Aim of this project / Problem Statement

This project teaches us how actually logistic regression algorithm is used to solve classification problems and how it is written in the sklearn library and the indepth mathematical intuition behind this algorithm because I have implemented this algorithm from scratch.

The entire theory for the implementation of can be understood by referring to my notes at following link (recommended as must read) : https://drive.google.com/drive/folders/14RdGvYg7camF7HhmStaq75q4JXSH4ocK?usp=sharing

The dataset used can be found at this link : https://drive.google.com/file/d/1XAcLuaKaBl3Z3VbPqIPlw39YPjD9TDIa/view?usp=sharing

By reading the theory you might have understood that ultimately the algorithm reduces to solving a convex unconstrained nonlinear optimization problem and we know there are several ways to solve this optimization problem problem so here we will be solving this optimization problem via these methods from scratch:

1. Gradient Descent
2. Stochastic Gradient Descent

You can find the code for all these at following link : https://colab.research.google.com/drive/1pG6GkKT_ypfXapUWZo1odxYPqH6tCjL0?usp=sharing

Code Design

The code can be divided into following subsections for clear understanding:

A. Importing Required Libraries :

You need to import some essential basic manipulation libraries to do some math operations and hence we are importing these libraries here.Some of these libraries are numpy, matplotlib and pandas.

B.Data Preprocessing :

1. First we use pandas library to read the dataset from the location where it has been uploaded

   The dataset used in this project consists of three independent features i.e. X1, X2 ,X3,X4 and a dependent feature Y. Our goal is to use these independent features to predict the values of dependent features.

2. The we are shuffling the data cause all the class 1 data are at the top and all the class 2 data are at the bottom

3. Since in the given dataset all the features are around the same scale we are not performing any feature scaling.

4. Train, validation and test splitting the original dataset as per 70:10:20 ration

C.Data Learning Without Using SKLearn Library :

1. Defining Our Evaluation Metric Class

This class is used to calculate the performance of our model.Now we know that there are various performance measures like accuracy, F1 Score , Precision , Recall etc etc.

The formula of accuracy is given by the following mathematical equation.

$$\text{Accuracy} = \frac{True\ Positives + Ture\ Negatives}{True\ Positives + Ture\ Negatives + False\ Positives + False\ Negatives}$$

The formula of precision is given by following mathematical equation

$$\text{Precision} = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

The formula of recall is given by following mathematical equation

$$\text{Recall} = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

The formula of F1 Score is given by following mathematical equation

F1 Score = harmonic mean of precision and recall =

$$\frac{Precision * Recall}{(Precision + Recall)/2}$$

Values of true positives , true negatives , false positives and false negatives can be calculated using the confusion matrix.

2. Using Gradient Descent (Vectorized Formula) as Optimization Method :

Here first I have created a class which performs this optimization for us. The name of the class is ‘GradientDescent’. This class contains several functions whose purpose is clearly mentioned in the code. To

understand the math and derivation of the vectorized formula used in this class the reader must go through the notes linked above. Once the reader has understood the mathematical derivation for the vectorized formula for solving logistic regression using the gradient descent provided in the notes the reader will take no time to understand the code with the additional help provided via comments in the code wherever necessary.

In this method we don't directly go to the minima but instead we reach close to the global minima of the cost/error function gradually by jumping from point to another point.The new point in every iteration is decided by the gradient at that old point.

The object of class ‘Gradient Descent’ and ‘Evaluation Metric’ is created to use both these classes

Performing Hyperparameter Tuning Using Validation Dataset

Now as we know the learning rate is a hyperparameter, hence I am doing hyperparameter tuning by trying out 3 different learning rates using validation dataset.

For Learning Rate = 10^(-5)

```
Cofficients : [ 0.60083013 -1.27146694 -0.64113134 -0.67618024 -0.21195492]
```

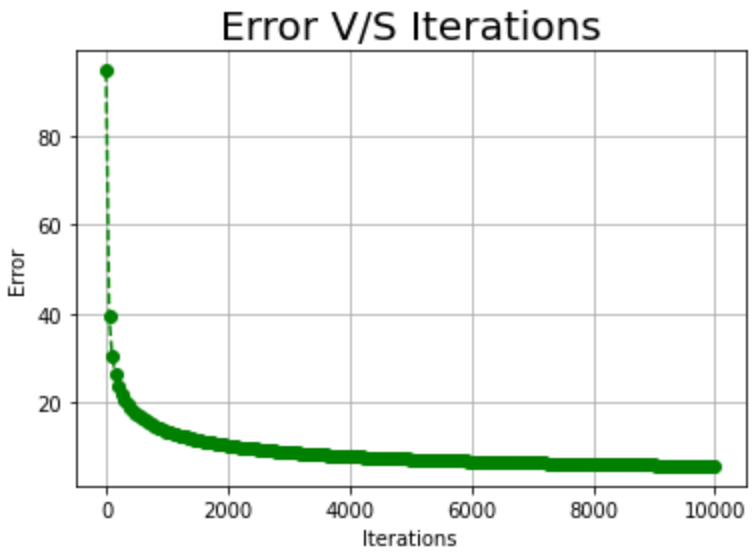

```
Total Error After Completing Training : 13.378683236387689

Evaluation Metrics for Validation Dataset

True Positives : 52
True Negatives : 81
False Positives : 3
False Negatives : 1
Accuracy : 0.9708029197080292
Precision : 0.9454545454545454
Recall : 0.9811320754716981
F1 score : 0.9629629629629629
```

For Learning Rate = 10^(-4)

```
Cofficients : [ 2.40788492 -2.3222907  -1.29197271 -1.57604107 -0.07007233]
```



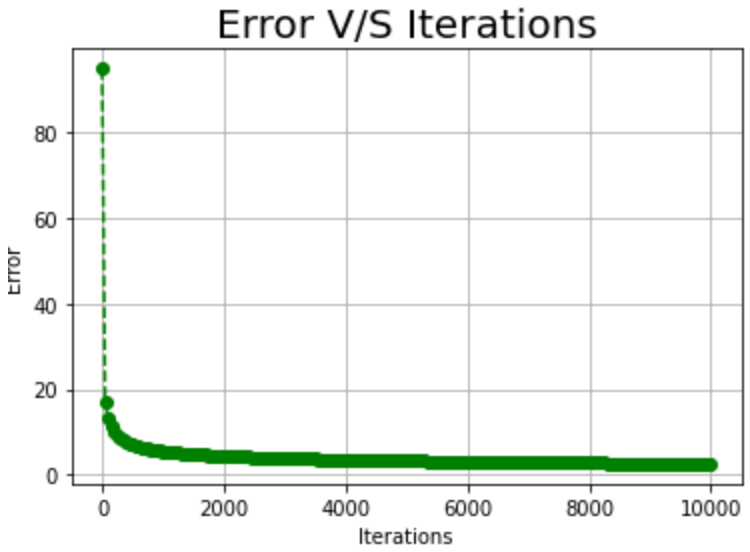

```
Total Error After Completing Training : 5.505329527738511

Evaluation Metrics for Validation Dataset

True Positives : 55
True Negatives : 81
False Positives : 0
False Negatives : 1
Accuracy : 0.9927007299270073
Precision : 1.0
Recall : 0.9821428571428571
F1 score : 0.990990990990991
```

For Learning Rate = 10^(-3)

```
Cofficients : [ 5.49428539 -5.16325035 -2.51014604 -3.37221418  0.18279112]
```




```
Total Error After Completing Training : 2.587597212769285

Evaluation Metrics for Validation Dataset

True Positives : 55
True Negatives : 82
False Positives : 0
False Negatives : 0
Accuracy : 1.0
Precision : 1.0
Recall : 1.0
F1 score : 1.0
```

We can conclude from the following graphs that as the learning rate decreases the number of iterations taken to reach the minima increases thus training time increases.

Since the evaluation metric for validation dataset with 0.001 learning rate is the best hence 0.001 is the best learning rate.Hence we will be using this learning rate for training and testing our model.

Train and Test Results using best hyperparameter

```
Cofficients : [ 5.98441778 -7.05218995 -3.63226529 -4.58283044 -0.48192649]

Evaluation Metrics for Training Dataset

True Positives : 432
True Negatives : 520
False Positives : 3
False Negatives : 5
Accuracy : 0.9916666666666667
Precision : 0.993103448275862
Recall : 0.988558352402746
F1 score : 0.9908256880733946

Evaluation Metrics for Testing Dataset

True Positives : 116
True Negatives : 155
False Positives : 3
False Negatives : 0
Accuracy : 0.9890510948905109
Precision : 0.9747899159663865
Recall : 1.0
F1 score : 0.9872340425531915
```

3. Using Stochastic Gradient Descent (Vectorized Formula) as Optimization Method :

Here first I have created a class which performs this optimization for us. The name of the class is 'StochasticGradientDescent'. This class contains several functions whose purpose is clearly mentioned in the code.Class StochasticGradientDescent in the code is exactly same just that instead of take error function as summation over all the training examples it considers error due to a randomly selected point in every iteration and updates the weights using the same update formula

Stochastic Error = (-Yi)ln(sigmoid(Wo+W1*X1+.....Wn*Xn)) - (1-Yi)ln(sigmoid(1-Wo+W1*X1+.....Wn*Xn))

In this method we don't directly go to the minima but instead we reach close to the global minima of the cost/error function gradually by

jumping from point to another point.The new point in every iteration is decided by the gradient at that old point.

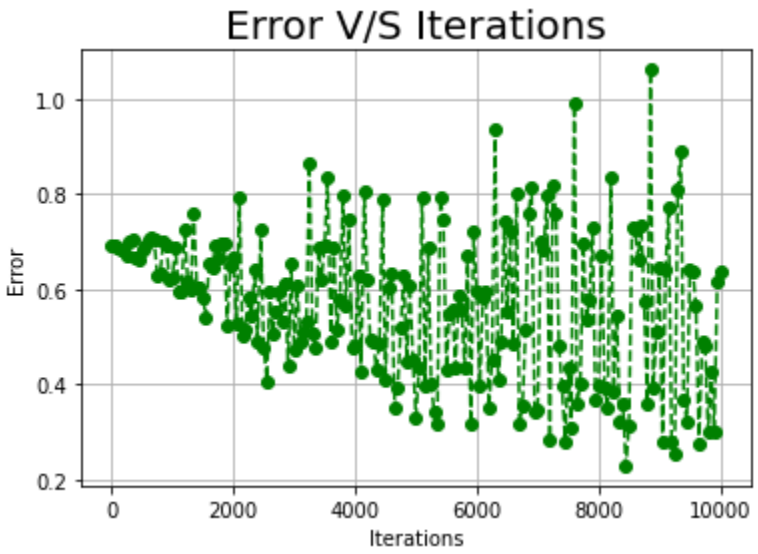
The object of class ‘StochasticGradientDescent’ and ‘Evaluation Metric’ is created to use both these classes

Performing Hyperparameter Tuning Using Validation Dataset

Now as we know the learning rate is a hyperparameter, hence I am doing hyperparameter tuning by trying out 3 different learning rates using validation dataset.

For Learning Rate = 10^(-5)

```
Cofficients : [-0.00625827 -0.09612068 -0.09381582  0.00987556  0.00133283]
```



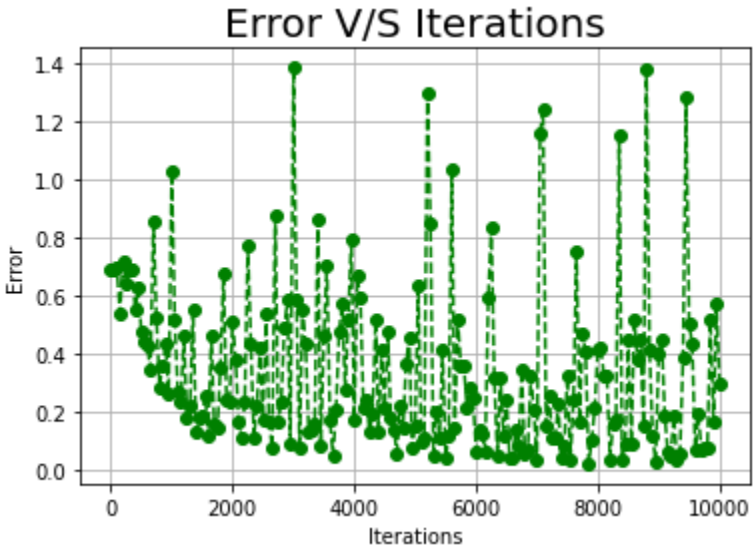
```
Total Error After Completing Training : 0.6356590996279623

Evaluation Metrics for Validation Dataset

True Positives : 38
True Negatives : 74
False Positives : 17
False Negatives : 8
Accuracy : 0.8175182481751825
Precision : 0.6909090909090909
Recall : 0.8260869565217391
F1 score : 0.7524752475247525
```

For Learning Rate = 10^(-4)

```
Cofficients : [ 0.01127746 -0.4869871  -0.23696884 -0.13138757 -0.06111139]
```



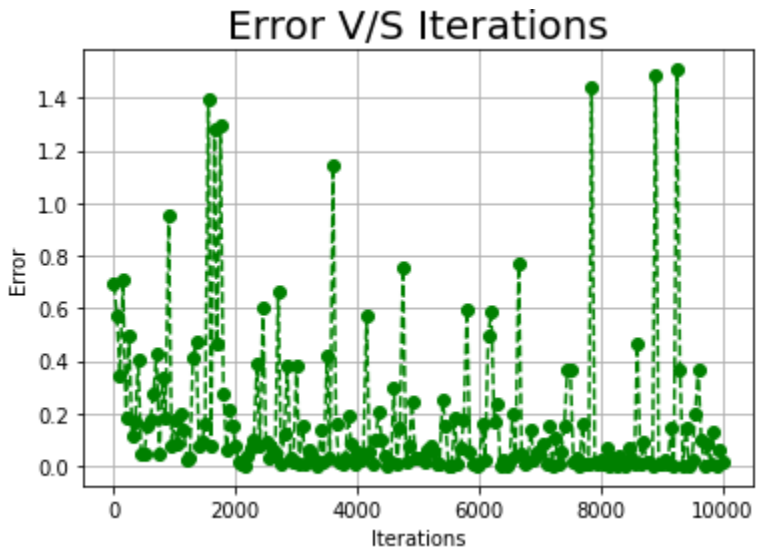

```
Total Error After Completing Training : 0.29565204295101766

Evaluation Metrics for Validation Dataset

True Positives : 46
True Negatives : 81
False Positives : 9
False Negatives : 1
Accuracy : 0.927007299270073
Precision : 0.8363636363636363
Recall : 0.9787234042553191
F1 score : 0.9019607843137255
```

For Learning Rate = 10^(-3)

```
Cofficients : [ 0.44591314 -1.15205961 -0.5897012  -0.59320507 -0.20466786]
```




```
Total Error After Completing Training : 0.016511927875883 74

Evaluation Metrics for Validation Dataset

True Positives : 52
True Negatives : 81
False Positives : 3
False Negatives : 1
Accuracy : 0.9708029197080292
Precision : 0.9454545454545454
Recall : 0.9811320754716981
F1 score : 0.9629629629629629
```

We can conclude from the following graphs that as the learning rate decreases the number of iterations taken to reach the minima increases thus training time increases.

Since the evaluation metric for validation dataset with 0.001 learning rate is the best hence 0.001 is the best learning rate.Hence we will be using this learning rate for training and testing our model.

Train and Test Results using best hyperparameter

```
Cofficients : [ 0.55613937 -1.18128594 -0.62835849 -0.66530589 -0.26611512]

Evaluation Metrics for Training Dataset

True Positives : 420
True Negatives : 516
False Positives : 15
False Negatives : 9
Accuracy : 0.975
Precision : 0.9655172413793104
Recall : 0.9790209790209791
F1 score : 0.9722222222222222

Evaluation Metrics for Testing Dataset

True Positives : 112
True Negatives : 155
False Positives : 7
False Negatives : 0
Accuracy : 0.9744525547445255
Precision : 0.9411764705882353
Recall : 1.0
F1 score : 0.9696969696969697
```

D.Data Learning Using SKLearn Library :

Firstly an object is created from a logistic regression class already defined in SKLearn Library.Then the train and test dataset were fitted and following results were obtained

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
[[-3.12393012e+00 -1.67774473e+00 -2.07587965e+00 -2.88981229e-03]]
[3.38302894]
```

```
[[155   0]
 [  1 118]]
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       155
           1       1.00      0.99      1.00       119

    accuracy                           1.00       274
   macro avg       1.00      1.00      1.00       274
weighted avg       1.00      1.00      1.00       274
```

Hence we can see that these values are very close to the one that I got via my implementation and this confirms that what I have implemented is absolutely correct.