Naive Bayes Classifier for Lung Cancer

March 11, 2025

[1]:
```
## Import the required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
import joblib
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import StandardScaler
```

[2]:
```
## Load the dataset
df = pd.read_csv("C:/Users/PC/OneDrive/Desktop/Data Science/Datasets/Datasets/
 ↪Diabetes.csv")
```

[3]:
```
## Inspect the first few observations of the dataset
df.head(10)
```

[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
5	5	116	74	0	0	25.6	
6	3	78	50	32	88	31.0	
7	10	115	0	0	0	35.3	
8	2	197	70	45	543	30.5	
9	8	125	96	0	0	0.0	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1

```
3                       0.167   21        0
4                       2.288   33        1
5                       0.201   30        0
6                       0.248   26        1
7                       0.134   29        0
8                       0.158   53        1
9                       0.232   54        1
```

[4]:
```
## Assess the structure of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[5]:
```
## Check for duplicates
df.duplicated().sum()
```

[5]:
```
0
```

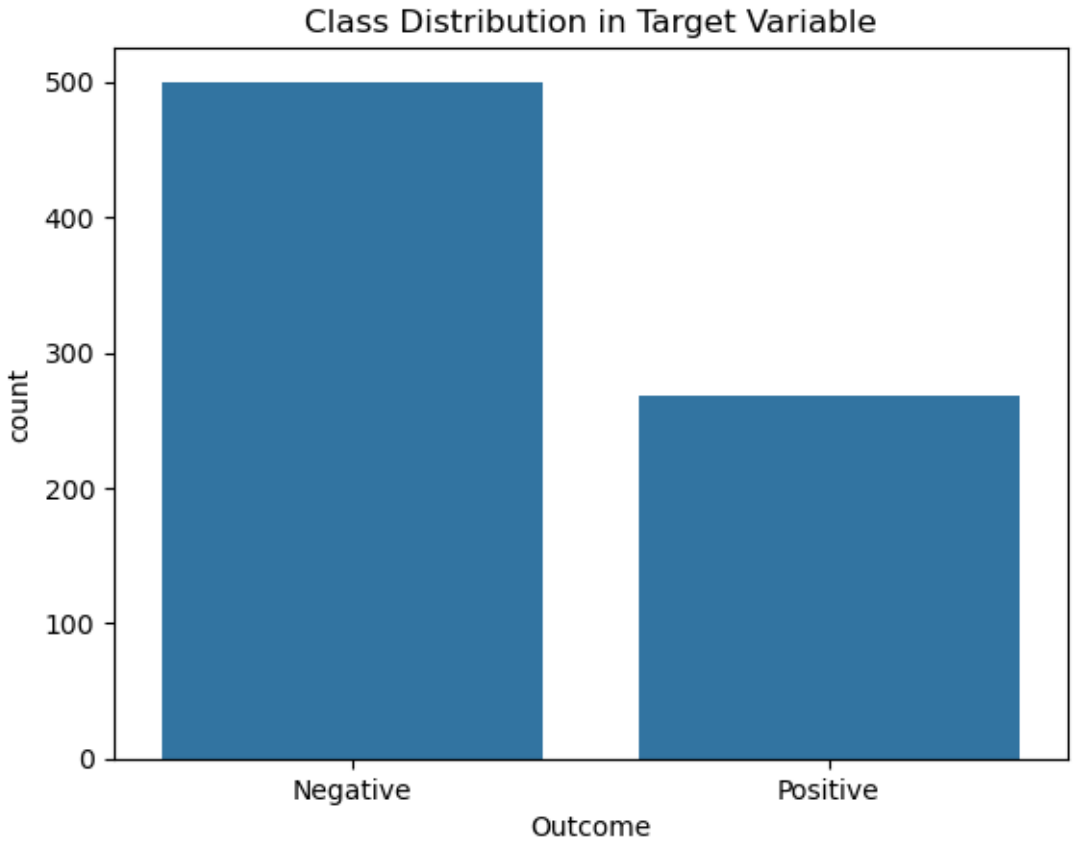
[7]:
```
## Check for missing values
df.isnull().sum()
```

[7]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[8]:
```
## ## Check target class distribution
print(df["Outcome"].value_counts())
```

```
Outcome
0    500
1    268
Name: count, dtype: int64
```

[9]:
```
# Visualize class distribution
sns.countplot(x=df['Outcome'])
plt.title("Class Distribution in Target Variable")
plt.xticks([0, 1], labels = ["Negative", "Positive"])
plt.show()
```



[10]:
```
## Summary statistics
df.describe()
```

[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	

```
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

                BMI  DiabetesPedigreeFunction         Age     Outcome
count   768.000000                768.000000  768.000000  768.000000
mean     31.992578                  0.471876   33.240885    0.348958
std       7.884160                  0.331329   11.760232    0.476951
min       0.000000                  0.078000   21.000000    0.000000
25%      27.300000                  0.243750   24.000000    0.000000
50%      32.000000                  0.372500   29.000000    0.000000
75%      36.600000                  0.626250   41.000000    1.000000
max      67.100000                  2.420000   81.000000    1.000000
```

[11]:
```
## Define Features and Target variable
X = df.drop(columns = ["Outcome"])
y = df["Outcome"]
```

[12]:
```
## Split the Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Check the shape
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
```

```
Training data shape: (614, 8)
Testing data shape: (154, 8)
```

[14]:
```
## Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[15]:
```
## Train the Naïve Bayes Model
# Initialize the classifier
nb_classifier = GaussianNB()

# Train (fit) the model
nb_classifier.fit(X_train, y_train)
```

[15]:
```
GaussianNB()
```

[16]:
```
## Make Predictions
y_pred = nb_classifier.predict(X_test)
```

[17]:
```
## Evaluate the Model
## Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.71
```

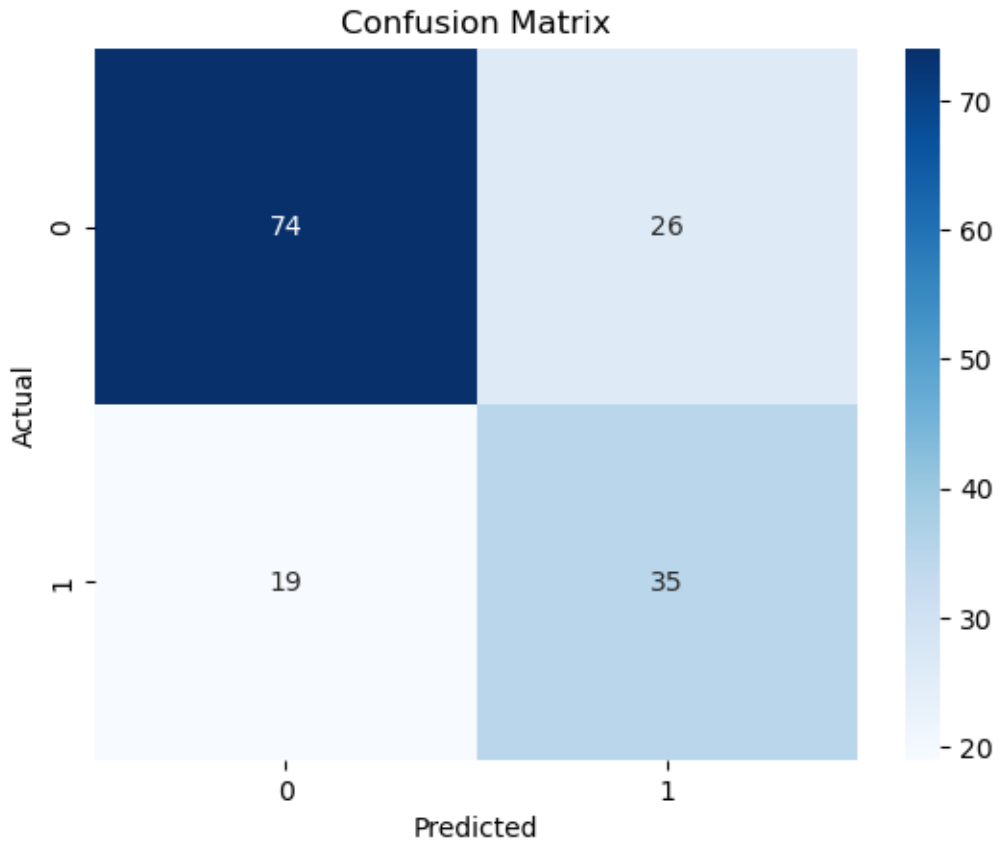
[18]:
```
## Classification Report
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.74      0.77       100
           1       0.57      0.65      0.61        54

    accuracy                           0.71       154
   macro avg       0.68      0.69      0.69       154
weighted avg       0.72      0.71      0.71       154
```

[19]:
```
## Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap="Blues", fmt='d')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```


Confusion Matrix
74
26
19
35
Actual
0
1
Predicted
0
1
70
60
50
40
30
20

[ ]: