ESE_2240_Lab_11

April 23, 2025

1 Lab 11: Face Recognition

[ ]:
```
import numpy as np
import scipy as scp
import matplotlib.pyplot as plt
import pickle as pkl
import os
import cv2
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
# Read in the images into directory called "att_faces" - make sure to upload␣
↪att_faces.zip to Drive
import zipfile
with zipfile.ZipFile("/content/drive/My Drive/att_faces.zip", "r") as zip_ref:
    zip_ref.extractall("att_faces")
```

2 4. Creating training and test set

2.1 4.1: Generate training and test sets

[ ]:
```
# Initialize X_train and X_test
X_train = np.zeros((10304, 360))
X_test = np.zeros((10304, 40))
```

[ ]:
```
train_idx = 0
test_idx = 0
y_train = np.zeros(360)
y_test = np.zeros(40)
for i in range(1, 41):  # s1 to s40
    # Populate training matrix
    for j in range(1, 10):  # images 1 to 9 for training
        img = plt.imread(f"att_faces/s{i}/{j}.pgm")
        img_flat = img.T.flatten()  # flip matrix (transpose) then flatten
```

```python
        X_train[:, train_idx] = img_flat
        y_train[train_idx] = i
        train_idx += 1
    img = plt.imread(f"att_faces/s{i}/10.pgm")  # 10th image for testing
    img_flat = img.T.flatten()
    X_test[:, test_idx] = img_flat
    y_test[test_idx] = i
    test_idx += 1
```

[ ]:
```python
print(X_train, y_train, X_test, y_test)
```

```
[[ 48.  60.  39. … 129. 125. 119.]
 [ 45.  58.  44. … 130. 121. 118.]
 [ 45.  68.  59. … 127. 122. 120.]
 …
 [ 46.  33.  28. …  95.  43.  88.]
 [ 47.  31.  27. …  92.  35.  92.]
 [ 46.  34.  29. …  93.  40.  85.]] [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  2.
2.  2.  2.  2.  2.  2.  2.  2.
  3.  3.  3.  3.  3.  3.  3.  3.  3.  4.  4.  4.  4.  4.  4.  4.  4.  4.
  5.  5.  5.  5.  5.  5.  5.  5.  5.  6.  6.  6.  6.  6.  6.  6.  6.  6.
  7.  7.  7.  7.  7.  7.  7.  7.  7.  8.  8.  8.  8.  8.  8.  8.  8.  8.
  9.  9.  9.  9.  9.  9.  9.  9.  9. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 11. 11. 11. 11. 11. 11. 11. 11. 11. 12. 12. 12. 12. 12. 12. 12. 12. 12.
 13. 13. 13. 13. 13. 13. 13. 13. 13. 14. 14. 14. 14. 14. 14. 14. 14. 14.
 15. 15. 15. 15. 15. 15. 15. 15. 15. 16. 16. 16. 16. 16. 16. 16. 16. 16.
 17. 17. 17. 17. 17. 17. 17. 17. 17. 18. 18. 18. 18. 18. 18. 18. 18. 18.
 19. 19. 19. 19. 19. 19. 19. 19. 19. 20. 20. 20. 20. 20. 20. 20. 20. 20.
 21. 21. 21. 21. 21. 21. 21. 21. 21. 22. 22. 22. 22. 22. 22. 22. 22. 22.
 23. 23. 23. 23. 23. 23. 23. 23. 23. 24. 24. 24. 24. 24. 24. 24. 24. 24.
 25. 25. 25. 25. 25. 25. 25. 25. 25. 26. 26. 26. 26. 26. 26. 26. 26. 26.
 27. 27. 27. 27. 27. 27. 27. 27. 27. 28. 28. 28. 28. 28. 28. 28. 28. 28.
 29. 29. 29. 29. 29. 29. 29. 29. 29. 30. 30. 30. 30. 30. 30. 30. 30. 30.
 31. 31. 31. 31. 31. 31. 31. 31. 31. 32. 32. 32. 32. 32. 32. 32. 32. 32.
 33. 33. 33. 33. 33. 33. 33. 33. 33. 34. 34. 34. 34. 34. 34. 34. 34. 34.
 35. 35. 35. 35. 35. 35. 35. 35. 35. 36. 36. 36. 36. 36. 36. 36. 36. 36.
 37. 37. 37. 37. 37. 37. 37. 37. 37. 38. 38. 38. 38. 38. 38. 38. 38. 38.
 39. 39. 39. 39. 39. 39. 39. 39. 39. 40. 40. 40. 40. 40. 40. 40. 40. 40.] [[ 34.
37. 104. … 108.  89. 125.]
 [ 35.  31. 102. … 102.  87. 124.]
 [ 34.  34. 107. … 105.  87. 121.]
 …
 [ 41.  27.  57. …  46. 107.  35.]
 [ 39.  67.  56. …  80. 107.  32.]
 [ 33. 133.  59. …  48. 109.  34.]] [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.
11. 12. 13. 14. 15. 16. 17. 18.
 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36.
```

37. 38. 39. 40.]

3 5. PCA on the training and test sets

3.1 5.1: PCA transform of training points

[ ]:
```
def get_K_eigenvectors(X, K):
  L, V = scp.sparse.linalg.eigs(X, k=K)
  return V[:,0:K]

def get_PCA_faces(face, mu, eigenfaces):
  f = np.ravel(face, order='F')
  m = np.ravel(mu, order='F')
  return np.real(np.matmul(eigenfaces.conj().T, (f - m)))
```

[ ]:
```
# Function that takes PCA of several components
# Returns 3 things: transformed training matrix using the top PCA components
# top K eigenfaces, the mean vector
def compute_covariance_custom(X):
    M = X.shape[1]
    mu = np.mean(X, axis=1)  # (D,)
    X_centered = X - mu[:, np.newaxis]  # Subtract mean from each column
    sigma = (1 / M) * X_centered @ X_centered.T  # Matrix multiplication
    return mu, sigma

def pca(X, k):
    mu, sigma = compute_covariance_custom(X)
    eigenfaces = get_K_eigenvectors(sigma, k)
    X_centered = X - mu[:, np.newaxis]
    projected = eigenfaces.T @ X_centered  # Project data onto top K␣
 ↪eigenvectors
    return projected, eigenfaces, mu
```

[ ]:
```
# Initialize projections, eigenfaces, means
ks = [1, 5, 10, 20, 50]
projections = []
eigenfaces = []
mus = []
```

[ ]:
```
for k in ks:
  # TODO: Use pca function to populate projections, eigenfaces, mu list
    projected, eigenface, mu = pca(X_train, k)
    projections.append(projected)
    eigenfaces.append(eigenface)
    mus.append(mu)
```

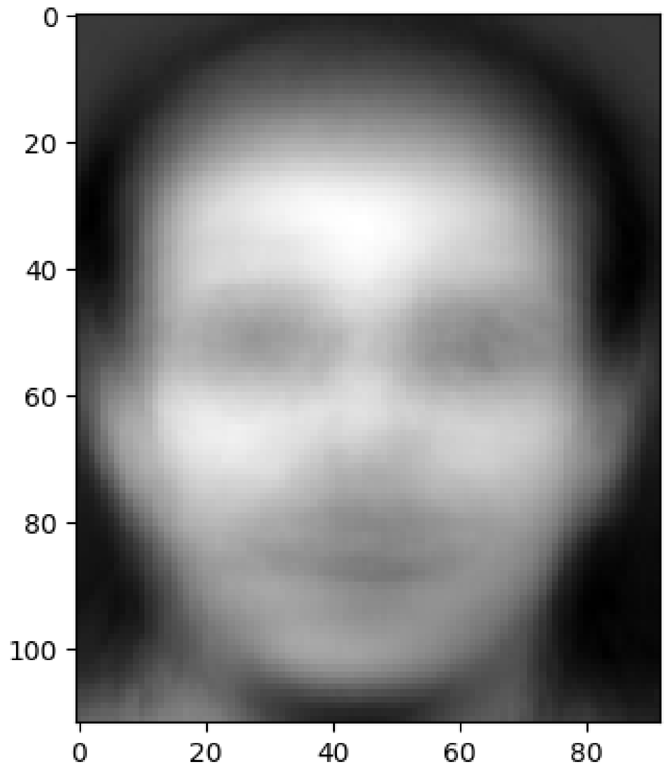
3.2 5.2: PCA transform of test point

[ ]:
```
test_projections = [] # This will have a set of projections for each k in ks
for k in ks:
  # TODO: Use function to take top k PCA of test projections, and add it to the␣
 ↪test_projections list
    Xtest_centered = X_test - mus[ks.index(k)][:, np.newaxis]  # Center the␣
 ↪test set with the mean vector for this k
    transformed_test = eigenfaces[ks.index(k)].T @ Xtest_centered  # Project␣
 ↪the test data onto the top k eigenvectors

    # Append the transformed test data to the list
    test_projections.append(transformed_test)
```

[ ]:
```
# Show the mean face
plt.imshow((mus[0].reshape(92,112)).T, cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7982fb589150>
```



4 6. Nearest neighbor classification

4.1 6.1: Nearest neighbor function

[ ]:
```
# Input: Training set, test set
# Returns: the index of the training sample closest to each vector in the test set
def pca_nearest_neighbor(train, test):
  size = test.shape[1]
  # TODO: Calculate the distance between each train set and each test set,
  # and find the index for each sample that minimizes this distance
  predictions = np.zeros((1, size), dtype=int)
  for i in range(size):
    distances = np.linalg.norm(train - test[:, i][:, np.newaxis], axis=0)
    predictions[0, i] = np.argmin(distances)

  return predictions
```

[ ]:
```
predictions = [] # This will hold predictions for each value k in ks
for i in range(len(ks)):
  predictions.append(pca_nearest_neighbor(projections[i], test_projections[i]))
```

[ ]:
```
def compute_accuracy(x):
  # TODO: Compute accuracy
  correct = 0
  for i in range(x.shape[1]):
      if y_train[int(x[0, i])] == y_test[i]:
          correct += 1
  return correct / y_test.shape[0]
```

[ ]:
```
accuracies = np.zeros(len(ks))
for i in range(len(ks)):
  accuracies[i] = compute_accuracy(predictions[i])
```

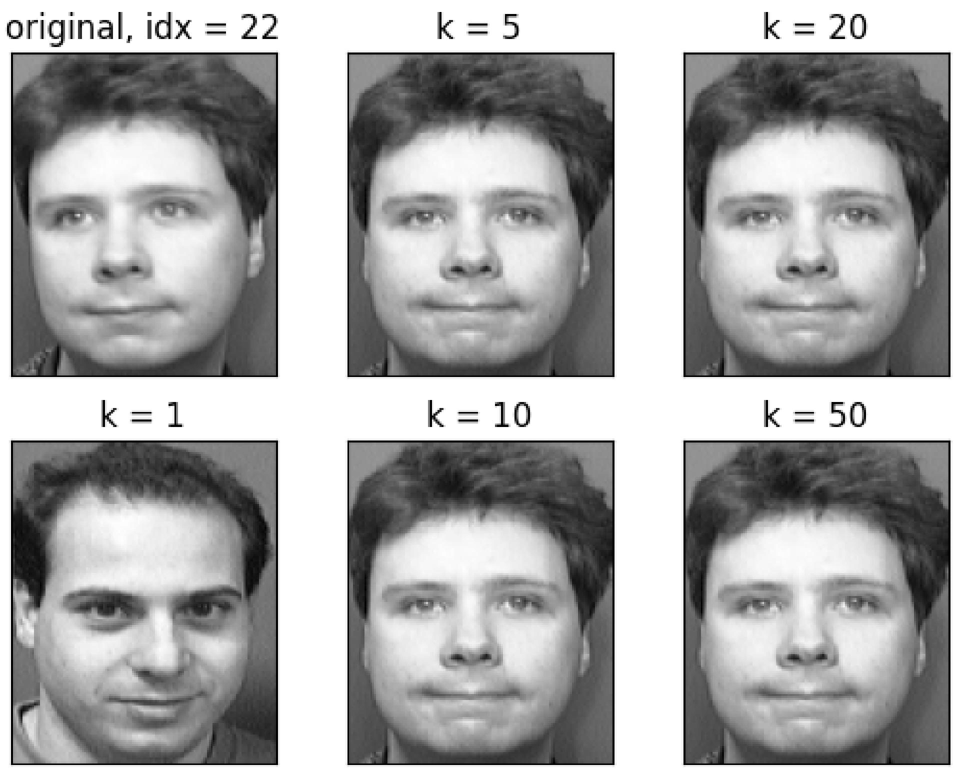
[ ]:
```
for i in range(len(ks)):
  print(f"k = {ks[i]}: accuracy = {accuracies[i]}")
```

```
k = 1: accuracy = 0.1
k = 5: accuracy = 0.825
k = 10: accuracy = 0.925
k = 20: accuracy = 0.95
k = 50: accuracy = 0.95
```

Check some predictions

[ ]:
```
idx = 22 # Indexes into test
fig, ax = plt.subplots(2, 3)
```

```python
ax[0][0].imshow(plt.imread("att_faces/s" + str(idx+1) + "/" + str(10)+".pgm"),
 ↪cmap = "gray")
ax[0][0].set_title(f"original, idx = {idx}")
for i in range(1, len(ks) + 1):
  ax[int(i%2)][int(i/2)].imshow(X_train[:,int(predictions[i - 1][0,idx])].
 ↪reshape(92,112).T,cmap="gray")
  ax[int(i%2)][int(i/2)].set_title(f"k = {ks[i - 1]}")
plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[]);
```



Interpret/Explaint he results: We can see that the correct images were identified with larger numbers of eigenfaces (only incorrect when k = 1)