Data Science II Assignment Individual

WS 2025

Bachelor: Mechatronik, Design & Innovation

5^{th} Semester

Lecturer: Daniel T. McGuiness, Ph.D

Groupe: BA-MECH-23-MLDS

Author: Lukas Heiss

January 20, 2026

Contents

1 Introduction

This assignment addresses the processing of the Olivetti Faces dataset using a combination of unsupervised and supervised learning methods. Starting with a basic K-Means clustering for structure analysis in section Q1 2, a further optimized pipeline with decision trees is developed, whose ideal parameters are determined through a systematic silhouette analysis in section Q2 3. The conclusion involves the use of principal component analysis (PCA) for efficiency improvement, as well as the application of Gaussian Mixture Models (GMM) to evaluate the generative properties of the model in creating new faces and detecting image anomalies in section Q3 4.

2 Q1 - Visualising Olivetti Faces

The first part of this project, the Olivetti Faces dataset is loaded and examined. The goal is to apply the K-Means algorithm to divide the 400 images into 40 clusters based on their pixel data, which corresponds to the number of 40 people contained in the dataset. The analysis focuses on how well the algorithm recognizes similarities in facial features. Finally, the distribution of images across the individual clusters is evaluated in order to assess and understand the grouping of the clusters.

2.1 CODE

Listing 1: Q1 - Olivetti Faces Clustering with K-Means

```
from sklearn.datasets import fetch_olivetti_faces
from sklearn.cluster import KMeans
from sklearn.discriminant_analysis import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# +++++++++++++++++++++++++++ Q1 - Olivetti Faces Clustering with K-Means
    +++++++++++++++++++++++++++
"""
We load the Olivetti faces dataset, and use K-Means for clustering. We
    also visualize the faces within a
selected cluster. Then, we count and print how many images are in each
    cluster.
"""

# --- loads Olivetti DS ---
data_faces = fetch_olivetti_faces()
X = data_faces.data
y = data_faces.target

# --- Split dataset in training and test set ---
# stratify mixes the classes in both sets equally, so that all persons
    are represented in both sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    stratify=y,
    test_size=0.2,
    random_state=42
)

#--- Model with k-means clustering ---
# 400 images of 40 different persons = 40 clusters
kmeans = KMeans(n_clusters=40, random_state=42)
kmeans.fit(X_train)

y_pred = kmeans.predict(X_test)
accuracy_Kmeans = accuracy_score(y_test, y_pred)
print(f"Kmeans accuracy: {accuracy_Kmeans:}")


# --- Function to plot the faces within a cluster ---
var = 0  # Cluster number to visualize
cluster_labels = kmeans.labels_
clusters = X_train[cluster_labels == var]


```

```python
# --- Counts how many images are in each cluster ---
cluster_counts = np.bincount(kmeans.labels_)
for i, count in enumerate(cluster_counts):
    print(f"Cluster {i} has {count} Images")

# --- Ploting the pictures in cluster var ---
plt.figure(figsize=(10, 2))
for i in range(min(20, len(clusters))):
    plt.subplot(1, 20, i + 1)
    plt.imshow(clusters[i].reshape(64, 64), cmap='gray')
    plt.axis('off')
plt.show()
```

2.2 RESULTS

The K-Means algorithm achieved an accuracy of 0.0625. The distribution of images across the 40 identified clusters is summarized in Table 2.1 and the pictures of the faces in the first cluster (Cluster 0) are shown in the Image 2.1.

Table 2.1: Distribution of Images per Cluster

Cluster	Count	Cluster	Count	Cluster	Count
Cluster 0	14	Cluster 14	17	Cluster 28	9
Cluster 1	11	Cluster 15	4	Cluster 29	9
Cluster 2	14	Cluster 16	3	Cluster 30	2
Cluster 3	14	Cluster 17	4	Cluster 31	11
Cluster 4	6	Cluster 18	4	Cluster 32	4
Cluster 5	8	Cluster 19	8	Cluster 33	8
Cluster 6	11	Cluster 20	5	Cluster 34	17
Cluster 7	5	Cluster 21	13	Cluster 35	5
Cluster 8	10	Cluster 22	6	Cluster 36	6
Cluster 9	8	Cluster 23	5	Cluster 37	5
Cluster 10	16	Cluster 24	12	Cluster 38	7
Cluster 11	4	Cluster 25	2	Cluster 39	2
Cluster 12	7	Cluster 26	6		
Cluster 13	8	Cluster 27	10		

Figure 2.1: Plot of the Faces in the first Cluster.

3 Q2 - Clustering the Faces

In the second section, the focus is on supervised learning to improve the classification of the Olivetti Faces. First, a Decision Tree Classifier is trained directly on the image data to measure accuracy in person recognition. Then, a pipeline is implemented that combines K-Means clustering as a preprocessing step with the Decision Tree. The core of this investigation lies in the systematic silhouette analysis, in which various cluster sizes ($k = 20$ to 160) were tested to determine the optimal size. The implementation is based on the methodological documentation of Scikit-Learn [1], whereby the use of the silhouettescore enabled an objective assessment of cluster quality (see table 3.1, which was also illustrated in the plot 3.1.

3.1 CODE

Listing 2: Q1 - Olivetti Faces Clustering with K-Means

```
from sklearn.datasets import fetch_olivetti_faces
from sklearn.cluster import KMeans
from sklearn.discriminant_analysis import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.pipeline import make_pipeline


# --- loads Olivetti DS ---
data_faces = fetch_olivetti_faces()
X = data_faces.data
y = data_faces.target

# --- Split dataset in training and test set ---
# stratify mixes the classes in both sets equally, so that all persons
    are represented in both sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    stratify=y,
    test_size=0.2,
    random_state=42
)

#--- Model with k-means clustering ---
# 400 images of 40 different persons = 40 clusters
kmeans = KMeans(n_clusters=40, random_state=42)
kmeans.fit(X_train)

y_pred = kmeans.predict(X_test)
accuracy_Kmeans = accuracy_score(y_test, y_pred)
print(f"Kmeans accuracy: {accuracy_Kmeans:}")



# +++++++++++++++++++++++++++ Q2 - Olivetti Faces using DT
    +++++++++++++++++++++++++++
"""
Training a Decision Tree Classifier on the Olivetti faces dataset to
    classify images of different persons.
Select the best cluster size for K-Means using silhouette score analysis.
"""

```

```
# --- DT Classifier ---
# max_leaf_nodes = 40 because, 40 diff. peaople
dt_Classifier = DecisionTreeClassifier(random_state=42, max_leaf_nodes=40)

# --- Train the model ---
dt_Classifier.fit(X_train, y_train)

# --- Predict on test set ---
y_pred_dt = dt_Classifier.predict(X_test)

# --- Calculate accuracy ---
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"DT accuracy: {accuracy_dt:}")

# --- Decision Tree Classifier Pipeline ---
DTKmeans_Pipeline = make_pipeline(
     KMeans(n_clusters=40, random_state=42),
     DecisionTreeClassifier(random_state=42)
)

# --- Train the pipeline ---
DTKmeans_Pipeline.fit(X_train, y_train)

# --- Predict Pipeline ---
y_pred_DTKmeans = DTKmeans_Pipeline.predict(X_test)
accuracy_DTKmeans = accuracy_score(y_test, y_pred_DTKmeans)
print(f"DT Kmeans Pipelime accuracy: {accuracy_DTKmeans:}")

from sklearn.metrics import silhouette_score

# --- Test via silhouette score to find the best cluster size for this DS ---
# for more Infos: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py
k_range = range(20, 161, 10) # Test 20, 30, ..., 160 clusters sizes
silhouette_scores = []

for k in k_range:
    kmeans_test = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans_test.fit_predict(X_train)

    # Callculate the score
    score = silhouette_score(X_train, labels)
    silhouette_scores.append(score)
    print(f"The Silhouette Score of k={k}: {score:.4f}")

# --- Plot thee silhouett scores ---
plt.figure(figsize=(8, 4))
plt.plot(k_range, silhouette_scores, "bo-")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Silhouette Score")
plt.title("Determine the Optimal k")
plt.grid(True)
plt.show()
```

3.2 RESULTS

The results of the silhouette analysis, shown in Figure 2.1 and detailed in Table 3.1, identify $k = 130$ as the optimal number of clusters for the Olivetti face dataset. At this point, the silhouette score reaches its maximum of 0.2148, indicating the most effective separation of the different facial groups. This achieves an accuracy in the pipeline with a decision tree classifier of 0.625. The continuous increase in the silhouette plot shows that significantly more clusters than the original 40 individuals are required to better capture variations in lighting and head position, providing the classifier with more precise features for more accurate identification.

Table 3.1: Silhouette Scores for different Cluster Sizes (k)

Number of Clusters (k)	**Silhouette Score**	**Number of Clusters (k)**	**Silhouette Score**
$k = 20$	0.1144	$k = 100$	0.2046
$k = 30$	0.1255	$k = 110$	0.2023
$k = 40$	0.1422	$k = 120$	0.2007
$k = 50$	0.1558	$k = 130$	**0.2148**
$k = 60$	0.1691	$k = 140$	0.1971
$k = 70$	0.1845	$k = 150$	0.1930
$k = 80$	0.1895	$k = 160$	0.1825
$k = 90$	0.1965		

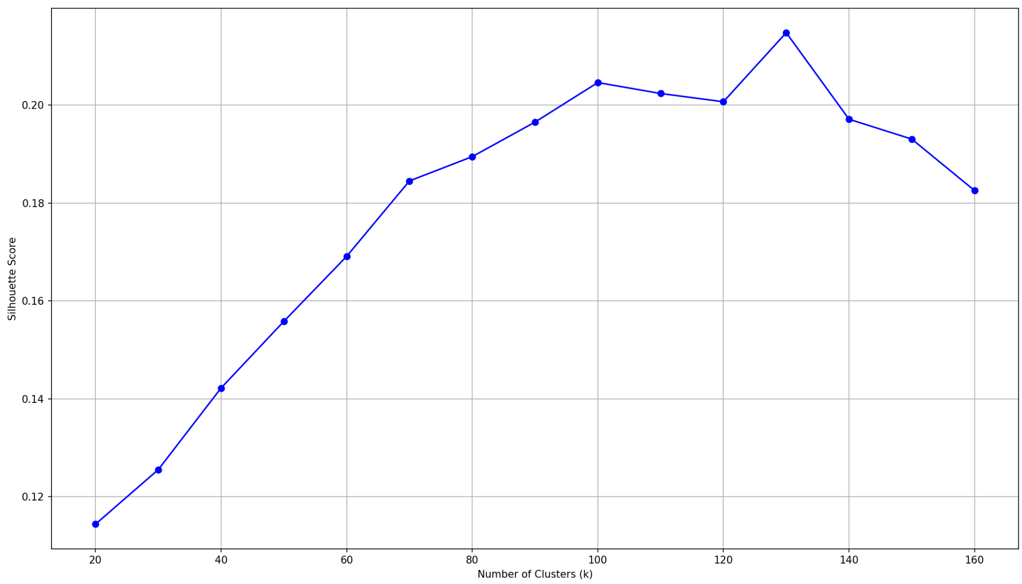

Figure 3.1: Plot of the Silhouette Score test.

4 Q3 - Gaussian Mixture Models

In the third and final part of the task, the focus is on improving the efficiency of the model as well as applying generative methods. Since working with high-dimensional image data (pixel values) is extremely demanding, principal component analysis (PCA) is used initially. The goal is to reduce the data in such a way that 95% of the variance is retained without losing significant information. On this reduced dataset, a Gaussian Mixture Model (GMM) is then implemented. Unlike K-means clustering, the GMM functions as a generative model, allowing the creation of new artificial faces, from the learned distribution. Another focus is on anomaly detection, which examines whether the model is capable of identifying manipulated images, such as an upside-down face, as anomalies based on atypical probability values.

The implementation of the Gaussian Mixture Model as well as the subsequent anomaly detection is based on the approaches of the notebook Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow [2] and also of the lecture book Data Science II [3]. In particular, the strategy of identifying anomalies through a threshold definition of the probability density was adopted in order to mathematically separate these anomaly cases from the normal ones.

4.1 CODE

Listing 3: Q1 - Olivetti Faces Clustering with K-Means

```
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture
import numpy as np


# +++++++++++++++++++++++++++ Q3 - Olivetti Faces Clustering with GMM
    +++++++++++++++++++++++++++
"""
We load the Olivetti faces dataset, apply PCA for dimensionality
    reduction,
and then use Gaussian Mixture Models (GMM) for clustering. PCA is used to
     reduce dimensionality
and therefore improve the time. We also generate new faces and perform
    anomaly detection by comparing
scores of normal and altered faces.

"""

# --- loads Olivetti DS ---
data_faces = fetch_olivetti_faces()
X = data_faces.data
y = data_faces.target


# --- Split dataset in training and test set ---
# stratify mixes the classes in both sets equally, so that all persons
    are represented in both sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    stratify=y,
    test_size=0.2,
    random_state=42
```

```
)

"""
PCA (Principal Component Analysis)
----------------------------------
Linear dimensionality reduction using Singular Value Decomposition of the
data to project it to a lower dimensional space.

Parameters
----------
n_components : float, we keep 95% of variance.
random_state : int = 42, 42 is the seed used by the random number generator.

"""
# --- PCA for dimensionality reduction ---
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

"""
GaussianMixture
-------------------
Gaussian Mixture Model for clustering.
Parameters
----------
n_components : int = 40, Number of mixture components.
random_state : int = 42, 42 is the seed used by the random number generator
"""
GaussianModel = GaussianMixture(n_components=40, random_state=42)

GaussianModel.fit(X_train_pca)
y_predict = GaussianModel.predict(X_test_pca)
accuracy_GMM = accuracy_score(y_test, y_predict)
print(f"GMM accuracy: {accuracy_GMM:}")

# --- Generate new faces ---
n_gen = 10
X_gen_pca, y_gen = GaussianModel.sample(n_samples=n_gen)
X_gen_original = pca.inverse_transform(X_gen_pca)

# gGenerated faces and plot the generated faces
plt.figure(figsize=(12, 3))
for i in range(n_gen):
    plt.subplot(1, n_gen, i + 1)
    plt.imshow(X_gen_original[i].reshape(64, 64), cmap='gray')
    plt.axis('off')
    plt.title(f"Gen {i+1}")
plt.suptitle("Generated Faces")
plt.show()

# Density of the faces
densities = GaussianModel.score_samples(X_test_pca)

# Determine threshold for anomaly detection
density_threshold = np.percentile(densities, 4)
# select a face to test
normal_face = X_test[0]
# Rotate the face
anomalous_face = np.flipud(normal_face.reshape(64, 64)).reshape(-1)

# Transform both faces using PCA
```

```python
test_faces_pca = pca.transform([normal_face, anomalous_face])

# Score both faces
test_scores = GaussianModel.score_samples(test_faces_pca)

# Print results
print(f"Threshold: {density_threshold:.2f}")
print(f"Normal face score: {test_scores[0]:.2f} -> Anomaly? {test_scores[0] < density_threshold}")
print(f"Anomalous face score: {test_scores[1]:.2f} -> Anomaly? {test_scores[1] < density_threshold}")

# Visualization
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title(f"Normal\nScore: {test_scores[0]:.2f}")
plt.imshow(normal_face.reshape(64, 64), cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title(f"Anomalous (Flipped)\nScore: {test_scores[1]:.2f}")
plt.imshow(anomalous_face.reshape(64, 64), cmap='gray')
plt.axis('off')

plt.show()
```

4.2 RESULTS

The results of the Gaussian Mixture Model show two very different aspects of model performance. The measured accuracy of 0.0125 is predictably low, as the GMM, like K-Means, is an unsupervised learning method. The model clusters faces based on statistical similarities without knowing the original assignment of individuals. The model demonstrates that it can generate images based on the data it has learned, this images are shown in 4.1.

Figure 4.1: Plot of the Faces generated with the GMM.

Another success of the model is evident in anomaly detection:

- The normal face achieves a score of about -**6532413.28**.
- The manipulated (inverted) face scores significantly lower, at about -**33042532.99**.

This massive difference in values of the following picture 4.2 demonstrates that the GMM has successfully learned the underlying structure of human faces. The inverted image is rated by the model as extremely unlikely and is thus reliably detected as an anomaly. This shows that despite the low classification accuracy, the GMM is a powerful tool for evaluating the data.

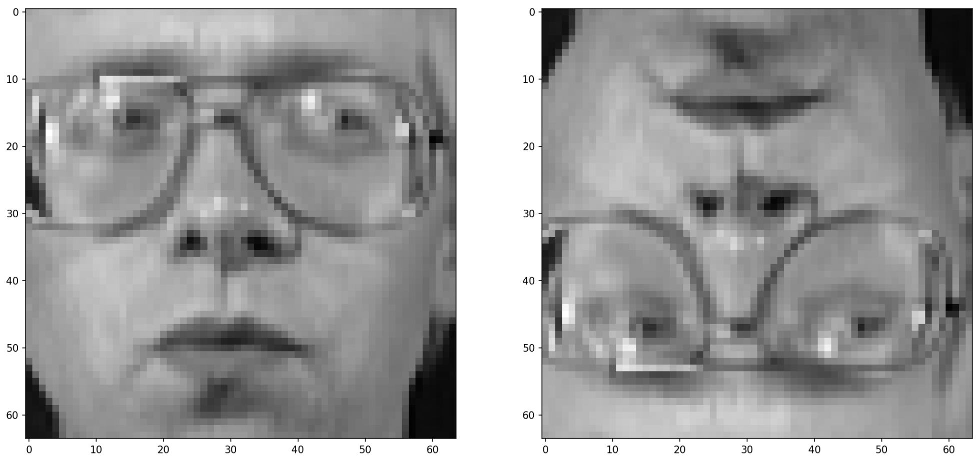

Figure 4.2: Plot of the 'normal' Faces and the Faces with the anomaly.

5 Conclusion

The analysis of the Olivetti Faces dataset demonstrates the effectiveness of combined machine learning methods. While Q1 introduced structural clustering using K-Means, Q2 was able to significantly improve classification accuracy in the pipeline through a systematic silhouette analysis. In Q3, anomaly detection using GMM and PCA was introduced. The complete source code and many more examples are available on GitHub at `https://github.com/luheiss/MachineLearning-DataScience_exercise`.

List of Figures

List of Tables

References

[1] scikit-learn developers, "scikit-learn," 2025. [Online]. Available: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py

[2] ageron, "Hands-on machine learning with scikit-learn, keras and tensorflow," 22. [Online]. Available: https://github.com/ageron/handson-ml2/blob/master/09_unsupervised_learning.ipynb

[3] P. Daniel T. McGuiness, "Data science ii," 2025. [Online]. Available: https://dtmc0945.github.io/L-MCI-BSc-Data-Science-II/DataScienceIILectureBookch5.html#x7-760005.3.1