Report on Handwritten Digit Classification Using PCA and Bayesian Decision Theory

a. Introduction

This report summarizes the results of a digit classification project that utilizes Principal Component Analysis (PCA) and Bayesian Decision Theory. The project employs a subset of images from the MNIST dataset, focusing on digits "5" and "6." The goal is to classify these two digits using dimensionality reduction and statistical modeling. The project comprises five main tasks, including data conditioning, PCA, dimension reduction, density estimation, and classification accuracy assessment.

Problem Statement:

In this project, we aim to classify handwritten digits "5" and "6" using dimensionality reduction through PCA and Bayesian Decision Theory. We have access to a modified subset of the MNIST dataset, consisting of 60,000 training images and 1,850 testing images for these two digits. The ultimate goal is to achieve accurate digit classification based on the features extracted through PCA.

Data Description:

Dataset: A subset of MNIST containing images of handwritten digits "5" and "6".
Training Set: 11,339 samples (5,421 for digit "5" and 5,918 for digit "6").
Testing Set: 1,850 samples (892 for digit "5" and 958 for digit "6").

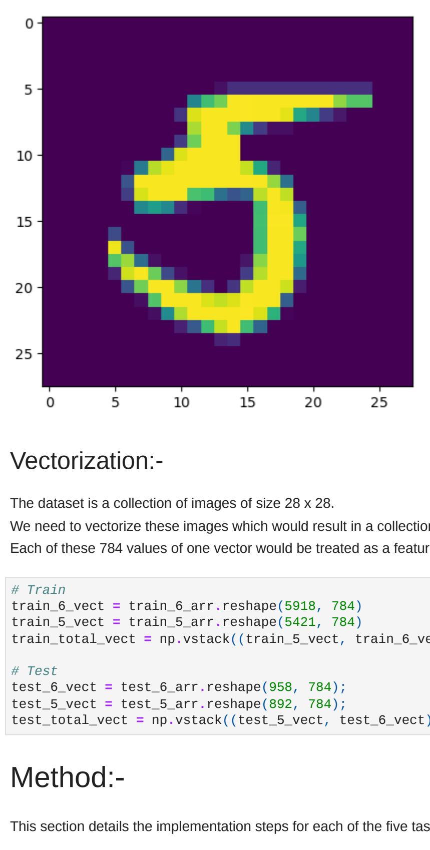
In [29]:
```
import scipy.io
import pandas as pd
import numpy as np
import numpy.matlib as npmat
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
import seaborn as sns
```

In [30]:
```
# Loading data
train_5_dict = scipy.io.loadmat("training_data_5.mat")
train_5_arr = train_5_dict['train_data_5'];
test_5_dict = scipy.io.loadmat("testing_data_5.mat")
test_5_arr = test_5_dict['test_data_5'];
```

In [31]:
```
train_6_dict = scipy.io.loadmat("training_data_6.mat")
train_6_arr = train_6_dict['train_data_6'];
test_6_dict = scipy.io.loadmat("testing_data_6.mat")
test_6_arr = test_6_dict['test_data_6'];
```

In [32]:
```
# Just to showcase how an MNIST image looks like
im1 = train_5_arr[500]
plt.imshow(im1)
```

Out[32]:
```
<matplotlib.image.AxesImage at 0x177e41ad0>
```



Vectorization:-

The dataset is a collection of images of size 28 x 28.
We need to vectorize these images which would result in a collection of vectors of size 784.
Each of these 784 values of one vector would be treated as a feature for that image

In [33]:
```
# Train
train_6_vect = train_6_arr.reshape(5918, 784)
train_5_vect = train_5_arr.reshape(5421, 784)
train_total_vect = np.vstack((train_5_vect, train_6_vect));

# Test
test_6_vect = test_6_arr.reshape(958, 784);
test_5_vect = test_5_arr.reshape(892, 784);
test_total_vect = np.vstack((test_5_vect, test_6_vect));
```

Method:-

This section details the implementation steps for each of the five tasks in the project.

Task 1: - Feature normalization

1.) Calculate the mean (mi) and standard deviation (STD) (si) for each feature (xi) using all training samples.
2.) Normalize all data samples (training and testing) using the computed mean and standard deviation: yi = (xi - mi)/si.

Formula: yi = (xi - mi)/si

Note:-

We use the same mean and standard deviation from the training set to normalize the testing data.

In [34]:
```
# Calculating mean and variance for training data.
# This would be used to normalize training as well as test data.
mean = np.mean(train_total_vect, axis = 0)
std = np.std(train_total_vect, axis = 0)
```

Note:-

Some values in the standard deviation array are 0, mostly because they are too small for python to store. Thus, assuming their eigen values will already be too small to be considered for principle componenets, we replace them with 1e-10.

In [35]:
```
# Replacing very small std deviations with 1e-10, which got rounded off to 0 due to decimal constraints.
std[std == 0] = 0.0000000001;
```

In [36]:
```
# Train
mean_total_mat = npmat.repmat(mean, 11339, 1);
std_total_mat = npmat.repmat(std, 11339, 1);
train_total_norm = np.divide((train_total_vect - mean_total_mat), std_total_mat)


# Test
mean_test_mat = npmat.repmat(mean, 1850, 1);
std_test_mat = npmat.repmat(std, 1850, 1);
test_total_norm = np.divide((test_total_vect - mean_test_mat), std_test_mat)
```

Task 2:- PCA using the training samples

- Compute the covariance matrix of the training samples.
- Perform eigenanalysis on the covariance matrix to identify principal components.

In [37]:
```
# Computing the covariance matrix.
cov_mat_total = np.cov(train_total_norm , rowvar = False)

# Computing eigen vector and eigen values
eigen_values_total , eigen_vectors_total = np.linalg.eigh(cov_mat_total)
```

Eigen analysis :-

1. First, we sort the eigenvalues.
2. Then, we choose the eigenvectors corresponding to the 2 largest eigenvalues, these are our Principle Components.

In [38]:
```
# Sorting eigenvalues, and storing the sorted indices
sorted_index_total = np.argsort(eigen_values_total)[::-1]

# Sorted eigenvalues
sorted_eigenvalue_total = eigen_values_total[sorted_index_total]

# Sorting Eigenvectors according to sorted indices obtained while sorting eigenvalues
sorted_eigenvectors_total = eigen_vectors_total[:,sorted_index_total]

# Now, we need to select the first 2 eigenvectors, corresponding to the 2 largest eigenvalues.
numOfComponents = 2
# Projection Matrix
eigenvector_subset_total = sorted_eigenvectors_total[:,0:numOfComponents]

principleComponent1 = eigenvector_subset_total[:, 0];
principleComponent1 = eigenvector_subset_total[:, 1];
```

Task 3:- Dimensionality Reduction using PCA

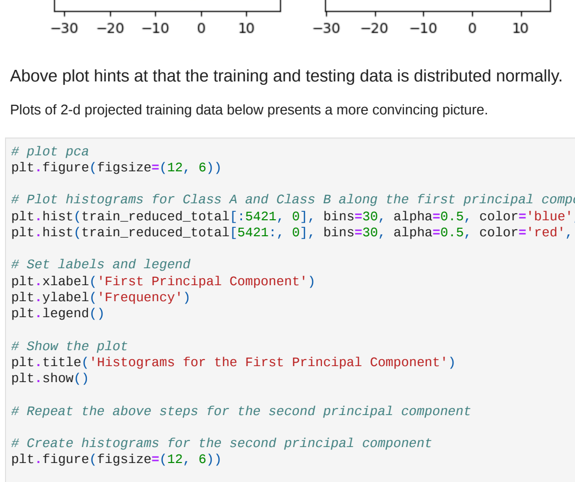
Here, we project our 784-dimensional data onto the 2 Principle components, and get a 2-d data.
We use the princimple components of training data only, even when we project test data.

1.) Consider 2D projections of samples onto the first and second principal components.
2.) Visualize the training and testing samples in this 2D space.
3.) Observe the clustering of the two classes in the 2D space.

In [54]:
```
# Projecting the 784-d training data onto 2 orthogonal eigenvectors in 2-d.

# Train
train_reduced_total = np.dot(eigenvector_subset_total.transpose(),train_total_norm.transpose()).transpose();

# Test
# Note:- We project testing data onto the same 2 principle components obtained from PCA over training data.
test_reduced_total = np.dot(eigenvector_subset_total.transpose(),test_total_norm.transpose()).transpose();
```

In [55]:
```
fig, axs = plt.subplots(1,2)

# Plotting reduced training data
axs[0].scatter(train_reduced_total[0:5421,0], train_reduced_total[0:5421,1], alpha = 0.1)
axs[0].scatter(train_reduced_total[5421:11338,0], train_reduced_total[5421:11338,1], alpha = 0.1)
axs[0].set_title('Training data after PCA')

# Plotting reduced testing data
axs[1].scatter(test_reduced_total[0:892,0], test_reduced_total[0:892,1], alpha = 0.1)
axs[1].scatter(test_reduced_total[892:,0], test_reduced_total[892:,1], alpha = 0.1)
axs[1].set_title('Testing data after PCA')
```

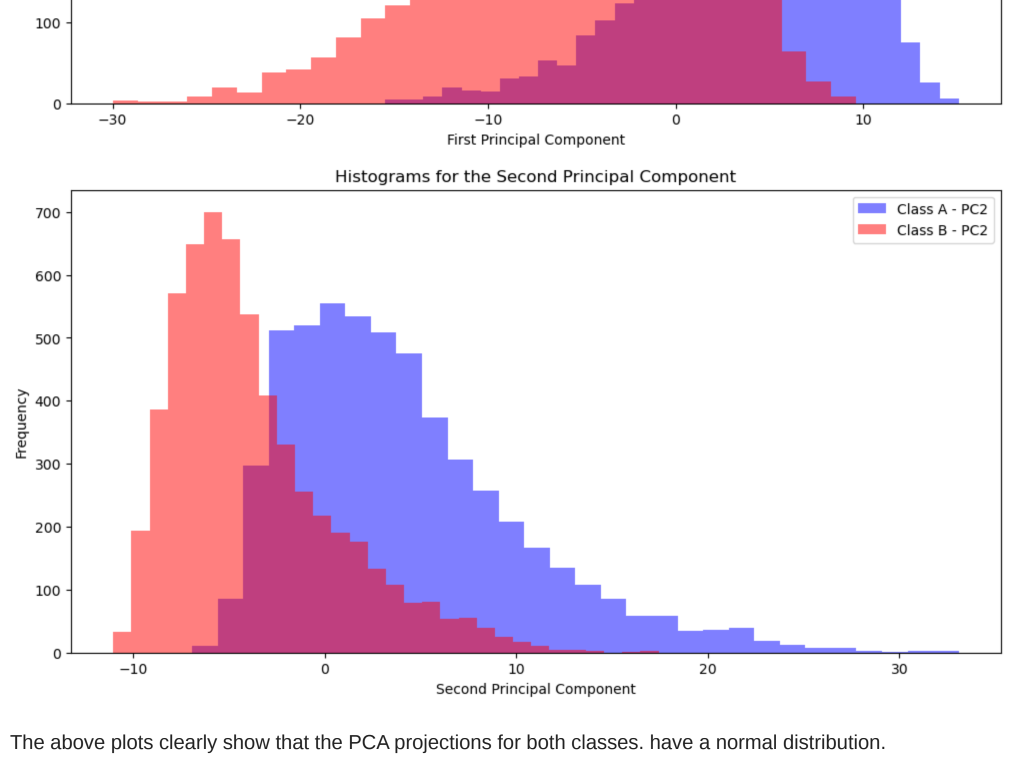
Out[55]:
```
Text(0.5, 1.0, 'Testing data after PCA')
```

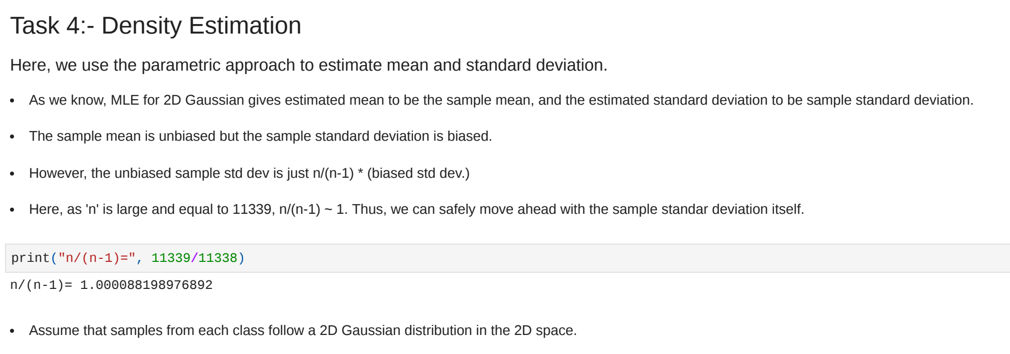


Above plot hints at that the training and testing data is distributed normally.

Plots of 2-d projected training data below presents a more convincing picture.

In [45]:
```
# plot pca
plt.figure(figsize=(12, 6))

# Plot histograms for Class A and Class B along the first principal component
plt.hist(train_reduced_total[:5421, 0], bins=30, alpha=0.5, color='blue', label='Class A - PC1')
plt.hist(train_reduced_total[5421:, 0], bins=30, alpha=0.5, color='red', label='Class B - PC1')

# Set labels and legend
plt.xlabel('First Principal Component')
plt.ylabel('Frequency')
plt.legend()

# Show the plot
plt.title('Histograms for the First Principal Component')
plt.show()

# Repeat the above steps for the second principal component

# Create histograms for the second principal component
plt.figure(figsize=(12, 6))

# Plot histograms for Class A and Class B along the second principal component
plt.hist(train_reduced_total[:5421, 1], bins=30, alpha=0.5, color='blue', label='Class A - PC2')
plt.hist(train_reduced_total[5421:, 1], bins=30, alpha=0.5, color='red', label='Class B - PC2')

# Set labels and legend
plt.xlabel('Second Principal Component')
plt.ylabel('Frequency')
plt.legend()

# Show the plot
plt.title('Histograms for the Second Principal Component')
plt.show()
```





The above plots clearly show that the PCA projections for both classes. have a normal distribution.

Task 4:- Density Estimation

Here, we use the parametric approach to estimate mean and standard deviation.

- As we know, MLE for 2D Gaussian gives estimated mean to be the sample mean, and the estimated standard deviation to be sample standard deviation.
- The sample mean is unbiased but the sample standard deviation is biased.
- However, the unbiased sample std dev is just n/(n-1) * (biased std dev.)
- Here, as 'n' is large and equal to 11339, n/(n-1) ~ 1. Thus, we can safely move ahead with the sample standar deviation itself.

In [46]:
```
print("n/(n-1)=", 11339/11338)
```

```
n/(n-1)= 1.000088198976892
```

- Assume that samples from each class follow a 2D Gaussian distribution in the 2D space.
- Estimate the parameters (mean and covariance) of the Gaussian distribution for each class using the training data.

In [47]:
```
def estimate_2d_normal_parameters(data):
    N = len(data)
    mean_estimate = np.zeros(2)
    for data_point in data:
        mean_estimate += data_point
    mean_estimate /= N

    cov_estimate = np.zeros((2, 2))
    for data_point in data:
        cov_estimate += np.outer(data_point - mean_estimate, data_point - mean_estimate)
    cov_estimate /= N

    return mean_estimate, cov_estimate
```

In [48]:
```
# Estimate the parameters of the 2D normal distribution
mean_estimate_5, cov_estimate_5 = estimate_2d_normal_parameters(train_reduced_total[0:5421, :])
mean_estimate_6, cov_estimate_6 = estimate_2d_normal_parameters(train_reduced_total[5421:, :])

mean_test_5, cov_test_5 = estimate_2d_normal_parameters(test_reduced_total[0:892, :])
mean_test_6, cov_test_6 = estimate_2d_normal_parameters(test_reduced_total[892:, :])
```

Task 5:- Bayesian Decision Theory for optimal classification

As stated in the problem, we need to do minimum error rate classification.

And we assume priors for both classes to be the same.
Thus, we simply new to compute p(xi|wi) for both classes and compare them.

Decision rule:- Select the class which has greater value for p(xi|wi)

In [49]:
```
# Adding labels to training and testing data
# Train
train_reduced_total = np.hstack((train_reduced_total, np.zeros((11339, 1))))
train_reduced_total[5421:, 2] = 1

# Test
test_reduced_total = np.hstack((test_reduced_total, np.zeros((1850, 1))))
test_reduced_total[892:, 2] = 1
```

In [50]:
```
def calculate_pdf(point, mean, covariance_matrix):
    pdf = multivariate_normal.pdf(point, mean=mean, cov=covariance_matrix)
    return pdf
```

In [51]:
```
def classify_point(point, mean_1, mean_2, covMat_1, covMat_2):
    # Calculate the probabilities for each class
    pdf_class1 = calculate_pdf(point, mean_1, covMat_1)
    pdf_class2 = calculate_pdf(point, mean_2, covMat_2)

    # Assign the point to the class with the higher probability
    if pdf_class1 > pdf_class2:
        return 0  # Class 0
    else:
        return 1  # Class 1
```

In [53]:
```
# Classify the training set and calculate accuracy
correct_train = 0
for point in train_reduced_total:
    true_class = point[-1]  # Assuming the class label is the last column
    predicted_class = classify_point(point[:-1], mean_estimate_5, mean_estimate_6, cov_estimate_5, cov_estimate_6)  # Exclude the class label for classification
    if true_class == predicted_class:
        correct_train += 1

accuracy_train = (correct_train / len(train_reduced_total)) * 100

# Classify the testing set and calculate accuracy
correct_test = 0
for point in test_reduced_total:
    true_class = point[-1]  # Assuming the class label is the last column
    predicted_class = classify_point(point[:-1], mean_estimate_5, mean_estimate_6, cov_estimate_5, cov_estimate_6)  # Exclude the class label for classification
    if true_class == predicted_class:
        correct_test += 1

accuracy_test = (correct_test / len(test_reduced_total)) * 100

print("Training accuracy in % :-", accuracy_train)
print("Testing accuracy in % :-", accuracy_test)
```

```
Training accuracy in % :- 94.27639121615663
Testing accuracy in % :- 93.83783783783784
```

Results:-

Training accuracy in % :- 94.27639121615663

Testing accuracy in % :- 93.83783783783784

d. Conclusion

In conclusion, this project successfully employed PCA and Bayesian Decision Theory to classify handwritten digits "5" and "6" from the MNIST dataset. Feature normalization, dimensionality reduction, and density estimation were essential steps in achieving accurate classification. The results and observations from each task provide valuable insights into the performance and behavior of the classification model. The report demonstrates the effectiveness of the chosen methodology in achieving the project's goals.

Rohan Chhibba

ASU ID:- 1229817786