Face Recognition

Yahia Ibrahim AlKaranshawy, Hossam Osama Iraqi, Ali Hassan Ali

Abril 29, 2025

1 Problem statement

It's required to implement a PCA, kmeans model and GMM model from scratch and compare the performance of the two approaches with the PCA using AT&T Database of Faces.

1.1 Dataset Loading and Preprocessing

We used the **ORL face dataset** (also known as the AT&T face dataset), which contains:

- **40 subjects**, each with **10 grayscale facial images**
- Each image has a resolution of **92×112 pixels**, resulting in **10304 features** when flattened

The dataset was loaded using OpenCV (cv2). Each image was converted to grayscale and flattened into a 1D vector of size 10304. These vectors were stacked to form the **data matrix** D, and corresponding subject IDs (1 to 40) were stored in the **label vector** y.

1.2 Dataset Splitting

To ensure balanced training and testing across all subjects, we split the dataset using the following strategy:

- For each subject (10 images), we selected:
  - **5 images** for **training** → **odd-indexed images** (1st, 3rd, 5th, 7th, 9th)
  - **5 images** for **testing** → **even-indexed images** (2nd, 4th, 6th, 8th, 10th)

This resulted in:

- **Training set**: 200 samples (5 per subject × 40 subjects)
- **Testing set**: 200 samples (5 per subject × 40 subjects)

2 PCA

implemented PCA **from scratch** to reduce the dimensionality of the facial dataset.

PCA Process:

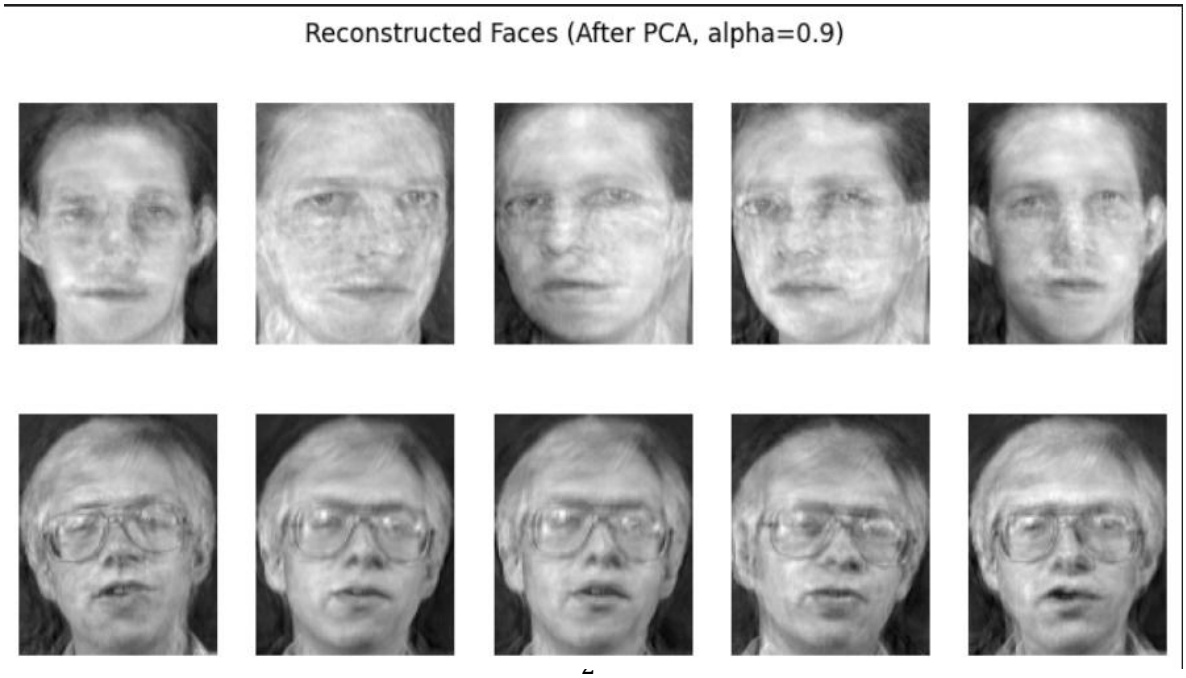
1. **Center the data** by subtracting the mean.
2. **Compute covariance matrix** of the centered data.
3. **Compute eigenvalues and eigenvectors** of the covariance matrix.
4. **Sort** them in descending order of eigenvalues.

5. **Choose the number of components** to retain a desired percentage α of the total variance.

Results for Different α (Variance Thresholds)

α (Variance Retained)	Number of Components
0.80 (80%)	36
0.85 (85%)	52
0.90 (90%)	76
0.95 (95%)	115

After PCA



3 GMM

This implementation fits a **Gaussian Mixture Model (GMM)** using the **Expectation-Maximization (EM)** algorithm.

Model Equation:

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

Steps:

1. Initialization

- Means μ_k: via KMeans
- Weights π_k: proportional to cluster sizes
- Covariances Σ_k: empirical with regularization

2. E-Step (Expectation):

Compute responsibilities:

$$r_{ik} = \frac{\pi_k \cdot \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_i|\mu_j, \Sigma_j)}$$

(log-sum-exp used for numerical stability)

3. M-Step (Maximization):

Update parameters using the computed responsibilities:

Weights:

$$\pi_k = \frac{1}{N} \sum_{i=1}^{N} r_{ik}$$

Means:

$$\mu_k = \frac{\sum_{i=1}^{N} r_{ik} x_i}{\sum_{i=1}^{N} r_{ik}}$$

Covariances:

$$\Sigma_k = \frac{\sum_{i=1}^{N} r_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{N} r_{ik}} + \text{reg}_{\text{covar}}$$

4. Convergence Check:

Stop if log-likelihood change is below tolerance:

$$\log p(X) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i | \mu_k, \Sigma_k) \right)$$

4 Bonus

As part of the bonus objective, we implemented a **simple autoencoder** to perform **nonlinear dimensionality reduction** on the facial dataset, followed by **K-Means** and **Gaussian Mixture Model (GMM)** clustering on the learned latent representations.

Autoencoder Architecture

The autoencoder was implemented using **PyTorch** and consists of:

- **Input layer:** 10304 neurons (92×112 pixel images)
- **Encoder:**
    - Linear(10304 → 1024) → ReLU

- Linear(1024 → 256) → ReLU
- Linear(256 → 50) → Latent space (bottleneck)

- **Decoder (mirror structure):**
  - Linear(50 → 256) → ReLU
  - Linear(256 → 1024) → ReLU
  - Linear(1024 → 10304) → Sigmoid (to normalize output to [0, 1])

The model was trained using **Mean Squared Error (MSE) loss** and **Adam optimizer** for 100 epochs on the training set.

Bottleneck Feature Extraction

Once trained, we used the encoder part of the autoencoder to extract **50-dimensional latent features** from the input data. These features capture essential facial characteristics while filtering out noise and redundancy.

Clustering in Latent Space

Using the 50D encoded features, we applied:

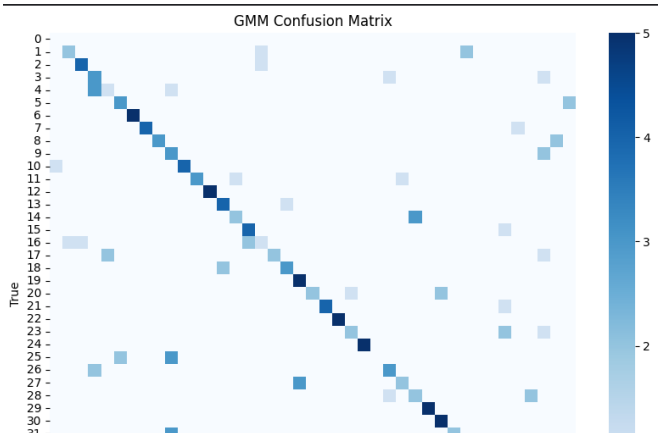
- **K-Means clustering** with k=40 (matching the number of subjects)
- **GMM clustering** with n_components=40

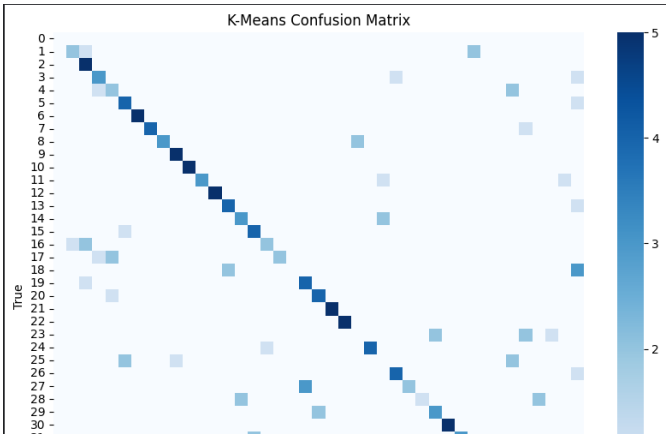
Both models were evaluated using **clustering accuracy** by aligning predicted clusters to true labels via the **Hungarian algorithm**.

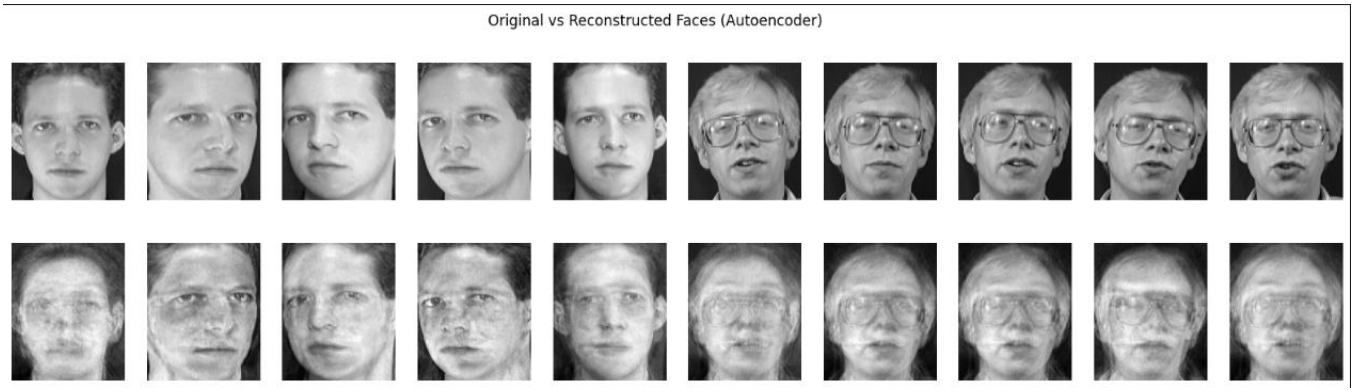
Clustering Performance

Algorithm	Accuracy (%)	F1 Score (Macro)
K-Means	*0.575*	*0.61547932330827*
GMM	*0.505*	*0.53598928848928*

Confusion matrix







Performance in both kmeans and gmm in autoencoder less than Kmeans and gmm when they applied on PCA

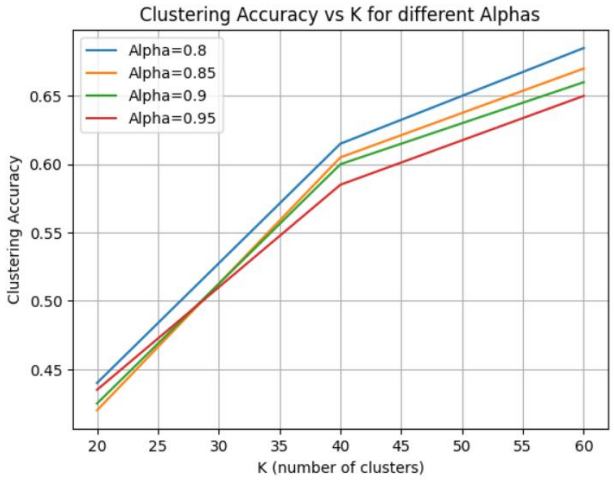
5 Evaluations

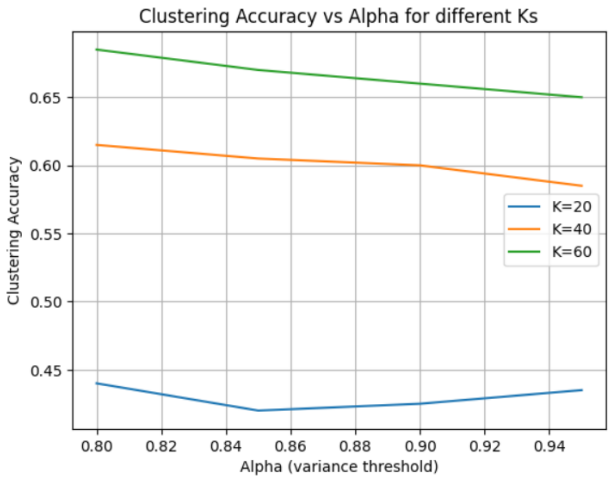
Alpha	K	Kmeans Accuracy	GMM Accuracy
0.80	20	0.4400	0.4250
0.80	40	0.6150	0.6400
0.80	60	0.6850	0.7650
0.85	20	0.4200	0.4350
0.85	40	0.6050	0.6900
0.85	60	0.6700	0.7300
0.90	20	0.4250	0.4450
0.90	40	0.6000	0.6700
0.90	60	0.6600	0.7050

0.95	20	0.4350	0.4550
0.95	40	0.5850	0.6350
0.95	60	0.6500	0.7300

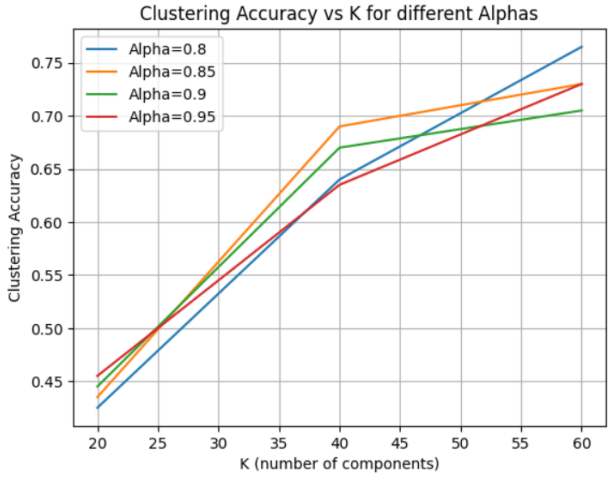
As we saw in this table, accuracy increases when K increase.

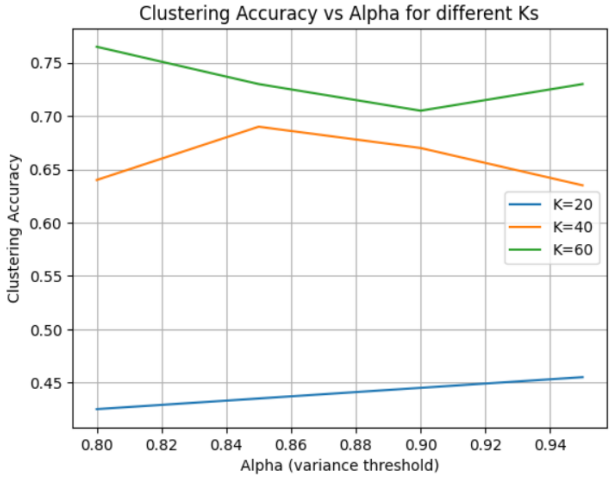
Kmenas





GMM





PCA =0.8