MNIST Handwritten Digits Recognition Using NB, Non-NB and KNN Classifiers

Objective

Objective of this project is to recognize the handwritten digits written in medical charts.

Importing Standard Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading Train Data

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

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

```
data = pd.read_csv('/content/drive/MyDrive/ITC Projects/MNIST/MNIST_train.csv')
```

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

```
print('Shape of the Dataframe: ', data.shape)
data.head()
```

```
Shape of the Dataframe:  (60000, 787)
```

	Unnamed: 0	index	labels	0	1	2	3	4	5	6	...	774	775	776	777	778	779	780	781	782	783
0	0	0	5	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	2	2	4	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	3	3	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	4	4	9	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 787 columns

```
data.columns
```

```
Index(['Unnamed: 0', 'index', 'labels', '0', '1', '2', '3', '4', '5', '6',
       ...
       '774', '775', '776', '777', '778', '779', '780', '781', '782', '783'],
      dtype='object', length=787)
```

```
# Dropping the index columns
data = data.drop(columns=['Unnamed: 0', 'index'], axis = 1)
```

```
# Creating Dataframes for labels 0, 1, and 8
data_0 = data[data['labels'] == 0]
data_1 = data[data['labels'] == 1]
data_8 = data[data['labels'] == 8]

print(data_0.shape)
print(data_1.shape)
print(data_8.shape)
```

```
(5923, 785)
(6742, 785)
(5851, 785)
```

Converting X_train dataframes to numpy arrays

```
# Converting X_train dataframes to numpy arrays
X_train = data.to_numpy()
X_train_0 = data_0.to_numpy()
X_train_1 = data_1.to_numpy()
X_train_8 = data_8.to_numpy()

# Segregating y_train variables from X_train arrays
y_train = X_train[:,0]
y_train_0 = X_train_0[:,0]
y_train_1 = X_train_1[:,0]
y_train_8 = X_train_8[:,0]

# Storing the pixel columns in X_train arrays
X_train = X_train[:, 1:]
X_train_0 = X_train_0[:, 1:]
X_train_1 = X_train_1[:, 1:]
X_train_8 = X_train_8[:, 1:]

# Shape of X_train arrays
print(X_train.shape)
print(X_train_0.shape)
print(X_train_1.shape)
print(X_train_8.shape)

# Shape of y_train arrays
print(y_train.shape)
print(y_train_0.shape)
print(y_train_1.shape)
print(y_train_8.shape)
```

```
(60000, 784)
(5923, 784)
(6742, 784)
(5851, 784)
(60000,)
(5923,)
(6742,)
(5851,)
```

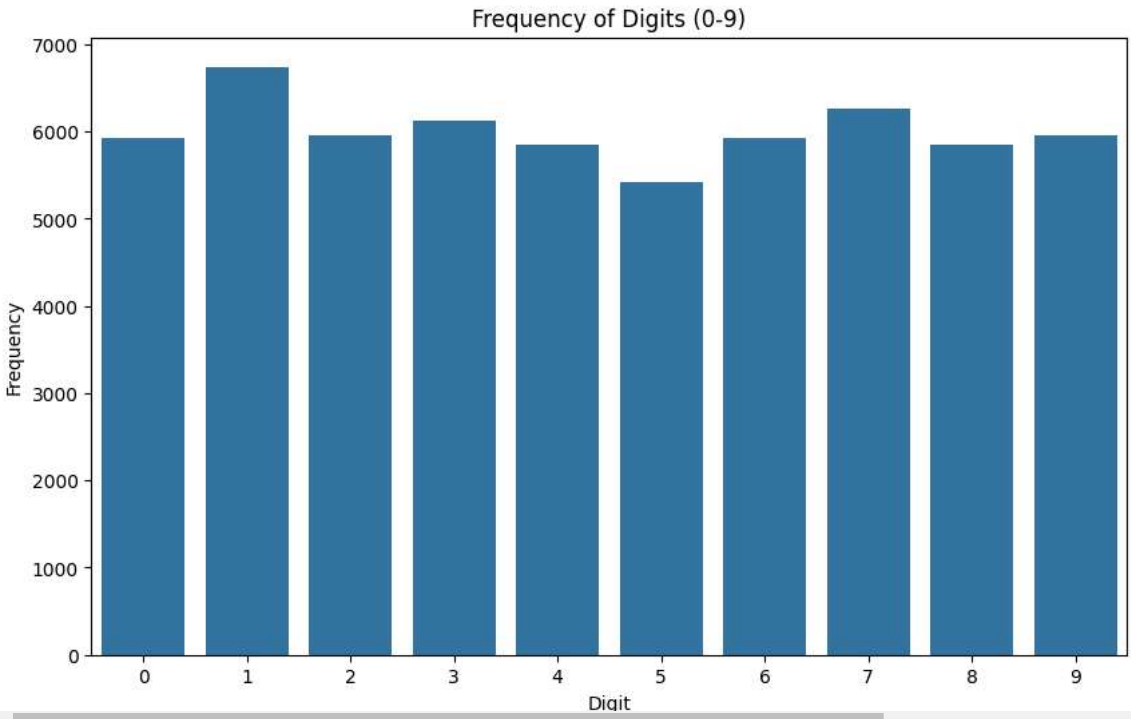
Data Visualizations

```
# Checking the unique values in y_train variable
y_train_unique = np.unique(y_train)
print(y_train_unique)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

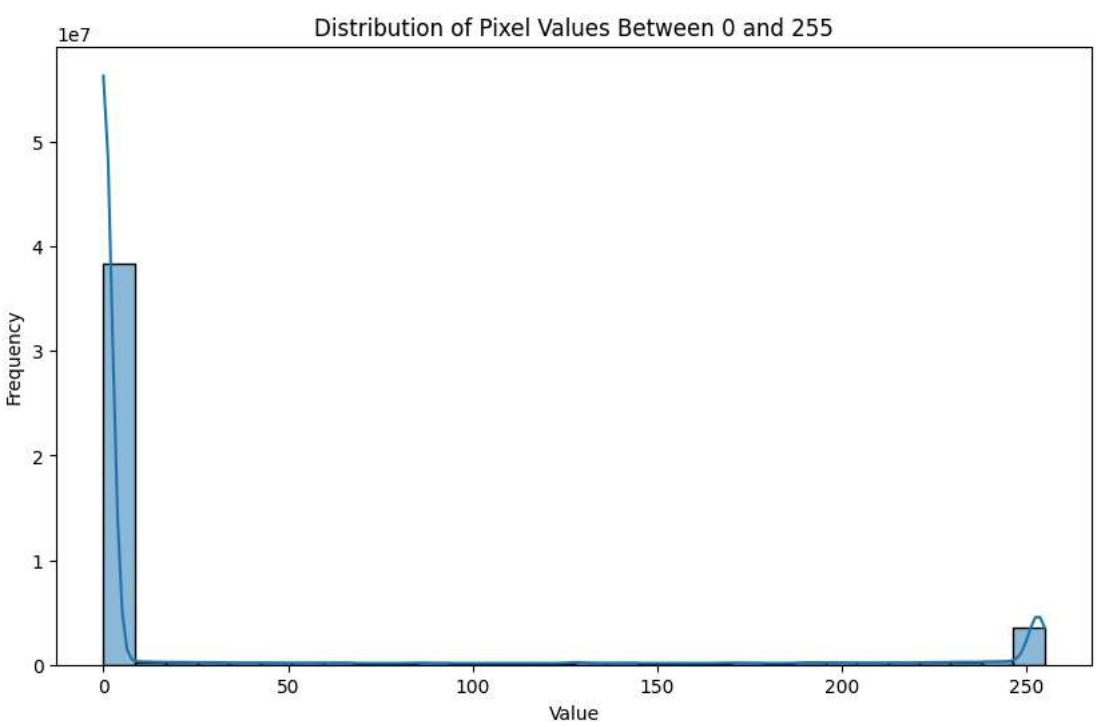
Frequency of Digits from 0 to 9

```
plt.figure(figsize = (10,6))
unique_values, counts = np.unique(y_train, return_counts=True)

# Create a bar plot
sns.barplot(x=unique_values, y=counts)

plt.xlabel("Digit")
plt.ylabel("Frequency")
plt.title("Frequency of Digits (0-9)")

plt.show()
```



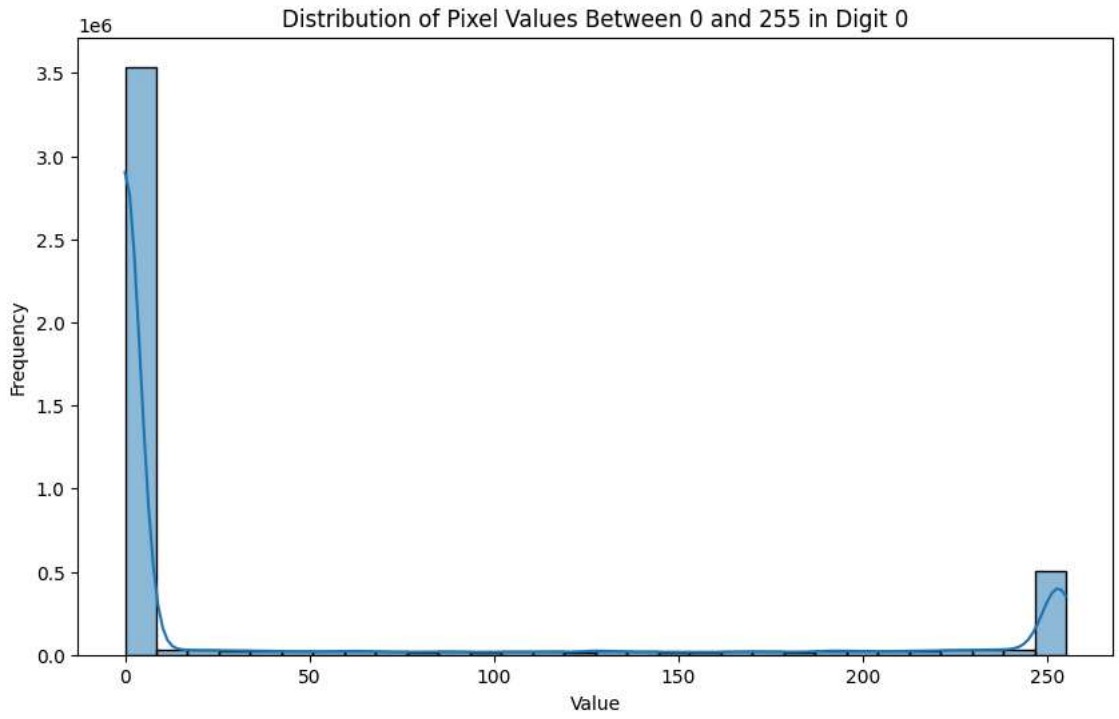
Distribution of Pixel Values Between 0 and 255

```
# Assuming X_train is a 2D array
X_train_flat = X_train.flatten()  # Flatten the 2D array into 1D

# Create the histogram
plt.figure(figsize=(10, 6))
sns.histplot(X_train_flat, bins=30, kde=True)
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.title("Distribution of Pixel Values Between 0 and 255")
plt.show()
```



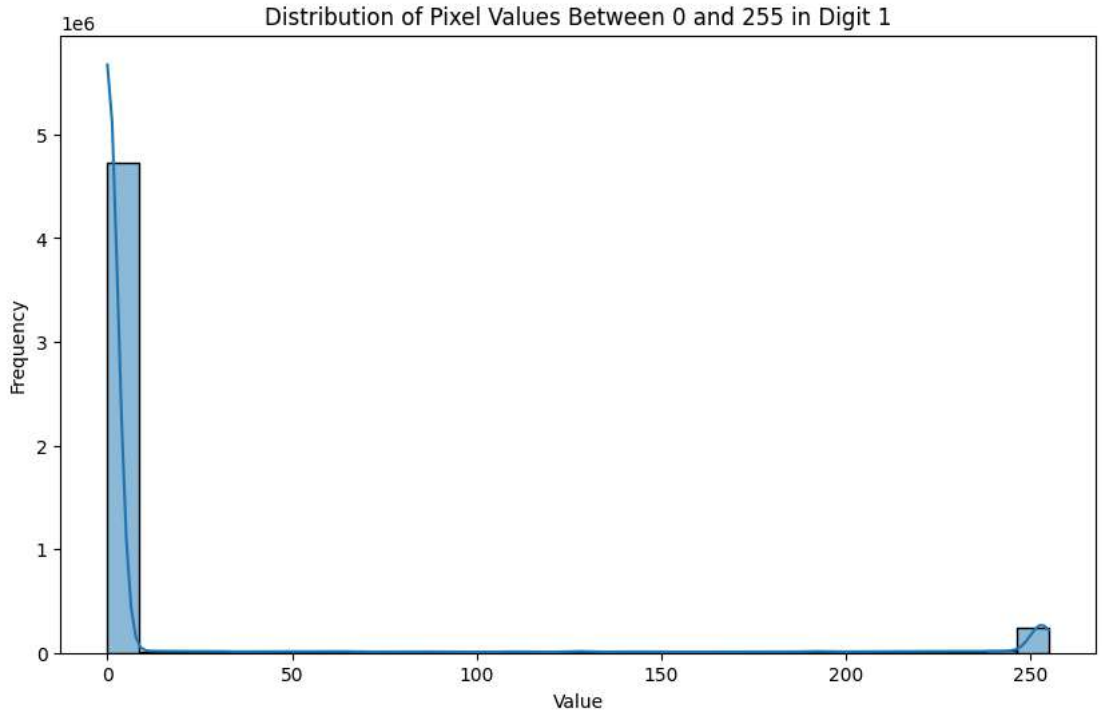
Distribution of Pixel Values Between 0 and 255 in Digit 0

```
X_train_0_flat = X_train_0.flatten()  # Flatten the 2D array into 1D

# Create the histogram
plt.figure(figsize=(10, 6))
sns.histplot(X_train_0_flat, bins=30, kde=True)
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.title("Distribution of Pixel Values Between 0 and 255 in Digit 0")
plt.show()
```



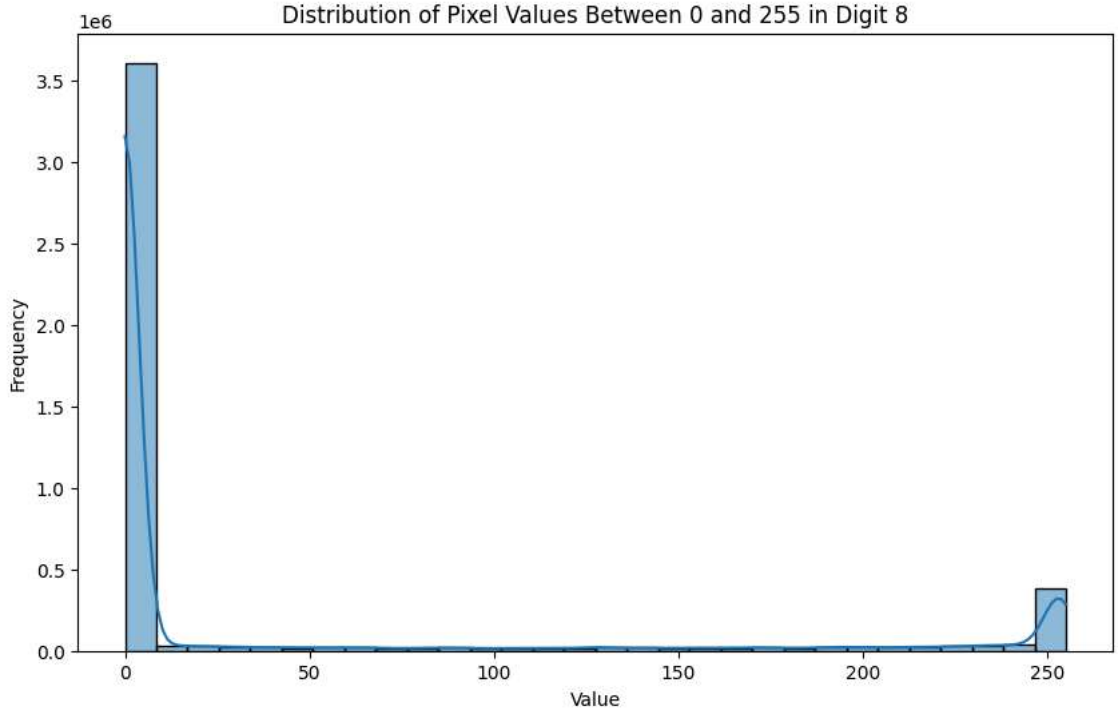
Distribution of Pixel Values Between 0 and 255 in Digit 1

```
X_train_1_flat = X_train_1.flatten()  # Flatten the 2D array into 1D

# Create the histogram
plt.figure(figsize=(10, 6))
sns.histplot(X_train_1_flat, bins=30, kde=True)
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.title("Distribution of Pixel Values Between 0 and 255 in Digit 1")
plt.show()
```



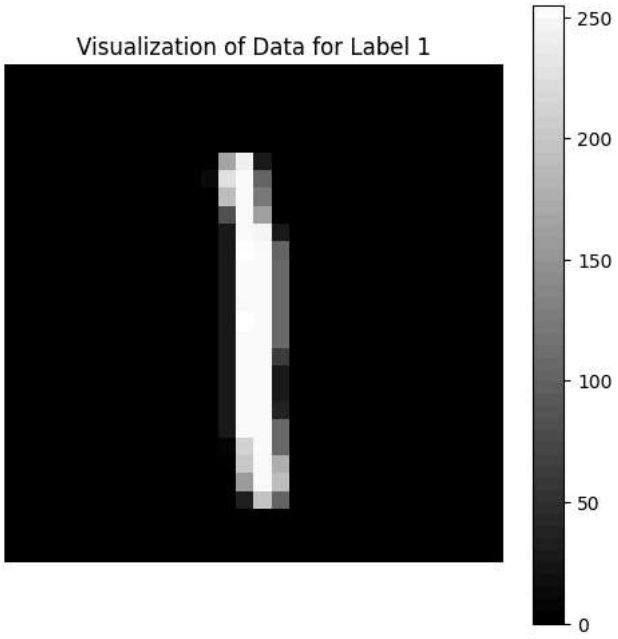
Distribution of Pixel Values Between 0 and 255 in Digit 8

```
X_train_8_flat = X_train_8.flatten()  # Flatten the 2D array into 1D

# Create the histogram
plt.figure(figsize=(10, 6))
sns.histplot(X_train_8_flat, bins=30, kde=True)
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.title("Distribution of Pixel Values Between 0 and 255 in Digit 8")
plt.show()
```



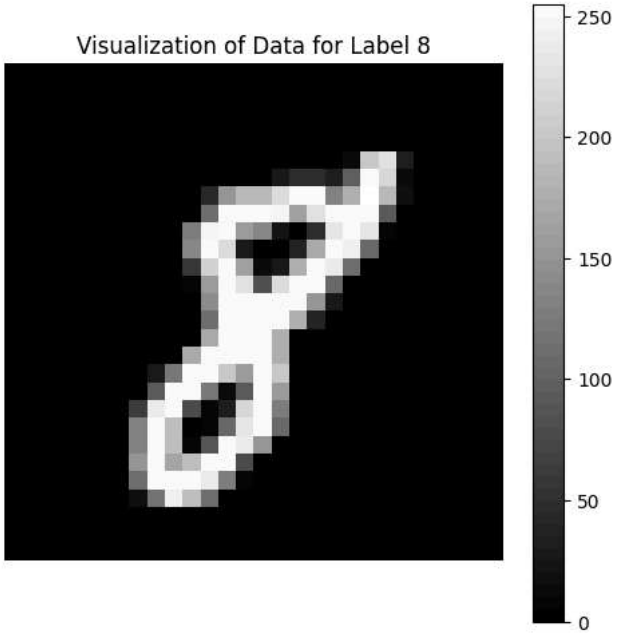
Visualization of Data for Label 1

```
example = X_train[14]
example_image = example.reshape(28, 28)

# Visualize the image
plt.figure(figsize=(6, 6))
plt.imshow(example_image, cmap='gray')
plt.colorbar()
plt.title("Visualization of Data for Label 1")
plt.axis('off')
plt.show()
```



Visualization of Data for Label 8

```
example = X_train[17]
example_image = example.reshape(28, 28)

# Visualize the image
plt.figure(figsize=(6, 6))
plt.imshow(example_image, cmap='gray')
plt.colorbar()
plt.title("Visualization of Data for Label 8")
plt.axis('off')
plt.show()
```



Naive Bayes Classifier

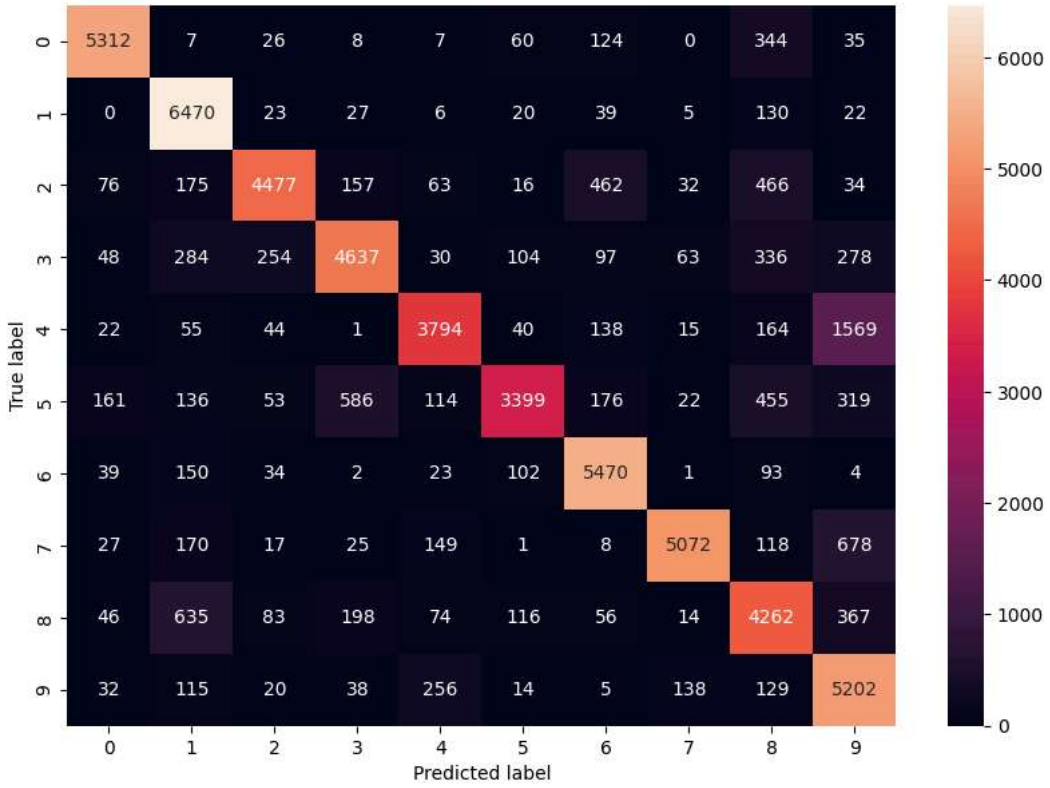
```
from scipy.stats import multivariate_normal as mvn


class GaussNB():
  def fit(self, X, y, epsilon = 1e-3):
    self.likelihoods = dict()
    self.priors = dict()

    self.K = set(y.astype(int))
    for k in self.K:
      X_k = X[y == k, :]
      self.likelihoods[k] = {"mean": X_k.mean(axis = 0), "cov": X_k.var(axis = 0)+epsilon}
      self.priors[k] = len(X_k)/len(X)
  def predict(self, X):
    N,D = X.shape
    P_hat = np.zeros((N, len(self.K)))

    for k, l in self.likelihoods.items():
      P_hat[:, k] = mvn.logpdf(X, l["mean"], l["cov"])+np.log(self.priors[k])

    return P_hat.argmax(axis = 1)


X_train_normalized = X_train/255


gnb = GaussNB()


gnb.fit(X_train_normalized,y_train,epsilon = 1e-2)



y_hat_train = gnb.predict(X_train_normalized)


def accuracy(y, y_hat):
  return np.mean(y == y_hat)


print('Naive Bayes Model Accuracy for Train data: ', round(accuracy(y_train, y_hat_train), 2)*100)
```

```
Naive Bayes Model Accuracy for Train data:  80.0
```

```
plt.figure(figsize=(10,7))
y_actu = pd.Series(y_train, name='Actual')
y_pred = pd.Series(y_hat_train, name='Predicted')
```

```
cm = pd.crosstab(y_actu, y_pred)
ax = sns.heatmap(cm, annot=True, fmt="d")
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 47.7222222222222, 'Predicted label')
```



Loading test data

```
test_data = pd.read_csv('/content/drive/MyDrive/ITC Projects/MNIST/MNIST_test.csv')
```

```
test_data = test_data.drop(columns = ['Unnamed: 0', 'index'], axis = 1)
```

```
X_test = test_data.to_numpy()
```

```
X_test
```

```
array([[7, 0, 0, ..., 0, 0, 0],
       [2, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       ...,
       [4, 0, 0, ..., 0, 0, 0],
       [5, 0, 0, ..., 0, 0, 0],
       [6, 0, 0, ..., 0, 0, 0]])
```

```
test_data.columns
```

```
Index(['labels', '0', '1', '2', '3', '4', '5', '6', '7', '8',
       ...
       '774', '775', '776', '777', '778', '779', '780', '781', '782', '783'],
      dtype='object', length=785)
```

```
y_test = X_test[:,0]
```

```
X_test = X_test[:,1:]
```

```
y_test
```

```
array([7, 2, 1, ..., 4, 5, 6])
```

```
X_test.shape
```

```
(10000, 784)
```

```
#Normalizing the X_test Data
X_test_Normalized = X_test/255
```

Testing using Naive Bayes

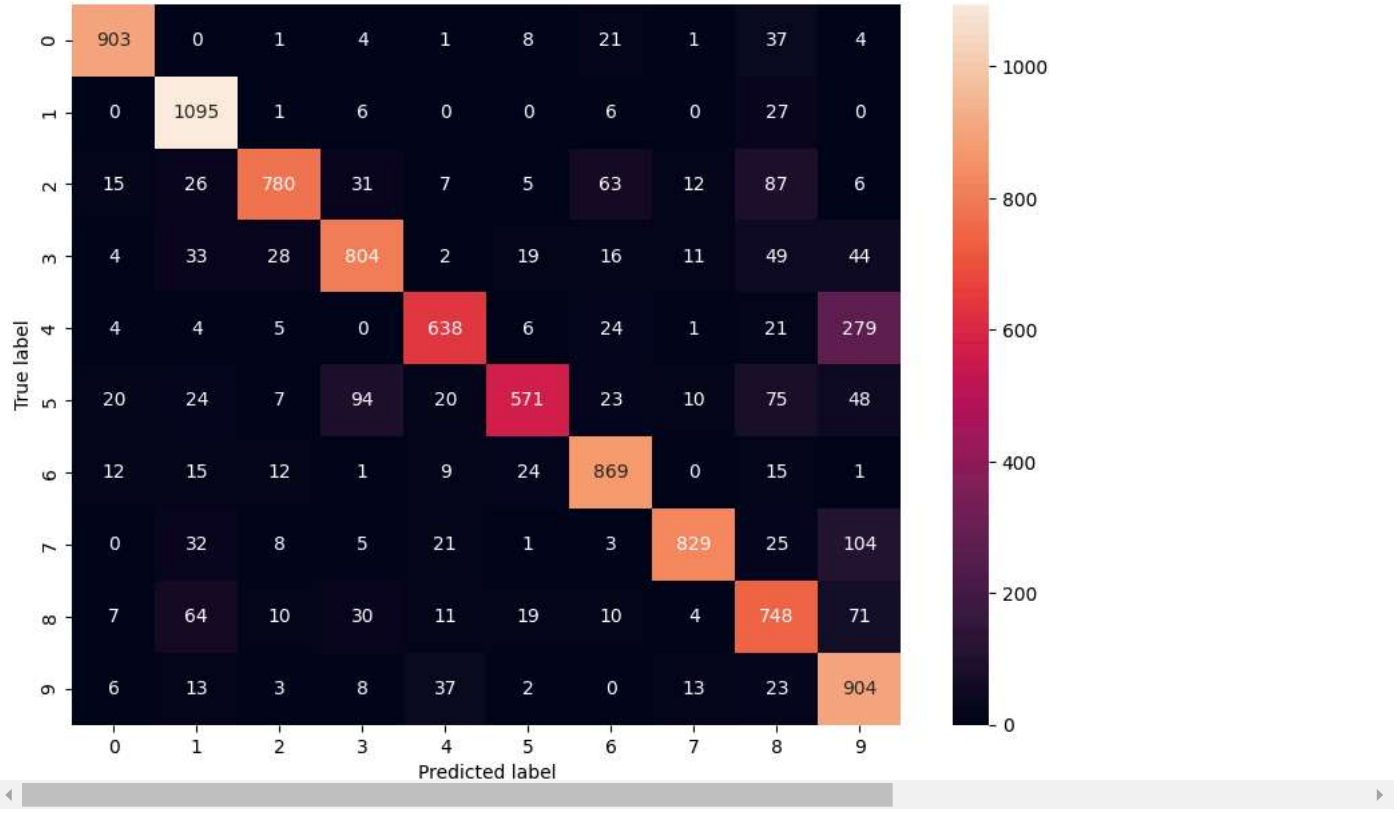
```
y_test_hat = gnb.predict(X_test_Normalized)
```

```
print('Naive Bayes Model Accuracy for Test data: ', round(accuracy(y_test, y_test_hat), 2)*100)
```

```
Naive Bayes Model Accuracy for Test data:  81.0
```

```
plt.figure(figsize=(10,7))
y_actu = pd.Series(y_test, name='Actual')
y_pred = pd.Series(y_test_hat, name='Predicted')
cm = pd.crosstab(y_actu, y_pred)
ax = sns.heatmap(cm, annot=True, fmt="d")
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 47.7222222222222, 'Predicted label')
```



Non-Naive Bayes

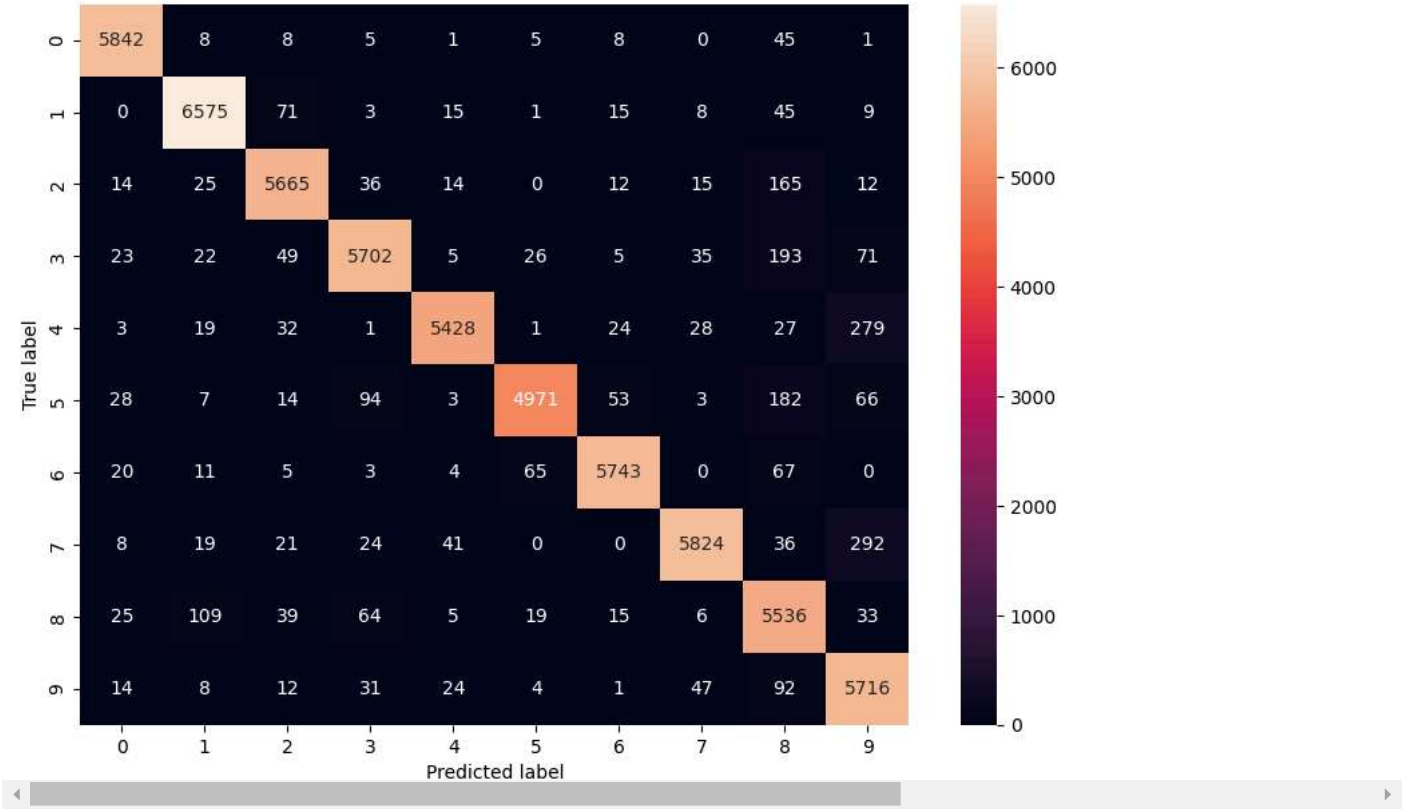
```
class GaussBayes():
  def fit(self, X, y, epsilon = 1e-3):
    self.likelihoods = dict()
    self.priors = dict()
    self.K = set(y.astype(int))

    for k in self.K:
      X_k = X[y == k, :]
      N_k,D = X_k.shape
      mu_k = X_k.mean(axis = 0)
```

```
        self.likelihoods[k] = {"mean": X_k.mean(axis = 0),
                               "cov": (1/(N_k-1))*np.matmul((X_k-mu_k).T, X_k-mu_k)+ epsilon*np.identity(D)}
        self.priors[k] = len(X_k)/len(X)
    def predict (self, X):
      N,D = X.shape
      P_hat = np.zeros((N,len(self.K)))

      for k, l in self.likelihoods. items():
        P_hat[:, k] = mvn.logpdf(X,l["mean"], l["cov"])+np.log(self.priors[k])


      return P_hat.argmax(axis = 1)


gbays_non_naive = GaussBayes()


gbays_non_naive.fit(X_train_normalized,y_train, epsilon = 5e-3)


y_hat_train_gbays = gbays_non_naive.predict(X_train_normalized)


print('Non_Naive Bayes Model Accuracy for Train data: ', round(accuracy(y_train, y_hat_train_gbays), 2)*100)
```

```
Non_Naive Bayes Model Accuracy for Train data:  95.0
```

```
plt.figure(figsize=(10,7))
y_actu = pd.Series(y_train, name='Actual')
y_pred = pd.Series(y_hat_train_gbays, name='Predicted')
cm = pd.crosstab(y_actu, y_pred)
ax = sns.heatmap(cm, annot=True, fmt="d")
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 47.7222222222222, 'Predicted label')
```



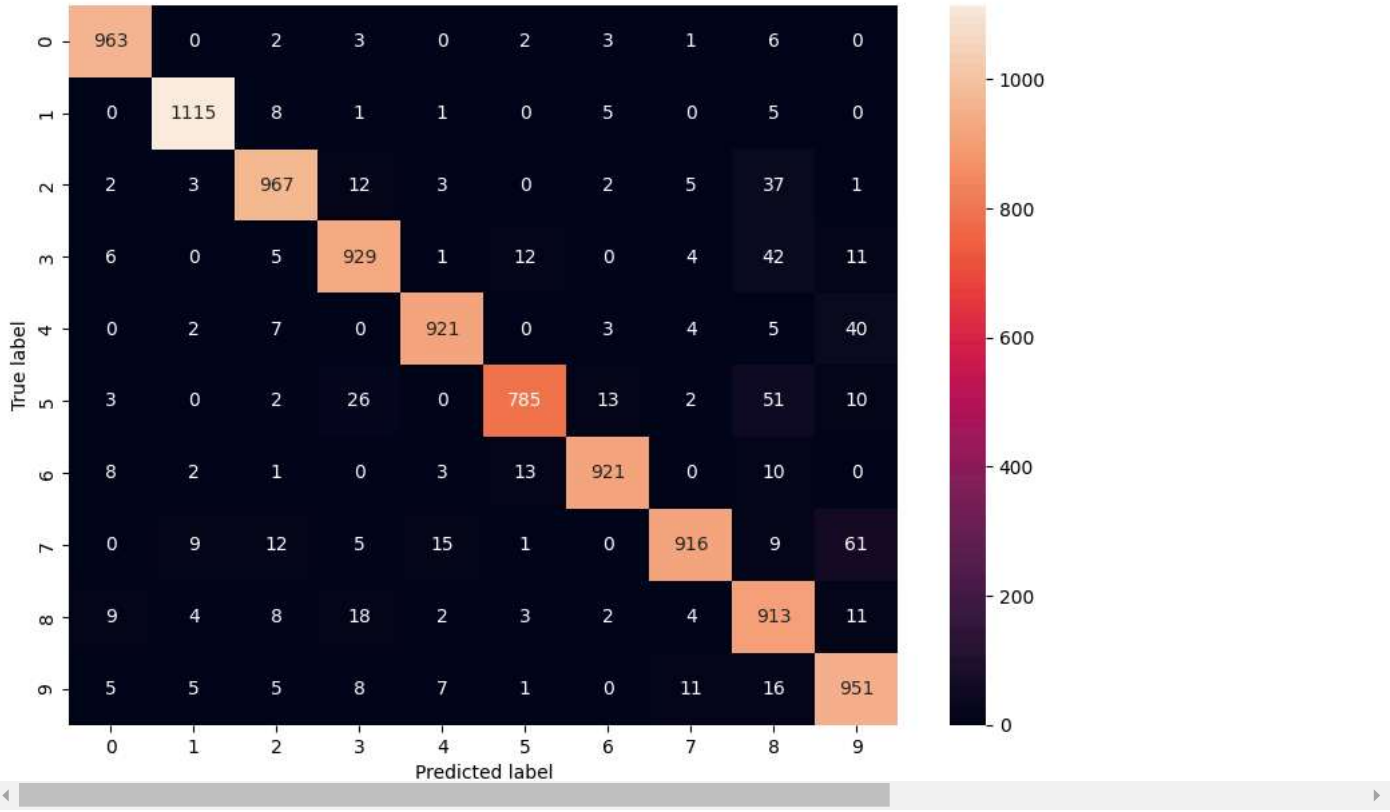
Testing using Non-Naive Bayes

```
y_hat_test_gbays = gbays_non_naive.predict(X_test_Normalized)


print('Non_Naive Bayes Model Accuracy for Test data: ', round(accuracy(y_test, y_hat_test_gbays), 2)*100)
```

```
Non_Naive Bayes Model Accuracy for Test data:  94.0
```

```
plt.figure(figsize=(10,7))
y_actu = pd.Series(y_test, name='Actual')
y_pred = pd.Series(y_hat_test_gbays, name='Predicted')
cm = pd.crosstab(y_actu, y_pred)
ax = sns.heatmap(cm, annot=True, fmt="d")
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 47.7222222222222, 'Predicted label')
```



K-Nearest Neighbours Classifiers

```
class KNNClassifier():
  def fit(self, X, y):
    self.X = X
```