Naive Bayes Classifier

Bayes's Theorem

According to the Wikipedia, In probability theory and statistics,* *Bayes's theorem*** *(alternatively *Bayes's law* or *Bayes's rule*) describes the probability of an event, based on prior knowledge of conditions that might be related to the event. Mathematically, it can be written as:

Where A and B are events and P(B)≠0

- P(A|B) is a conditional probability: the likelihood of event A occurring given that B is true.
- P(B|A) is also a conditional probability: the likelihood of event B occurring given that A is true.
- P(A) and P(B) are the probabilities of observing A and B respectively; they are known as the marginal probability.

Let's understand it with the help of an example:

The problem statement:

There are two machines which manufacture bulbs. Machine 1 produces 30 bulbs per hour and machine 2 produce 20 bulbs per hour. Out of all bulbs produced, 1 % turn out to be defective. Out of all the defective bulbs, the share of each machine is 50%. What is the probability that a bulb produced by machine 2 is defective?

We can write the information given above in mathematical terms as:

The probability that a bulb was made by Machine 1, P(M1)=30/50=0.6

The probability that a bulb was made by Machine 2, P(M2)=20/50=0.4

The probability that a bulb is defective, P(Defective)=1%=0.01

The probability that a defective bulb came out of Machine 1, P(M1 | Defective)=50%=0.5

The probability that a defective bulb came out of Machine 2, P(M2 | Defective)=50%=0.5

Now, we need to calculate the probability of a bulb produced by machine 2 is defective i.e., P(Defective | M2). Using the Bayes Theorem above, it can be written as:

$$P(Defective|M2) = \frac{P(M2|Defective) * P(Defective)}{P(M2)}$$

Substituting the values, we get: $P(Defective|M2) = \frac{0.5*0.01}{0.4} = 0.0125$

Task for you is to calculate the probability that a bulb produced by machine 1 is defective.

We'll extend this same understanding to understand the Naïve Baye's Algorithm.

Algorithm steps:

1. Let's consider that we have a binary classification problem i.e., we have two classes in our data as shown below.

2. Now suppose if we are given with a new data point, to which class does that point belong to?

3. The formula for a point 'X' to belong in class1 can be written as:

Where the numbers represent the order in which we are going to calculate different probabilities.
4. A similar formula can be utilised for class 2 as well.
5. Probability of class 1 can be written as:
$P(class1) = \frac{Numberofpointsinclass1}{Totalnumberofpoints} = \frac{16}{26} = 0.62$
6. For calculating the probability of X, we draw a circle around the new point and see how many points(excluding the new point) lie inside that circle.


The points inside the circle are considered to be similar points.
$P(X) = \frac{Numberofsimilarobservation}{TotalObservations} = \frac{3}{26} = 0.12$ 7. Now, we need to calculate the probability of a point to be in the circle that we have made given that it's of class 1.
$P(X|Class1) = \frac{Numberofpointsinclass1insidethecircle}{Totalnumberofpointsinclass1} = \frac{1}{16} = 0.06$ 8. We can substitute all the values into the formula in step 3. We get: $P(Class1|X) = \frac{0.06*0.62}{0.12} = 0.31$ 9. And if we calculate the probability that X belongs to Class2, we'll get 0.69. It means that our point belongs to class 2.

The Generalization for Multiclass:

The approach discussed above can be generalised for multiclass problems as well. Suppose, P1, P2, P3…Pn are the probabilities for the classes C1,C2,C3…Cn, then the point X will belong to the class for which the probability is maximum. Or mathematically the point belongs to the result of : $argmax(P1, P2, P3 \ldots . Pn)$

The Difference

You can notice a major difference in the way in which the Naïve Bayes algorithm works form other classification algorithms. It does not first try to learn how to classify the points. It directly uses the label to identify the two separate classes and then it predicts the class to which the new point shall belong.

Why it is called Naïve Bayes?

The entire algorithm is based on Bayes's theorem to calculate probability. So, it also carries forward the assumptions for the Bayes's theorem. But those assumptions(that the features are independent) might not always be true when implemented over a real-world dataset. So, those assumptions are considered *Naïve* and hence the name.

Gaussian Naive Bayes

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. Go back to the normal distribution lecture to review the formulas for the Gaussian/Normal Distribution.

For example of using the Gaussian Distribution, suppose the training data contain a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class. Let μ_c be the mean of the values in x associated with class c, and let σ^2_c be the variance of the values in x associated with class c. Then, the probability distribution of some value given a class, p(x=v|c), can be computed by plugging v into the equation for a Normal distribution parameterized by μ_c and σ^2_c. That is:

$$p(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}}\, e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

Python Implementation

In [1]:
```
#Let's start with importing necessary libraries

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
#from sklearn.linear_model  import Ridge,Lasso,RidgeCV, LassoCV, ElasticNe
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, r
import matplotlib.pyplot as plt
import seaborn as sns
import scikitplot as skl
sns.set()
```

In [2]:
```
data = pd.read_csv("diabetes.csv") # Reading the Data
data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

In [3]:
```
data.describe()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

we can see there few data for columns Glucose, Insulin, skin thickness, BMI and Blood Pressure which have value as 0. That's not possible. You can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

In [4]:
```
# replacing zero values with the mean of the column
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressur
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness
```

In [5]:
```
# Handling the Outliers

q = data['Pregnancies'].quantile(0.98)
# we are removing the top 2% data from the Pregnancies column
data_cleaned = data[data['Pregnancies']<q]
q = data_cleaned['BMI'].quantile(0.99)
# we are removing the top 1% data from the BMI column
data_cleaned  = data_cleaned[data_cleaned['BMI']<q]
q = data_cleaned['SkinThickness'].quantile(0.99)
# we are removing the top 1% data from the SkinThickness column
data_cleaned  = data_cleaned[data_cleaned['SkinThickness']<q]
q = data_cleaned['Insulin'].quantile(0.95)
# we are removing the top 5% data from the Insulin column
data_cleaned  = data_cleaned[data_cleaned['Insulin']<q]
q = data_cleaned['DiabetesPedigreeFunction'].quantile(0.99)
# we are removing the top 1% data from the DiabetesPedigreeFunction column
data_cleaned  = data_cleaned[data_cleaned['DiabetesPedigreeFunction']<q]
q = data_cleaned['Age'].quantile(0.99)
# we are removing the top 1% data from the Age column
data_cleaned  = data_cleaned[data_cleaned['Age']<q]
```

In [6]:
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data_cleaned:
    if plotnumber<=9 :
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(data_cleaned[column])
        plt.xlabel(column,fontsize=20)
        #plt.ylabel('Salary',fontsize=20)
    plotnumber+=1
plt.show()
```

```
C:\Users\HOME\Anaconda3\lib\site-packages\seaborn\distributions.py:255
7: FutureWarning: `distplot` is a deprecated function and will be remo
ved in a future version. Please adapt your code to use either `displot
` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\HOME\Anaconda3\lib\site-packages\seaborn\distributions.py:255
7: FutureWarning: `distplot` is a deprecated function and will be remo
ved in a future version. Please adapt your code to use either `displot
` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\HOME\Anaconda3\lib\site-packages\seaborn\distributions.py:255
7: FutureWarning: `distplot` is a deprecated function and will be remo
ved in a future version. Please adapt your code to use either `displot
` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\HOME\Anaconda3\lib\site-packages\seaborn\distributions.py:255
```

In [7]:
```
X = data.drop(columns = ['Outcome'])
y = data['Outcome']
```

In [8]:
```
# we need to scale our data as well

scalar = StandardScaler()
X_scaled = scalar.fit_transform(X)
```

In [9]:
```
# This is how our data Looks now after scaling.
X_scaled
```

Out[9]:
```
array([[ 0.63994726,  0.86527574, -0.0210444 , ...,  0.16725546,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.20598931, -0.51658286, ..., -0.85153454,
        -0.36506078, -0.19067191],
       [ 1.23388019,  2.01597855, -0.68176235, ..., -1.33182125,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 , -0.02240928, -0.0210444 , ..., -0.90975111,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.14197684, -1.01212132, ..., -0.34213954,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.94297153, -0.18622389, ..., -0.29847711,
        -0.47378505, -0.87137393]])
```

In [11]:
```
# now we will check for multicollinearity using VIF(Variance Inflation fac
vif = pd.DataFrame()
vif["Features"] = X.columns
vif["vif"] = [variance_inflation_factor(X_scaled,i) for i in range(X_scale

#let's check the values
vif
```

Out[11]:

	Features	vif
0	Pregnancies	1.431075
1	Glucose	1.347308
2	BloodPressure	1.247914
3	SkinThickness	1.450510
4	Insulin	1.262111
5	BMI	1.550227
6	DiabetesPedigreeFunction	1.058104
7	Age	1.605441

All the VIF values are less than 5 and are very low. That means no multicollinearity. Now, we can go ahead with fitting our data to the model. Before that, let's split our data in test and training set.

In [12]:
```
x_train,x_test,y_train,y_test = train_test_split(X_scaled,y, test_size= 0.
```

In [13]:
```
model = GaussianNB()
```

In [14]:
```
model.fit(x_train,y_train)
```

Out[14]:
```
GaussianNB()
```

In [ ]:
```
import pickle
# Writing different model files to file
with open( 'modelForPrediction.sav', 'wb') as f:
    pickle.dump(model,f)

with open('standardScalar.sav', 'wb') as f:
    pickle.dump(scalar,f)
```

In [15]:
```
y_pred = model.predict(x_test)
```

In [16]:
```
print(accuracy_score(y_test, y_pred))
```

```
0.7922077922077922
```

In [17]:
```
# Confusion Matrix
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

Out[17]:
```
array([[90,  9],
       [23, 32]], dtype=int64)
```

In [18]:
```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

In [19]:
```
# Breaking down the formula for Accuracy
Accuracy = (true_positive + true_negative) / (true_positive +false_positive
Accuracy
```

Out[19]:
```
0.7922077922077922
```

In [20]:
```
# Precison
Precision = true_positive/(true_positive+false_positive)
Precision
```

Out[20]:
```
0.9090909090909091
```

In [21]:
```
# Recall
Recall = true_positive/(true_positive+false_negative)
Recall
```

Out[21]:
```
0.7964601769911505
```

In [22]:
```
# F1 Score
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

Out[22]:
```
0.8490566037735849
```

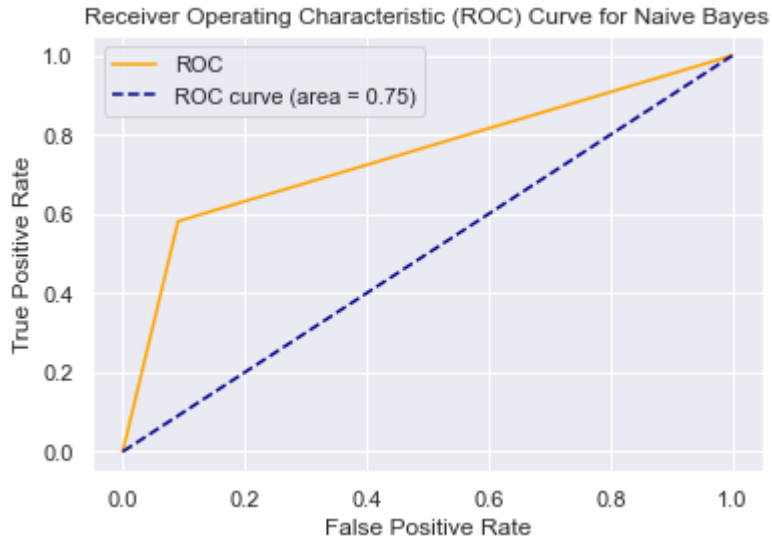
In [23]:
```
# Area Under Curve
auc = roc_auc_score(y_test, y_pred)
auc
```

Out[23]:
```
0.7454545454545454
```

So far we have been doing grid search to maximise the accuracy of our model. Here, we'll follow a different approach. We'll create two models, one with Logistic regression and other with Naïve Bayes and we'll compare the AUC. The algorithm having a better AUC shall be considered for production deployment.

In [30]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

In [31]:
```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Naive Bayes')
plt.legend()
plt.show()
```



In [24]:
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()

log_reg.fit(x_train,y_train)
```

Out[24]:
```
LogisticRegression()
```

In [25]:
```
y_pred_logistic = log_reg.predict(x_test)
```

In [26]:
```
accuracy_logistic = accuracy_score(y_test,y_pred_logistic)
accuracy_logistic
```

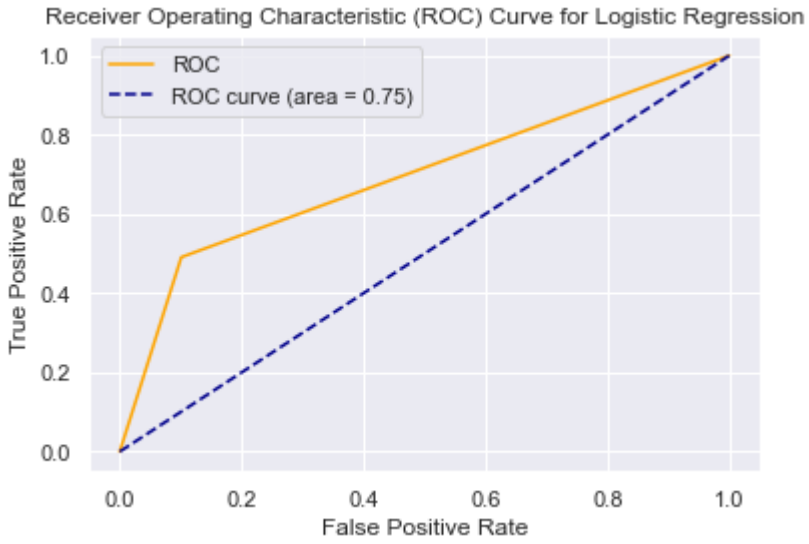
Out[26]:
```
0.7532467532467533
```

In [27]:
```
# Confusion Matrix
conf_mat = confusion_matrix(y_test,y_pred_logistic)
conf_mat
```

Out[27]:
```
array([[89, 10],
       [28, 27]], dtype=int64)
```

In [28]:
```
# ROC
fpr_logistic, tpr_logistic, thresholds_logistic = roc_curve(y_test, y_pred
```

In [29]:
```
plt.plot(fpr_logistic, tpr_logistic, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Logistic Regre
plt.legend()
plt.show()
```



In [32]:
```
from sklearn.metrics  import roc_auc_score
```

In [33]:
```
auc_naive=roc_auc_score(y_test,y_pred)
auc_naive
```

Out[33]:
```
0.7454545454545454
```

In [34]:
```
auc_logistic=roc_auc_score(y_test,y_pred_logistic)
auc_logistic
```

Out[34]:
```
0.694949494949495
```

Here, you can see that the AUC for Naïve Bayes is more. So, we'll take that as our production-ready model.

Advantages:

- Naive Bayes is extremely fast for both training and prediction as they not have to learn to create separate classes.
- Naive Bayes provides a direct probabilistic prediction.
- Naive Bayes is often easy to interpret.
- Naive Bayes has fewer (if any) parameters to tune

Disadvantages:

- The algorithm assumes that the features are independent which is not always the scenario
- Zero Frequency i.e. if the category of any categorical variable is not seen in training data set even once then model assigns a zero probability to that category and then a

prediction cannot be made.

In [ ]:

In [ ]: