MNIST Handwritten Digit Classification Using Naive Bayes

Indian Institute of Technology Goa

Course: Probability and Statistics for Computer Science[CS561]
Course Code: CS561
Academic Year: 2023-2024

Course Instructor: Dr. Satyanath Bhat

Submitted by: Karthik Pai S [2313101]
Ramya R Shet [2313102]

Contents

1 Introduction

This report presents the implementation of a Naive Bayes classifier on the MNIST handwritten digit dataset. The objective is to classify 28* 28-pixel greyscale images into one of the 10 digits from 0-9.

The MNIST dataset is a handwritten dataset that consists of 60,000 handwritten digits for training the machine learning model and 10,000 handwritten digits for testing the model. Each image consists of a 28*28-pixel greyscale image corresponding to a particular single digit number. The goal of the project is to create a classifier which can predict the single digit number stored in the image.

When applying Naïve Bayes for image classification with the MNIST dataset, several steps are taken to preprocess the data, train the model and make predictions. Firstly, the MNIST dataset is loaded from the keras module. Then the data is preprocessed by flattening the 28*28-pixel images into 1 dimensional array and normalizing them to a common scale between 0 and 1. Gaussian Naïve Bias classifier is trained using 60,000 of the handwritten training datasets. Mean and standard deviation is calculated for each pixel of each digit class in the training set. This mean and standard deviation is used to model the Gaussian Distribution of pixels for each digit class. The classifier tries to predict each test dataset to the appropriate digit class by calculating its likelihood of belonging to a particular digit class using the Gaussian Distribution parameters. Naïve Bayes formula is used to compute the posterior probability for each digit class. The test dataset is predicted to belong to belong to the digit class with the highest probability. The model's accuracy is tested using the test dataset. A confusion matrix is plotted to understand how correctly the model is predicting on the test images.

2 Naive Bayes Classification

The Naïve Bayes Classifier is a supervised machine learning algorithm, it used for classification task. It is a probabilistic classifier based on Bayes Theorem. Bayes theorem gives the probability of an event based on new information. This is used by Naïve Bayes to make a prediction based on the input, where the input provides the new information.

2.1 Naive Bayes Assumption

Naïve Bayes assumes that the features that is the pixels used to represent the digits are independent of each other, given the digit label. Basically, it considers that the pixels of the digit images are unrelated given the digit class is known. However, this is not the case in reality, as the pixels of an image are not completely independent of each other. This assumption allows Naïve Bayes to have easier computation as shown below:

$$P(x_1, x_2, x_3, \dots, x_{784}|C) = P(x_1|C) \times P(x_2|C) \times P(x_3|C) \times \dots \times P(x_{784}|C)$$

Where $\mathrm{P}(\mathrm{x}_1,\ \mathrm{x}_2,\ \mathrm{x}_3,\ \dots,\ \mathrm{x}_{784}\ |\ \mathrm{C})$ is the joint probability of all the pixels

$P(x_i|C)$ is the probability of the pixel xi given the digit label C

$x_1, x_2, x_3, \dots, x_{784}$ are the pixels in the dataset.

Thus, due to the assumption that the pixels are independent of each other the joint probability given the digit label can be written as product of individual probabilities given the digit label.

2.2 Image Classification

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Here x represents the image, basically it's a 28*28=784 vector representing all the 784 pixels of the input image.

$P(c|x)$ is the posterior probability that the class is c given that the data vector is x.

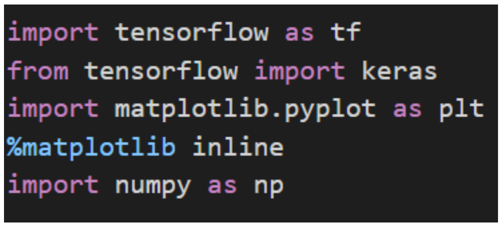
$\mathrm{P}(x|c)$ is the probability that the data vector x belongs to class c.

We have to check the posterior probability for all the digit class for a given data set using the above equation and the data set is predicted to belong to the digit class with the highest probability.

$P(x)$ is constant for all the digit labels so the numerator $P(x|c)P(c)$ becomes the contributing factor.

3 Building the Naïve Bayes Classifier

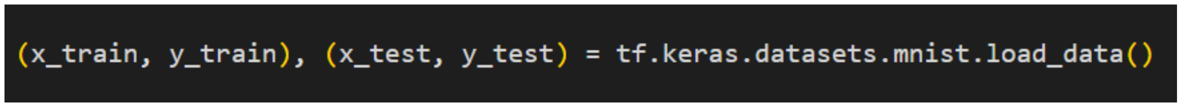
3.1 Loading the MNIST Data



```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

FIG 3.1 Imports

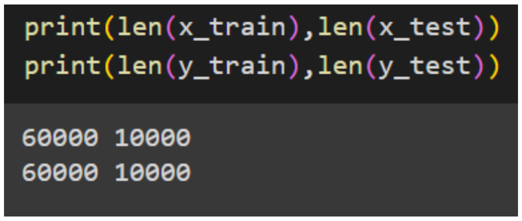
Firstly, the MNIST dataset set is loaded from the keras module after importing the required modules.



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

FIG 3.2 Loading the training and test set

The data set is divided into 60,000 training set data and 10,000 test set data

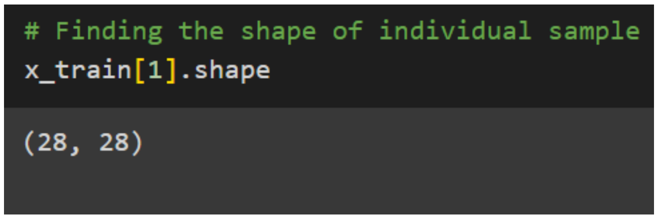


```
print(len(x_train),len(x_test))
print(len(y_train),len(y_test))
```

```
60000 10000
60000 10000
```

FIG 3.3 Data set length for training and test set.

Each data set represent a 28*28 greyscale image.

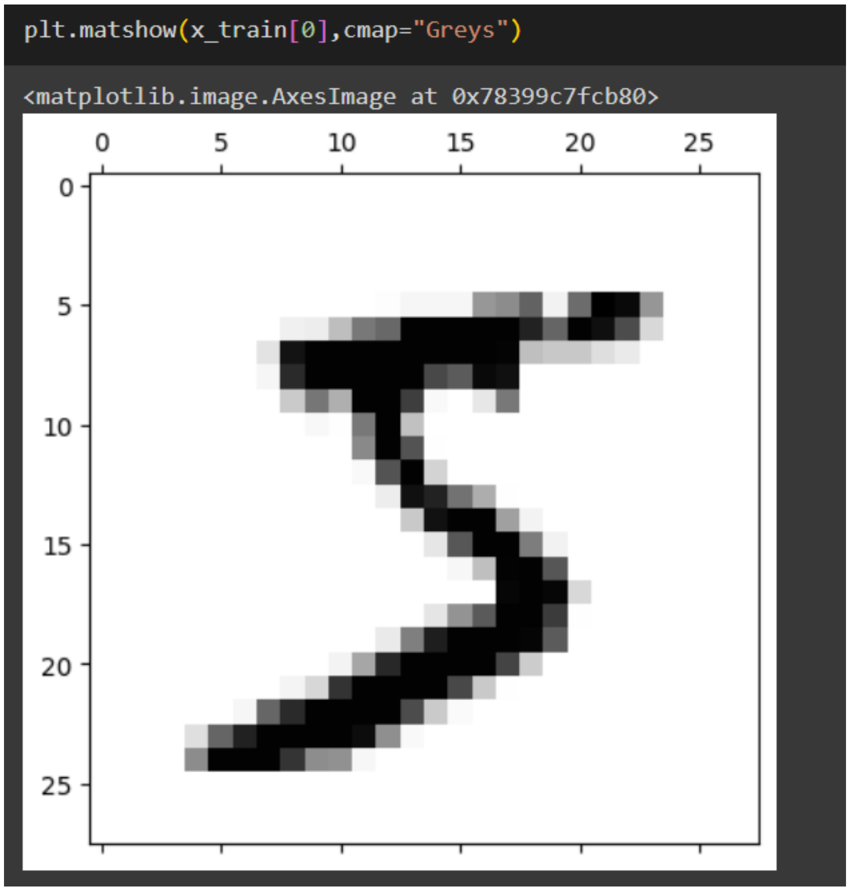


```
# Finding the shape of individual sample
x_train[1].shape
```

```
(28, 28)
```

FIG 3.4 Image size

The value of each pixel is from the rage 0 to 255. Following is plot of a sample data set.



```
plt.matshow(x_train[0],cmap="Greys")
```

```
<matplotlib.image.AxesImage at 0x78399c7fcb80>
```

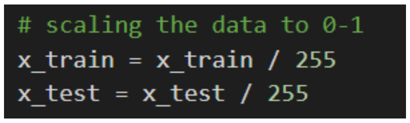
FIG 3.5 Sample greyscale image.

3.2 Data preprocessing

Both the training and test data are normalized and flattened.

3.2.1 Normalization

The training and test data are normalized by dividing it by 255.

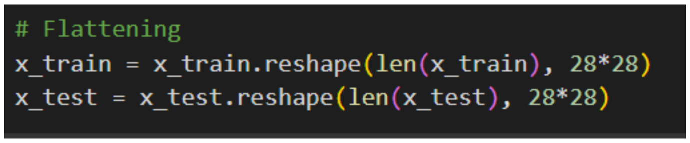


```
# scaling the data to 0-1
x_train = x_train / 255
x_test = x_test / 255
```

FIG 3.6 Normalization of the data set

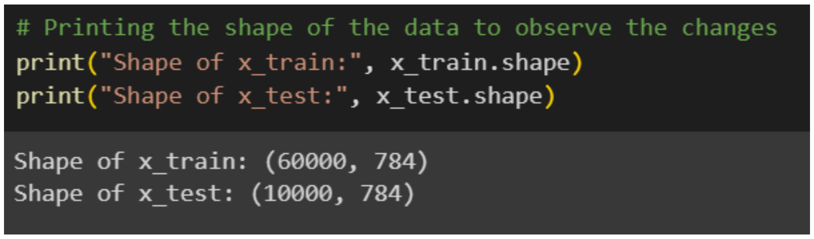
3.2.2 Flattening

The 2D data set images are converted to 1D vectors, thus converting the 28*28-pixel 2D images to 784-pixel 1D vector.



```
# Flattening
x_train = x_train.reshape(len(x_train), 28*28)
x_test = x_test.reshape(len(x_test), 28*28)
```

FIG 3.7 Flattening of the data set



```
# Printing the shape of the data to observe the changes
print("Shape of x_train:", x_train.shape)
print("Shape of x_test:", x_test.shape)
```

```
Shape of x_train: (60000, 784)
Shape of x_test: (10000, 784)
```

FIG 3.8 Data after processing

3.3 Training the Naive Bayes Classifier

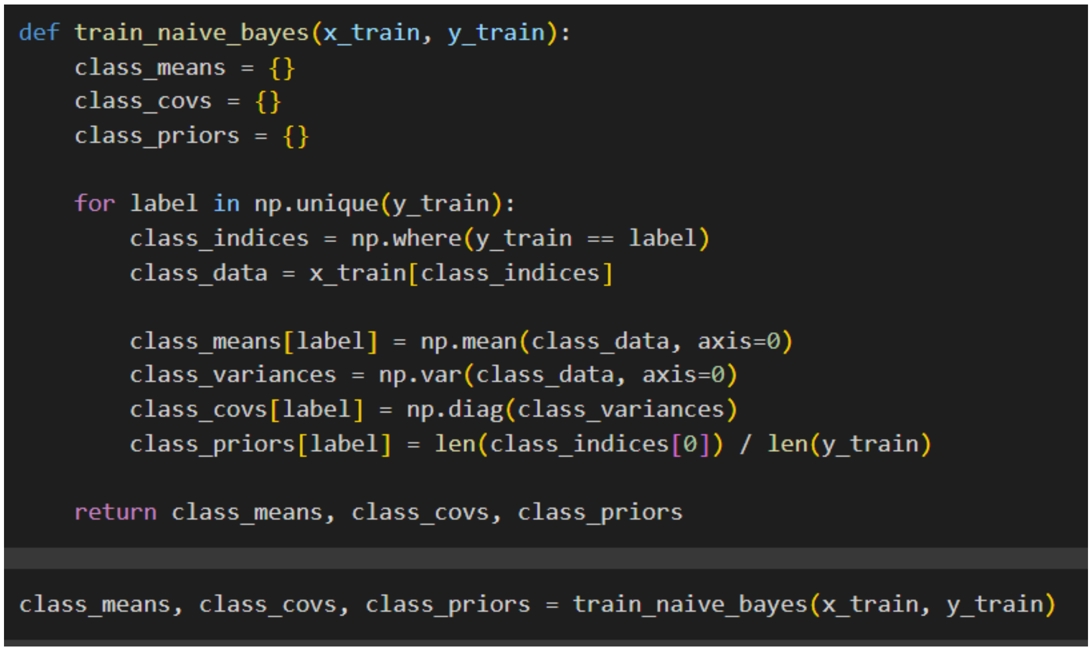
Multivariate Gaussian is used to model the data.

$$P(x|c) = \frac{1}{\sqrt{(2\pi)^D|\Sigma|}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Probabilities tend to be very small (populating near 0) when the number of dimensions is very large, thus we can make use of log of the Multivariate Gaussian formula instead of the Multivariate Gaussian to get:

$$\log P(x \mid c) = -\frac{D}{2}\ln(2\pi) - \frac{1}{2} ln|\Sigma| - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$$

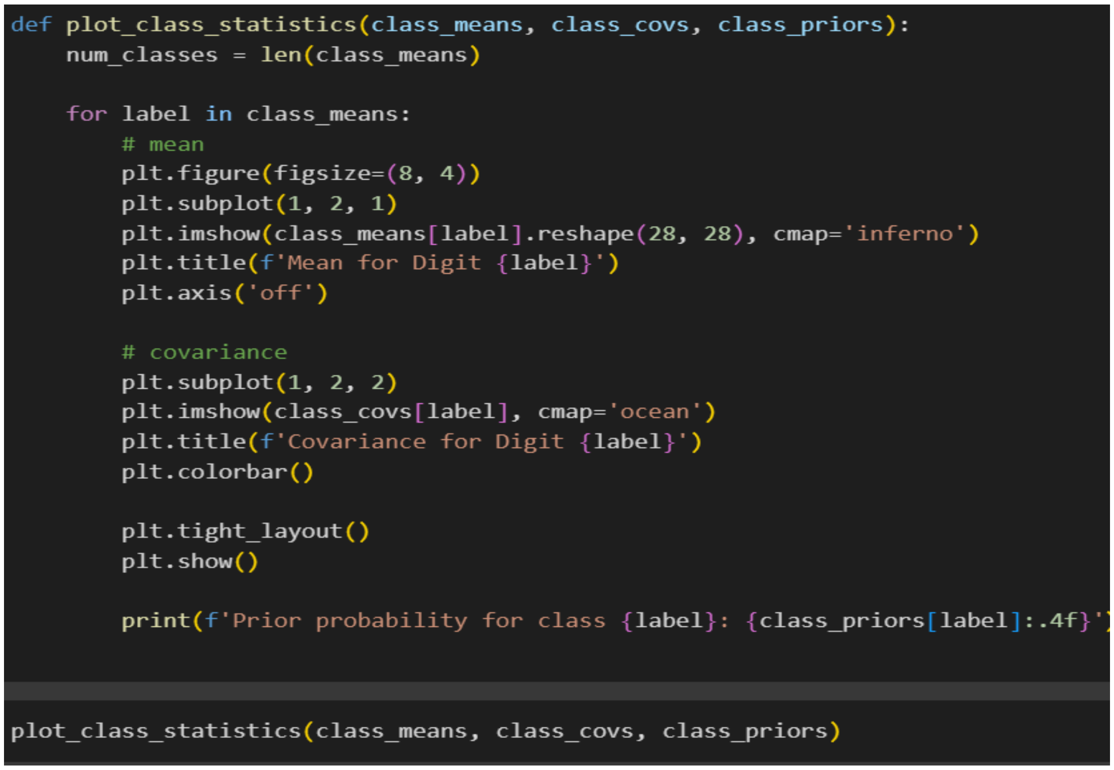
The mean and the covariance matrix to substitute in the above formula is calculated by using the training set data for all the digit class.



```python
def train_naive_bayes(x_train, y_train):
    class_means = {}
    class_covs = {}
    class_priors = {}

    for label in np.unique(y_train):
        class_indices = np.where(y_train == label)
        class_data = x_train[class_indices]

        class_means[label] = np.mean(class_data, axis=0)
        class_variances = np.var(class_data, axis=0)
        class_covs[label] = np.diag(class_variances)
        class_priors[label] = len(class_indices[0]) / len(y_train)

    return class_means, class_covs, class_priors
```

```python
class_means, class_covs, class_priors = train_naive_bayes(x_train, y_train)
```

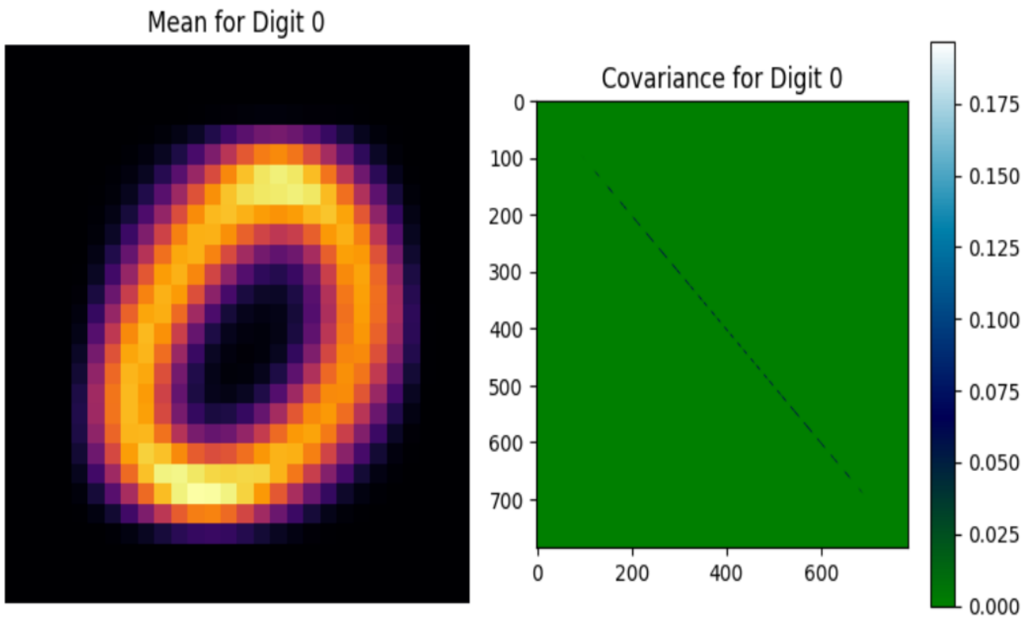
FIG 3.9 Calculating the mean, covariance and probability of occurrence of a image of a particular digit for the training set.



```python
def plot_class_statistics(class_means, class_covs, class_priors):
    num_classes = len(class_means)

    for label in class_means:
        # mean
        plt.figure(figsize=(8, 4))
        plt.subplot(1, 2, 1)
        plt.imshow(class_means[label].reshape(28, 28), cmap='inferno')
        plt.title(f'Mean for Digit {label}')
        plt.axis('off')

        # covariance
        plt.subplot(1, 2, 2)
        plt.imshow(class_covs[label], cmap='ocean')
        plt.title(f'Covariance for Digit {label}')
        plt.colorbar()

        plt.tight_layout()
        plt.show()

        print(f'Prior probability for class {label}: {class_priors[label]:.4f}'
```

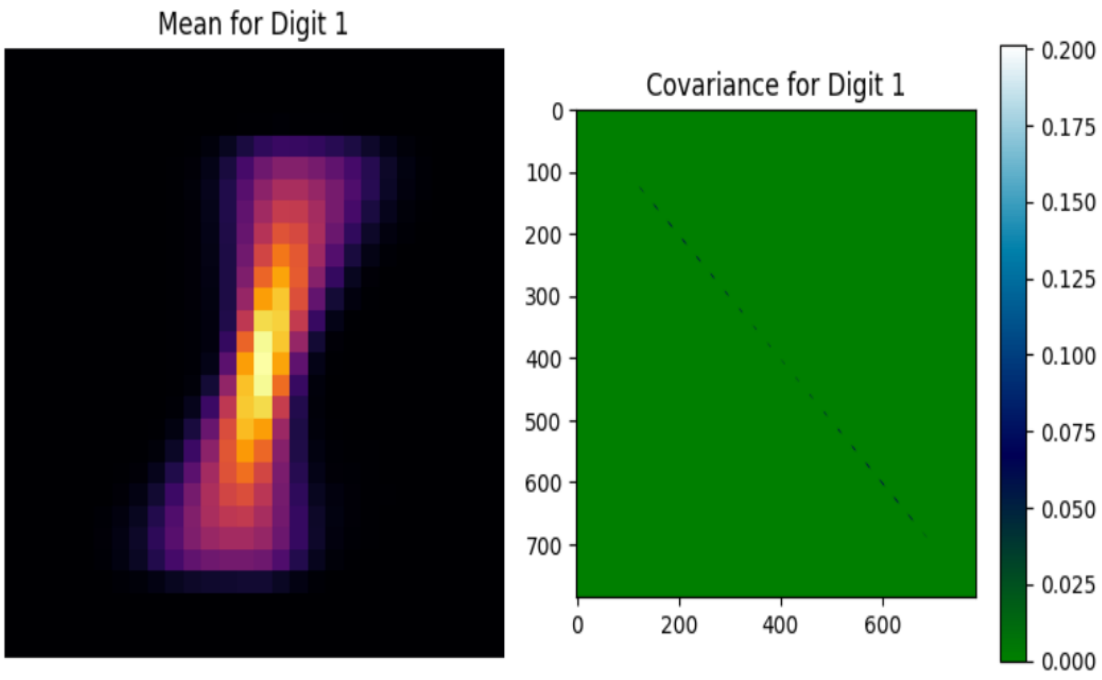
```python
plot_class_statistics(class_means, class_covs, class_priors)
```

FIG 3.10 Code for plotting mean and covariance for all digit class



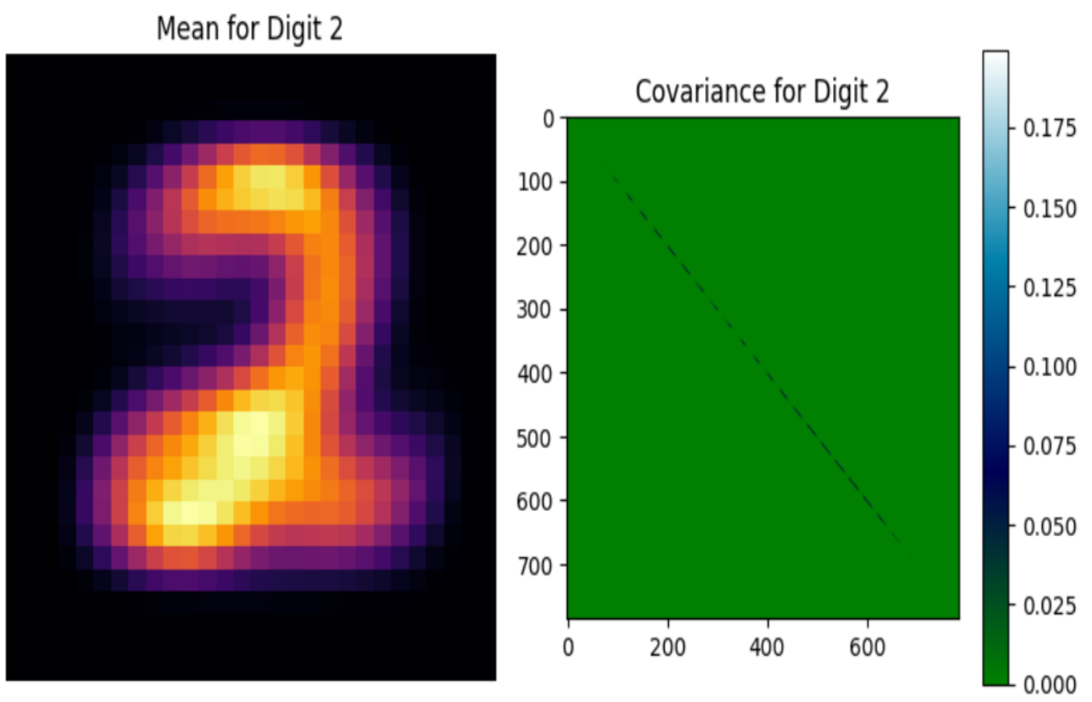
```
Prior probability for class 0: 0.0987
```

FIG 3.11 Plot of mean and covariance for 0 class



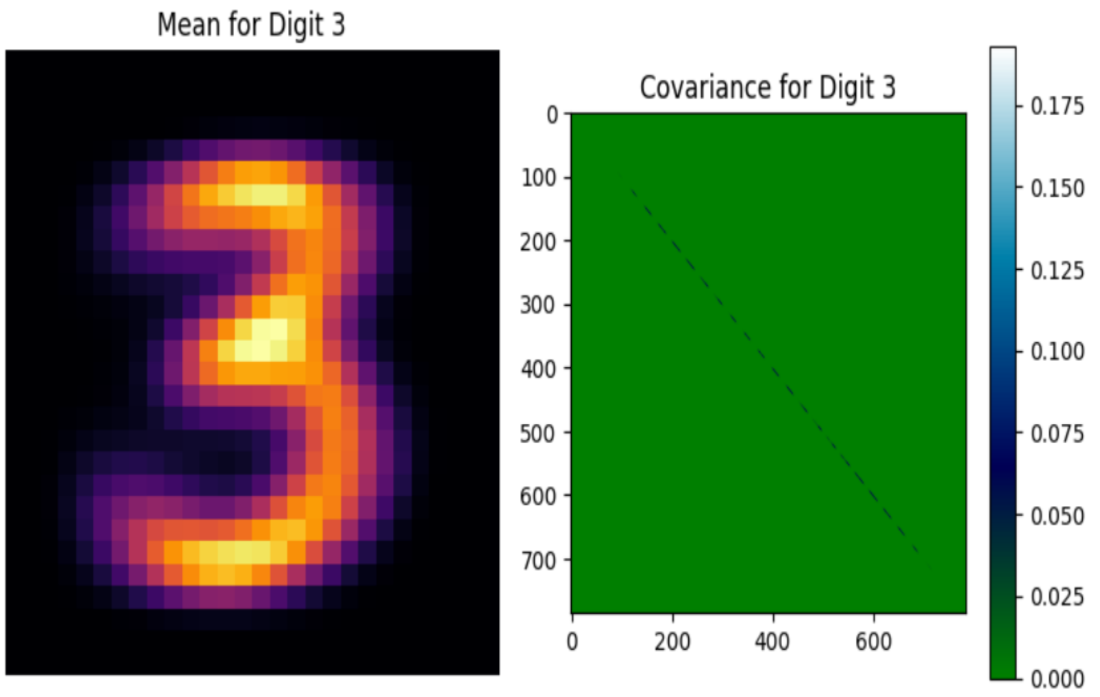
```
Prior probability for class 1: 0.1124
```

FIG 3.12 Plot of mean and covariance for 1 class



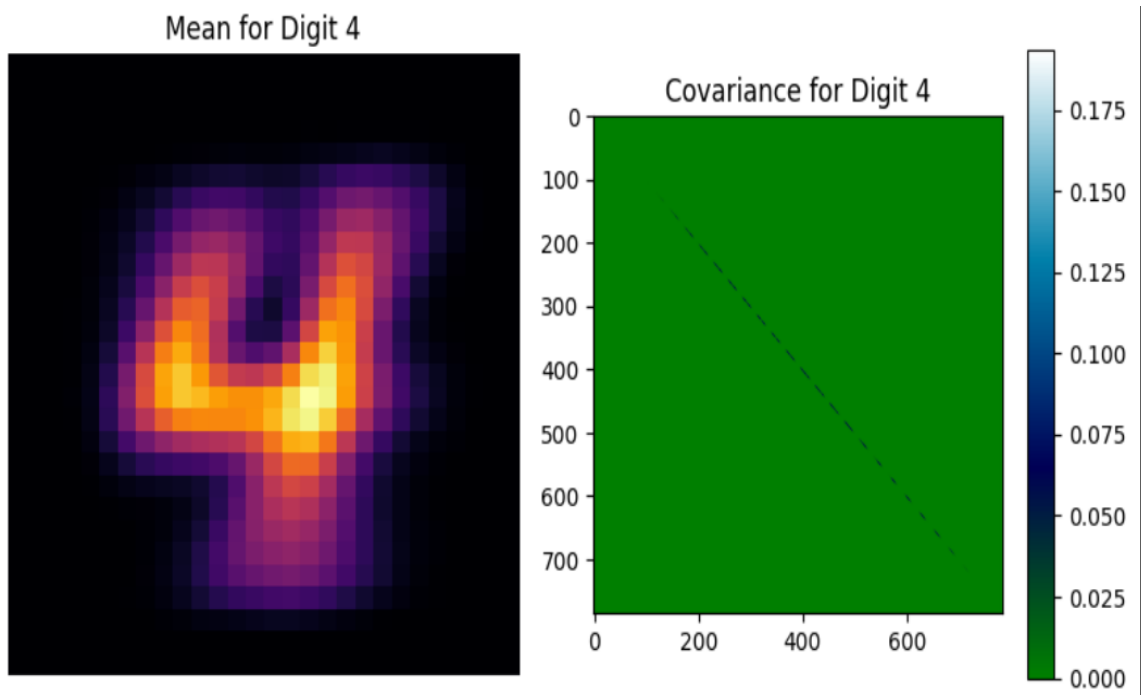
```
Prior probability for class 2: 0.0993
```

FIG 3.13 Plot of mean and covariance for 2 class



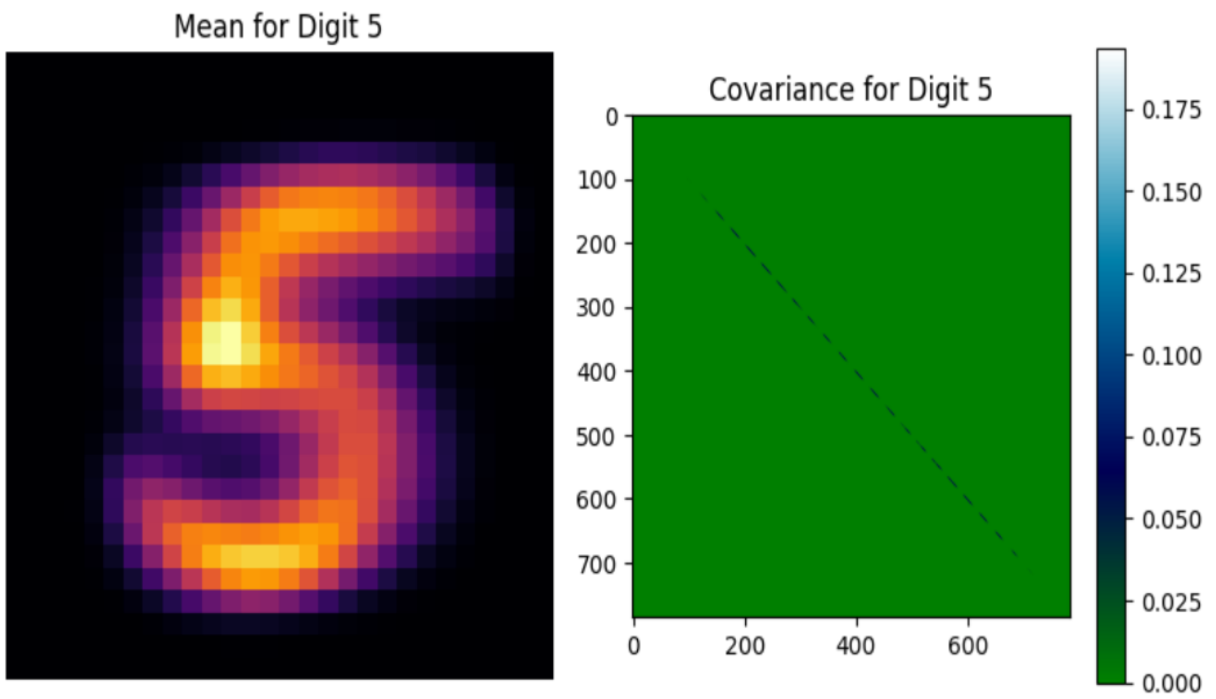
```
Prior probability for class 3: 0.1022
```

FIG 3.14 Plot of mean and covariance for 3 class



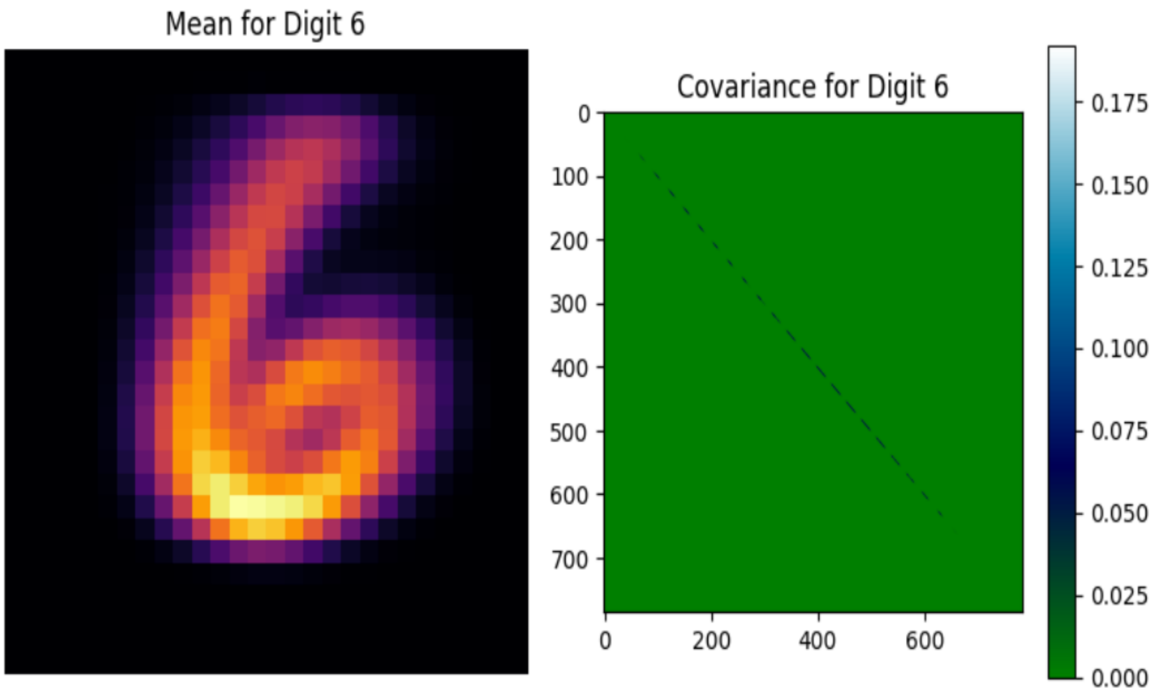
```
Prior probability for class 4: 0.0974
```

FIG 3.15 Plot of mean and covariance for 4 class



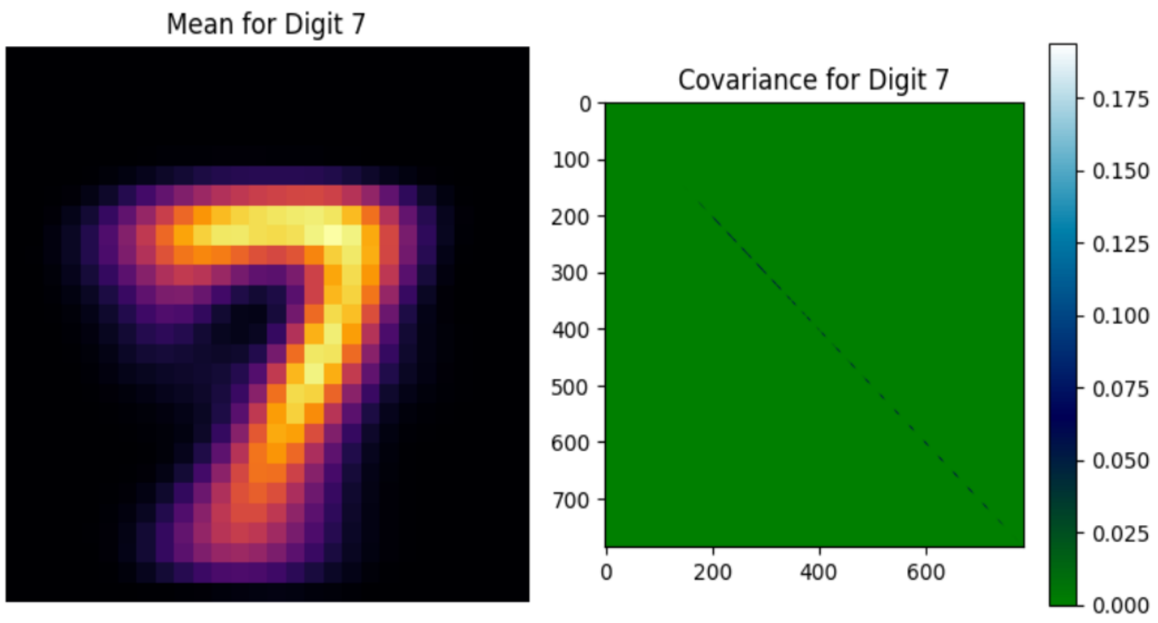
```
Prior probability for class 5: 0.0903
```

FIG 3.16 Plot of mean and covariance for 5 class



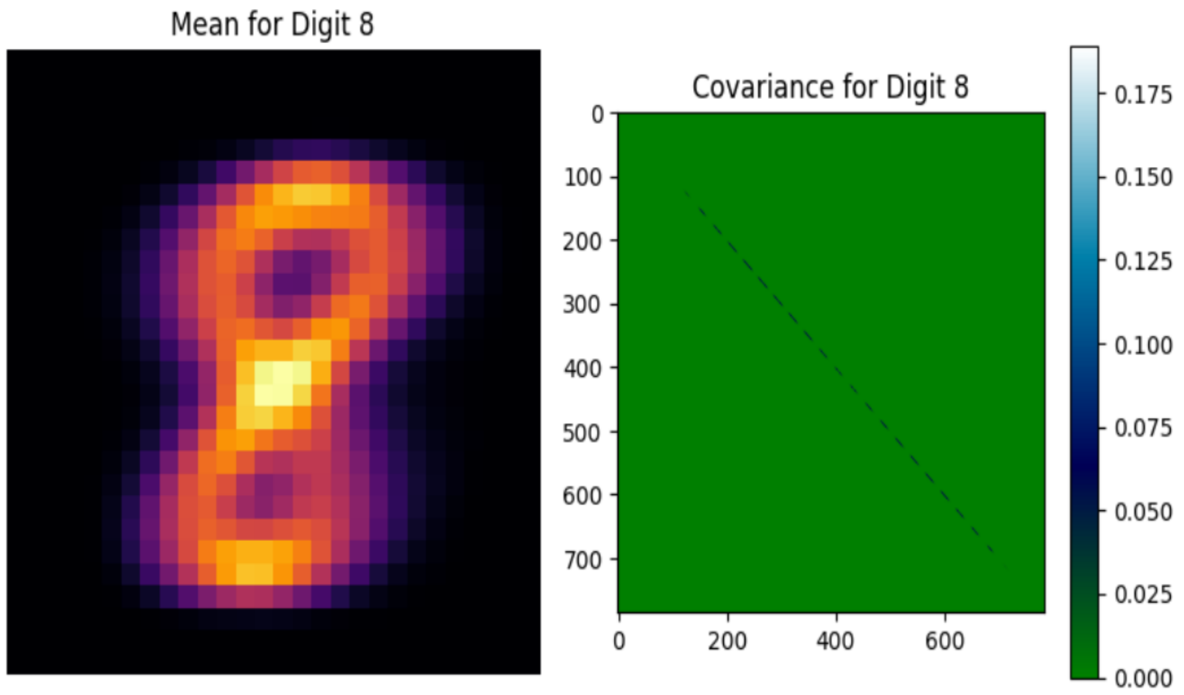
```
Prior probability for class 6: 0.0986
```

FIG 3.17 Plot of mean and covariance for 6 class



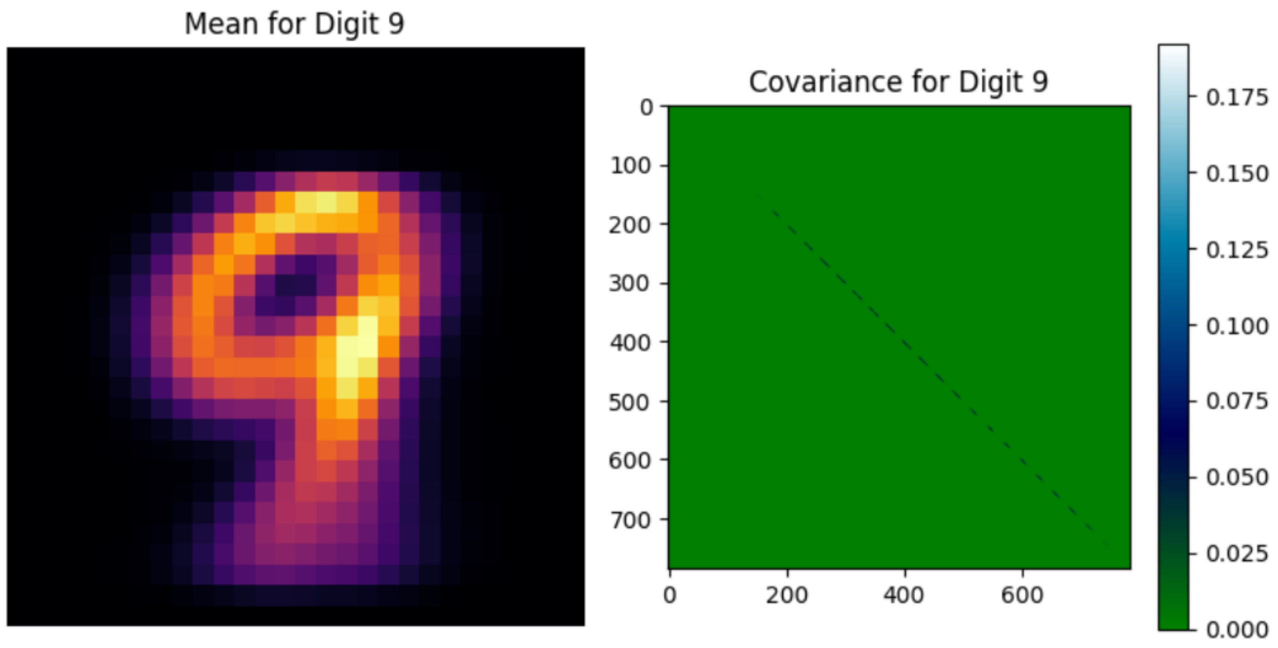
```
Prior probability for class 7: 0.1044
```

FIG 3.18 Plot of mean and covariance for 7 class



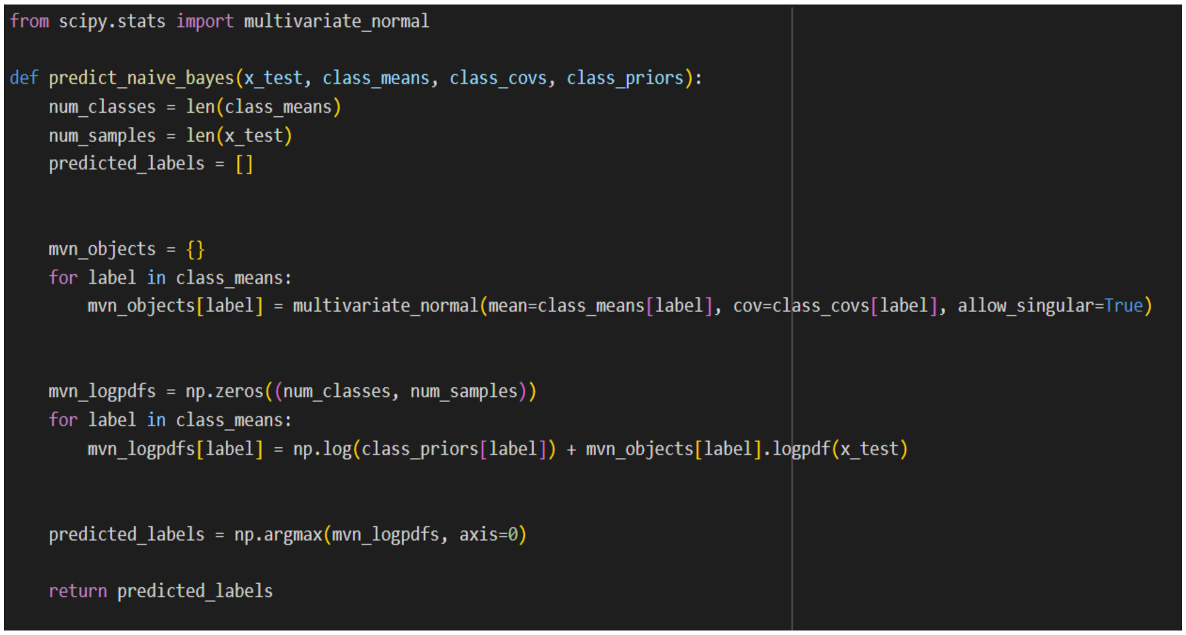
```
Prior probability for class 8: 0.0975
```

FIG 3.19 Plot of mean and covariance for 8 class



```
Prior probability for class 9: 0.0992
```

FIG 3.20 Plot of mean and covariance for 9 class



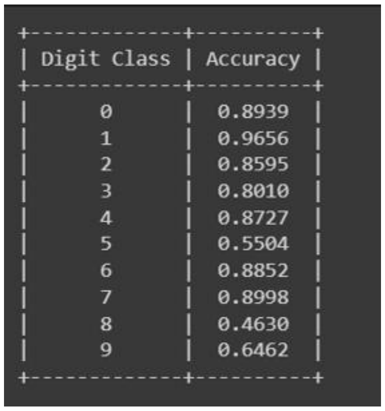
```python
from scipy.stats import multivariate_normal

def predict_naive_bayes(x_test, class_means, class_covs, class_priors):
    num_classes = len(class_means)
    num_samples = len(x_test)
    predicted_labels = []


    mvn_objects = {}
    for label in class_means:
        mvn_objects[label] = multivariate_normal(mean=class_means[label], cov=class_covs[label], allow_singular=True)


    mvn_logpdfs = np.zeros((num_classes, num_samples))
    for label in class_means:
        mvn_logpdfs[label] = np.log(class_priors[label]) + mvn_objects[label].logpdf(x_test)


    predicted_labels = np.argmax(mvn_logpdfs, axis=0)

    return predicted_labels
```

FIG 3.20 Code for training the Multivariate Gaussian

4 Results

The following results were obtained after using the Naïve Bayes Classifier to predict on the test MNIST set and an accuracy of 78.92% was obtained.

4.1 Accuracy table

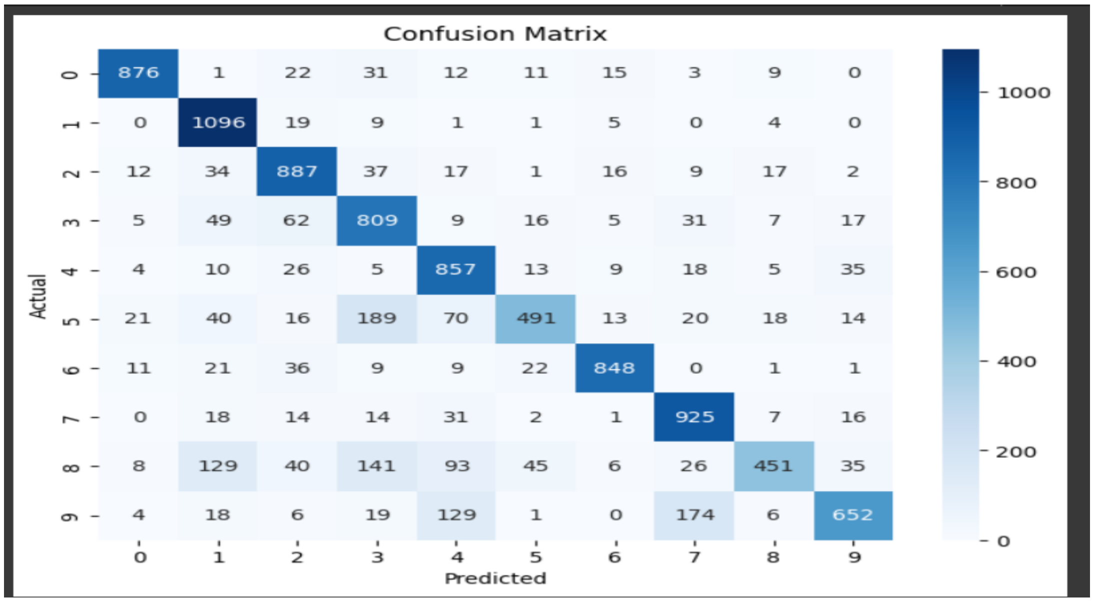


```
+-------------+----------+
| Digit Class | Accuracy |
+-------------+----------+
|      0      |  0.8939  |
|      1      |  0.9656  |
|      2      |  0.8595  |
|      3      |  0.8010  |
|      4      |  0.8727  |
|      5      |  0.5504  |
|      6      |  0.8852  |
|      7      |  0.8998  |
|      8      |  0.4630  |
|      9      |  0.6462  |
+-------------+----------+
```

Figure 4.1: Accuracy table

The accuracy table gives us the information of accuracy of prediction for individual digit class.

4.2 Confusion Matrix



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	876	1	22	31	12	11	15	3	9	0
1	0	1096	19	9	1	1	5	0	4	0
2	12	34	887	37	17	1	16	9	17	2
3	5	49	62	809	9	16	5	31	7	17
4	4	10	26	5	857	13	9	18	5	35
5	21	40	16	189	70	491	13	20	18	14
6	11	21	36	9	9	22	848	0	1	1
7	0	18	14	14	31	2	1	925	7	16
8	8	129	40	141	93	45	6	26	451	35
9	4	18	6	19	129	1	0	174	6	652

Figure 4.2: Confusion Matrix

Confusion matrix helps us understand where our classifier is getting confused and what digit is getting confused for what.

4.3 Percentage Error in Prediction

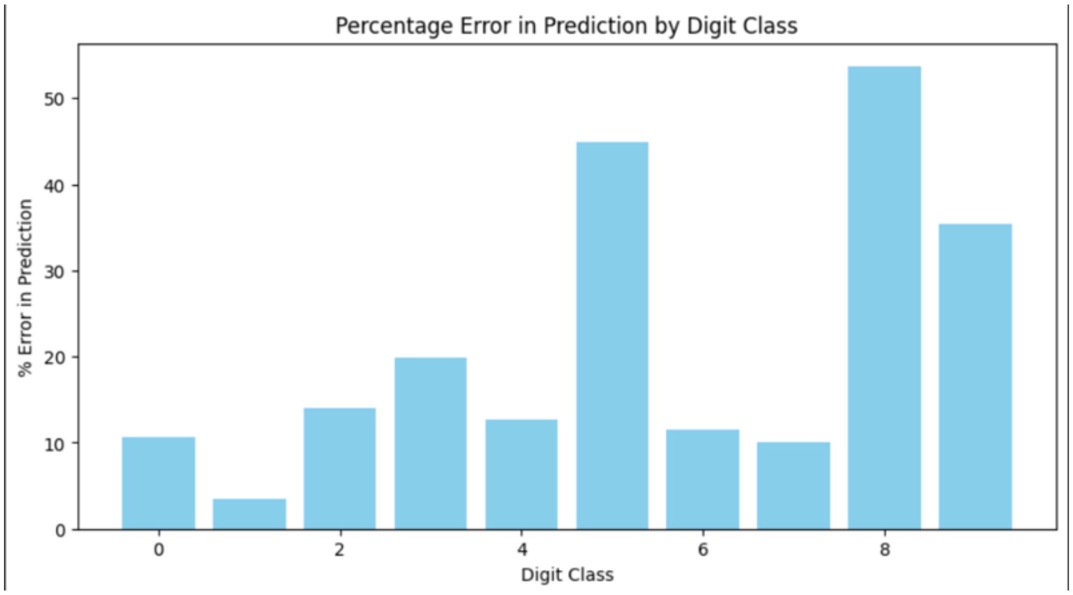


Figure 4.3: Percentage error in prediction.

The graph gives the percentage error in prediction for all the digit class.

5 Conclusion and Future Scope

5.1 Conclusion

Applying Naïve Bayes to the MNIST dataset helped us build an understanding on how probabilistic models can classify handwritten digits. It provided insights into the simplicity and limitations of assuming independence among pixels of the image data. The model provided an accuracy of 78.92% as it struggled to understand the relationship between pixels.

5.2 Future Scope

The projects future scope involves advancing beyond Naïve Bayes to more complex algorithms like deep learning. This could help to capture the relationship between the pixels and provide a higher accuracy.

References

[1] https://scikit-learn.org/stable/modules/naive_bayes.html.

[2] https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[3] https://en.wikipedia.org/wiki/Multivariate_normal_distribution

[4] https://web.stanford.edu/class/cs124/lec/naivebayes