Logistic Regression

1.Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

2.Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

3.Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logisticregression is used for solving the classification problems.

4.In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

5.Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

6.Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.

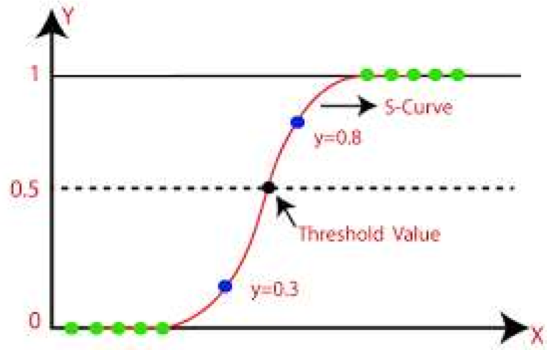
Logistic Function (Sigmoid Function):

1. sigmoid function is a mathematical function used to map the predicted values to probabilities.

2.It maps any real value into another value within a range of 0 and 1.

3.The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.

4.In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.



Assumptions for Logistic Regression:

1.The dependent variable must be categorical in nature. 2.The independent variable should not have multi-collinearity

In [27]:
```
import pandas as pd
import numpy as np
```

In [28]:
```
data={
    "x" : [0.1,0.2,0.3,0.4,0.6,0.7,0.8,0.9],
    'y' : [0,0,0,0,1,1,1,1]
}
```

In [29]:
```
data
```

Out[29]:
```
{'x': [0.1, 0.2, 0.3, 0.4, 0.6, 0.7, 0.8, 0.9], 'y': [0, 0, 0, 0, 1, 1, 1, 1]}
```

In [30]:
```
df=pd.DataFrame(data=data)
```

In [31]:
```
df
```

Out[31]:

	x	y
0	0.1	0
1	0.2	0
2	0.3	0
3	0.4	0
4	0.6	1
5	0.7	1
6	0.8	1
7	0.9	1

In [32]:

```
df['sum_xy']=df['x']*df['y']
```

In [33]:

```
df['sqr_x'] = df['x']** 2
```

In [34]:

```
df
```

Out[34]:

	x	y	sum_xy	sqr_x
0	0.1	0	0.0	0.01
1	0.2	0	0.0	0.04
2	0.3	0	0.0	0.09
3	0.4	0	0.0	0.16
4	0.6	1	0.6	0.36
5	0.7	1	0.7	0.49
6	0.8	1	0.8	0.64
7	0.9	1	0.9	0.81

Sigmoid

$$y_{pred} = \frac{1}{1 + e^{-xcap}}$$

In [213…

```
class logistic_Regression:
    def __init__(self,df):
        self.n = len(df)
        self.sum_x = df['x'].sum()
        self.sum_y = df['y'].sum()
        self.sum_xy = df['sum_xy'].sum()
        self.sqr_x = df['sqr_x'].sum()
        self.sumx_h_2 = sum_x ** 2


    def m_val(self,n, sum_x, sum_y, sum_xy, sqr_x, sumx_h_2):   ## Find m value
        self.num_m = n*((sum_xy))-(sum_x)*(sum_y)
        self.den_m = n*((sqr_x))-(sumx_h_2)
        m = num_m / den_m
        return m

    def b_val(self,n, sum_x, sum_y):       ## Find b value
        self.num_b = (sum_y) - m*(sum_x)
        self.den_b = n
        self.b = self.num_b / self.den_b
        return self.b

    x_cap = [(m*x + b) for x in df['x']]
    print("x_cap values:",x_cap)

    def sigmoid(x_cap):
```

```
        return [(1/(1+np.exp(-xcap))) for xcap in x_cap]

    ypred = sigmoid(x_cap)
    print("sigmoid values:",ypred)

    def final(ypred):
        re = [1 if val>=0.5 else 0 for val in ypred]
        return re

    ypred = final(ypred)
    print("y predicted values:",ypred)

    ytrue = df.y.values
    print("y true values:",ytrue)


    def conf_matrix(self,ytrue,ypred):           ## Confusion matrix
        TP = 0
        FP = 0
        TN = 0
        FN = 0

        for i in range(len(ypred)):
            if ytrue[i]==ypred[i]==1:
                TP += 1
            if ypred[i]==1 and ytrue[i]!=ypred[i]:
                FP += 1
            if ytrue[i]==ypred[i]==0:
                TN += 1
            if ypred[i]==0 and ytrue[i]!=ypred[i]:
                FN += 1

        return(TP, FP, TN, FN)
    #print("confusion matrix:",conf_matrix(ytrue, ypred))

    def Diagnostics_1(self,TP,FP,FN,TN): # Evalutions

        accuracy_1 = (TP+TN)/(TP+FP+FN+TN)       # for 1
        precission_1 = (TP)/(TP+FP)
        recall_1 = (TP)/(TP+FN)
        f1_score_1 = 2*((precission_1*recall_1)/(precission_1+recall_1))

        return accuracy_1,precission_1,recall_1,f1_score_1

    def Diagnostics_0(self,TP,FP,FN,TN):

        accuracy_0 = (TP+TN)/(TP+FP+FN+TN)       # for 0
        precission_0 = (TN)/(TN+FN)
        recall_0 = (TN)/(TN+FP)
        f1_score_0 = 2*((precission_0*recall_0)/(precission_0+recall_0))

        return accuracy_0,precission_0,recall_0,f1_score_0
```

```
x_cap values: [-0.16666666666666705, -2.7755575615628914e-16, 0.1666666666666664, 0.3333
3333333333326, 0.6666666666666666, 0.8333333333333334, 1.0000000000000004, 1.16666666666
6667]
```

```
sigmoid values: [0.45842951678320004, 0.5, 0.5415704832167998, 0.5825702064623147, 0.660
7563687658172, 0.6970592839654074, 0.731058578630005, 0.7625419716560975]
y predicted values: [0, 1, 1, 1, 1, 1, 1, 1]
y true values: [0 0 0 0 1 1 1 1]
```

In [214…

```
m_obj = logistic_Regression(df)    ## object declaration for m(slope)
m = m_obj.m_val(n,sum_x, sum_y, sum_xy, sqr_x, sumx_h_2)   # obj.methodname()
print("m value:",m)                                       # print m value
```

```
m value: 1.6666666666666676
```

In [215…

```
b_obj = logistic_Regression(df)   ## object declaration for b(constant)
b = b_obj.b_val(n, sum_x, sum_y)            # obj.methodname()
print("b value:",b)
```

```
b value: -0.3333333333333338
```

In [216…

```
d_obj1=logistic_Regression(df)  ## Accuracy object
accuracy_1,precission_1,recall_1,f1_score_1 = d_obj1.Diagnostics_1(TP,FP,FN,TN)
accuracy_1,precission_1,recall_1,f1_score_1
```

Out[216…

```
(0.625, 0.5714285714285714, 1.0, 0.7272727272727273)
```

In [217…

```
d_obj2=logistic_Regression(df)  ## Accuracy object
accuracy_0,precission_0,recall_0,f1_score_0 = d_obj2.Diagnostics_0(TP,FP,FN,TN)
accuracy_0,precission_0,recall_0,f1_score_0
```

Out[217…

```
(0.625, 1.0, 0.25, 0.4)
```

In [218…

```
from sklearn.metrics import classification_report
print(classification_report(ytrue,ypred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.25      0.40         4
           1       0.57      1.00      0.73         4

    accuracy                           0.62         8
   macro avg       0.79      0.62      0.56         8
weighted avg       0.79      0.62      0.56         8
```

In [243…

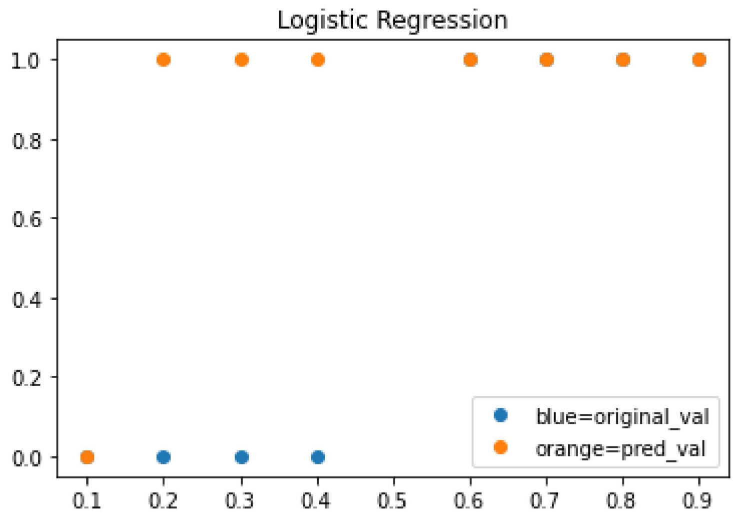
```
#graph

import matplotlib.pyplot as plt

x_val = df['x']
ytrue = df['y']
y_pre = ypred

#plt.plot(x_val,ytrue,color='g')
plt.scatter(x_val,ytrue,label='blue=original_val')    # for original values
plt.scatter(x_val,ypred,label='orange=pred_val')     # for predicted values
plt.title("Logistic Regression")
```

```
plt.legend()
plt.show()
```



In [ ]:

In [ ]:

In [ ]: