```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

from sklearn import preprocessing
from sklearn.linear_model import  SGDClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
hinge_loss
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from scipy.stats import multivariate_normal as mvn
from sklearn.metrics import f1_score
```

```
mtrain = pd.read_csv("/content/MNIST_train.csv")
```

```
mtest = pd.read_csv("/content/MNIST_test.csv - MNIST_test.csv.csv")
```

```
mtrain.shape
```

```
(60000, 787)
```

```
print('null values = ', mtrain.isnull().sum().sum())
```

```
null values =  0
```

```
mtrain = mtrain.drop(columns=['Unnamed: 0', 'index'])
```

```
mtrain
```

```
       labels  0  1  2  3  4  5  6  7  8  ...  774  775  776  777  778  
779  \
0           5  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0  
0   
1           0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0  
0   
2           4  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0  
0   
3           1  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0  
0   
4           9  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0  
0   
...       ... .. .. .. .. .. .. .. .. ..  ...  ...  ...  ...  ...  ...  
...   
59995       8  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0  
0   
59996       3  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0  
0   
59997       5  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0  
0   
59998       6  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0  
0   
```

```
59999       8  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
0

       780  781  782  783
0        0    0    0    0
1        0    0    0    0
2        0    0    0    0
3        0    0    0    0
4        0    0    0    0
...    ...  ...  ...  ...
59995    0    0    0    0
59996    0    0    0    0
59997    0    0    0    0
59998    0    0    0    0
59999    0    0    0    0

[60000 rows x 785 columns]
```

```
print('null values = ', mtrain.isnull().sum().sum())
#mtrain = mtrain.dropna()
#print('null values = ', mtrain.isnull().sum().sum())
```

```
null values =  0
```

```
mtest = mtest.drop(columns=['Unnamed: 0', 'index'])
```

```
mtest
```

```
      labels  0  1  2  3  4  5  6  7  8  ...  774  775  776  777  778
779  \
0          7  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
0
1          2  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
0
2          1  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
0
3          0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
0
4          4  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
0
...      ... .. .. .. .. .. .. .. .. ..  ...  ...  ...  ...  ...  ...
...
9995       2  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
0
9996       3  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
0
9997       4  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
0
9998       5  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
0
9999       6  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
```

```
0
```

```
      780  781  782  783
0       0    0    0    0
1       0    0    0    0
2       0    0    0    0
3       0    0    0    0
4       0    0    0    0
...   ...  ...  ...  ...
9995    0    0    0    0
9996    0    0    0    0
9997    0    0    0    0
9998    0    0    0    0
9999    0    0    0    0

[10000 rows x 785 columns]
```

```
mtest.isnull().sum().sum()
```

```
0
```

```
print(mtrain.dtypes)
print(mtest.dtypes)
```

```
labels    int64
0         int64
1         int64
2         int64
3         int64
          ...
779       int64
780       int64
781       int64
782       int64
783       int64
Length: 785, dtype: object
labels    int64
0         int64
1         int64
2         int64
3         int64
          ...
779       int64
780       int64
781       int64
782       int64
783       int64
Length: 785, dtype: object
```

```
X = mtrain[mtrain.columns[1:]]
y = mtrain['labels']
```

```
X
```

```
       0  1  2  3  4  5  6  7  8  9  ...  774  775  776  777  778  779  
780  \
0      0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    
0   
1      0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    
0   
2      0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    
0   
3      0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    
0   
4      0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    
0   
...   .. .. .. .. .. .. .. .. .. ..  ...  ...  ...  ...  ...  ...  ...  
...   
59995  0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    
0   
59996  0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    
0   
59997  0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    
0   
59998  0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    
0   
59999  0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    
0   

       781  782  783  
0        0    0    0  
1        0    0    0  
2        0    0    0  
3        0    0    0  
4        0    0    0  
...    ...  ...  ...  
59995    0    0    0  
59996    0    0    0  
59997    0    0    0  
59998    0    0    0  
59999    0    0    0  

[60000 rows x 784 columns]
```

Non-Naive Gauss Bayes Classifier

```
X2 = X
y2 = y
```

```
type(X2)
```

```
pandas.core.frame.DataFrame
```

```
X2 = X2.values.reshape((-1, 28*28))
X2 = preprocessing.normalize(X2)
print(X2)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

```
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():
      #Bayes Theorem computation
      P_hat[:,k] = mvn.logpdf(X, l['mean'],l['cov']) +
np.log(self.priors[k])

    return P_hat.argmax(axis=1)
```

```
X2.shape
```

```
(60000, 784)
```

```
gbayes = GaussBayes()
```

```
gbayes.fit(X2, y2)
```

```
X_mtest = mtest[mtest.columns[1:]]
y_mtest = mtest['labels']
```

```
X_mtest1 = X_mtest.values.reshape((-1, 28*28))
X_mtest1 = preprocessing.normalize(X_mtest1)
```

```
y_hat2 = gbayes.predict(X_mtest1)
```

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

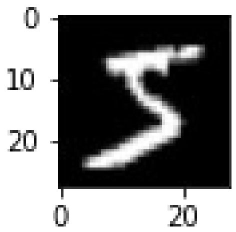
```
accuracy(y_mtest, y_hat2)
```

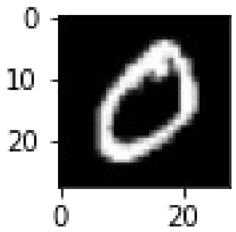
```
0.9576
```

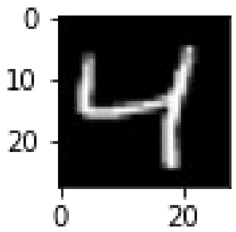
```
X0 = X.to_numpy()
X_mtest = X_mtest.to_numpy()
```

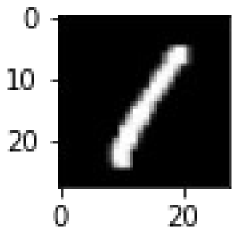
```
from keras.datasets import mnist
from matplotlib import pyplot

#loading
(train_X, train_y), (test_X, test_y) = mnist.load_data()

#shape of dataset
print('X_train: ' + str(train_X.shape))
print('Y_train: ' + str(train_y.shape))
print('X_test:  '  + str(test_X.shape))
print('Y_test:  '  + str(test_y.shape))

#plotting
from matplotlib import pyplot
for i in range(9):
  pyplot.subplot(330 + 1 + i)
  pyplot.imshow(train_X[i], cmap=pyplot.get_cmap('gray'))
  pyplot.show()
```

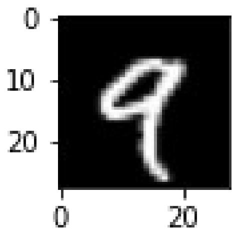
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
X_train: (60000, 28, 28)
Y_train: (60000,)
X_test:  (10000, 28, 28)
Y_test:  (10000,)
```

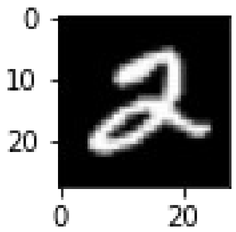

0
10
20
0
20

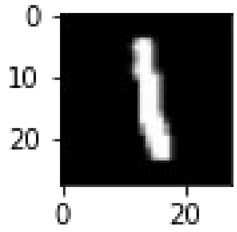

0
10
20
0
20

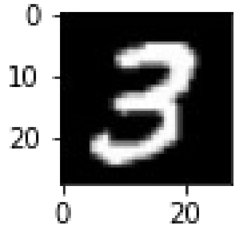

0
10
20
0
20

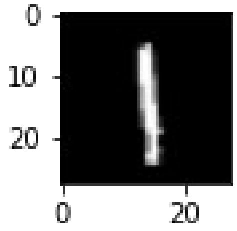

0
10
20
0
20


0
10
20
0
20


0
10
20
0
20







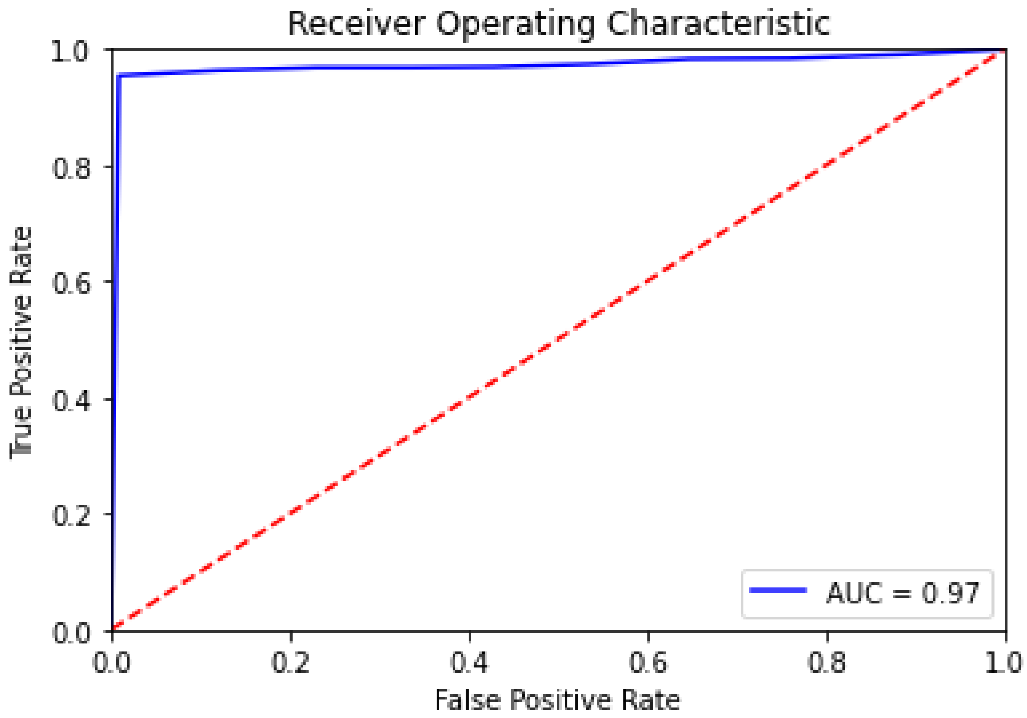
Model Evaluation

```
print("Accuracy : ",accuracy(y_mtest, y_hat2))
print("Precision Score : ", precision_score(y_mtest, y_hat2,
pos_label='positive', average='macro'))
print("Recall Score : ", recall_score(y_mtest, y_hat2,
pos_label='positive', average='macro'))
print(f"F1 Score :  {f1_score(y_mtest, y_hat2, average='macro')}")
```

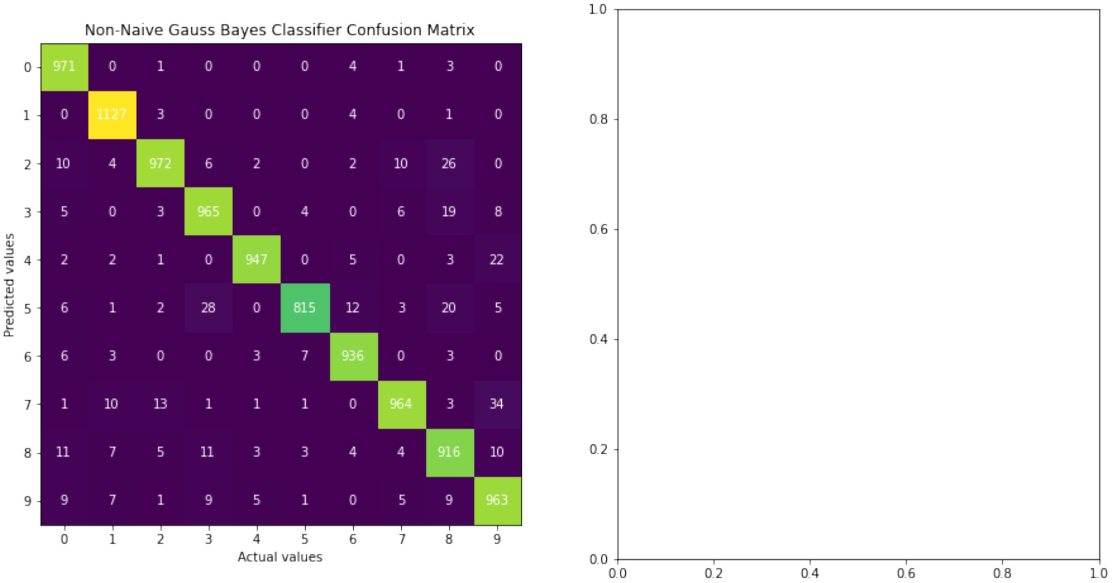
```
Accuracy :  0.9576
Precision Score :  0.9581210776051872
Recall Score :  0.9568750194116571
F1 Score :  0.9572150329042397
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/
_classification.py:1396: UserWarning: Note that pos_label (set to
'positive') is ignored when average != 'binary' (got 'macro'). You may
use labels=[pos_label] to specify a single positive class.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification
.py:1396: UserWarning: Note that pos_label (set to 'positive') is
ignored when average != 'binary' (got 'macro'). You may use
labels=[pos_label] to specify a single positive class.
  warnings.warn(
# Importing all necessary libraries
from sklearn.metrics import roc_curve, auc

#class_probabilities = classifier.predict_proba(X_mtest)
preds = y_hat2

fpr, tpr, threshold = roc_curve(y_mtest, preds, pos_label=9)
roc_auc = auc(fpr, tpr)

# Printing AUC
print(f"AUC for our classifier is: {roc_auc}")

# Plotting the ROC
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
AUC for our classifier is: 0.9714988636913535
```



```
def plot_confusion_matrix(fig, ax, conf_matrix, title, xlabel, ylabel,
n_image=0):
    im = ax[n_image].imshow(conf_matrix)

    ax[n_image].set_xticks(np.arange(10))
    ax[n_image].set_yticks(np.arange(10))

    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            text = ax[n_image].text(j, i, conf_matrix[i, j],
                           ha="center", va="center", color="w")

    ax[n_image].set_xlabel(xlabel)
    ax[n_image].set_ylabel(ylabel)
    ax[n_image].set_title(title)

testing_MNIST_targets = y_mtest
fed_prediction = y_hat2

fed_conf_matrix = confusion_matrix(testing_MNIST_targets,
fed_prediction)
#local_conf_matrix = confusion_matrix(testing_MNIST_targets,
local_prediction)


fig, axs = plt.subplots(nrows=1, ncols=2,figsize=(15,8))



plot_confusion_matrix(fig, axs, fed_conf_matrix,
                      "Non-Naive Gauss Bayes Classifier Confusion
Matrix",
                      "Actual values", "Predicted values", n_image=0)

# plot_confusion_matrix(fig, axs, local_conf_matrix,
#                        "Local Perceptron Confusion Matrix",
#                        "Actual values", "Predicted values",
n_image=1)
```



```
# plt.figure(figsize=(12,8))
# plt.scatter(X0[:,0], X0[:,1], c=y2, alpha=0.5, s=10)
# plt.scatter(X_mtest[:,0], X_mtest[:,1], c=y_hat2, alpha=0.5, s=10)
# #plt.axis([X2.min(), X2.max(), y2.min(), y2.max()])
```

KNN

```
class KNNClassifier():

  def fit(self, X, y):
    self.X = X
    self.y = y


  def predict(self, X, K, epsilon = 1e-3):
    N = len(X)
    y_hat = np.zeros(N)

    for i in range(N):
      dist2 = np.sum((self.X-X[i])**2,axis=1)
      idxt  = np.argsort(dist2)[:K]
      gamma_k=1/(np.sqrt(dist2[idxt]+epsilon))
      y_hat[i] = np.bincount(self.y[idxt], weights=gamma_k).argmax()


    return y_hat
```

```
np.bincount([0,1,1,2], weights = [(1/2),(1/7),(1/6),(1/9)])
```

```
array([0.5       , 0.30952381, 0.11111111])
```

```
knn = KNNClassifier()
```

```
print(X.shape)
print(y.shape)
```

```
(60000, 784)
(60000,)
```

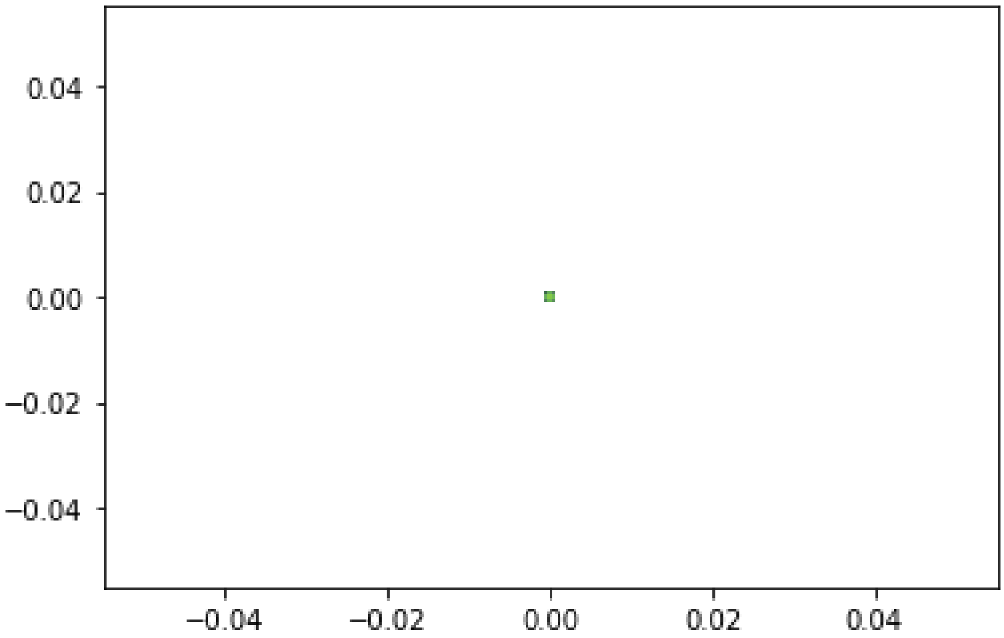
```
X3 = X2
y3 = y2
```

```
knn.fit(X3,y3)
```

```
y_hat3 = knn.predict(X3,10) #k=10
```

```
plt.figure()
plt.scatter(X3[:,0], X3[:,1], c=y_hat3, alpha= 0.5, s= 6)
```

```
<matplotlib.collections.PathCollection at 0x7ff11140aa90>
```



```
accuracy(y3,y_hat3)
```

```
1.0
```

```
X_vis3 = np.random.uniform(-6,6,(int(N*10),D))
```

```
------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-159-6b51110a3a4e> in <module>
----> 1 X_vis3 = np.random.uniform(-6,6,(int(N*10),D))

NameError: name 'N' is not defined
```

```
y_hat_vis3.shape
```

```
y_hat_vis3 =knn.predict(X_vis3,100)
plt.figure()
plt.scatter(X_vis3[:,0],X_vis3[:,1], c=y_hat_vis3, alpha =0.5, s=6)
```