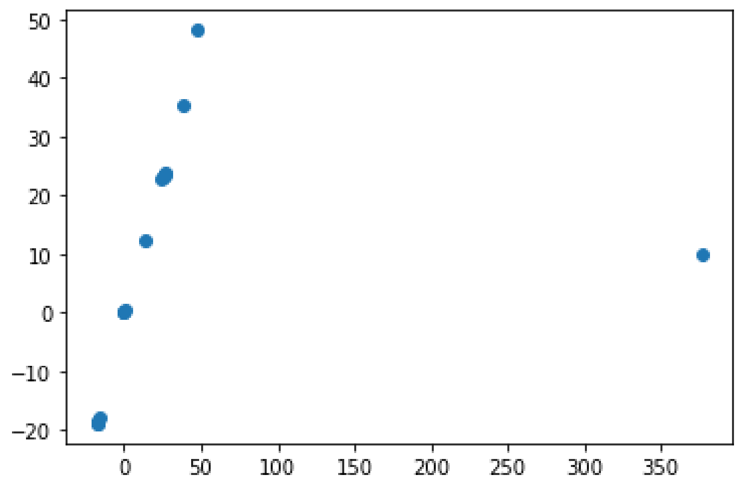
As we can see the MSE for the Ridge is the lowest, so it performs better than the other two.

6/Support Vector Machine

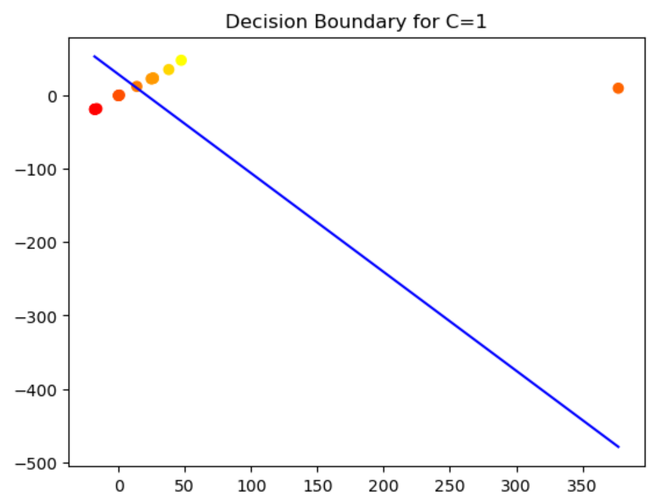
The algorithm uses Support Vector Machines (SVM) with a linear kernel to find an optimal hyperplane for classifying positive and negative examples. The parameter `C` controls the balance between training error and hyperplane margin.

We start the code by loading the code and defining X and y. We plot the data using 'utils4.plotData' and we trained the svm models with differents 'C'. Finally, the decision boundaries are plot using 'utils4.visualizeBoundaryLinear'.

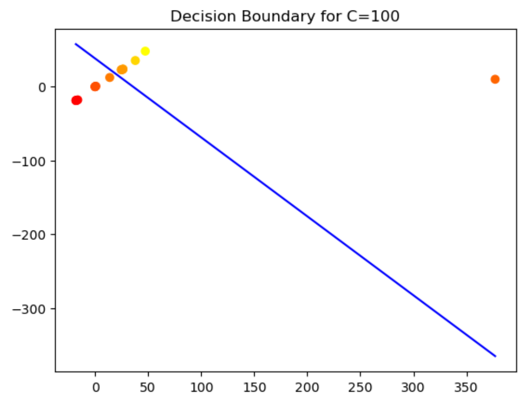
Here is the result we obtain:

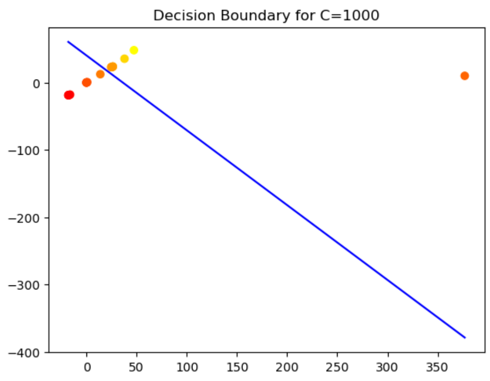


We observe that the data plot gives us only positive features. The data is not well, there is a point alone on the right.



For C=1, we can see that the decision boundary didn't separates well the data in two.





We observe the same thing for C=100 and C=1000

We can conclude by saying that the method didn't work well with our dataset. There is no negative point, and the decision boundary didn't separate well the point for different parameter C.

7/ Gaussian Kernel (SVM)

The code demonstrates the application of a Support Vector Machine (SVM) with a Gaussian kernel for non-linear classification.

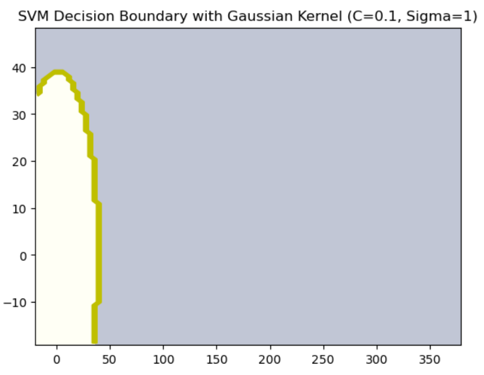
In the code, we defined the Gaussian Kernel function, test it with two sample data points, we trained the svm with different values of 'C' and 'sigma' and visualized the decision boundaries.

The first result we obtain are:

```
Gaussian Kernel between x1 = [47.32178  48.109274], x2 = [37.917452 35.424735], sigma = 1:
        7.190226890408274e-55
```

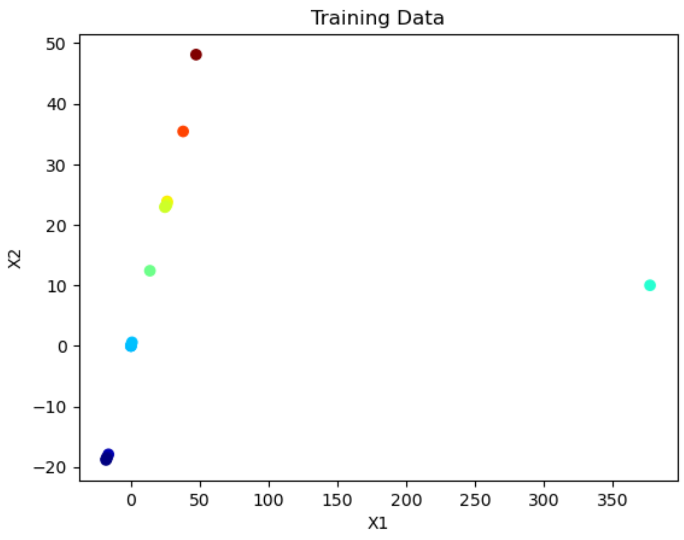
The output is small, so the similarity for the two inputs will be far apart between them. Their contributions to the decision boundary will be minimal.

The second result:



For C=0.1 and sigma=1, with see that the yellow circle who encircle positive feature is on the bottom left where the data plot has the positive point. It seems to be a nonlinear decision boundary.

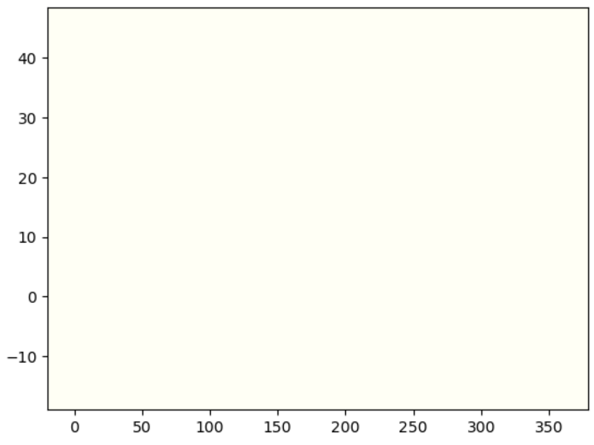
The third result are:



We obtain the same plot as the first one. We only have positive point.

The next result is:

Best C: 0.01
Best Sigma: 0.01
The best combination should be C=0.01 and sigma= 0.01.



We obtain only a yellow background for the plot. The parameter is nor suitable or the dataset is complex.

In conclusion, the Gaussian Kernel method may not be the best choice for this dataset, and further exploration or other kernel methods might be required to improve performance.

8/Neural network

The code loads a dataset, transposes it for the proper format, and visualizes a subset of examples. The 'data.npy' file is used to load the data, which is then transposed to have examples as rows and features as columns. A grayscale image representation is used to display a random subset of examples with their features. The displayed images are reshaped to a 9x2 grid for visualization.

Here is the result:

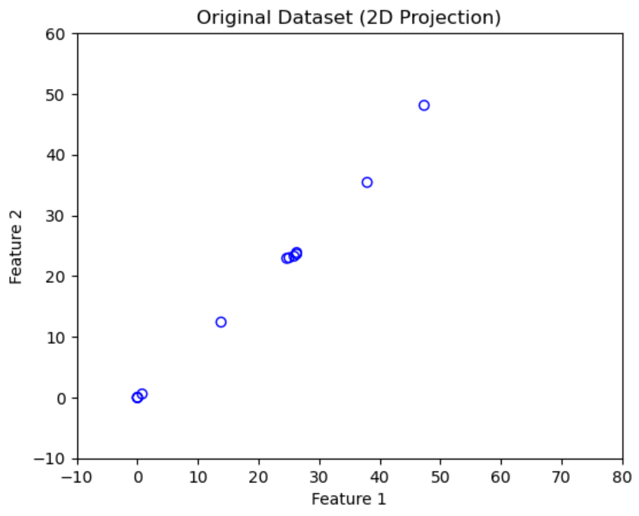
As we can see we found the 18 exemples as we proposed.

9/ Principal Component Analysis

Principal Component Analysis (PCA) works by first normalizing the input data to achieve zero mean and unit variance for each feature. It then computes the covariance matrix to capture the relationships between the features.

First, we load the dataset, then we visualize the projection in 2D. The definition of the feature normalization is given, the definition of pca function, we then normalize the feaure using the definition, and we applied pca to obtain the principal components U, the singular values S and the unitary matrices V. Then the Pca are displayed, we visualize the components and print the top eigenvectors.

The first result we have is :



We can see that the data that we plot form a linear plot from 0 to 50. This linear form suggests a strong correlation of the dataset.

The second result:

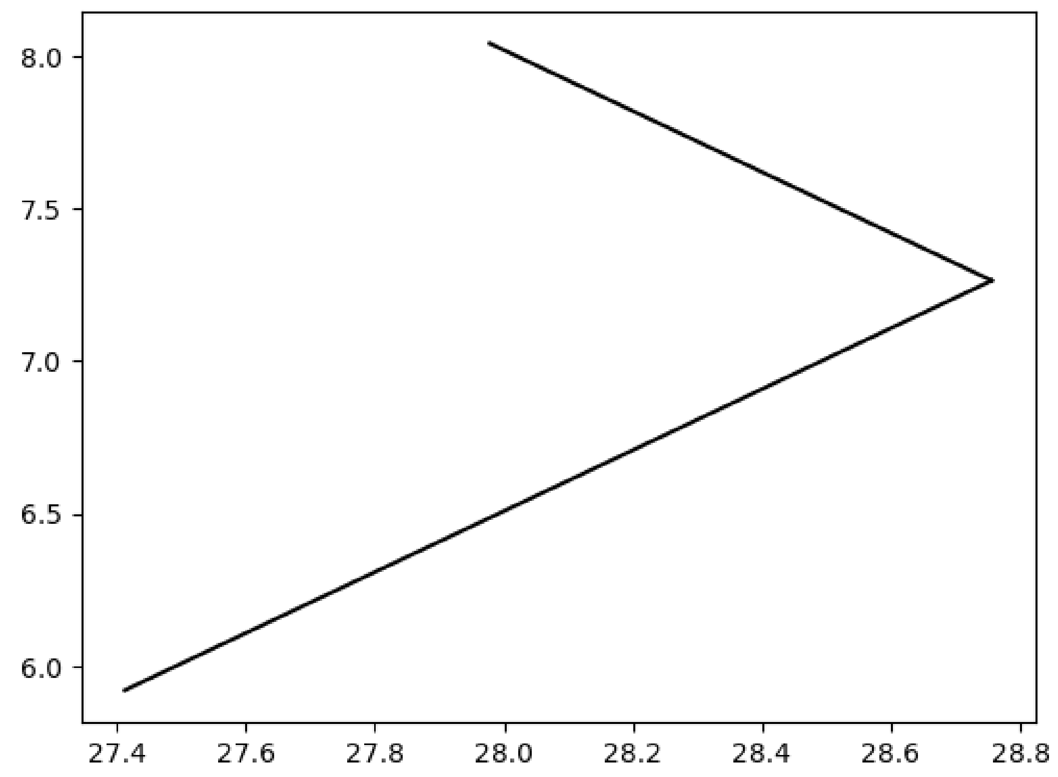
```
Principal Components (U):
[[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]

Singular Values (S):
[1.266796 0.733204]

Unitary Matrices (V):
[[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
```

The matrix U tell us that they are two principal components, they indicate the direction of the dataset.

The matrix S represents the importance of each principal components. In this case we have two values, and the bigger one is the most important.

The matrix V has the same form as U, it provides additional information about the dataset.

The third result are:



The top line represents the primary axis of variation, extending from the mean mu along the direction of the first principal component U[:, 0] , scaled by 1.5 times its singular value (S[0]). Similarly, the bottom line represents the secondary axis of variation, aligned with the direction of the second principal component (U[:, 1]), scaled by 1.5 times its singular value (S[1]). The length and direction of each line convey the importance and orientation of the corresponding principal component, providing a visual representation of how these components influence the overall structure and distribution of the data set.

The last result:

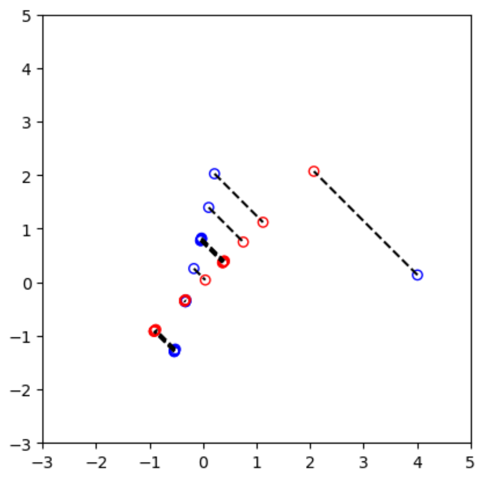
Top eigenvector:
U = [-0.70710678 -0.70710678]
It gives us the direction in the dataset captured by the principal component.

10/Dimension reduction

To do the dimension reduction we start by defining the project data function and recover data function. We start by visualizing the dataset. We put K=1, and project and recover the data, then we plot it. We now do it for our dataset 'data.npy'.

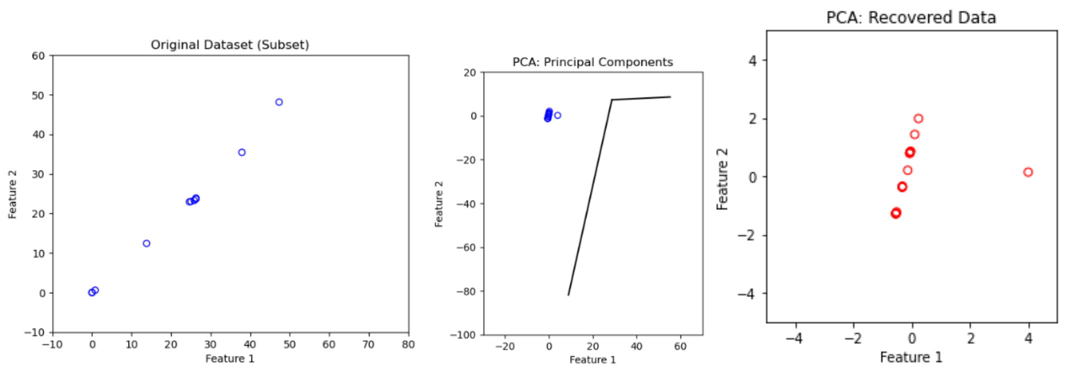
The first results:

Projection of the first example: [-1.5843611]
Approximation of the first example: [1.12031248 1.12031248]



As we have seen in the first example both representations show the principal component. Here the value in red represents the expected value that we will use, they are the value of the eigenvalue of the principal component, and they have the most information. The rest are the rest of the value of the dataset that are not well recovered and that can't be used. In the result, the projection value is -1.5843611 for the first example, and the approximation is [1.12031248, 1.12031248].

The second results:



The initial plot illustrates a portion of the original dataset in a two-dimensional space. The second plot displays the principal components obtained through PCA on the normalized features. The blue points represent the original data, and the black lines indicate the components. The third plot exhibits the recovered data after projecting onto the first two principal components. The recovered data is represented by the red points, demonstrating the ability of PCA to capture essential features and simplify the representation in a reduced-dimensional space.

The code effectively demonstrates dimensionality reduction and the ability to recover data using PCA.

11/K-means clustering

The Python code provided implements the K-means clustering algorithm. It includes functions for finding the closest centroids, computing new centroid positions, and randomly initializing centroids.

We start the algorithm by defining the function to find the closest centroids. We initialize the data; we choose 2 clusters. We then applied the function to find the centroids. Compute centroid function is defined, and the input and output are chosen. We applied this function and displayed it. We then run the runkMeans function from utils for 10 iterations.We initialize the centroids we the function associated and visualize all.

The first result:

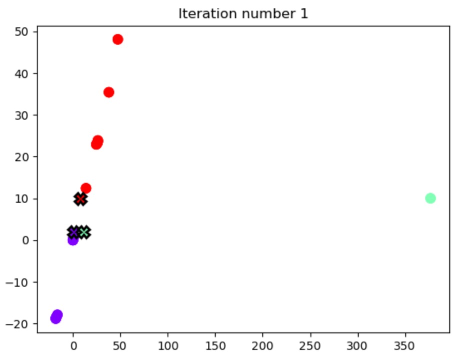
Closest centroids for the first 3 examples:

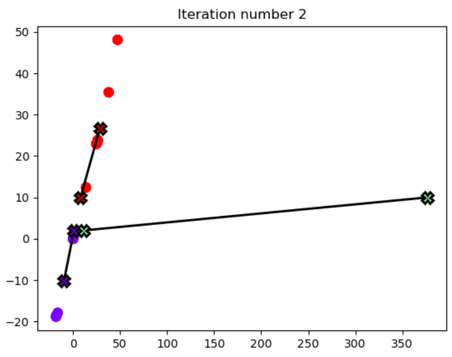
```
[2 2 0]
```
It represents each closest centroids for their respective example.

The second result:

Centroids computed after initial finding of closest centroids:
```
[[ -9.61495232 -10.21198269]
 [377.          10.        ]]
```
We observe that the second centroid is huge compared to the first.

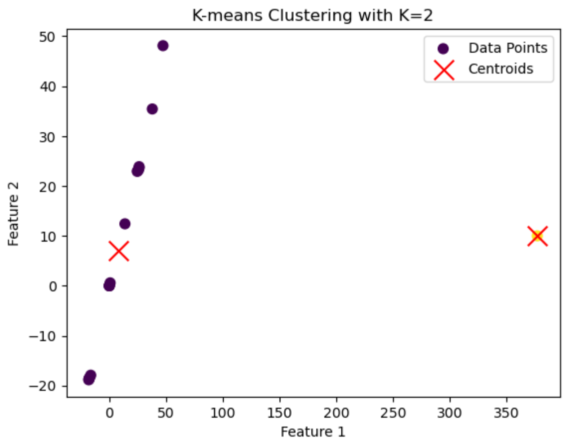
The third result:





We observe that the centroid is not currently well shown, they are not centred well for the first iteration, it takes 2 iterations to be centred.

The last result:

Centroids computed after initial finding of closest centroids:
```
[[  8.27119183   7.10212734]
 [377.          10.        ]]
```


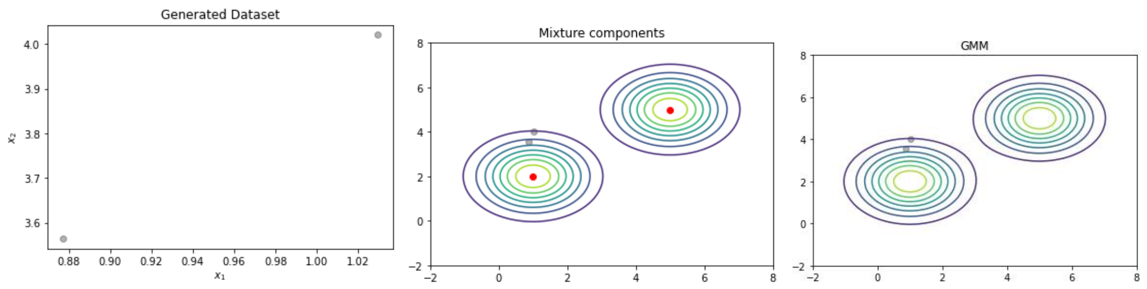
The K-Means algorithm produced centroids [8.27, 7.10] and [377, 10] with K=2 clusters and 10 iterations. However, the second centroid's unusually large value suggests potential convergence issues or initialization problems. The scatter plot visualizes data points colored by their assigned clusters, with centroids marked in red. Further investigation or algorithm adjustments may be necessary for improved results.

We can conclude that in overall the method is good with 2 clusters, the centroid seems well placed. The only issue is the high value of the second centroid which can be related to a problem in the initialization.

12/Gaussian mixture models

The Gaussian Mixture Model (GMM) is a probabilistic model that assumes that the data is generated from a mixture of several Gaussian distributions. Each Gaussian component represents a cluster in the data. The Expectation-Maximization (EM) algorithm is commonly used to estimate the parameters of a GMM, such as the means, covariances, and weights.

The first result is:



The data is well generated, there is 2 clusters in the data as we want. There are two mixture components, and they keep this number in the GMM. The two clusters are in the left mixture component. We can say that the GMM successfully models the distribution of the data.

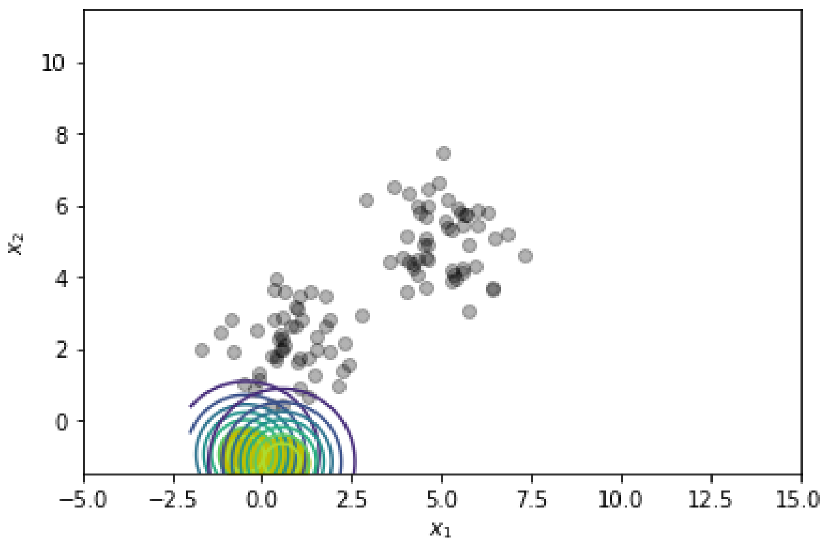
We start by generating the dataset, then we visualize the GMM components. W initialize the means, covariances and weights for EM algorithm, initialize the list which store the Negative Log-likelihood values. We iterate te EM algorithm to find is convergence. And then we do a final GMM visualization, and we plot the NLL.

The second result:

```
Initial mean vectors (one per row):
[[-0.64828282 -0.13094691]
 [-0.06686261 -0.13860127]]
```

The output displays the initial mean vectors for a Gaussian Mixture Model (GMM) with two clusters (K=2). These vectors indicate the starting points in the feature space for each cluster and are randomly generated to represent the initial guess for the cluster centers before refinement through the Expectation-Maximization (EM) algorithm.

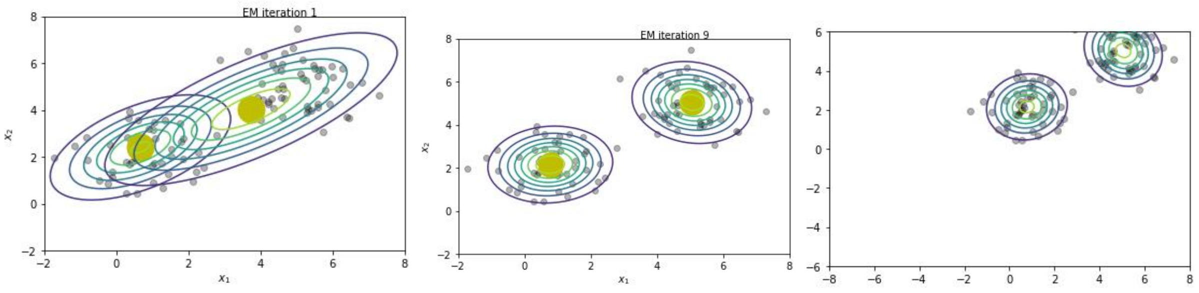
The third result:



We observe that in this plot, the two circles intertwine, and most of the points are not inside the two circles. The EM algorithm need to adjust the parameters to better fit the data in other steps.

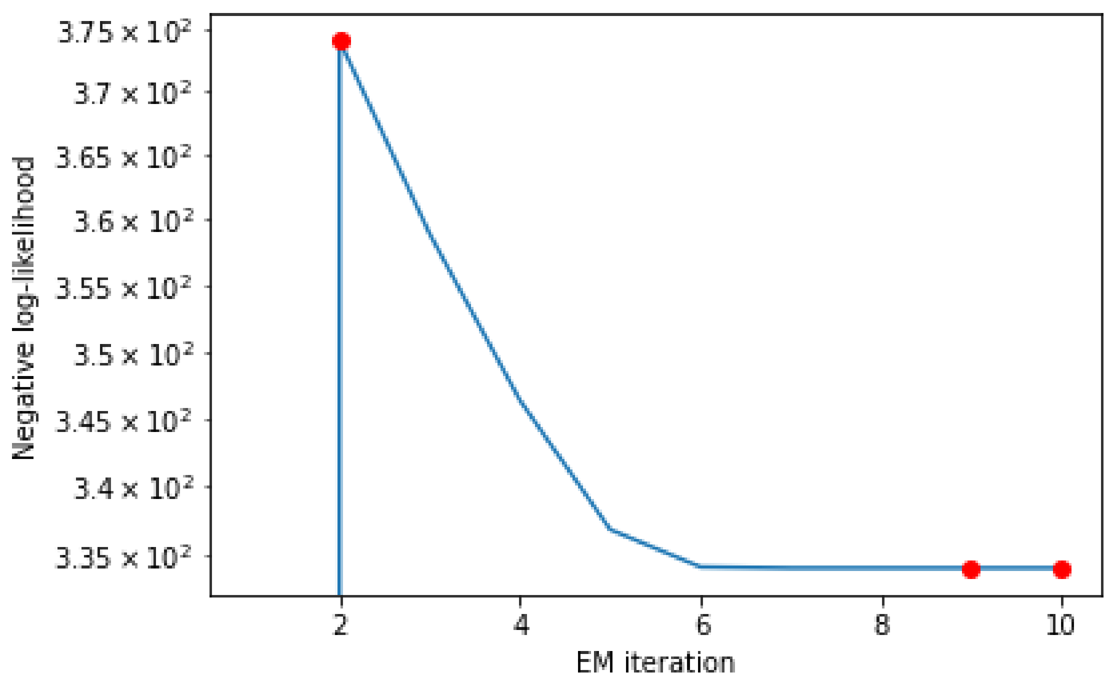
The fourth result:

The are 10 iterations:



It Converged after iteration 9
With this we observe that the EM algorithm had adjust the plot by steps and after 14 iterations it bec ame more precise and stabilized.

The last result:



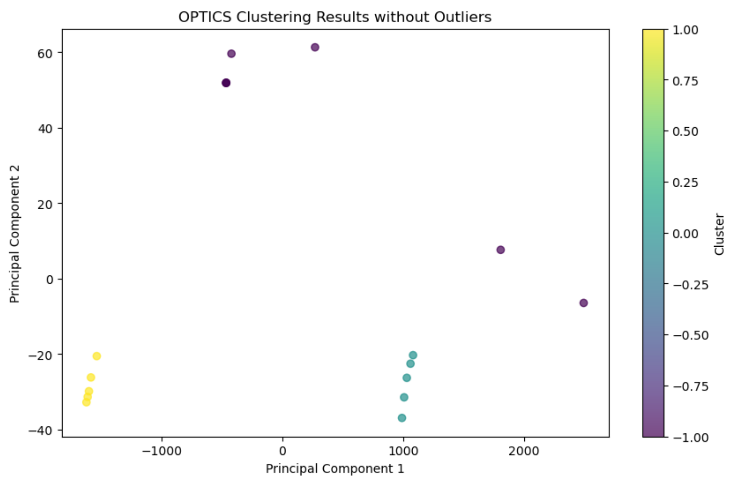
The graph shows the negative log-likelihood (NLL) of the Gaussian Mixture Model (GMM) during the Expectation-Maximization (EM) iterations. The y-axis is logarithmic. As the EM algorithm progresses (x-axis), the NLL decreases, indicating an improvement in the model's fit to the data. The red dots highlight specific iterations: the first point (2, 3.75e2) corresponds to the initial negative log-likelihood (NLL); the second point (9, 3.35e2) represents a later iteration, and the third point (10, 3.35e2) marks the convergence point after 14 iterations.
Indicates that the expectation-maximization (EM) algorithm has reached a stable solution, and further iterations do not significantly reduce the negative log-likelihood (NLL).
We can conclude that the method worked well with our dataset, but it takes 14 iterations to reach a stable solution.

13/OPTICS (Ordering Points To Identify Clustering Structure)

OPTICS is a density-based clustering algorithm that extends DBSCAN. It calculates reachability distances for each point, highlighting regions of varying density. The resulting hierarchical ordering of points reveals clusters without requiring a predefined cluster count. OPTICS is adept at identifying clusters with diverse shapes and densities in spatial data.

We start by loading the dataset, then we applied the pca analysis to reduce the data to 2. We do the OPTICS initialization with the parameters ‘min_samples’, ‘xi’, and ‘min_clusters_size’. We obtained the clusters from optics and visualize the result and plot the reachability distances.

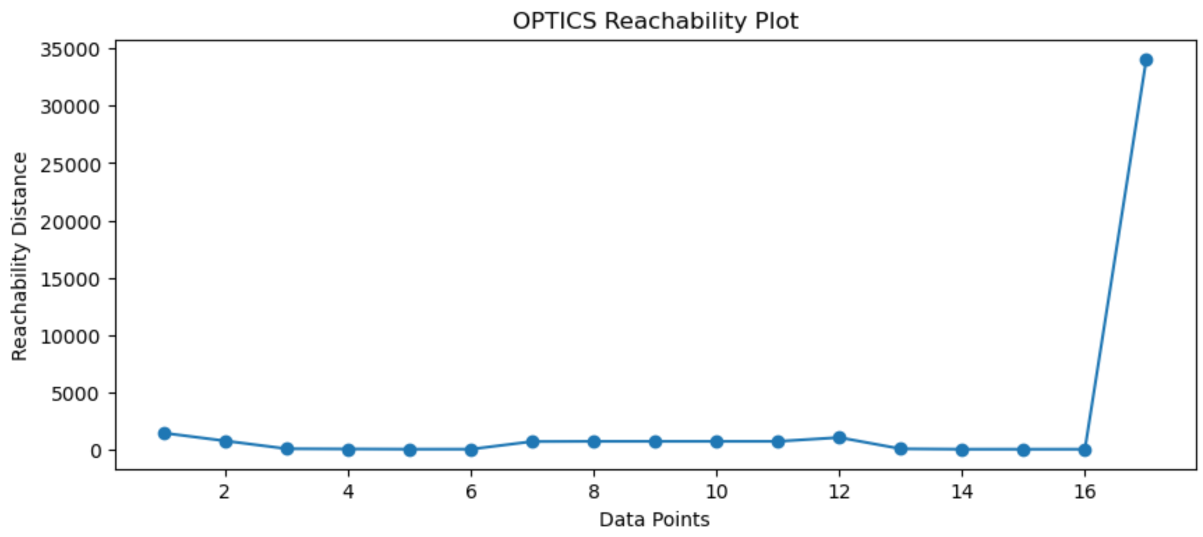
The first result is:



We observe 3 distinct group yellow, blue, and purple.

They seem to have 3 clusters.

We also have:



There is a constant region from x=0 to 16, with a low reachability, this means that the clusters are densely packed.

At x=18, we have a peak to y= 35000, which means that the distance between points will increase.

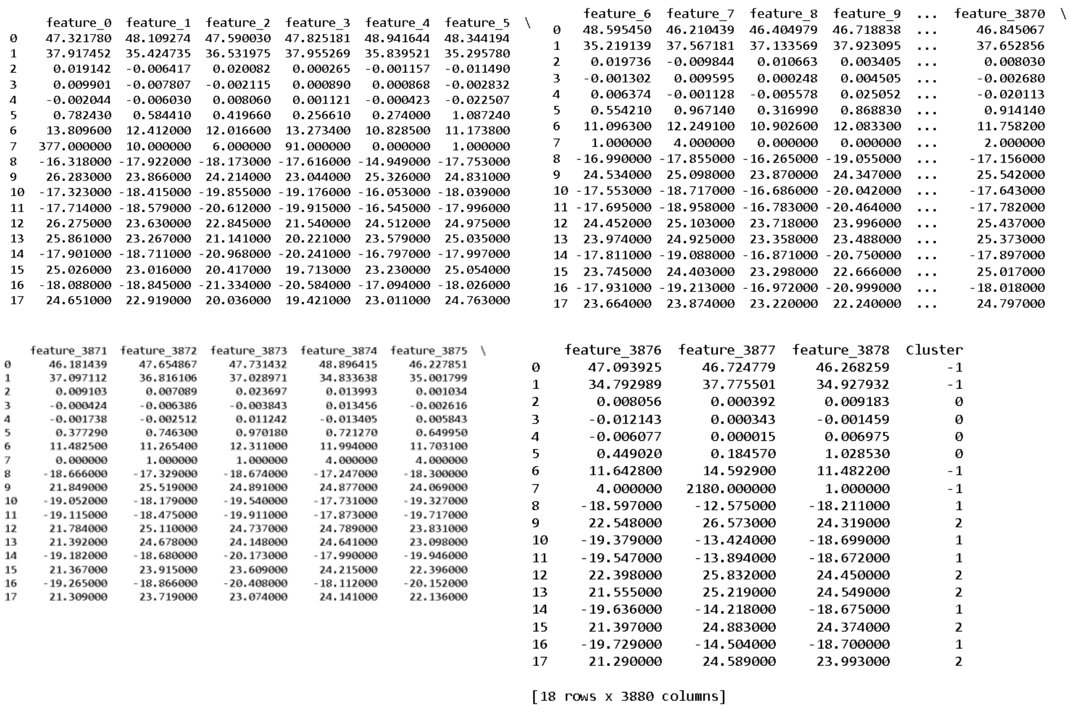
This method is efficient to indicate the presence of clusters and to establish the distance between them. It revealed that at x=18, there is a boundary between clusters.

14/DBScan

DBSCAN is a density-based clustering algorithm that groups data points based on their spatial density in the feature space. The algorithm identifies clusters as regions of high data point density and can discover clusters of arbitrary shapes without requiring a predetermined number of clusters.

We start by loading the data and standardizes it using 'StandardScaler'. We applied the DBScan clustering by defining its parameter. We display it in a plot and the use the function to find the k-nearest neighbours and plot it.

First result:



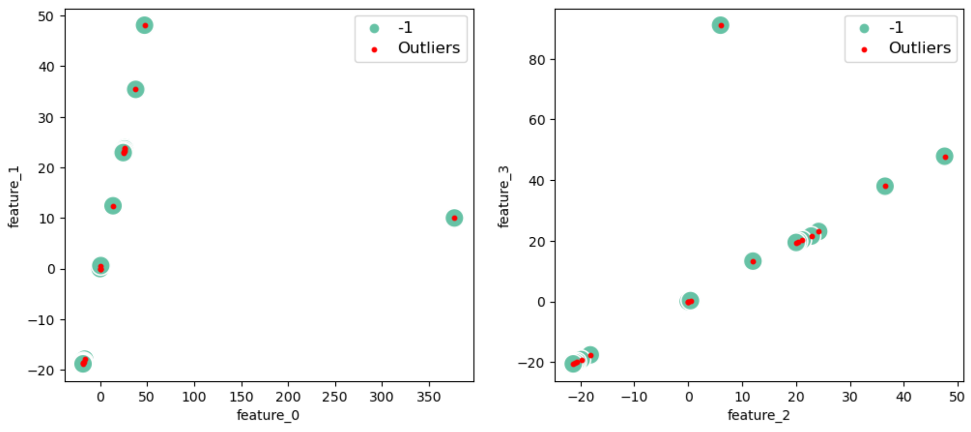
```
   feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  \
0   47.321780  48.109274  47.590030  47.825181  48.941644  48.344194
1   37.917452  35.424735  36.531975  37.955269  35.839521  35.295780
2    0.019142  -0.006417   0.020082   0.000265  -0.001157  -0.011490
3    0.009901  -0.007807  -0.002115   0.000890   0.000868  -0.002832
4   -0.002044  -0.006030   0.008060   0.001121  -0.000423  -0.022507
5    0.782430   0.584410   0.419660   0.256610   0.274000   1.087240
6   13.809600  12.412000  12.016600  13.273400  10.828500  11.173800
7  377.000000  10.000000   6.000000  91.000000   0.000000   1.000000
8  -16.318000 -17.922000 -18.173000 -17.616000 -14.949000 -17.753000
9   26.283000  23.866000  24.214000  23.044000  25.326000  24.831000
10 -17.323000 -18.415000 -19.855000 -19.176000 -16.053000 -18.039000
11 -17.714000 -18.579000 -20.612000 -19.915000 -16.545000 -17.996000
12  26.275000  23.630000  22.845000  21.540000  24.512000  24.975000
13  25.861000  23.267000  21.141000  20.221000  23.579000  25.035000
14 -17.901000 -18.711000 -20.968000 -20.241000 -16.797000 -17.997000
15  25.026000  23.016000  20.417000  19.713000  23.230000  25.054000
16 -18.088000 -18.845000 -21.334000 -20.584000 -17.094000 -18.026000
17  24.651000  22.919000  20.036000  19.421000  23.011000  24.763000

\   feature_6  feature_7  feature_8  feature_9  ...  feature_3870  \
0   48.595450  46.210439  46.404979  46.718838  ...     46.845067
1   35.219139  37.567181  37.133569  37.923095  ...     37.652856
2    0.019736  -0.009844   0.010663   0.003405  ...      0.008030
3   -0.001302   0.009595   0.000248   0.004505  ...     -0.002680
4    0.006374  -0.001128  -0.005578   0.025052  ...     -0.020113
5    0.554210   0.967140   0.316990   0.868830  ...      0.914140
6   11.096300  12.249100  10.902600  12.083300  ...     11.758200
7    1.000000   4.000000   0.000000   0.000000  ...      2.000000
8  -16.990000 -17.855000 -16.265000 -19.055000  ...    -17.156000
9   24.534000  25.098000  23.870000  24.347000  ...     25.542000
10 -17.553000 -18.717000 -16.686000 -20.042000  ...    -17.643000
11 -17.695000 -18.958000 -16.783000 -20.464000  ...    -17.782000
12  24.452000  25.103000  23.718000  23.996000  ...     25.437000
13  23.974000  24.925000  23.358000  23.488000  ...     25.373000
14 -17.811000 -19.088000 -16.871000 -20.750000  ...    -17.897000
15  23.745000  24.403000  23.298000  22.666000  ...     25.017000
16 -17.931000 -19.213000 -16.972000 -20.999000  ...    -18.018000
17  23.664000  23.874000  23.220000  22.240000  ...     24.797000

    feature_3871  feature_3872  feature_3873  feature_3874  feature_3875  \
0      46.181439     47.654867     47.731432     48.896415     46.227851
1      37.097112     36.816106     37.028971     34.833638     35.001799
2       0.009103      0.007089      0.023697      0.013993      0.001034
3      -0.000424     -0.006386     -0.003843      0.013456     -0.002616
4      -0.001738     -0.002512      0.011242     -0.013405      0.005843
5       0.377290      0.746300      0.970180      0.721270      0.649950
6      11.482500     11.265400     12.311000     11.994000     11.703100
7       0.000000      1.000000      1.000000      4.000000      4.000000
8     -18.666000    -17.329000    -18.674000    -17.247000    -18.300000
9      21.849000     25.519000     24.891000     24.877000     24.069000
10    -19.052000    -18.179000    -19.540000    -17.731000    -19.327000
11    -19.115000    -18.475000    -19.911000    -17.873000    -19.717000
12     21.784000     25.110000     24.737000     24.789000     23.831000
13     21.392000     24.678000     24.148000     24.641000     23.098000
14    -19.182000    -18.680000    -20.173000    -17.990000    -19.946000
15     21.367000     23.915000     23.609000     24.215000     22.396000
16    -19.265000    -18.866000    -20.408000    -18.112000    -20.152000
17     21.309000     23.719000     23.074000     24.141000     22.136000

    feature_3876  feature_3877  feature_3878  Cluster
0      47.093925     46.724779     46.268259       -1
1      34.792989     37.775501     34.927932       -1
2       0.008056      0.000392      0.009183        0
3      -0.012143      0.000343     -0.001459        0
4      -0.006077      0.000015      0.006975        0
5       0.449020      0.184570      1.028530        0
6      11.642800     14.592900     11.482200       -1
7       4.000000   2180.000000      1.000000       -1
8     -18.597000    -12.575000    -18.211000        1
9      22.548000     26.573000     24.319000        2
10    -19.379000    -13.424000    -18.699000        1
11    -19.547000    -13.894000    -18.672000        1
12     22.398000     25.832000     24.450000        2
13     21.555000     25.219000     24.549000        2
14    -19.636000    -14.218000    -18.675000        1
15     21.397000     24.883000     24.374000        2
16    -19.729000    -14.504000    -18.700000        1
17     21.290000     24.589000     23.993000        2

[18 rows x 3880 columns]
```

This shows us the dataset, it has 18 data point with 3880 features. We also saw the assigned clusters for each of one.

The second result:

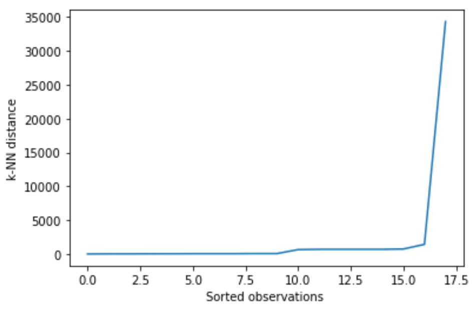
Cluster Counts:

```
   Cluster
-1      18
```



We can observe that there is only 1 cluster with 18 data points. In our case it’s seem that all the data point are identified as outliers.

The last result:



This graph shows the distance to the nearest k-th neighbour. When y=0 there is a dense region of data point in the clusters but after x=16, the number increase and there is less point near each other int the clusters.

Conclusion

In this project, we explored various clustering methods to understand their functionality. We analysed our dataset and evaluated each algorithm's performance based on factors such as cluster shapes, sizes, and noise tolerance. The results emphasized the importance of selecting appropriate algorithms based on the data's nature. Overall, this project contributes to the broader understanding of clustering techniques and their practical applications. It provides valuable insights for data analysis and pattern recognition in diverse domains.