Assignment 4

Gaussian Mixture Models

COMP257(SEC. 003)

Yoo Sun Song
301091906

1.Use PCA preserving 99% of the variance to reduce the dataset's dimensionality.

```
# load the Olivetti faces dataset
from sklearn.datasets import fetch_olivetti_faces

olivetti_faces = fetch_olivetti_faces(shuffle=True, random_state=96)
X = olivetti_faces.data

print('X Shape: ', X.shape)
```

→ X Shape: (400, 4096)

```
#1. Use PCA preserving 99% of the variance to reduce the dataset's dimensionality.
from sklearn.decomposition import PCA

pca = PCA(n_components=0.99, whiten=True)
X_pca = pca.fit_transform(X)

print('X Shape: ', X_pca.shape)
```

→ X Shape: (400, 260)

Principal Component Analysis (PCA) is used to perform dimensionality reduction on the dataset X. By setting n_components to 0.99, the PCA transformation retains 99% of the variance in the data. The whiten=True parameter ensures that the principal components are uncorrelated and have unit variance. As a result, the transformed dataset, X_pca, has been reduced to 260 principal components from its original number of features while still preserving most of the information. This indicates that the majority of the dataset's variance can be captured in these 260 components. Reducing dimensionality in this manner can help speed up algorithms and remove noise from the data.

2.Determine the most suitable *covariance_type* for the dataset.

Print the results for the current covariance type

```
print('\nCovariance_type: ', cov_type)
print('AIC: ', gmm.aic(X_pca))
print('BIC: ', gmm.bic(X_pca))
print('------------------------------')
```

→ Result:

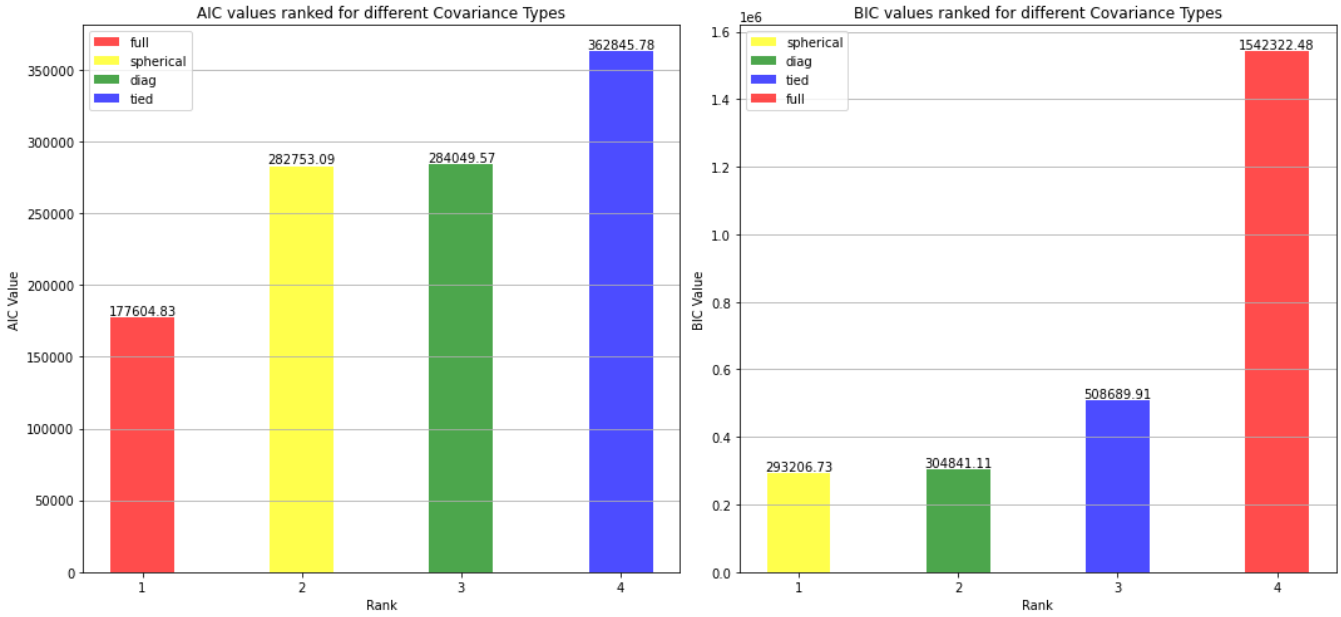
```
Covariance_type:  full
AIC:  177604.82619827567
BIC:  1542322.4780354185
------------------------------

Covariance_type:  tied
AIC:  362845.7843216754
BIC:  508689.907408454
------------------------------

Covariance_type:  diag
AIC:  284049.5673960773
BIC:  304841.1062219628
------------------------------

Covariance_type:  spherical
AIC:  282753.08775213757
BIC:  293206.73340101336
------------------------------
```

Covariance Types:

① full: Each component has its own general covariance matrix.
② tied: All components share the same general covariance matrix.
③ diag: Each component has its own diagonal covariance matrix.
④ spherical: Each component has its own single variance.

Analyzing the results, the full covariance type has the lowest AIC value. A lower AIC indicates a better model fit. However, in terms of BIC, spherical presents the lowest value.

Rank covariance types based on AIC and BIC values.



The bar charts visualize the AIC and BIC values ranked for different covariance types.

① For AIC, the 'full' covariance type demonstrates the lowest value, suggesting it's the most optimal choice according to this criterion.
② In contrast, the 'tied' covariance type has the highest AIC, implying it might be less suited for the given data.
③ Examining the BIC chart, the 'spherical' covariance type scores the lowest, suggesting it may be the best fit per this metric.
④ Conversely, the 'full' covariance type shows an exceedingly high BIC value, indicating potential overfitting or an overly complex model.

Determines the best covariance type for Gaussian Mixture Models based on AIC and BIC values.

```
print(f"Best covariance type based on AIC: {best_covariance_types_aic}")
print(f"Best covariance type based on BIC: {best_covariance_types_bic}")
```

→

Best covariance type based on AIC: full
Best covariance type based on BIC: spherical

The results indicate a discrepancy between the AIC and BIC criteria for determining the best covariance type for the Gaussian Mixture Model:

① According to AIC, the 'full' covariance type is deemed optimal, emphasizing a better fit to the data.
② On the other hand, BIC favors the 'spherical' covariance type, suggesting it provides a balance between model fit and complexity.

3.Determine the minimum number of clusters that best represent the dataset using either AIC or BIC.

The best number of clusters for AIC

```
print(f"Best number of clusters according to AIC: {best_n_clusters_aic}")
```

→ Best number of clusters according to AIC: 8

The best number of clusters for BIC

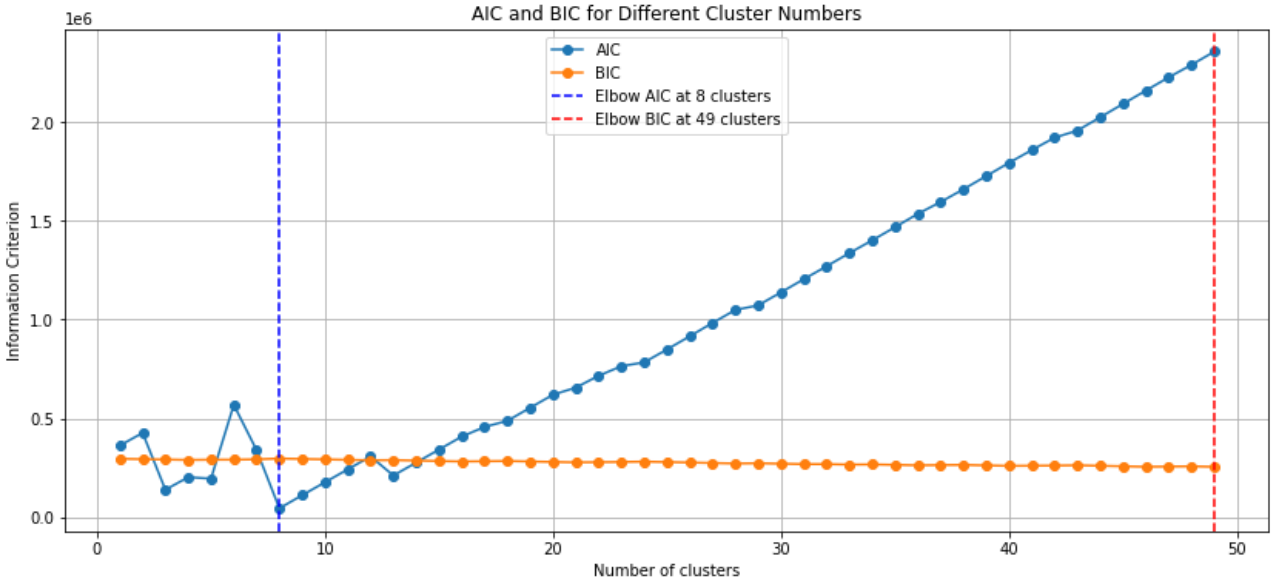
```
print(f"Best number of clusters according to BIC: {best_n_clusters_bic}")
```

→ Best number of clusters according to BIC: 49

The results reveal differing recommendations on the optimal number of clusters for the data:

① According to AIC, the best model has 8 clusters, suggesting this number provides a better fit to the data without being overly complex.
② BIC recommends a model with 49 clusters, implying that despite its increased complexity, this model is more suitable.

4.Plot the results from (2) and (3).



The graph visualizes the AIC and BIC values for varying cluster numbers.

① AIC has a noticeable elbow at 8 clusters, indicating an optimal point before diminishing returns in added complexity.
② However, BIC shows a steady increase but has a marked inflection at 49 clusters, suggesting this might be an optimal number of clusters according to BIC.
③ The difference in the elbow points for AIC and BIC indicates a trade-off between model simplicity and data fit.
④ While AIC emphasizes a more parsimonious model with fewer clusters, BIC suggests that a more complex model with more clusters might be beneficial.

5.Output the hard clustering for each instance.

Using the BIC optimal number of clusters

print("Hard clustering labels for each instance (Using BIC):", hard_clusters_bic)

→ Hard clustering labels for each instance (Using BIC): [28 4 10 4 2 4 10 10 28 42 10 28 3 5 4 5 28 10 10 10 5 47 4 2

12 27 10 28 4 4 12 4 36 10 28 23 2 28 42 23 22 5 10 4 10 10 10 10

47 10 4 1 10 3 4 10 28 12 40 10 3 31 4 10 6 6 28 3 4 42 47 10

28 6 4 21 22 42 13 19 28 4 4 10 31 0 36 23 4 10 28 28 4 19 42 4

5 18 5 10 4 10 10 4 24 17 5 19 4 10 42 4 4 10 8 10 23 48 37 34

4 3 10 10 10 28 10 42 30 19 4 9 4 5 4 5 12 10 10 42 4 23 10 22

5 4 42 5 41 14 23 10 4 4 5 4 4 10 4 10 10 10 10 10 10 4 4 28

33 3 10 3 28 10 5 10 19 42 43 10 28 5 12 4 23 23 12 3 10 4 10 10

10 10 3 19 35 4 4 10 28 10 10 10 10 10 19 3 23 10 10 10 46 23 10 5

5 5 10 4 5 26 5 28 28 28 10 42 3 10 23 10 18 46 10 19 10 42 10 3

37 10 10 28 22 28 10 10 28 3 10 42 4 3 7 42 10 3 5 10 4 18 10 19

10 5 4 12 3 10 42 23 10 10 28 18 23 10 10 28 21 28 20 4 10 3 10 4

10 10 10 2 10 1 10 4 5 4 4 28 10 19 4 33 4 10 32 4 28 5 3 28

42 42 42 1 10 10 3 10 4 5 4 4 10 4 15 10 4 10 4 5 12 4 10 40

10 10 10 42 42 10 42 12 10 11 12 4 42 28 3 44 4 22 5 39 42 4 28 4

36 29 28 1 4 16 23 5 42 36 47 45 42 21 4 10 10 10 4 18 28 5 10 38

4 42 10 4 10 23 41 5 10 25 3 4 10 5 4 5]

The output showcases the hard clustering labels for each instance based on the BIC criteria. The labels range from 0 to 48, consistent with the optimal number of clusters (49) determined by BIC. Each number represents a specific cluster to which an instance has been assigned. For example, the first instance belongs to cluster 28, the second to cluster 4, and so forth. Observing the frequency and distribution of these labels can provide insights into the density and characteristics of each cluster. It's evident that certain clusters, like 10 and 4, appear frequently, which could imply they encompass a broader or more general set of instances compared to others.

Using the AIC optimal number of clusters

print("Hard clustering labels for each instance (Using AIC):", hard_clusters_aic)

→Hard clustering labels for each instance (Using AIC): [5 5 4 5 2 5 6 6 4 5 5 4 4 6 4 4 4 4 6 5 6 4 6 2 6 4 5 5 6 4 5 6 5 4 5 4 2

5 4 4 4 6 4 5 4 5 3 5 4 3 5 1 5 5 5 6 4 6 5 5 4 5 5 5 7 7 4 4 5 5 5 5 4 7

5 4 5 5 6 5 3 5 6 5 5 0 5 5 5 4 5 6 5 5 5 5 6 6 6 6 6 5 6 6 4 5 4 5 5 5 6

4 6 5 6 6 4 6 5 5 6 4 4 4 4 4 5 4 5 3 5 5 4 3 6 6 5 6 5 4 6 6 5 4 4 5 5 6

5 6 3 4 5 5 6 6 6 5 5 5 5 5 4 5 4 6 4 5 5 5 6 4 5 4 6 4 5 5 6 6 4 5 4 4 4

4 4 4 4 5 4 6 3 5 4 6 5 4 5 4 4 5 5 3 5 5 4 6 4 6 6 5 6 6 5 6 6 6 6 5 4 5 6

5 4 3 5 5 4 4 5 4 4 5 5 5 4 6 6 5 4 5 6 5 5 5 6 6 5 5 6 4 5 5 5 5 4 5 1 4 6

5 5 5 5 5 6 6 5 5 4 6 4 5 6 4 5 6 5 6 6 4 4 5 6 5 4 5 4 6 4 5 4 2 5 1 4 5

4 4 4 6 6 5 6 4 4 3 4 6 6 5 5 5 5 6 4 1 5 5 4 4 6 6 4 6 4 5 4 6 5 5 6 4 5

5 6 4 4 4 5 6 4 6 5 3 5 6 3 5 4 4 4 5 4 5 5 6 6 5 6 4 5 5 5 1 5 5 6 5 4 6

3 5 5 5 5 5 5 4 5 5 5 6 5 4 4 4 5 6 4 6 4 6 5 6 4 6 6 5 5 6]

The output showcases the hard clustering labels for each instance when utilizing the AIC criteria. The labels range from 0 to 7, implying an optimal choice of 8 clusters as informed by AIC. Each integer represents a particular cluster that an instance is classified into. For example, the first instance is classified into cluster 5, the next also to cluster 5, and so on. By examining the distribution and frequency of these labels, one can gather insights into the nature and density of each cluster. Notably, clusters like 5 and 4 appear recurrently, suggesting that they might represent more generalized or dominant patterns within the data. In contrast, others such as 0 or 7 might represent more niche or specific patterns.

6.Output the soft clustering for each instance.

Using the BIC optimal number of clusters

```
print("Soft clustering probabilities for each instance (Using BIC):", soft_clusters_bic)
```

→ Soft clustering probabilities for each instance (Using BIC): [[0.00000000e+000 4.10059641e-052 4.69223644e-080 ... 2.91833678e-241

8.32508616e-077 0.00000000e+000]

[0.00000000e+000 6.91873316e-054 7.22908059e-086 ... 9.43220276e-249

6.96378820e-077 0.00000000e+000]

[0.00000000e+000 2.57340983e-037 2.53778991e-052 ... 4.78663111e-192

4.52627438e-051 0.00000000e+000]

...

[0.00000000e+000 3.39221683e-036 1.94699423e-056 ... 7.86300231e-203

7.34141887e-054 0.00000000e+000]

[0.00000000e+000 2.21847140e-062 6.44746206e-093 ... 3.68005132e-267

1.85160862e-085 0.00000000e+000]

[0.00000000e+000 5.39486446e-034 2.03322781e-045 ... 3.96334780e-188

4.45260877e-050 0.00000000e+000]]

The output depicts the soft clustering probabilities for each instance based on the BIC criteria. In soft clustering, instead of assigning each data point to a specific cluster definitively, a probability or degree of membership to each cluster is given. From the output, we observe very small probabilities, many close to zero, indicating a high certainty in the clustering for many instances. The presence of values like 0.00000000e+000 suggests certain clusters have almost no likelihood of containing specific data points. Conversely, the non-zero probabilities denote the degree of membership of an instance to the respective clusters. In general, this probabilistic approach provides a nuanced understanding, allowing us to ascertain not just where data points likely belong, but also how confidently they fit within those clusters.

Using the AIC optimal number of clusters

```
print("Soft clustering probabilities for each instance (Using AIC):", soft_clusters_aic)
```

→ Soft clustering probabilities for each instance (Using AIC): [[0. 0. 0. ... 1. 0. 0.]

[0. 0. 0. ... 1. 0. 0.]

[0. 0. 0. ... 0. 0. 0.]

...

[0. 0. 0. ... 1. 0. 0.]

[0. 0. 0. ... 1. 0. 0.]

[0. 0. 0. ... 0. 1. 0.]]

The soft clustering probabilities for each instance using AIC are starkly different from those using BIC. For AIC, the probabilities are much more decisive, often showing clear membership to one cluster with values of 1, and 0 for others. This indicates a high certainty in cluster assignments when AIC is used. In contrast, the BIC-based probabilities are distributed over a range, with many minuscule values and some closer to 1, indicating less certainty in some assignments. This shows that BIC offers a more probabilistic approach, highlighting the degree of confidence in each cluster assignment. Thus, while AIC provides a definitive clustering, BIC offers a more nuanced perspective on the clustering probabilities.

7.Use the model to generate some new faces (using the *sample()* method), and visualize them (use the *inverse_transform()* method to transform the data back to its original space based on the PCA method used).

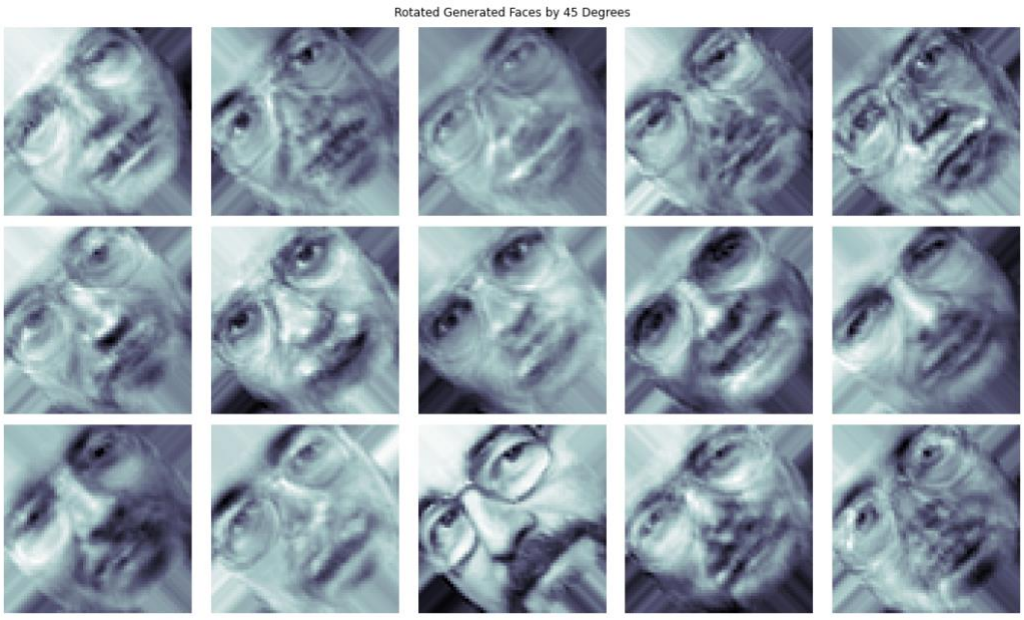
```
# Number of new faces to be generated
num_new_faces = 15

# Use the sample() method of the Gaussian Mixture Model to generate new samples (new faces in PCA space) Using BIC
new_samples, _ = gmm_best_bic.sample(n_samples=num_new_faces)

# Transform the samples from PCA space back to the original space using the inverse_transform() method of the PCA
new_faces = pca.inverse_transform(new_samples)

# Plotting the generated faces
plt.figure(figsize=(15, 9))

# Loop through the generated faces and plot them
for i, face in enumerate(new_faces):
    plt.subplot(3, 5, i + 1)
    plt.imshow(face.reshape(64, 64), cmap='bone')  # Reshape the face data to 64x64 and display as an image
    plt.axis('off')  # Hide the axis

plt.tight_layout()

plt.show()
```

The code snippet provided appears to deal with the generation of new facial images using a Gaussian Mixture Model (GMM) trained on face data that has been processed with Principal Component Analysis (PCA). After training the GMM with the BIC criterion, 15 new face samples are generated in the reduced PCA space. These generated samples, which represent faces in a compressed form, are then transformed back into the original data space, resulting in a set of novel face images. These images are then visualized using Matplotlib. The accompanying image showcases an array of faces, possibly a demonstration of the synthesized outputs, which are quite diverse yet still possess discernible facial features.

8.Modify some images (e.g., rotate, flip, darken).

Rotate the new faces generated by 45 degrees without reshaping them

```
from scipy.ndimage import rotate
import cv2
rotated_images = [rotate(image.reshape(64, 64), 45, reshape=False, mode='nearest') for image in new_faces]
plt.figure(figsize=(15, 9))
for index, face in enumerate(rotated_images):
    plt.subplot(3, 5, index + 1)
    plt.imshow(face, cmap='bone')
    plt.axis('off')
plt.suptitle('Rotated Generated Faces by 45 Degrees')
plt.tight_layout()
plt.show()
```



The code applies a rotation transformation to previously generated facial images. Utilizing the scipy.ndimage.rotate function, each image is rotated 45 degrees without changing its dimensions. The mode argument is set to 'nearest' to handle edge boundaries. After rotation, the processed images are displayed in a grid layout using Matplotlib, as evidenced by the provided image. The visual grid showcases an array of faces titled "Rotated Generated Faces by 45 Degrees", demonstrating the results of this transformation.

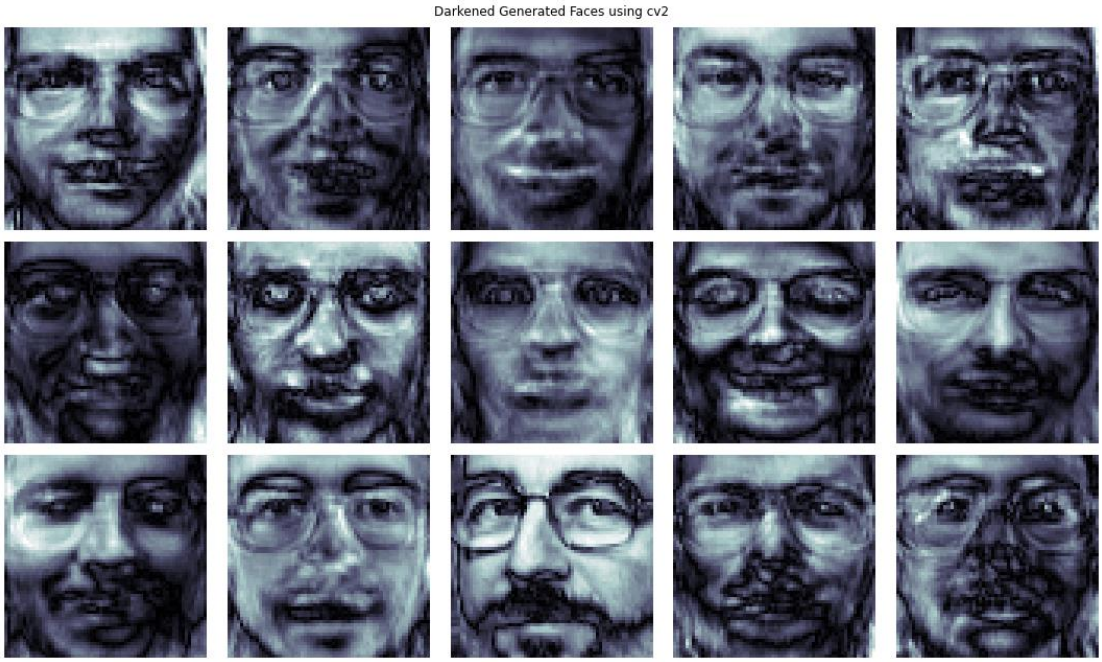
Flip the new faces horizontally.

```
flipped_images = [np.fliplr(image.reshape(64, 64)) for image in new_faces]
plt.figure(figsize=(15, 9))
for index, face in enumerate(flipped_images):
    plt.subplot(3, 5, index + 1)
    plt.imshow(face, cmap='bone')
    plt.axis('off')
plt.suptitle('Flipped Generated Faces')
plt.tight_layout()
plt.show()
```

The code snippet takes a series of generated facial images and horizontally flips each image using the np.fliplr function. This operation mirrors the image about its vertical axis, giving the appearance of a left-to-right inversion. After the flipping process, the images are visualized in a grid layout using Matplotlib. The output display, titled "Flipped Generated Faces", presents a compilation of these horizontally flipped faces, underscoring the impact of this basic image transformation.

Darken the images.

```
scaled_images = [image.reshape(64, 64) * 255 for image in new_faces]
darkened_images = [cv2.convertScaleAbs(image, alpha=0.5, beta=-50) for image in scaled_images]
plt.figure(figsize=(15, 9))
for index, face in enumerate(darkened_images):
    plt.subplot(3, 5, index + 1)
    plt.imshow(face, cmap='bone')
    plt.axis('off')
plt.suptitle('Darkened Generated Faces using cv2')
plt.tight_layout()
plt.show()
```



The code manipulates a set of generated facial images, specifically darkening them with the help of the OpenCV library. First, each image's pixel values are rescaled by multiplying them by 255, a process often needed when handling normalized image data. Then, the cv2.convertScaleAbs function is employed to lessen the images' brightness by adjusting the alpha and beta values; the provided values of alpha=0.5 and beta=-50 collectively reduce the brightness. The end result, titled "Darkened Generated Faces using cv2", displays these darkened facial images in a grid format, demonstrating the effect of this particular transformation.

9.Determine if the model can detect the anomalies produced in (8) by comparing the output of the *score_samples()* method for normal images and for anomalies).

Using AIC

```
# Compute the log likelihood of each sample in the original images using the best GMM model trained with AIC
original_scores = gmm_best_aic.score_samples(pca.transform(new_faces.reshape(num_new_faces, -1)))

# Flatten the modified images to convert them from 2D (64x64) to 1D (4096)
rotated_2d = np.array([img.flatten() for img in rotated_images])
flipped_2d = np.array([img.flatten() for img in flipped_images])
darkened_2d = np.array([img.flatten() for img in darkened_images])

# Transform these flattened images back into the PCA space
rotated_pca = pca.transform(rotated_2d)
flipped_pca = pca.transform(flipped_2d)
darkened_pca = pca.transform(darkened_2d)

# Compute the log likelihood of each sample in the modified images using the best GMM model trained with AIC
rotated_scores = gmm_best_aic.score_samples(rotated_pca)
flipped_scores = gmm_best_aic.score_samples(flipped_pca)
darkened_scores = gmm_best_aic.score_samples(darkened_pca)

# Display the average log likelihoods for each set of images
print("Average score for original images:", np.mean(original_scores))
print("Average score for rotated images:", np.mean(rotated_scores))
print("Average score for flipped images:", np.mean(flipped_scores))
print("Average score for darkened images:", np.mean(darkened_scores))
```

→

```
Average score for original images: -46416483.96538188
Average score for rotated images: -124248889.83814168
Average score for flipped images: -44817787.35156987
Average score for darkened images: -840932824663.3296
```

① The Gaussian Mixture Model (gmm_best_aic) reveals that the original and flipped images have comparable likelihood scores, suggesting that flipping doesn't significantly alter the inherent characteristics recognized by the GMM.

② The rotated images have a notably lower average score, implying a substantial change in features due to rotation, making them less consistent with the GMM's expectations.

③ The average score for darkened images is exponentially lower than the others, indicating that darkening profoundly impacts the features, making them highly inconsistent with the original data the GMM was trained on.

Using BIC

```
original_scores = gmm_best_bic.score_samples(pca.transform(new_faces.reshape(num_new_faces, -1)))

# Flatten the modified images to convert them from 2D (64x64) to 1D (4096)
rotated_2d = np.array([img.flatten() for img in rotated_images])
flipped_2d = np.array([img.flatten() for img in flipped_images])
darkened_2d = np.array([img.flatten() for img in darkened_images])

# Transform these flattened images back into the PCA space
rotated_pca = pca.transform(rotated_2d)
flipped_pca = pca.transform(flipped_2d)
darkened_pca = pca.transform(darkened_2d)

# Compute the log likelihood of each sample in the modified images using the best GMM model trained with
BIC
rotated_scores = gmm_best_bic.score_samples(rotated_pca)
flipped_scores = gmm_best_bic.score_samples(flipped_pca)
darkened_scores = gmm_best_bic.score_samples(darkened_pca)

# Display the average log likelihoods for each set of images
print("Average score for original images:", np.mean(original_scores))
print("Average score for rotated images:", np.mean(rotated_scores))
print("Average score for flipped images:", np.mean(flipped_scores))
print("Average score for darkened images:", np.mean(darkened_scores))
```

① Using the gmm_best_bic, the original images have the highest average likelihood score, suggesting that they align most closely with the characteristics of the original dataset on which the GMM was trained.

② The scores for rotated and flipped images are lower, indicating that these transformations lead to variations in the data that the GMM identifies as less typical, with rotation causing a more significant deviation than flipping.

③ The darkened images have an extremely low score, demonstrating that darkening drastically changes the recognized features, making them vastly different from what the GMM trained on and, thus, highly atypical.

Github link: https://github.com/yooSunChaCha07/Gaussian-Mixture-Models.git