MM20B007 Tutorial 7

Importing necessary packages

```
import time
import random
import sklearn
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.metrics import accuracy_score, f1_score
from sklearn.naive_bayes import GaussianNB, ComplementNB
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import MinMaxScaler

# set random seeds
random.seed = 42
np.random.seed = 42
sns.set_style('darkgrid')
```

Dataset Generation

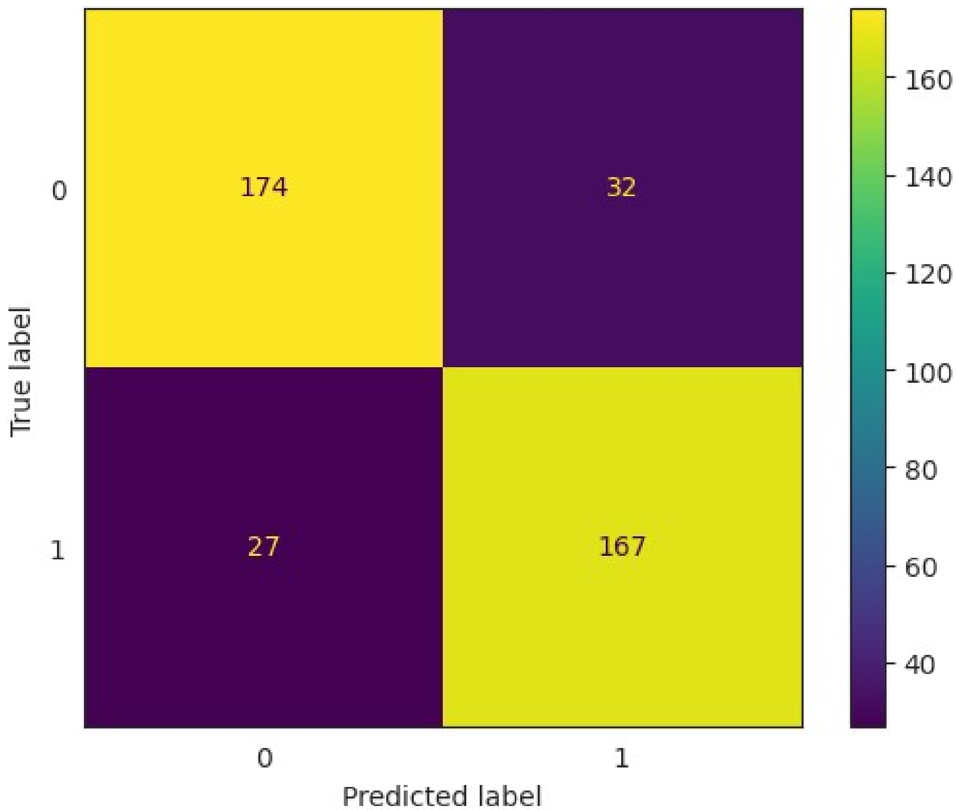
```
### DO NOT EDIT ###
def data_gen(n_data, n_class, weights = None, f_tot = 6, f_info = 6,
f_red = 0, class_sep = 1, name = 'data1_nb'):
    X,Y = make_classification(n_samples = n_data,
                              n_features = f_tot,
                              n_classes = n_class,
                              n_informative = f_info,
                              n_redundant = f_red,
                              n_clusters_per_class = 1,
                              weights = weights,
                              hypercube = True,
                              random_state = 42,
                              class_sep= class_sep)
    return X, Y
```

Refer to the tutorial notebook for details. Create datasets with number of features - 1000, 2000, 3000, 4000 and 5000. For this question use the same dataset generation parameters as Dataset 1 in notebook. Change only the number of features (total and informative) per datapoint. Use time module to calculate runtime of Naive Bayes for each dataset created above and plot results

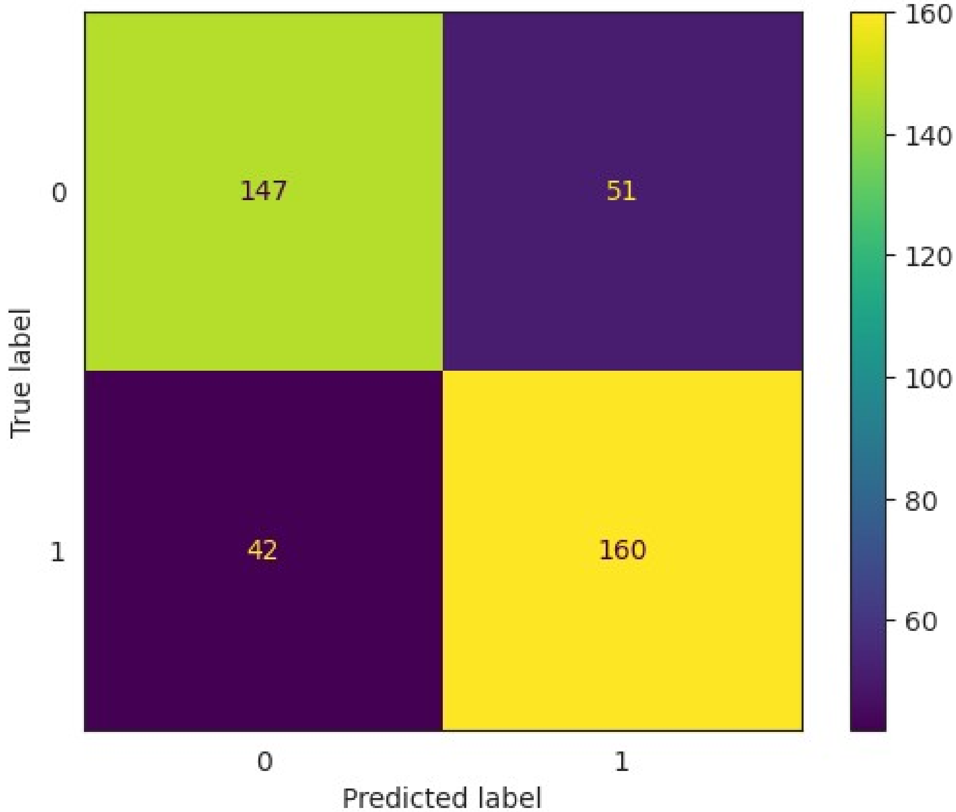
```python
### DO NOT EDIT ###
# Creating the datasets
n_samples = 2000
n_classes = 2

# DATASET 1 - 1000 features
X1, Y1 = data_gen(n_data = n_samples,
                  n_class = n_classes,
                  weights = None,
                  f_tot = 1000,
                  f_info = 1000,
                  name = 'data1_nb')

# DATASET 2 - 2000 features
X2, Y2 = data_gen(n_data = n_samples,
                  n_class = n_classes,
                  weights = None,
                  f_tot = 2000,
                  f_info = 2000,
                  name = 'data2_nb')

# DATASET 3 - 3000 features
X3, Y3 = data_gen(n_data = n_samples,
                  n_class = n_classes,
                  weights = None,
                  f_tot = 3000,
                  f_info = 3000,
                  name = 'data3_nb')

# DATASET 4 - 4000 features
X4, Y4 = data_gen(n_data = n_samples,
                  n_class = n_classes,
                  weights = None,
                  f_tot = 4000,
                  f_info = 4000,
                  name = 'data4_nb')

# DATASET 5 - 5000 features
X5, Y5 = data_gen(n_data = n_samples,
                  n_class = n_classes,
                  weights = None,
                  f_tot = 5000,
                  f_info = 5000,
                  name = 'data5_nb')
```

```python
### DO NOT EDIT ###
# splitting each dataset into test and train sets
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1,
test_size=0.2, random_state=42, shuffle=True)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2, Y2,
test_size=0.2, random_state=42, shuffle=True)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(X3, Y3,
test_size=0.2, random_state=42, shuffle=True)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(X4, Y4,
test_size=0.2, random_state=42, shuffle=True)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(X5, Y5,
test_size=0.2, random_state=42, shuffle=True)

### DO NOT EDIT ###
def NBClassifier(X_train, Y_train, X_test, Y_test, d, conf = True):
    tick = time.time()
    model = GaussianNB()
    model.fit(X_train, Y_train)
    tock = time.time()

    Y_pred = model.predict(X_test)
    acc = accuracy_score(Y_test, Y_pred)
    f1 = f1_score(Y_test, Y_pred)
    cm = confusion_matrix(Y_test, Y_pred)

    print('**************************************')
    print('Dataset:', d)
    print(f"Accuracy: {acc:0,.4f}")
    print(f"F1 score: {f1:0,.4f}")

    sns.set_style("white")
    if conf:
        disp = ConfusionMatrixDisplay(confusion_matrix=cm)
        disp.plot()
        plt.show()

    return tock-tick

times = []

time1 = NBClassifier(X1_train, Y1_train, X1_test, Y1_test, 1)
print('Time Taken (seconds):', time1)
times.append(time1)
```

```
**************************************
Dataset: 1
Accuracy: 0.8525
F1 score: 0.8499
```



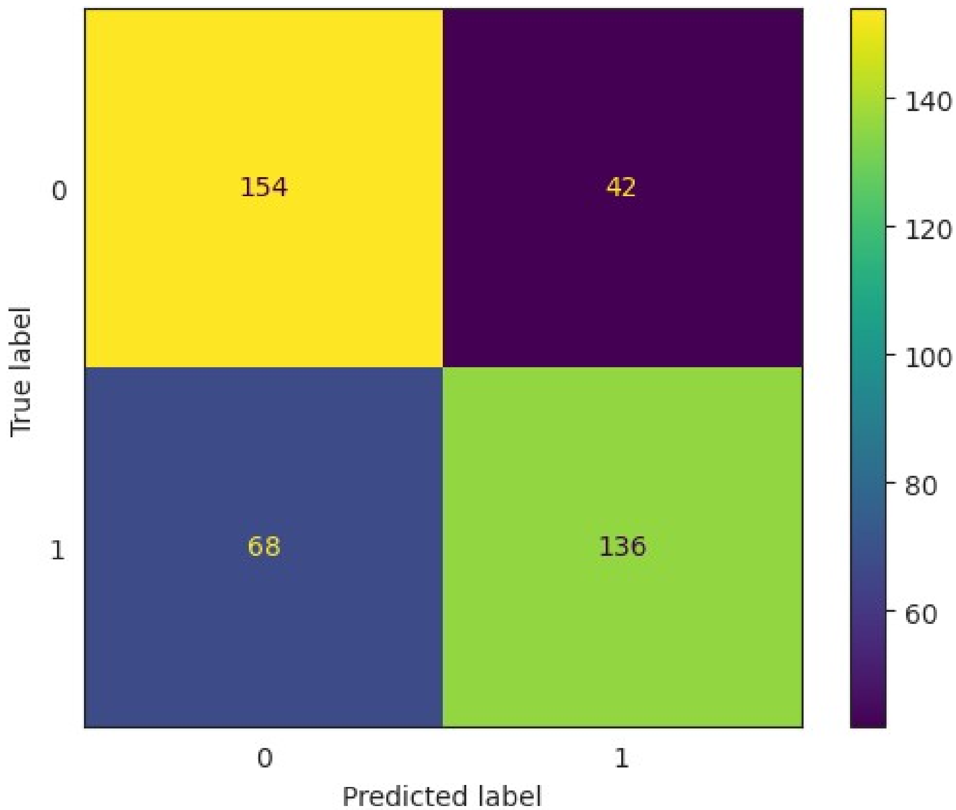

```
Time Taken (seconds): 0.01887035369873047

time2 = NBClassifier(X2_train, Y2_train, X2_test, Y2_test, 2)
print('Time Taken (seconds):', time2)
times.append(time2)

**************************************
Dataset: 2
Accuracy: 0.7675
F1 score: 0.7748
```



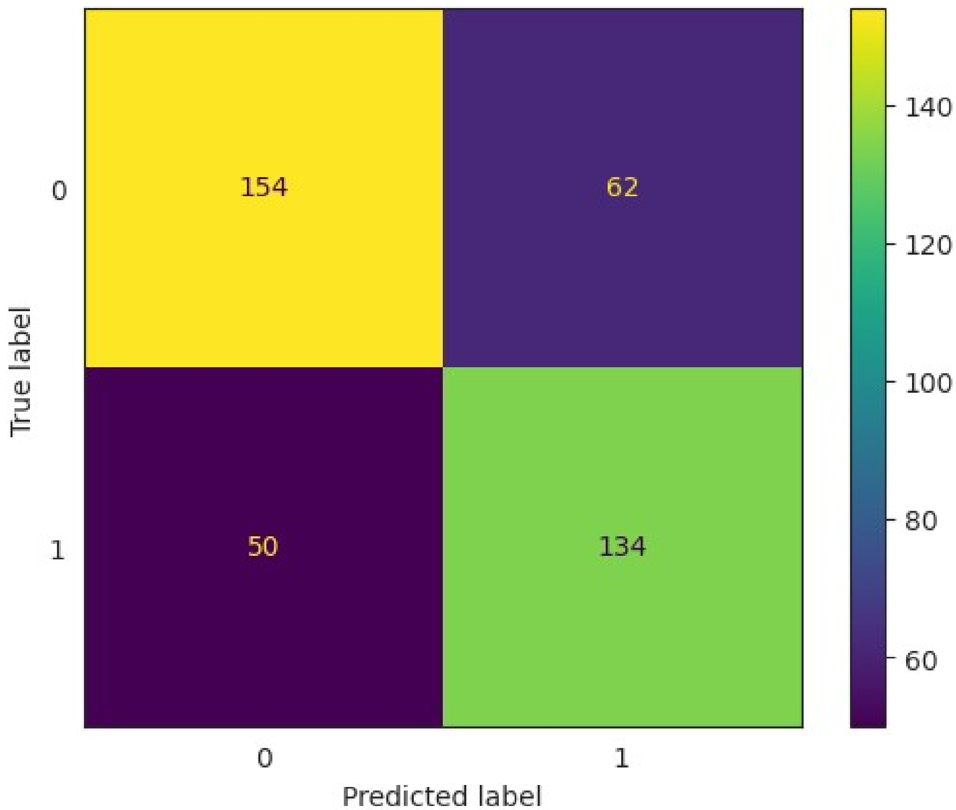

```
Time Taken (seconds): 0.03926229476928711
```

```
time3 = NBClassifier(X3_train, Y3_train, X3_test, Y3_test, 3)
print('Time Taken (seconds):', time3)
times.append(time3)
```

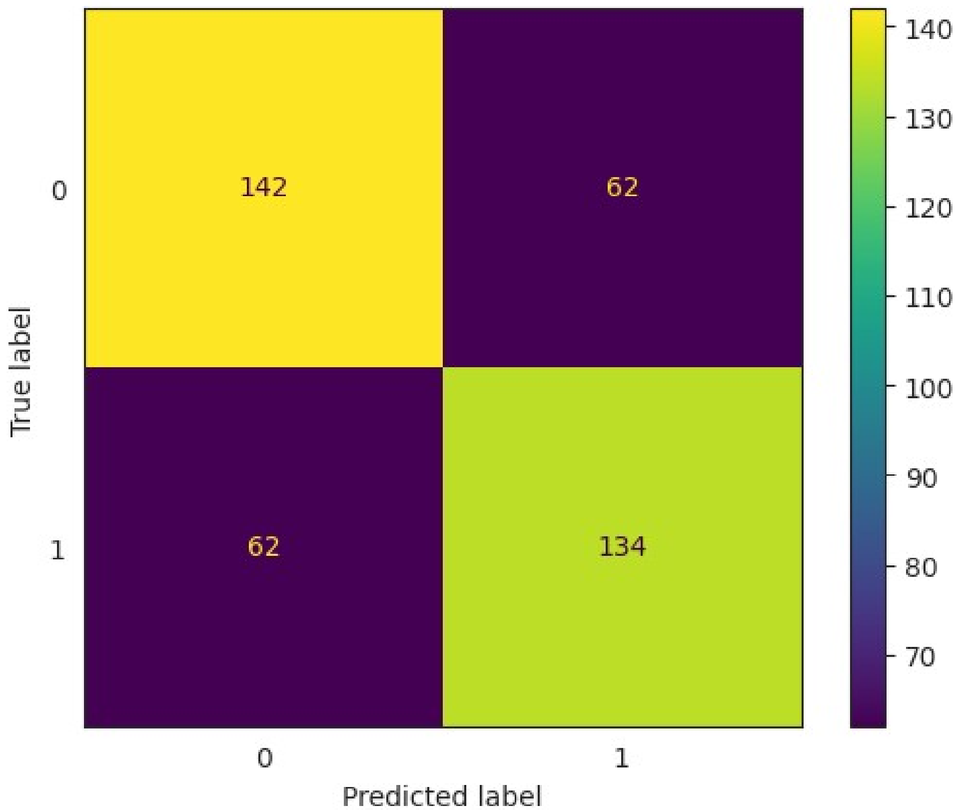
```
**************************************
Dataset: 3
Accuracy: 0.7250
F1 score: 0.7120
```



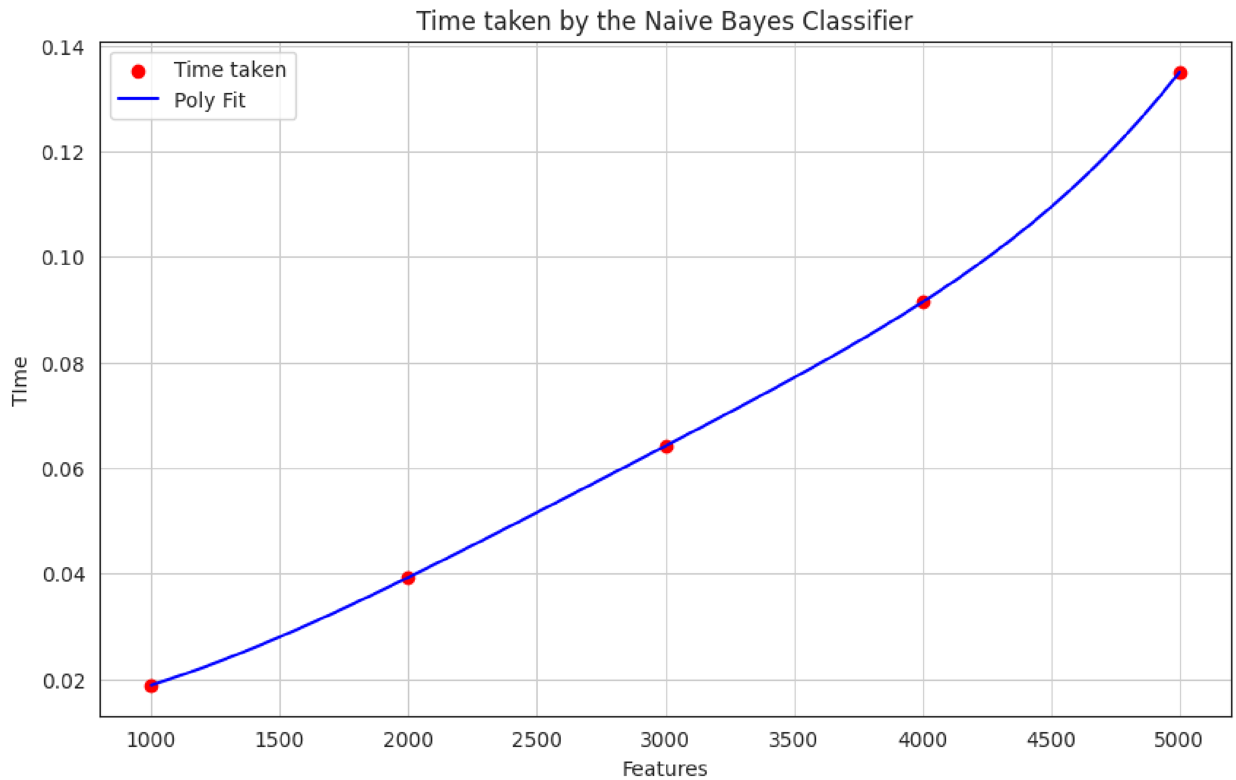

```
Time Taken (seconds): 0.06415295600891113

time4 = NBClassifier(X4_train, Y4_train, X4_test, Y4_test, 4)
print('Time Taken (seconds):', time4)
times.append(time4)

**************************************
Dataset: 4
Accuracy: 0.7200
F1 score: 0.7053
```




```
Time Taken (seconds): 0.09140825271606445

time5 = NBClassifier(X5_train, Y5_train, X5_test, Y5_test, 5)
print('Time Taken (seconds):', time5)
times.append(time5)

************************************
Dataset: 5
Accuracy: 0.6900
F1 score: 0.6837
```




```
Time Taken (seconds): 0.13501858711242676

features = [1000, 2000, 3000, 4000, 5000]

coeff = np.polyfit(features, times, 5)
poly = np.poly1d(coeff)

feature_gen = np.linspace(min(features), max(features), 100)
time_gen = poly(feature_gen)

plt.figure(figsize=(10, 6))
plt.scatter(features, times, label='Time taken', color='red',
marker='o')
plt.plot(feature_gen, time_gen, color='blue', label='Poly Fit')
plt.xlabel('Features')
plt.ylabel('TIme')
plt.title('Time taken by the Naive Bayes Classifier')
plt.legend()
plt.grid(True)
plt.show()

/usr/local/lib/python3.10/dist-packages/IPython/core/
interactiveshell.py:3553: RankWarning: Polyfit may be poorly
conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
```



For the datasets with class imbalance used in the tutorial today (Dataset 3 and Dataset 4), try improving the performance in the specificity in the Gaussian naive bayes model. [Hint: Set the priors beforehand]

```
# DATASET 6 - 20:80 class imbalanced
X6, Y6 = data_gen(n_data = n_samples,
                  n_class = n_classes,
                  weights = [0.2, 0.8],
                  f_tot = 1000,
                  f_info = 1000,
                  name = 'data6_nb')

# DATASET 7 - 10:90 class imbalanced
X7, Y7 = data_gen(n_data = n_samples,
                  n_class = n_classes,
                  weights = [0.1, 0.9],
                  f_tot = 1000,
                  f_info = 1000,
                  name = 'data7_nb')
```

```
# splitting each dataset into test and train sets
X6_train, X6_test, Y6_train, Y6_test = train_test_split(X6, Y6,
test_size=0.2, random_state=42, shuffle=True)
X7_train, X7_test, Y7_train, Y7_test = train_test_split(X7, Y7,
test_size=0.2, random_state=42, shuffle=True)

def NBClassifier_2(X_train, Y_train, X_test, Y_test, prior, d, conf =
True):
    model = GaussianNB(priors = prior)
    model.fit(X_train, Y_train)

    Y_pred = model.predict(X_test)
    acc = accuracy_score(Y_test, Y_pred)
    f1 = f1_score(Y_test, Y_pred)
    cm = confusion_matrix(Y_test, Y_pred)
    tn, fp, fn, tp = cm.ravel()
    specificity = tn / (tn+fp)

    print('**************************************')
    print('Dataset:', d)
    print(f"Accuracy: {acc:0,.4f}")
    print(f"Specificity: {specificity:0,.4f}")

    sns.set_style("white")
    if conf:
        disp = ConfusionMatrixDisplay(confusion_matrix=cm)
        disp.plot()
        plt.show()
    return [acc, f1, specificity]

time6 = NBClassifier_2(X6_train, Y6_train, X6_test, Y6_test, None, 6,
conf = False)
time7 = NBClassifier_2(X7_train, Y7_train, X7_test, Y7_test, None, 7,
conf = False)

**************************************
Dataset: 6
Accuracy: 0.8675
Specificity: 0.4521
**************************************
Dataset: 7
Accuracy: 0.9125
Specificity: 0.2093
```

Now tunning the prior values

```
priors = []
for i in range(0, 105, 5):
  random_priors = [round(i / 100, 2), round(1 - i / 100, 2)]  #
Generates two random values
```

```
  # random_priors /= random_priors.sum()  # Normalize to ensure the sum is 1
  priors.append(list(random_priors))
priors
```

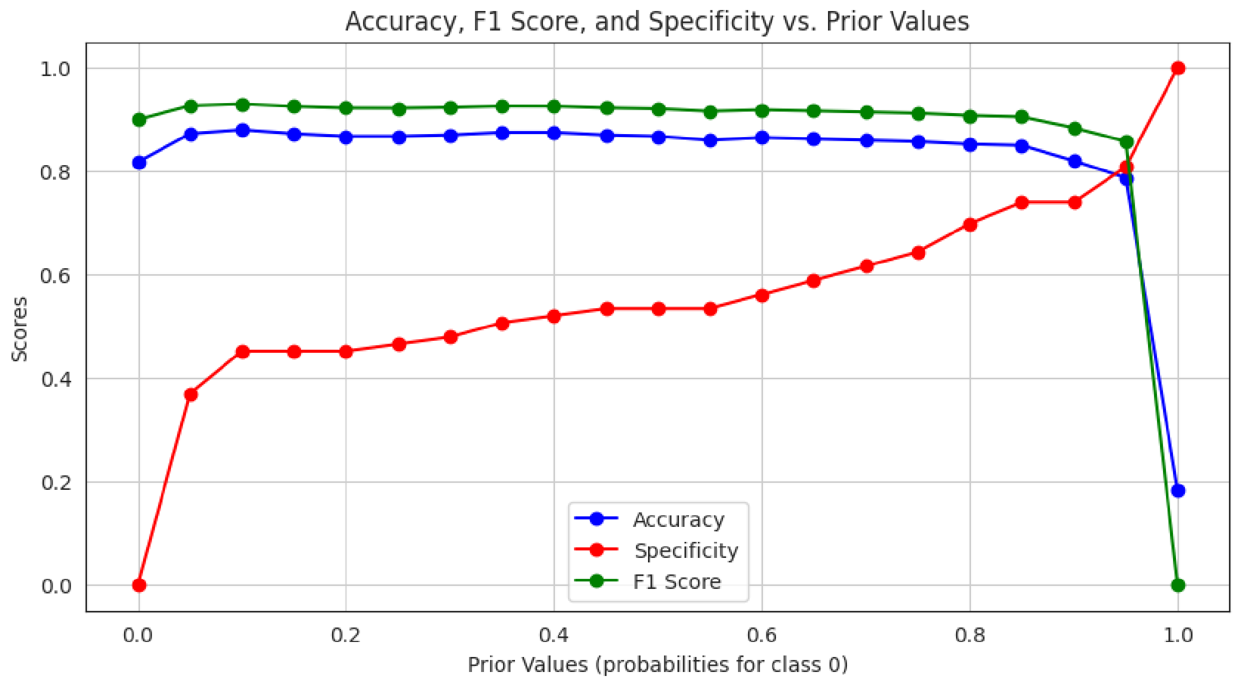
```
[[0.0, 1.0],
 [0.05, 0.95],
 [0.1, 0.9],
 [0.15, 0.85],
 [0.2, 0.8],
 [0.25, 0.75],
 [0.3, 0.7],
 [0.35, 0.65],
 [0.4, 0.6],
 [0.45, 0.55],
 [0.5, 0.5],
 [0.55, 0.45],
 [0.6, 0.4],
 [0.65, 0.35],
 [0.7, 0.3],
 [0.75, 0.25],
 [0.8, 0.2],
 [0.85, 0.15],
 [0.9, 0.1],
 [0.95, 0.05],
 [1.0, 0.0]]
```

```
accuracy_d6 = []
f1_d6 = []
specifities_d6 = []
for items in priors:
  time6 = NBClassifier_2(X6_train, Y6_train, X6_test, Y6_test, items, 6, conf = False)
  accuracy_d6.append(time6[0])
  f1_d6.append(time6[1])
  specifities_d6.append(time6[2])
```

```
**************************************
Dataset: 6
Accuracy: 0.8175
Specificity: 0.0000
**************************************
Dataset: 6
Accuracy: 0.8725
Specificity: 0.3699
**************************************
Dataset: 6
Accuracy: 0.8800
Specificity: 0.4521
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/naive_bayes.py:514:
RuntimeWarning: divide by zero encountered in log
  jointi = np.log(self.class_prior_[i])

**************************************
Dataset: 6
Accuracy: 0.8725
Specificity: 0.4521
**************************************
Dataset: 6
Accuracy: 0.8675
Specificity: 0.4521
**************************************
Dataset: 6
Accuracy: 0.8675
Specificity: 0.4658
**************************************
Dataset: 6
Accuracy: 0.8700
Specificity: 0.4795
**************************************
Dataset: 6
Accuracy: 0.8750
Specificity: 0.5068
**************************************
Dataset: 6
Accuracy: 0.8750
Specificity: 0.5205
**************************************
Dataset: 6
Accuracy: 0.8700
Specificity: 0.5342
**************************************
Dataset: 6
Accuracy: 0.8675
Specificity: 0.5342
**************************************
Dataset: 6
Accuracy: 0.8600
Specificity: 0.5342
**************************************
Dataset: 6
Accuracy: 0.8650
Specificity: 0.5616
**************************************
Dataset: 6
Accuracy: 0.8625
Specificity: 0.5890
**************************************
Dataset: 6
```

```
Accuracy: 0.8600
Specificity: 0.6164
**************************************
Dataset: 6
Accuracy: 0.8575
Specificity: 0.6438
**************************************
Dataset: 6
Accuracy: 0.8525
Specificity: 0.6986
**************************************
Dataset: 6
Accuracy: 0.8500
Specificity: 0.7397
**************************************
Dataset: 6
Accuracy: 0.8200
Specificity: 0.7397
**************************************
Dataset: 6
Accuracy: 0.7875
Specificity: 0.8082
**************************************
Dataset: 6
Accuracy: 0.1825
Specificity: 1.0000

/usr/local/lib/python3.10/dist-packages/sklearn/naive_bayes.py:514:
RuntimeWarning: divide by zero encountered in log
  jointi = np.log(self.class_prior_[i])

# Extract the prior values to use as x-coordinates for the plot
prior_x = [prior[0] for prior in priors]

# Create the plot
plt.figure(figsize=(10, 5))
plt.plot(prior_x, accuracy_d6, marker='o', linestyle='-', color='b',
label='Accuracy')
plt.plot(prior_x, specifities_d6, marker='o', linestyle='-',
color='r', label='Specificity')
plt.plot(prior_x, f1_d6, marker='o', linestyle='-', color='g',
label='F1 Score')
plt.title("Accuracy, F1 Score, and Specificity vs. Prior Values")
plt.xlabel("Prior Values (probabilities for class 0)")
plt.ylabel("Scores")
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```




```
accuracy_d7 = []
f1_d7 = []
specifities_d7 = []
for items in priors:
  time7 = NBClassifier_2(X7_train, Y7_train, X7_test, Y7_test, items,
7, conf = False)
  accuracy_d7.append(time7[0])
  f1_d7.append(time7[1])
  specifities_d7.append(time7[2])
```

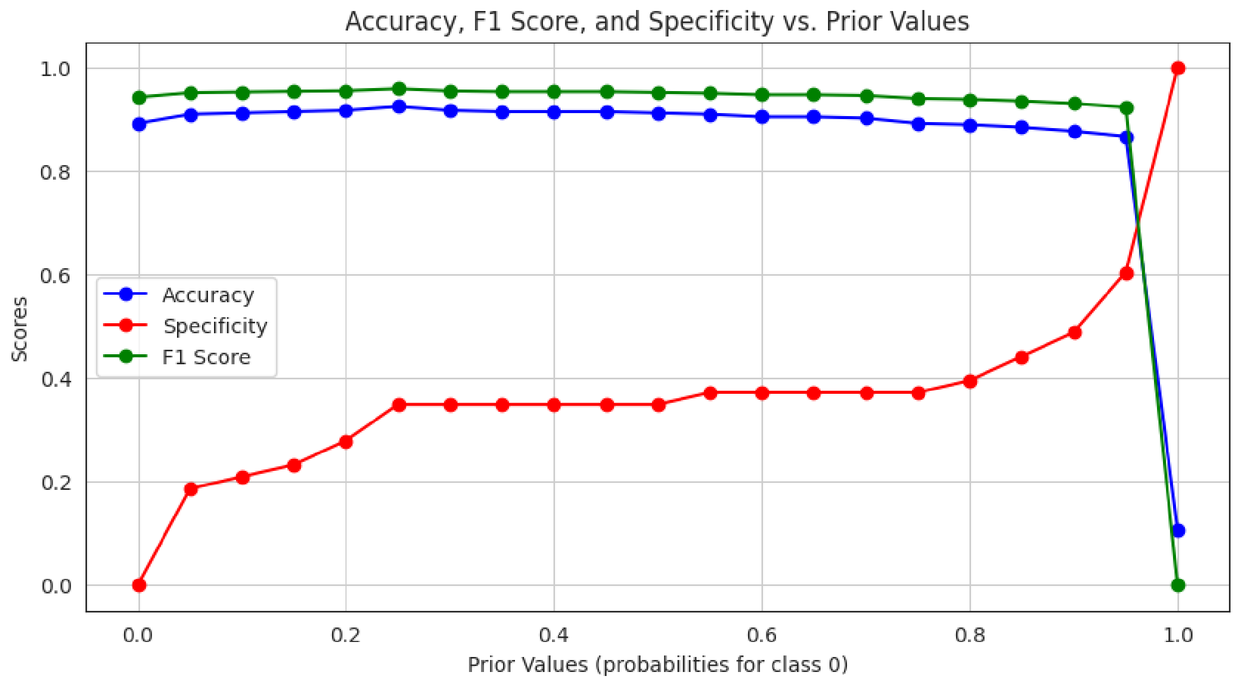
```
/usr/local/lib/python3.10/dist-packages/sklearn/naive_bayes.py:514:
RuntimeWarning: divide by zero encountered in log
  jointi = np.log(self.class_prior_[i])

**************************************
Dataset:  7
Accuracy: 0.8925
Specificity: 0.0000
**************************************
Dataset:  7
Accuracy: 0.9100
Specificity: 0.1860
**************************************
Dataset:  7
Accuracy: 0.9125
Specificity: 0.2093
**************************************
Dataset:  7
Accuracy: 0.9150
```

```
Specificity: 0.2326
**************************************
Dataset: 7
Accuracy: 0.9175
Specificity: 0.2791
**************************************
Dataset: 7
Accuracy: 0.9250
Specificity: 0.3488
**************************************
Dataset: 7
Accuracy: 0.9175
Specificity: 0.3488
**************************************
Dataset: 7
Accuracy: 0.9150
Specificity: 0.3488
**************************************
Dataset: 7
Accuracy: 0.9150
Specificity: 0.3488
**************************************
Dataset: 7
Accuracy: 0.9150
Specificity: 0.3488
**************************************
Dataset: 7
Accuracy: 0.9125
Specificity: 0.3488
**************************************
Dataset: 7
Accuracy: 0.9100
Specificity: 0.3721
**************************************
Dataset: 7
Accuracy: 0.9050
Specificity: 0.3721
**************************************
Dataset: 7
Accuracy: 0.9050
Specificity: 0.3721
**************************************
Dataset: 7
Accuracy: 0.9025
Specificity: 0.3721
**************************************
Dataset: 7
Accuracy: 0.8925
Specificity: 0.3721
```

```
**************************************
Dataset: 7
Accuracy: 0.8900
Specificity: 0.3953
**************************************
Dataset: 7
Accuracy: 0.8850
Specificity: 0.4419
**************************************
Dataset: 7
Accuracy: 0.8775
Specificity: 0.4884
**************************************
Dataset: 7
Accuracy: 0.8675
Specificity: 0.6047
**************************************
Dataset: 7
Accuracy: 0.1075
Specificity: 1.0000
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/naive_bayes.py:514:
RuntimeWarning: divide by zero encountered in log
  jointi = np.log(self.class_prior_[i])
```

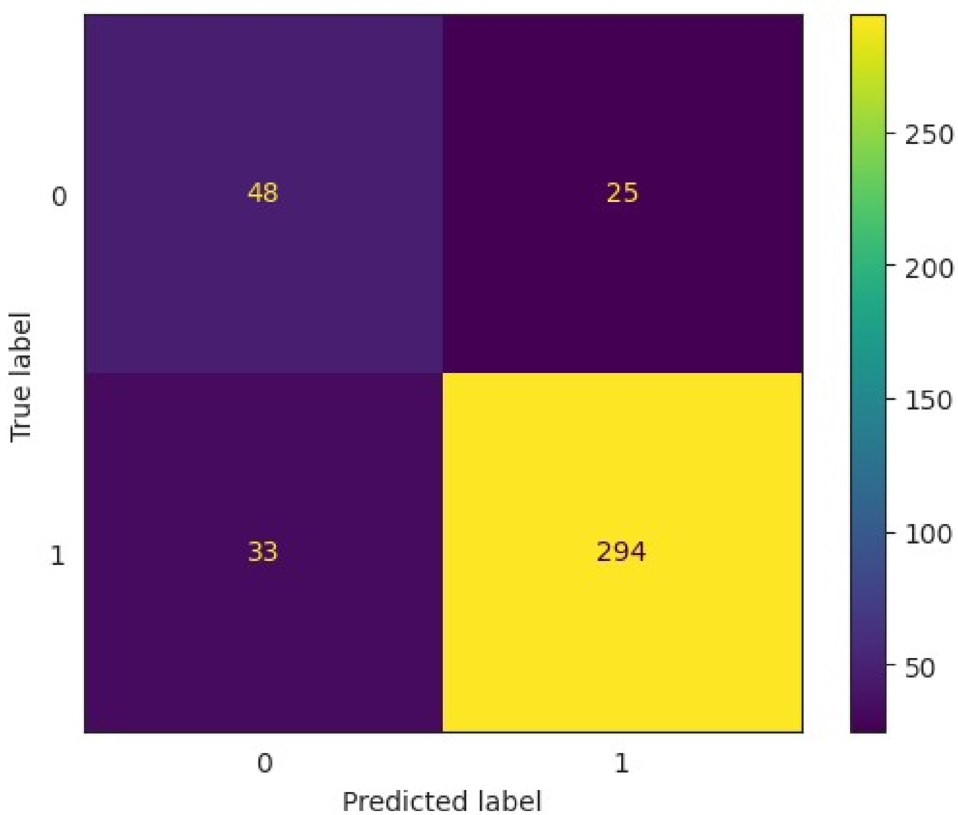
```
# Extract the prior values to use as x-coordinates for the plot
prior_x = [prior[0] for prior in priors]

# Create the plot
plt.figure(figsize=(10, 5))
plt.plot(prior_x, accuracy_d7, marker='o', linestyle='-', color='b',
label='Accuracy')
plt.plot(prior_x, specifities_d7, marker='o', linestyle='-',
color='r', label='Specificity')
plt.plot(prior_x, f1_d7, marker='o', linestyle='-', color='g',
label='F1 Score')
plt.title("Accuracy, F1 Score, and Specificity vs. Prior Values")
plt.xlabel("Prior Values (probabilities for class 0)")
plt.ylabel("Scores")
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



Use ComplementNB class in sklearn to fit a naive bayes classifier and see whether the performance (f1-score) increases for the imbalanced dataset.

```
def complementNB(X_train, Y_train, X_test, Y_test, d, conf = True):
    model = ComplementNB()
    model.fit(X_train, Y_train)

    Y_pred = model.predict(X_test)
    acc = accuracy_score(Y_test, Y_pred)
    f1 = f1_score(Y_test, Y_pred)
    cm = confusion_matrix(Y_test, Y_pred)
    tn, fp, fn, tp = cm.ravel()
    specificity = tn / (tn+fp)


    print('Dataset:', d)
    print(f"Accuracy: {acc:0,.4f}")
    print(f"Specificity: {specificity:0,.4f}")
    print(f'Classes known to classifier: {model.classes_}')
    print(f'Prior probabilities: {np.e ** model.class_log_prior_}')

    print('**************************************')
    sns.set_style("white")
    if conf:
```

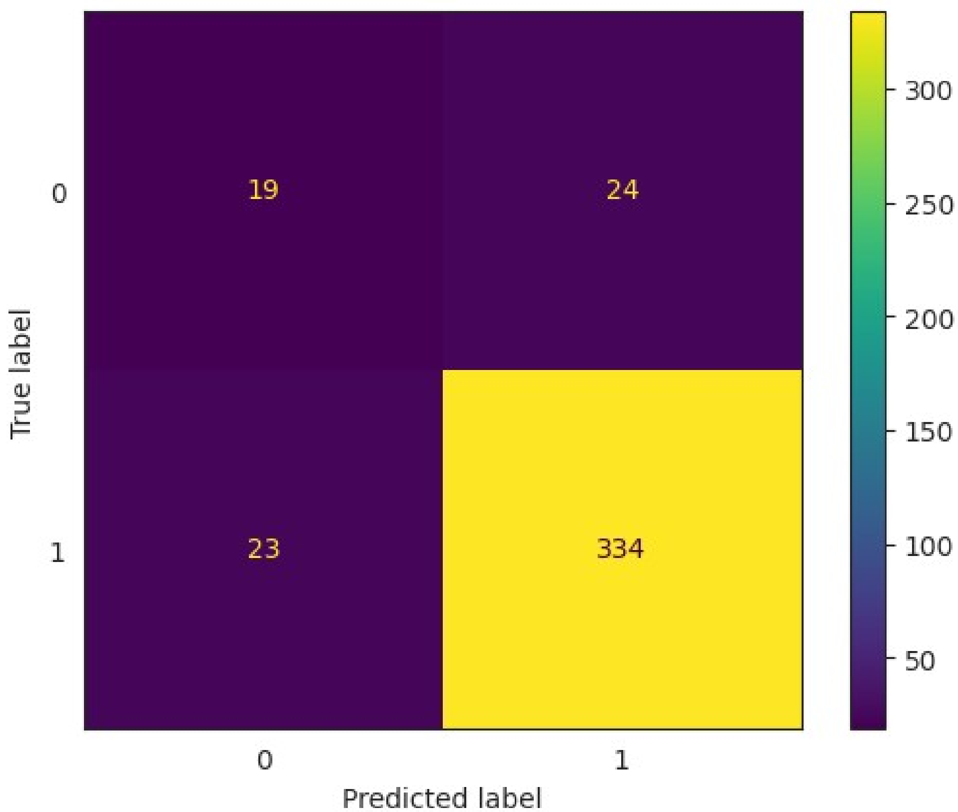
```
        disp = ConfusionMatrixDisplay(confusion_matrix=cm)
        disp.plot()
        plt.show()
    return [acc, f1, specificity]

scaler = MinMaxScaler()

X6_train_scaled = scaler.fit_transform(X6_train)
X6_test_scaled = scaler.transform(X6_test)

X7_train_scaled = scaler.fit_transform(X7_train)
X7_test_scaled = scaler.transform(X7_test)

res_d6 = complementNB(X6_train_scaled, Y6_train, X6_test_scaled,
Y6_test, 6, conf = True)
print(res_d6)

Dataset: 6
Accuracy: 0.8550
Specificity: 0.6575
Classes known to classifier: [0 1]
Prior probabilities: [0.208125 0.791875]
**************************************
```



```
[0.855, 0.9102167182662539, 0.6575342465753424]
```

```
res_d7 = complementNB(X7_train_scaled, Y7_train, X7_test_scaled,
Y7_test, 7, conf = True)
```

```
Dataset: 7
Accuracy: 0.8825
Specificity: 0.4419
Classes known to classifier: [0 1]
Prior probabilities: [0.1025 0.8975]
**************************************
```



So ComplementNB has better specificity than GaussianNB for both imbalanced dataset, hence there is an increase in the performance.