David Zoro and Jordan White

Professor Hwang

EE 443

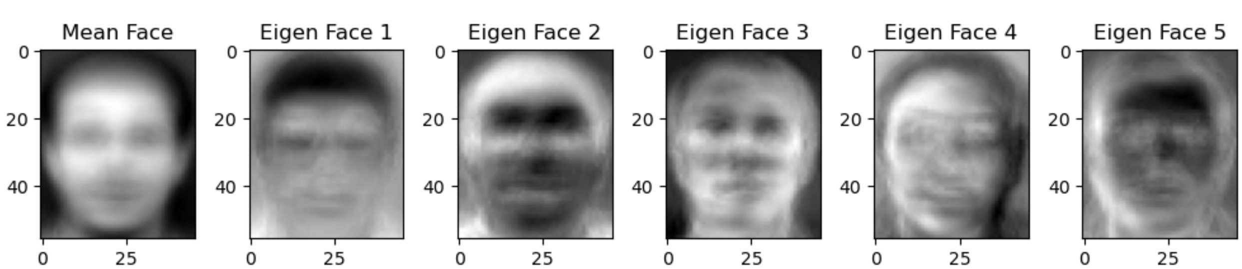
13 April 2023

HW1

Problem 1-a:

```python
# 1Part 1a - Question One
# Data standardization
X_STD = X_train.copy()
X_mean = np.mean(X_train, axis=0)
X_STD -= X_mean


# Part 1a - Question Two
# computing covariance matrix
covariant_matrix = np.cov(np.transpose(X_STD))

# Part 1a - Question Three
# Eigen-decomposition
eigen_values, eigen_vectors = np.linalg.eigh(covariant_matrix) # this is 2 arrays, index 0 is eigenvectors, 1 is eigenvalues
eigen_vectors = eigen_vectors[:, ::-1]

# Part 1a - Question Four
# selecting principal components
def top_k_eigen_vectors(k, vectors):
    new_vectors = np.transpose(vectors)
    new_vectors = -1 * new_vectors[:k]
    return new_vectors

# Part 1a - Question Five
# Dimension reduction
eigen_vectors_250 = top_k_eigen_vectors(250, eigen_vectors)
```

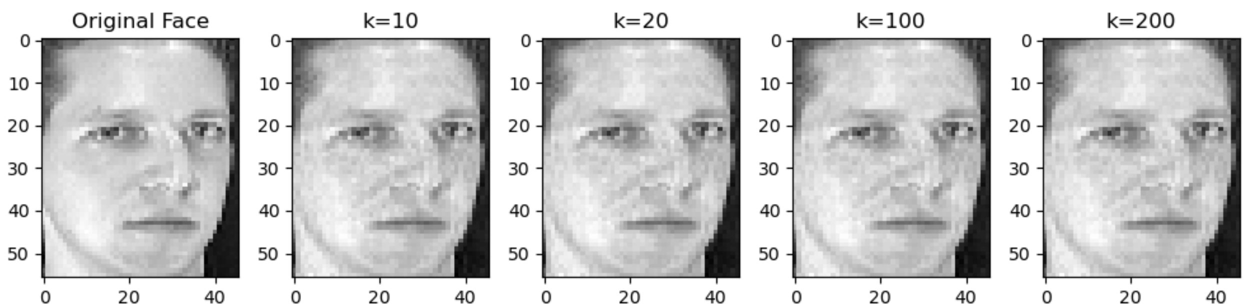
Problem 1-b and 1-c:



5 largest eigenvalues are [[0.00606061 0.00630507 0.00564232 ... 0.01547486 0.01263168
  0.0131702]
 [-0.0295488 -0.02944688 -0.02955176 ... 0.01432871 0.02047787
  0.01874106]

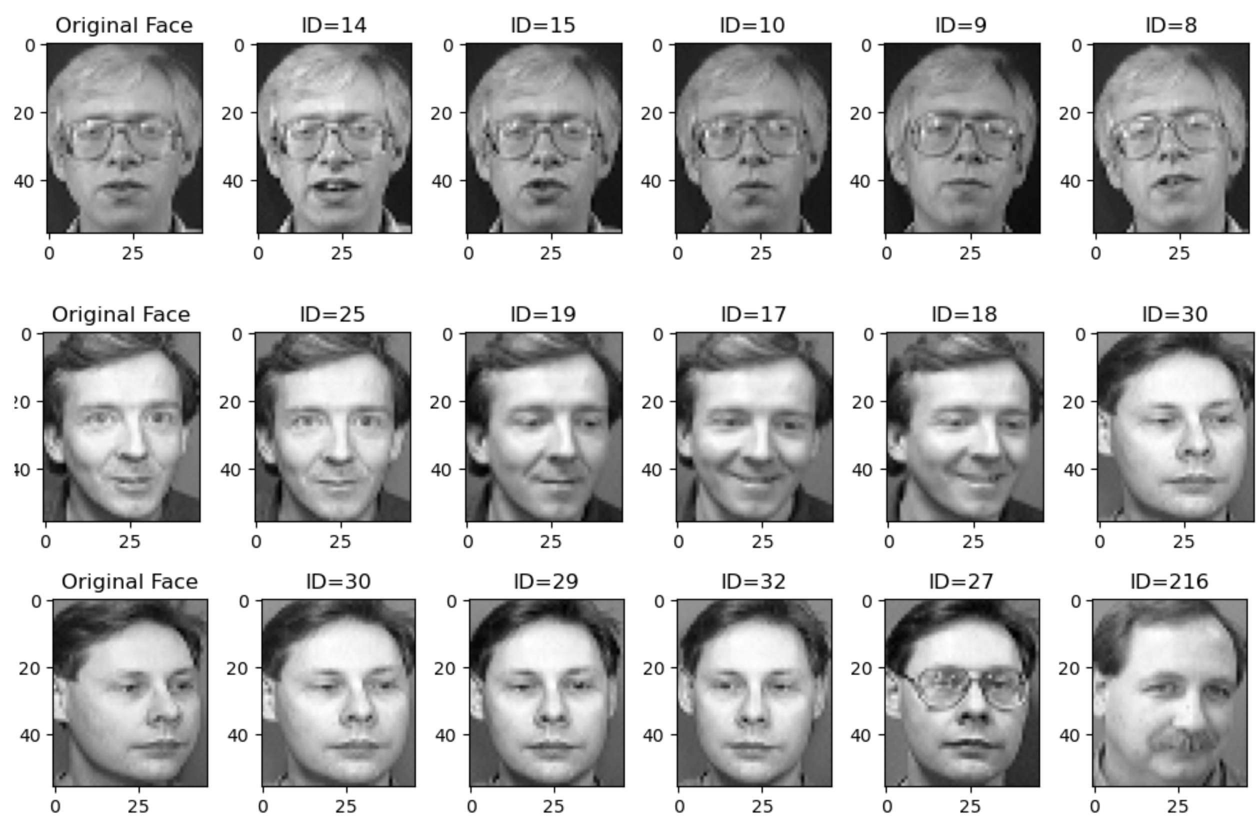
[-0.03985461 -0.03970156 -0.03961594 ... -0.02412773 -0.02921861
 -0.03115513]
[0.02173746 0.02187729 0.02211515 ... -0.02984455 -0.02604688
 -0.02497663]
[-0.0060077 -0.00644463 -0.00562654 ... 0.01114323 0.00942016
  0.00796383]]

Problem 1-d:



MSE: 25.332578244184187
MSE: 25.332578244184187
MSE: 25.332578244184187
MSE: 25.332578244184187

Problem 1-e:



Number of matching identities among the 5 nearest training images for the first three testing images: 0

Problem 2-a:

```python
# to calculate accuracy
def accuracy_score(y_labels, predicted_labels):
    return np.sum(y_labels == predicted_labels)/len(y_labels)

# implementation of the pca class
class PCA:
    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None

    def fit(self, X):
        self.mean = np.mean(X, axis=0)
        X = X - self.mean

        covariance_matrix = np.cov(X.T)
        eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

        eig_pairs = [(np.abs(eigenvalues[i]), eigenvectors[:, i]) for i in range(len(eigenvalues))]
        eig_pairs.sort(key=lambda x: x[0], reverse=True)

        self.components = np.array([eig_pairs[i][1] for i in range(self.n_components)]).T

    def transform(self, X):
        X = X - self.mean
        return np.dot(X, self.components)

    def fit_transform(self, X):
        self.fit(X)
        return self.transform(X)
```

```python
# implementation of KMeans
class CustomKMeans:
    def __init__(self, n_clusters, n_init=1, max_iter=300, tol=1e-4):
        self.n_clusters = n_clusters
        self.init = 'fps'
        self.n_init = n_init
        self.max_iter = max_iter
        self.tol = tol
        self.cluster_centers_ = None
        self.labels_ = None

    @staticmethod
    def _fps_sampling(X, k): # this is our implementation of furthest point sampling
        centroids = [X[0]]
        for _ in range(k - 1):
            dist = cdist(X, centroids).min(axis=1)
            idx = np.argmax(dist)
            centroids.append(X[idx])
        return np.array(centroids)

    def _initialize_centroids(self, X): # this initializes the centroids
        if self.init == 'fps':
            return self._fps_sampling(X, self.n_clusters)
        else:
            raise ValueError(f"Unknown initialization method '{self.init}'")

    def _assign_points(self, X, centroids): # correct assignment of points
        return np.argmin(cdist(X, centroids), axis=1)

    def _update_centroids(self, X, assignments): # updating the centroids
        return np.array([X[assignments == i].mean(axis=0) for i in range(self.n_clusters)])

    def _k_means_single_run(self, X): # singly running k means
        centroids = self._initialize_centroids(X)
        assignments = self._assign_points(X, centroids)
```

```python
        for _ in range(self.max_iter):
            new_centroids = self._update_centroids(X, assignments)
            new_assignments = self._assign_points(X, new_centroids)

            if np.linalg.norm(new_centroids - centroids) < self.tol:
                break

            centroids = new_centroids
            assignments = new_assignments

        return centroids, assignments

    def fit(self, X): # fitting data
        best_inertia = None
        best_centroids = None
        best_assignments = None

        for _ in range(self.n_init):
            centroids, assignments = self._k_means_single_run(X)
            inertia = np.sum([np.linalg.norm(X[assignments == i] - centroids[i])**2 for i in range(self.n_clusters)])

            if best_inertia is None or inertia < best_inertia: # placeholder
                best_inertia = inertia
                best_centroids = centroids
                best_assignments = assignments

        self.cluster_centers_ = best_centroids
        self.labels_ = best_assignments
        return self

    def predict(self, X): # predictions
        return self._assign_points(X, self.cluster_centers_)
```

```python
# Load data
train_data = pd.read_csv("HW1_2/mnist_train.csv")
test_data = pd.read_csv("HW1_2/mnist_test.csv")

X_train = train_data.iloc[:, 1:].values
y_train = train_data.iloc[:, 0].values
X_test = test_data.iloc[:, 1:].values
y_test = test_data.iloc[:, 0].values

# PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)


# furthest point sampling
def fps_sampling(X, k):
    centroids = [X[0]]
    for q in range(k - 1):
        dist = cdist(X, centroids).min(axis=1)
        idx = np.argmax(dist)
        centroids.append(X[idx])
    return np.array(centroids)
```

```python
# kmeans function, for returning centroids and assignments
def k_means(X, k):
    centroids = fps_sampling(X, k)
    assignments = np.argmin(cdist(X, centroids), axis=1)
    updated = True

    while updated:
        updated = False
        new_centroids = np.array([X[assignments == i].mean(axis=0) for i in range(k)])
        new_assignments = np.argmin(cdist(X, new_centroids), axis=1)

        if not np.array_equal(assignments, new_assignments):
            updated = True
            centroids = new_centroids
            assignments = new_assignments

    return centroids, assignments
```

Problem 2-d:

```python
# 2d
# Tests k_means algorithm on different numbers of clusters
def test_k_and_pca(k_values, pca_dim, X_train, y_train, X_test, y_test):
    # Loops over each number of clusters in a list of cluster values
    for k in k_values:
        pca = PCA(n_components=pca_dim) # Initializes the PCA
        X_train_pca = pca.fit_transform(X_train) # Utilizes the PCA on X_Train
        X_test_pca = pca.transform(X_test) # Utilizes the PCA on X_test
        centroids, assignments = k_means(X_train_pca, k) # Implements k_means given a number of clusters and returns a tuple
        train_accuracy = evaluate_accuracy(assignments, y_train, k) # Tests the accuracy of the predicted labels

        kmeans = CustomKMeans(n_clusters=k, n_init=1)  # Initializes the k_means algorithm
        kmeans.fit(X_train_pca) # Trains the algorithm of X_train_pca
        test_assignments = kmeans.predict(X_test_pca) # Predicts the clusters for the algorithm
        test_accuracy = evaluate_accuracy(test_assignments, y_test, k) # Evaluates the accuracy of the predicted labels to the labels

        print(f"k = {k}, Train Accuracy: {train_accuracy:.4f}, Test Accuracy: {test_accuracy:.4f}")
```