CSE455 Machine Learning

Homework 3

Sedef Erdoğdu, 1801042102

May 13, 2023

0.1 Import Necessary Libraries

[1]:

```
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import KernelPCA
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
```

```
2023-05-13 22:55:55.963536: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-05-13 22:55:56.039377: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-05-13 22:55:56.040642: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2023-05-13 22:55:58.561717: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

0.2 Read Dataset

Load MNIST Dataset

[2]:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

0.3 PCA Implementation

[3]:

```
def pca(X):
    mean = np.mean(X, axis=0)


    X_normalized = X - mean

    U, S, Vt = np.linalg.svd(X_normalized)
    weights = S ** 2 / (X.shape[0] - 1)

    vectors = Vt.T

    return mean, weights, vectors
```

0.4 Using PCA before Classification

[4]:
```
x_train.shape
```

[4]:
```
(60000, 28, 28)
```

[5]:
```
x_test.shape
```

[5]:
```
(10000, 28, 28)
```

[6]:
```
X_train = x_train.reshape(x_train.shape[0], -1)
X_test = x_test.reshape(x_test.shape[0], -1)

print('Train Shape:',X_train.shape)
print('Class Shape:',X_test.shape)
```

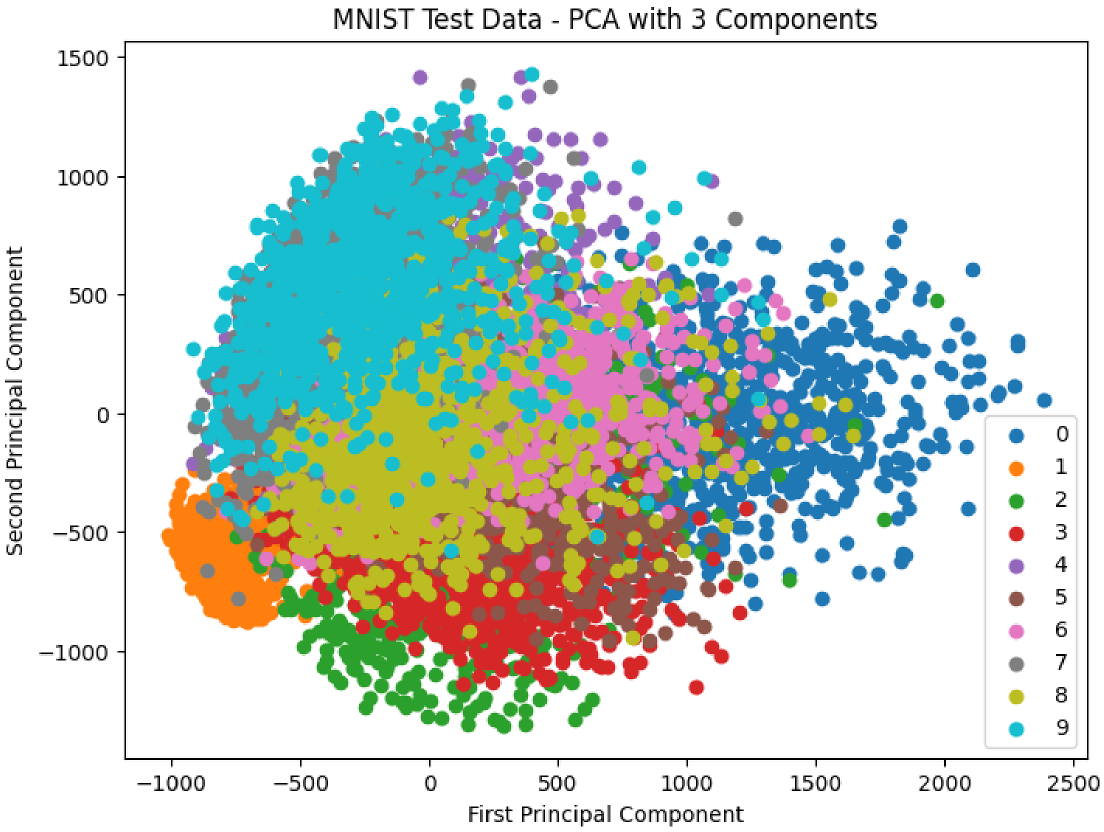
```
Train Shape: (60000, 784)
Class Shape: (10000, 784)
```

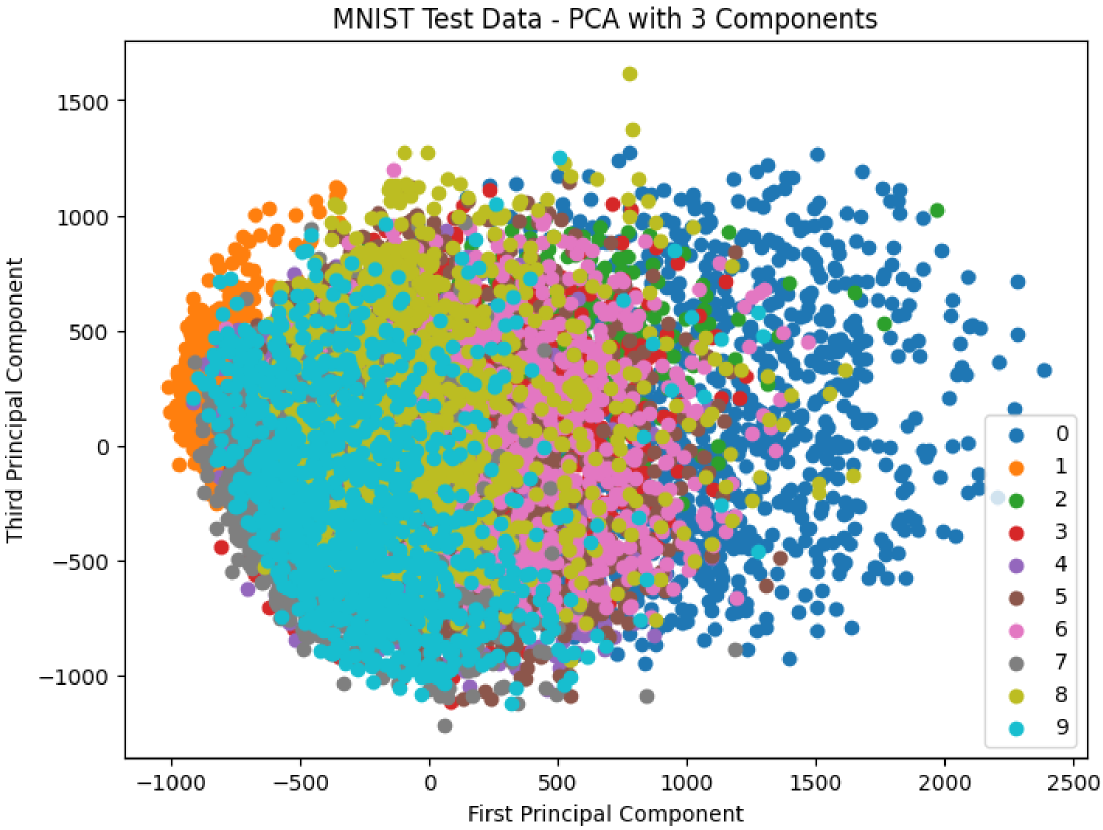
Selecting a subset of data

[7]:
```
subset_size = 1000
indices = np.random.choice(X_train.shape[0], size=subset_size, replace=False)
x_train_subset = X_train[indices]
y_train_subset = y_train[indices]
```

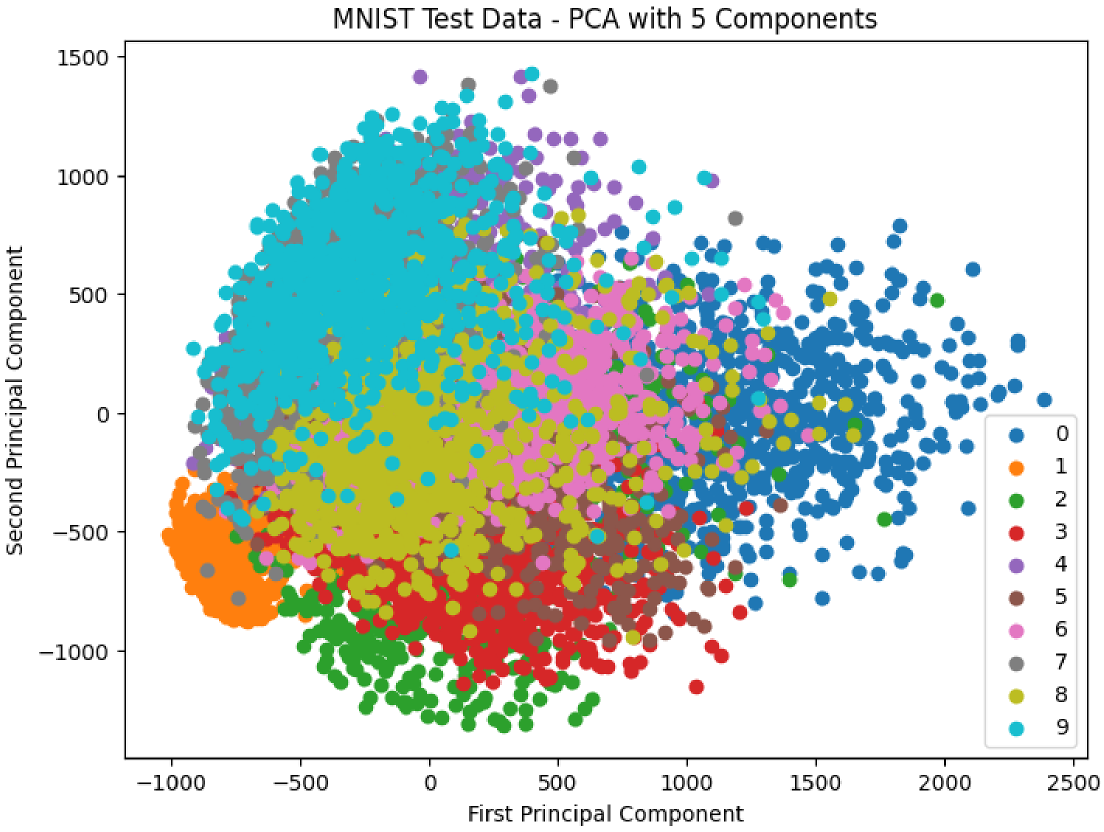
[8]:
```
num_components = [3, 5, 10, 20]  # Choose the number of components for reduction

for n in num_components:
    mean, weights, vectors = pca(x_train_subset)

    reduced_X_train = np.dot(x_train_subset - mean, vectors[:, :n])
    reduced_X_test = np.dot(X_test - mean, vectors[:, :n])


    plt.figure(figsize=(8, 6))

    for i in range(10):
        indices = np.where(y_test == i)[0]
        plt.scatter(reduced_X_test[indices, 0], reduced_X_test[indices, 1],␣
 ↪label=str(i))
```

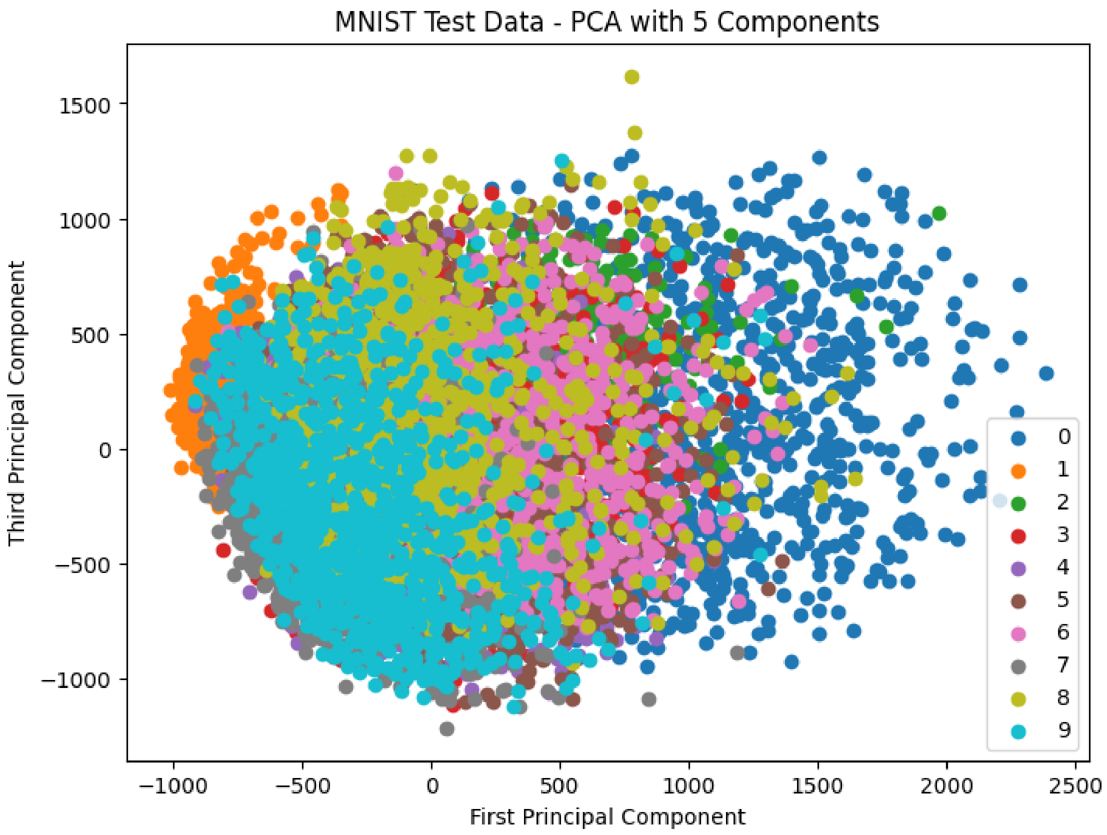
```
    plt.xlabel("First Principal Component")
    plt.ylabel("Second Principal Component")
    plt.legend()
    plt.title("MNIST Test Data - PCA with {} Components".format(n))
    plt.show()

    if n >= 3:
        plt.figure(figsize=(8, 6))
        for i in range(10):
            indices = np.where(y_test == i)[0]
            plt.scatter(reduced_X_test[indices, 0], reduced_X_test[indices, 2],␣
↪label=str(i))
        plt.xlabel("First Principal Component")
        plt.ylabel("Third Principal Component")
        plt.legend()
        plt.title("MNIST Test Data - PCA with {} Components".format(n))
        plt.show()

    # Feed reduced features to Random Forest and perform classification with␣
↪cross-validation
    rf = RandomForestClassifier(n_estimators=100)

    accuracies = []

    # Perform cross-validation
    kf = KFold(n_splits=5,shuffle=True)

    for train_index, test_index in kf.split(reduced_X_train):
        x_cv_train, x_cv_test = reduced_X_train[train_index],␣
↪reduced_X_train[test_index]
        y_cv_train, y_cv_test = y_train_subset[train_index],␣
↪y_train_subset[test_index]

        rf.fit(x_cv_train, y_cv_train)

        y_pred = rf.predict(x_cv_test)
        accuracy = accuracy_score(y_cv_test, y_pred)
        accuracies.append(accuracy)



    # Calculate the average accuracy over all folds
    avg_accuracy = np.mean(accuracies)

    print("Average accuracy with {} components: {:.2f}%".format(n, avg_accuracy␣
↪* 100))
```


MNIST Test Data - PCA with 3 Components
Second Principal Component
First Principal Component
1500
1000
500
0
−500
−1000
−1000
−500
0
500
1000
1500
2000
2500
● 0
● 1
● 2
● 3
● 4
● 5
● 6
● 7
● 8
● 9

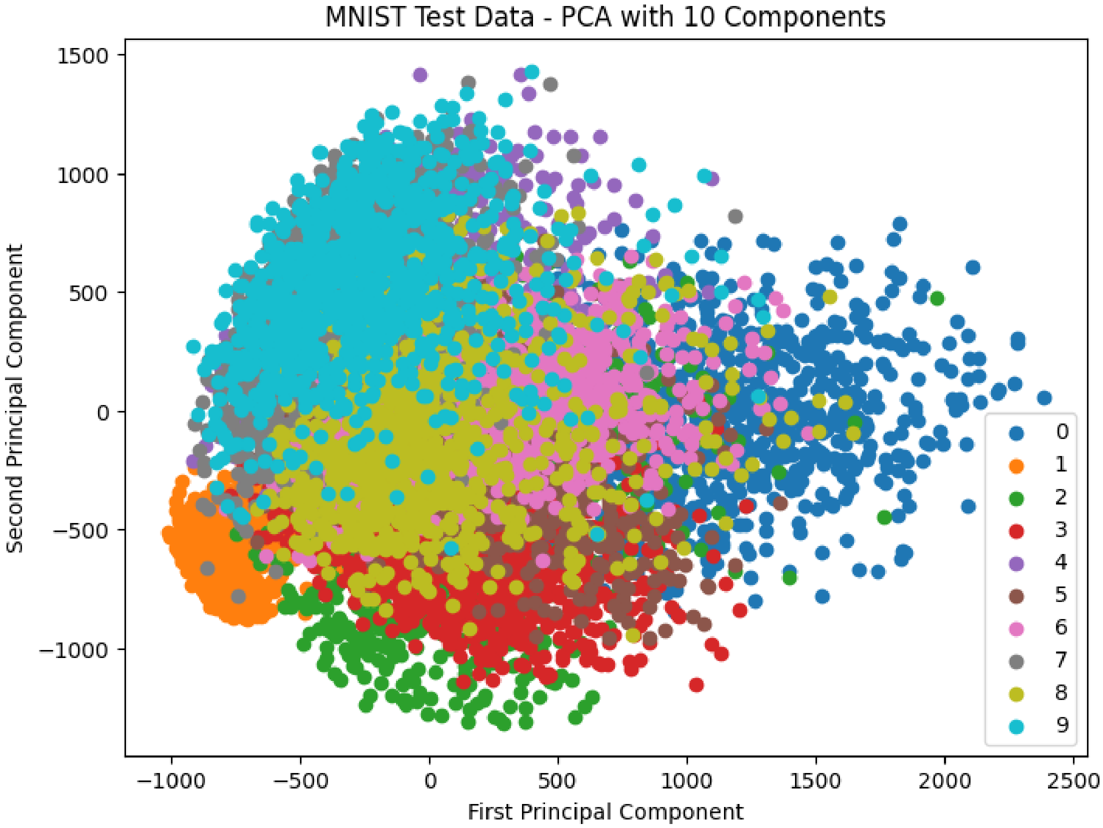


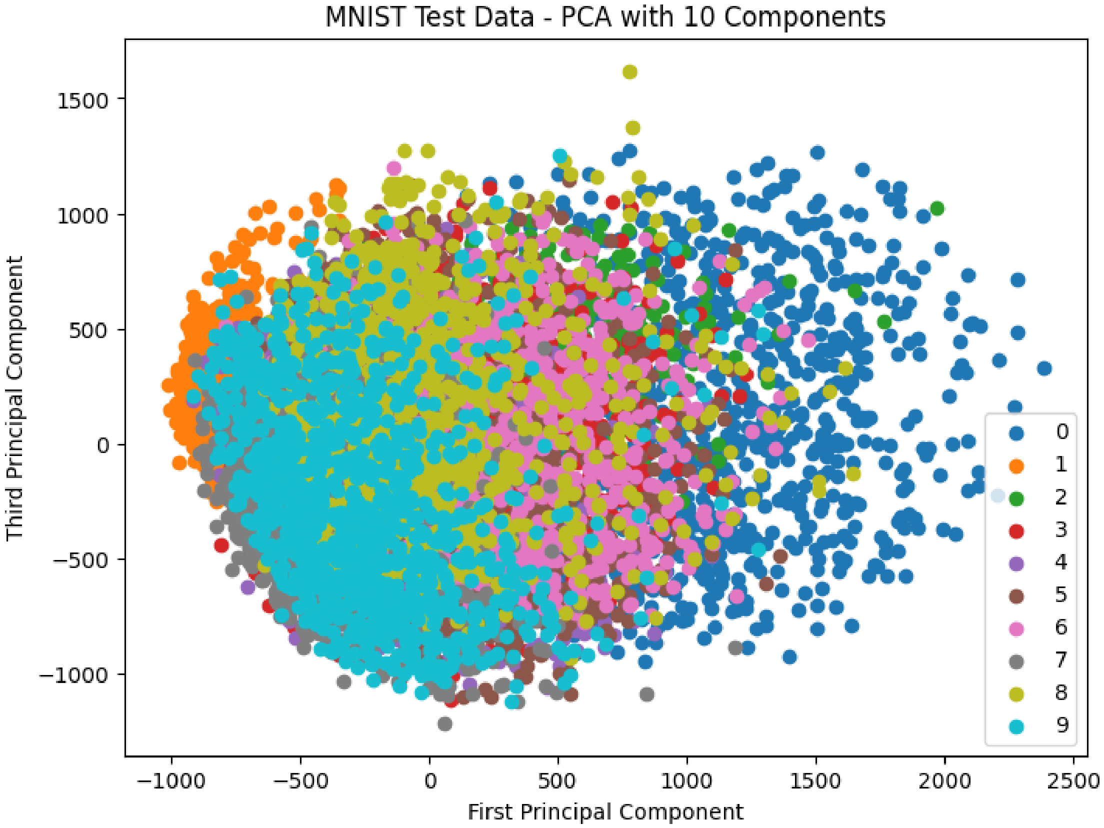
```
Average accuracy with 3 components: 44.00%
```


MNIST Test Data - PCA with 5 Components
1500
1000
500
0
−500
−1000
Second Principal Component
−1000
−500
0
500
1000
1500
2000
2500
First Principal Component
0
1
2
3
4
5
6
7
8
9

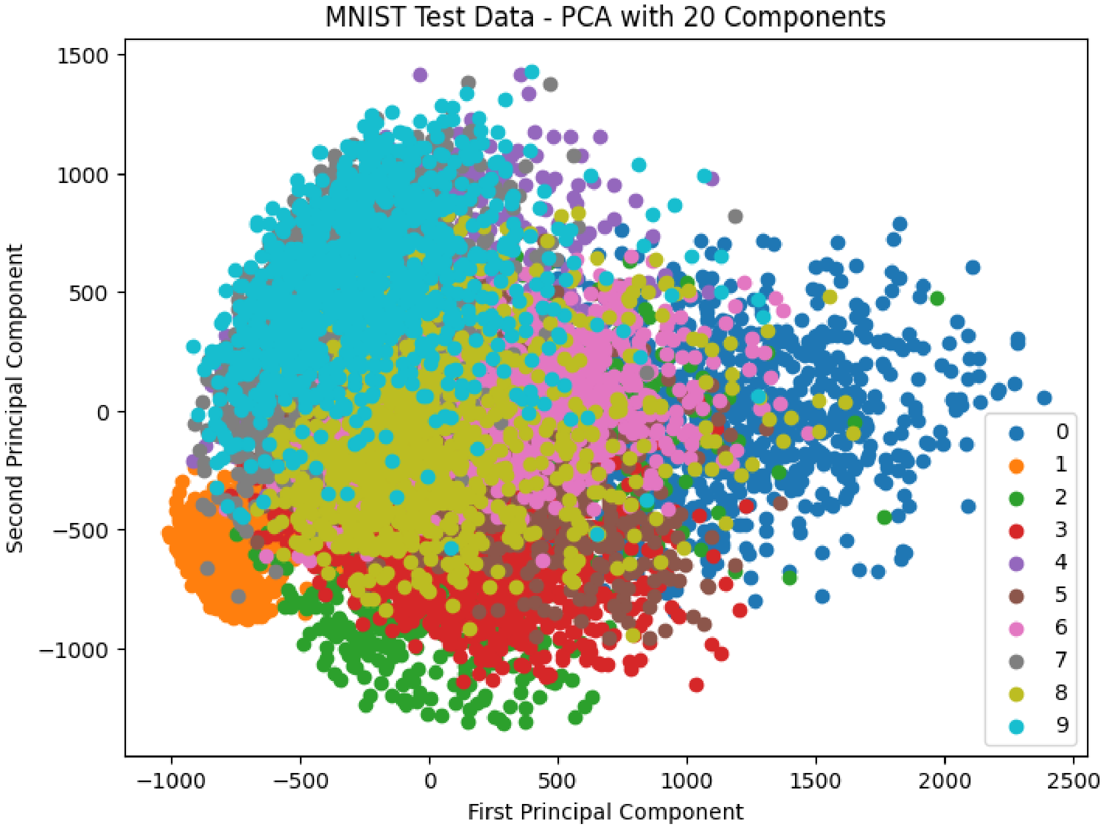


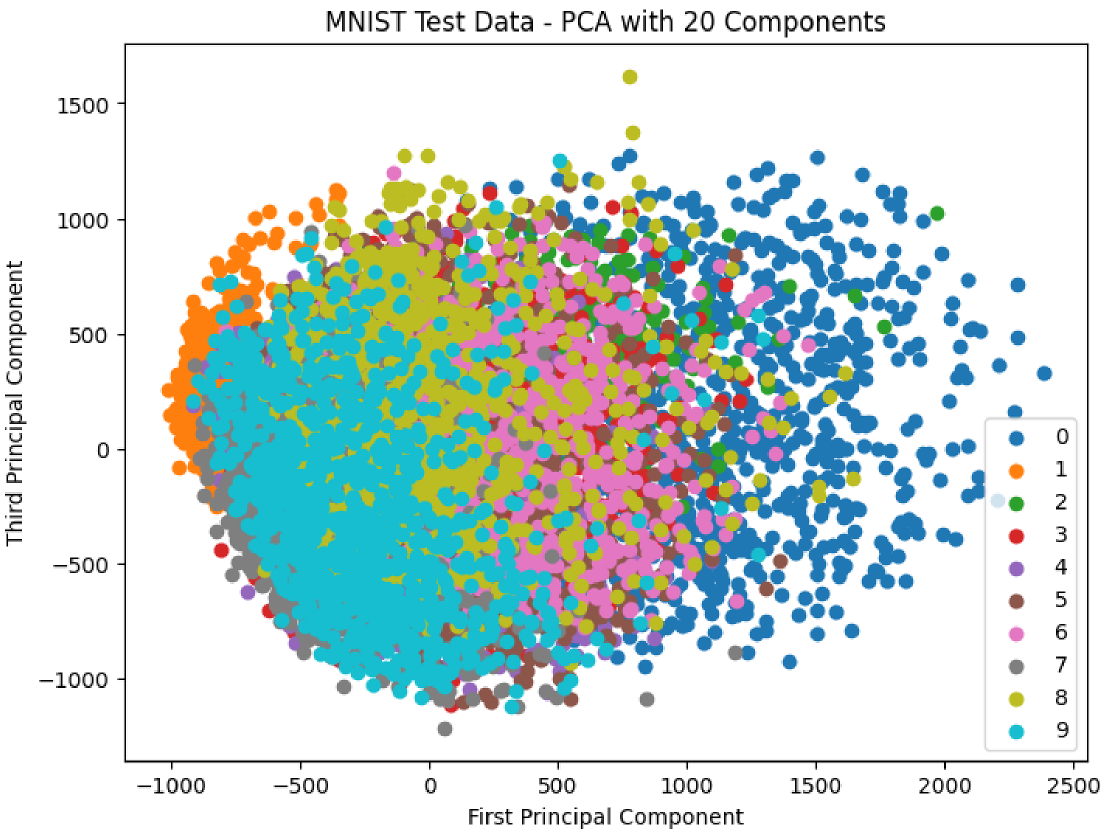
```
Average accuracy with 5 components: 65.20%
```


MNIST Test Data - PCA with 10 Components
Second Principal Component
First Principal Component
1500
1000
500
0
−500
−1000
−1000
−500
0
500
1000
1500
2000
2500
0
1
2
3
4
5
6
7
8
9



```
Average accuracy with 10 components: 81.20%
```


MNIST Test Data - PCA with 20 Components
Second Principal Component
First Principal Component
1500
1000
500
0
−500
−1000
−1000
−500
0
500
1000
1500
2000
2500
0
1
2
3
4
5
6
7
8
9

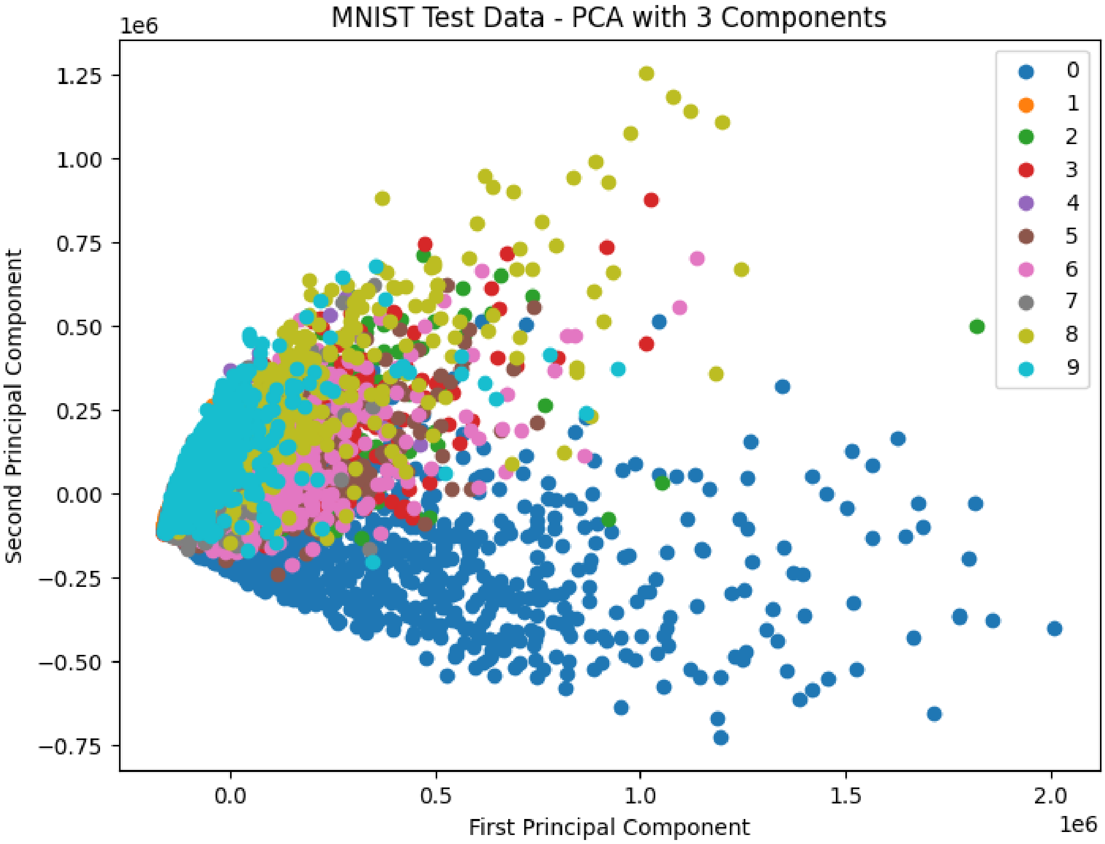


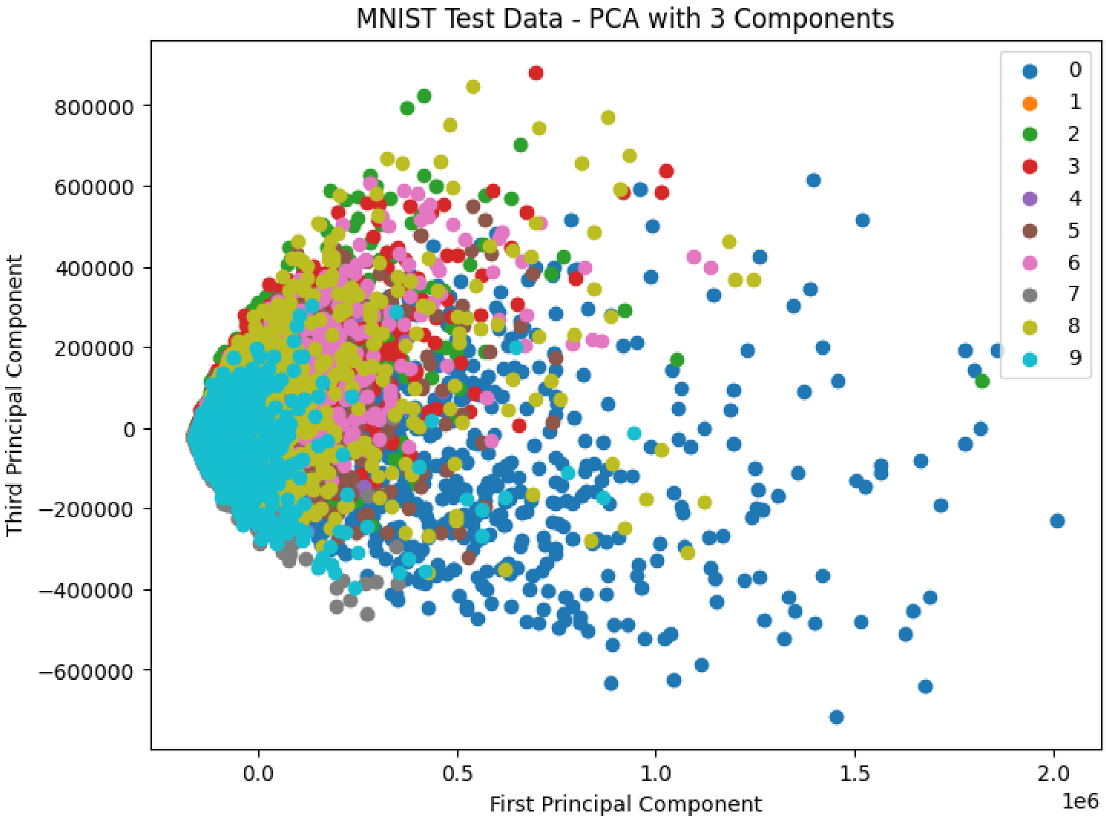
```
Average accuracy with 20 components: 85.40%
```

0.5 Non-linear Versions of PCA

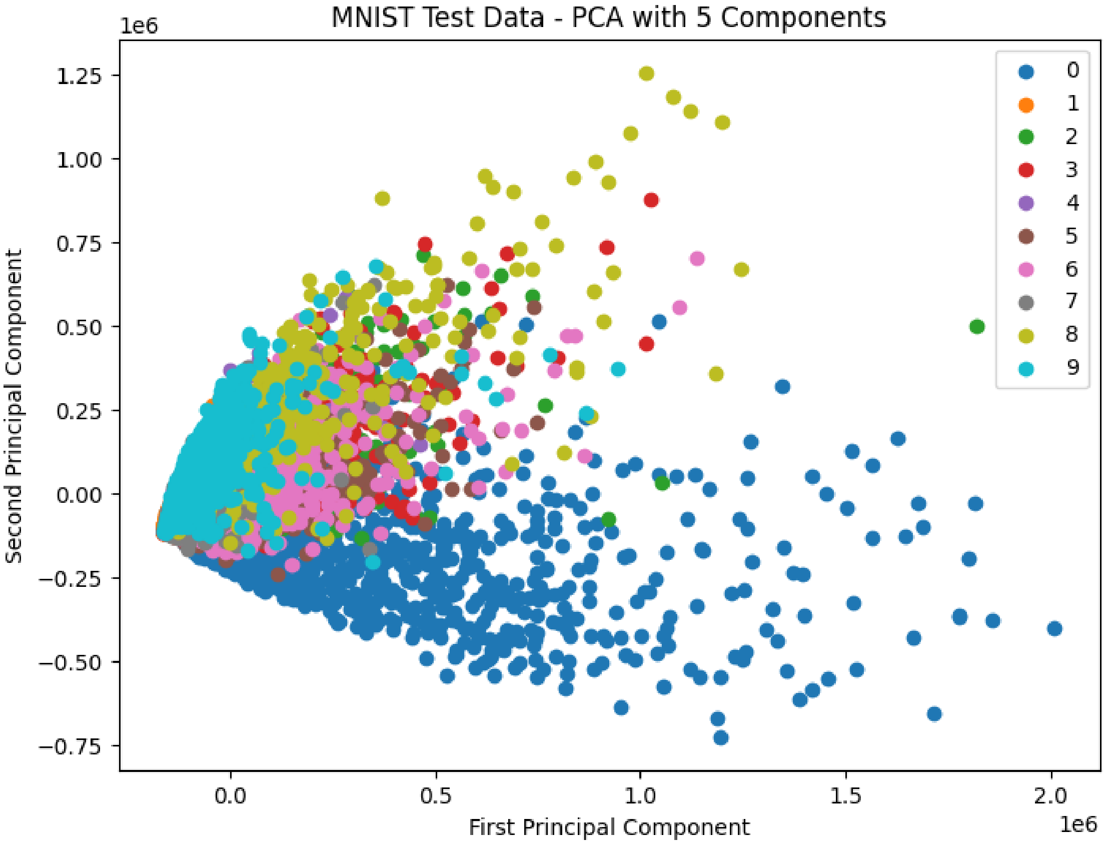
```
[9]: num_components = [3, 5, 10, 20]  # Choose the number of components for reduction



     for n in num_components:
         kpca = KernelPCA(n_components=n, kernel='poly', degree=3)
         reduced_X_train = kpca.fit_transform(x_train_subset)
         reduced_X_test = kpca.transform(X_test)

         plt.figure(figsize=(8, 6))

         for i in range(10):
             indices = np.where(y_test == i)[0]
             plt.scatter(reduced_X_test[indices, 0], reduced_X_test[indices, 1],␣
      ↪label=str(i))
         plt.xlabel("First Principal Component")
         plt.ylabel("Second Principal Component")
         plt.legend()
```

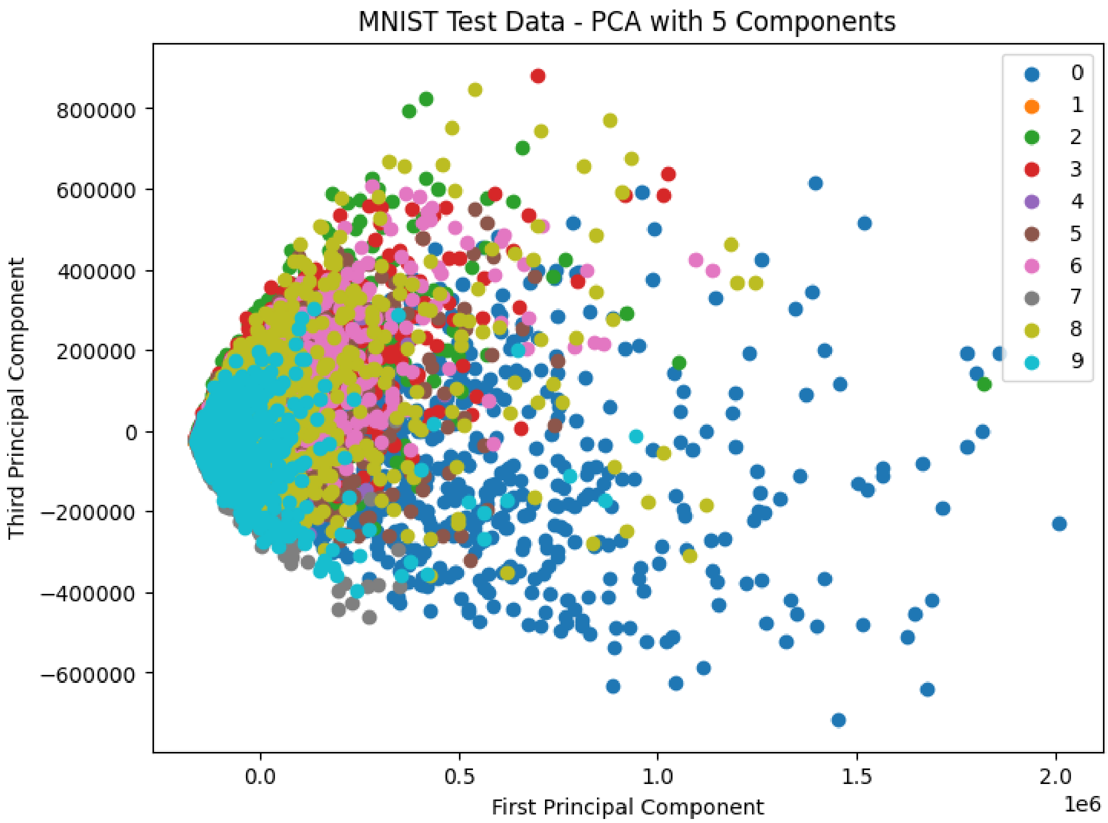
```
    plt.title("MNIST Test Data - PCA with {} Components".format(n))
    plt.show()

    if n >= 3:
        plt.figure(figsize=(8, 6))
        for i in range(10):
            indices = np.where(y_test == i)[0]
            plt.scatter(reduced_X_test[indices, 0], reduced_X_test[indices, 2],␣
↪label=str(i))
        plt.xlabel("First Principal Component")
        plt.ylabel("Third Principal Component")
        plt.legend()
        plt.title("MNIST Test Data - PCA with {} Components".format(n))
        plt.show()

    # Feed reduced features to Random Forest and perform classification with␣
↪cross-validation
    rf = RandomForestClassifier(n_estimators=100)

    accuracies = []

    # Perform cross-validation
    kf = KFold(n_splits=5,shuffle=True)

    for train_index, test_index in kf.split(reduced_X_train):

        x_cv_train, x_cv_test = reduced_X_train[train_index],␣
↪reduced_X_train[test_index]
        y_cv_train, y_cv_test = y_train_subset[train_index],␣
↪y_train_subset[test_index]


        rf.fit(x_cv_train, y_cv_train)


        y_pred = rf.predict(x_cv_test)
        accuracy = accuracy_score(y_cv_test, y_pred)
        accuracies.append(accuracy)



    # Calculate the average accuracy over all folds
    avg_accuracy = np.mean(accuracies)

    print("Average accuracy with {} components: {:.2f}%".format(n, avg_accuracy␣
↪* 100))
```


MNIST Test Data - PCA with 3 Components
1e6
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Second Principal Component
0.0
0.5
1.0
1.5
2.0
1e6
First Principal Component
0
1
2
3
4
5
6
7
8
9

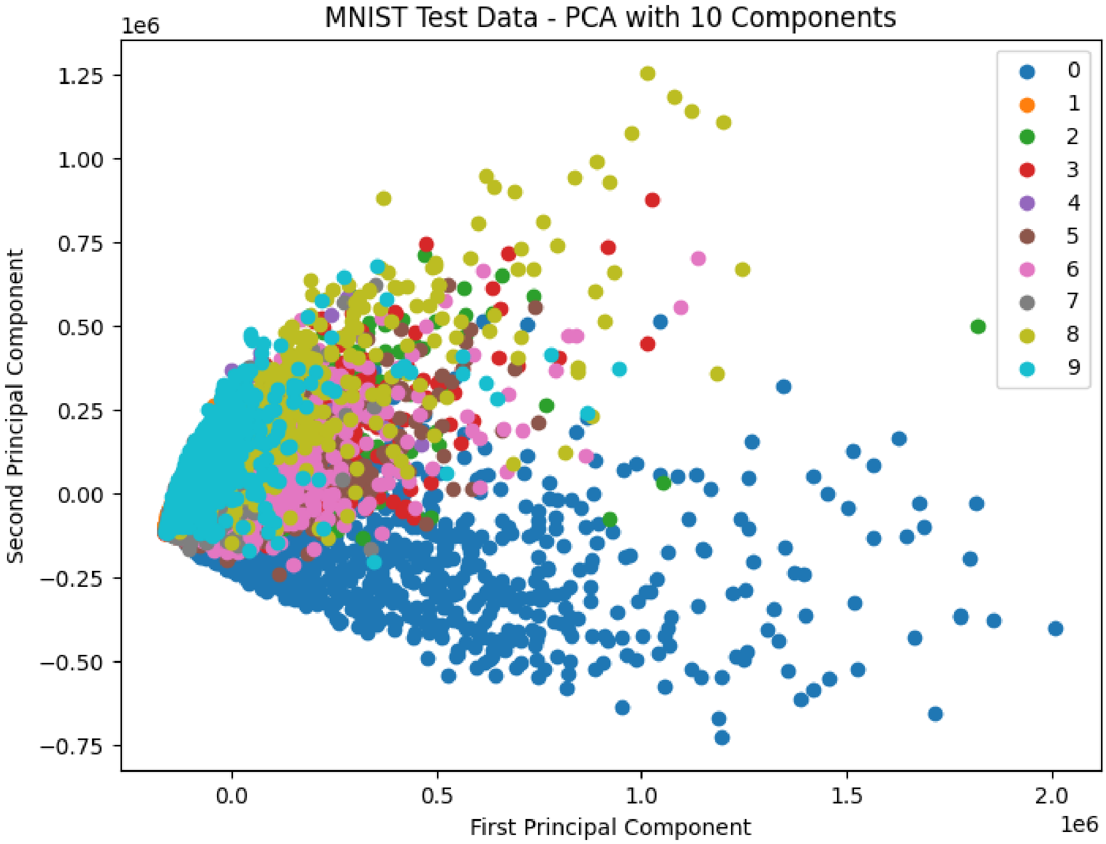


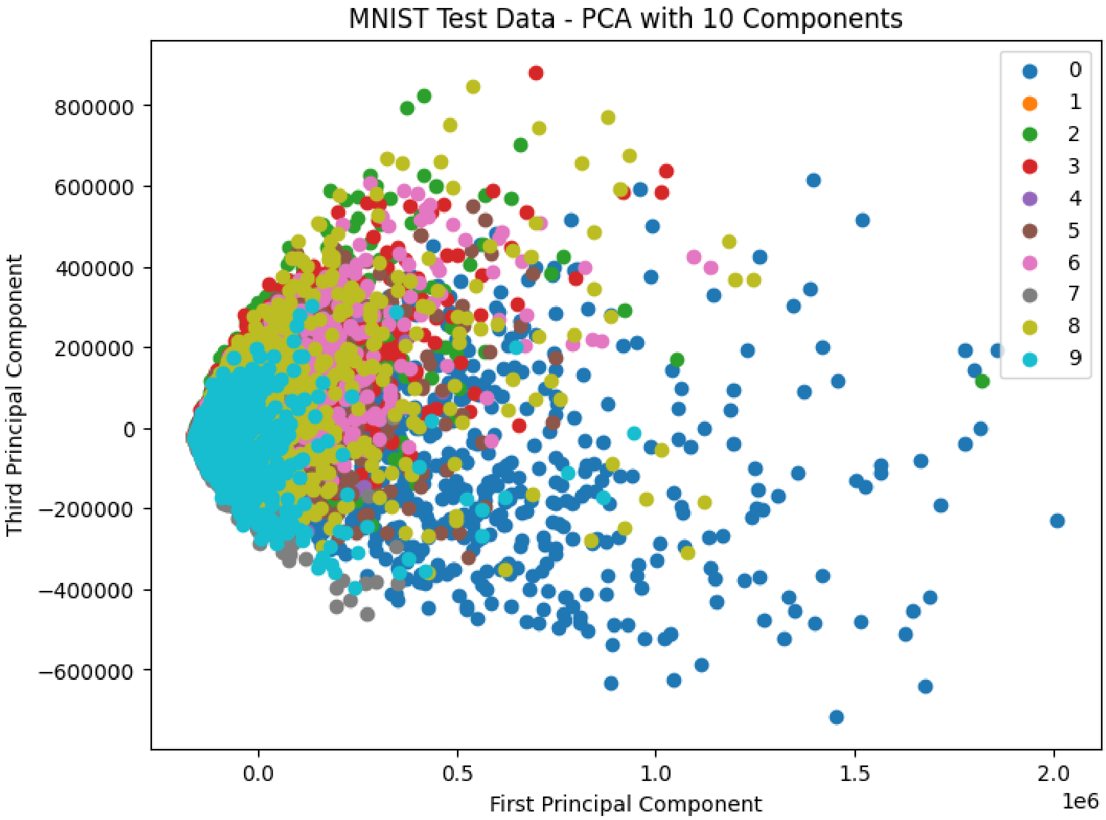
```
Average accuracy with 3 components: 37.10%
```


MNIST Test Data - PCA with 5 Components
1e6
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Second Principal Component
0.0
0.5
1.0
1.5
2.0
1e6
First Principal Component
0
1
2
3
4
5
6
7
8
9

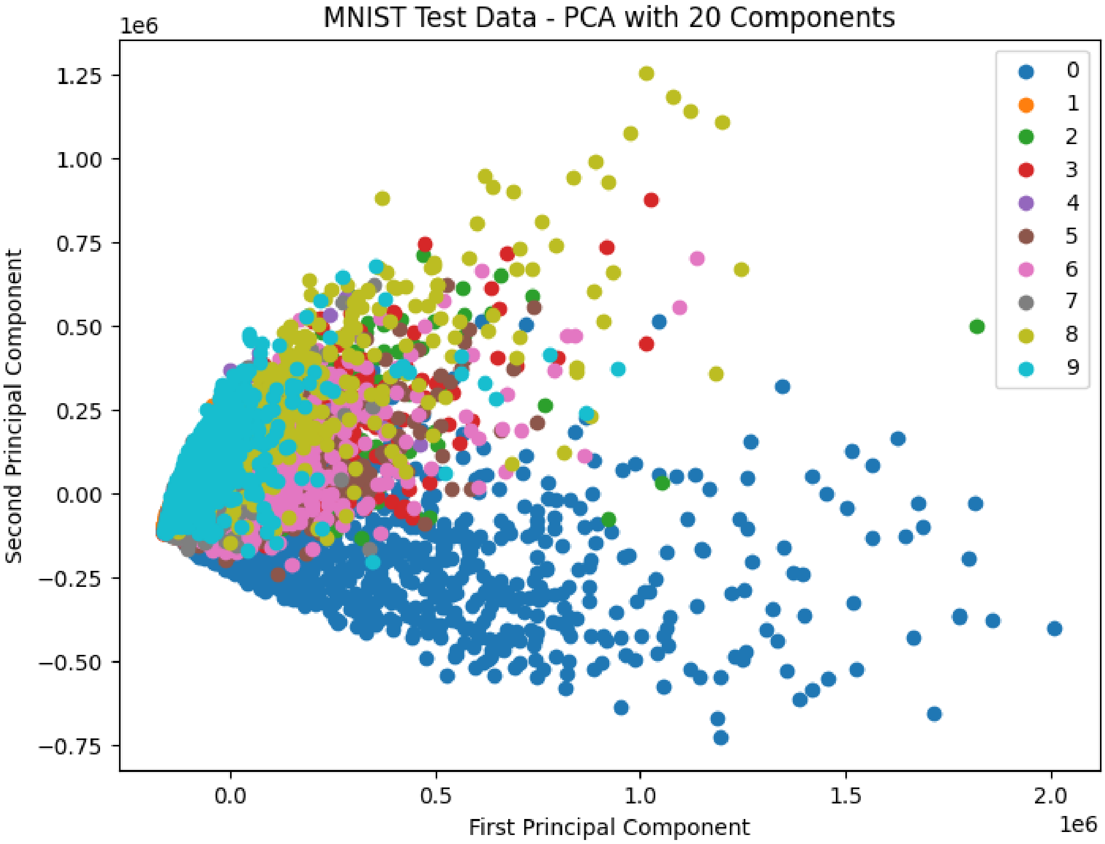


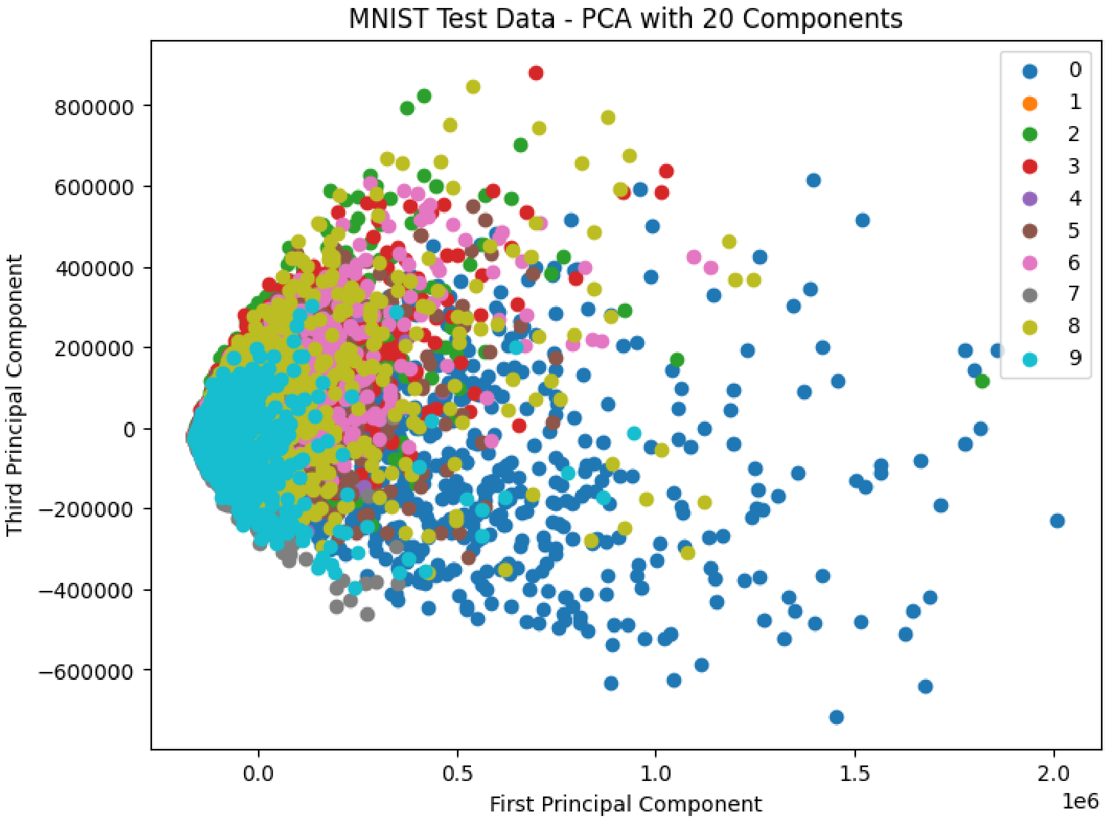
```
Average accuracy with 5 components: 61.10%
```


MNIST Test Data - PCA with 10 Components
1e6
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Second Principal Component
0.0
0.5
1.0
1.5
2.0
1e6
First Principal Component
0
1
2
3
4
5
6
7
8
9



```
Average accuracy with 10 components: 77.20%
```


MNIST Test Data - PCA with 20 Components
1e6
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Second Principal Component
0.0
0.5
1.0
1.5
2.0
1e6
First Principal Component
0
1
2
3
4
5
6
7
8
9



```
Average accuracy with 20 components: 82.50%
```