CSE 574: INTRODUCTION TO MACHINE LEARNING

(FALL 2019)

Prof. S. N. Srihari
University at Buffalo

PROJECT 1 REPORT

Submitted by:

Vishva Nitin Patel

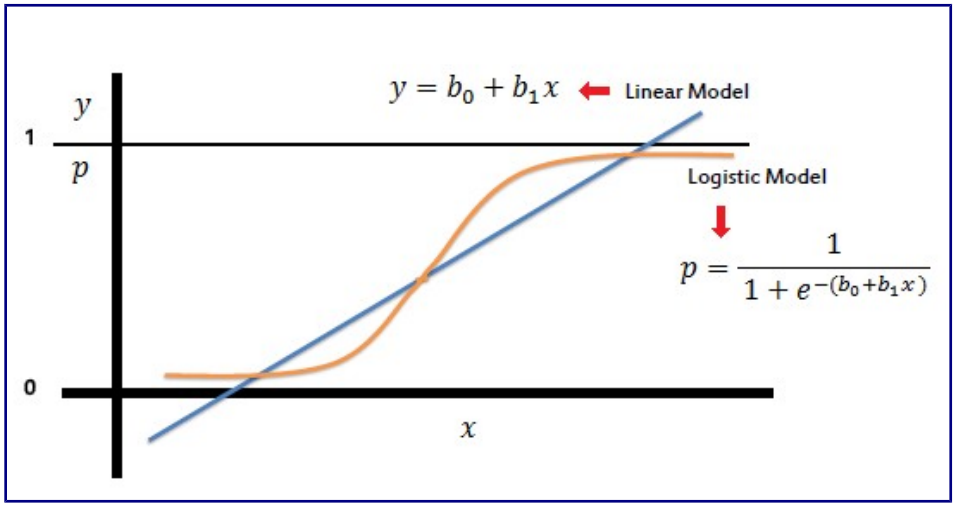
ABSTRACT:

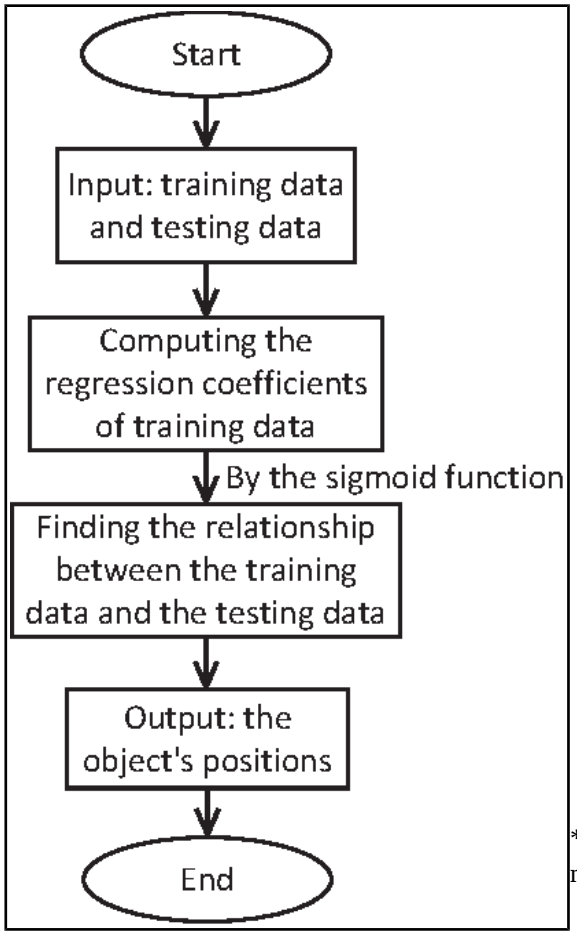
Presented in this report is the utilization of Logistic Regression for classification. The Wisconsin Diagnostic Breast Cancer data set is being used to extract the required features. I have tried to classify suspected fine needle aspirate cells into two classes Benign and Malignant using Logistic Regression as the classifier. I have implemented the classification program in python from scratch and findings of the same along with the related code snippets are attached herewith. Steps involved were: Feature extraction, Data Partitioning, Training using Logistic Regression, Tuning Hyper Parameters, Testing the Machine Learning scheme on Testing set.

1. Introduction:

Logistic Regression is a Machine Learning algorithm that is used for the classification problems, and is based on the concept of probability. Logistic Regression uses a cost function, this cost function can be defined as the Sigmoid function or the logistic function. The cost function is delimited between 0 and 1. We use sigmoid to map predictions to probabilities, the function maps any real value into another value between 0 and 1. Hence, using Logistic regression we can train the model to predict and in turn classify a given data set into two (or more) major categories using the sigmoid function (In this case the two classes are Benign and Malignant).

The main difference between Logistic regression and Linear Regression is that Logistic Regression is used when the dependent variable is binary in nature. While Linear regression is used when the dependent variable is continuous and linear in nature.

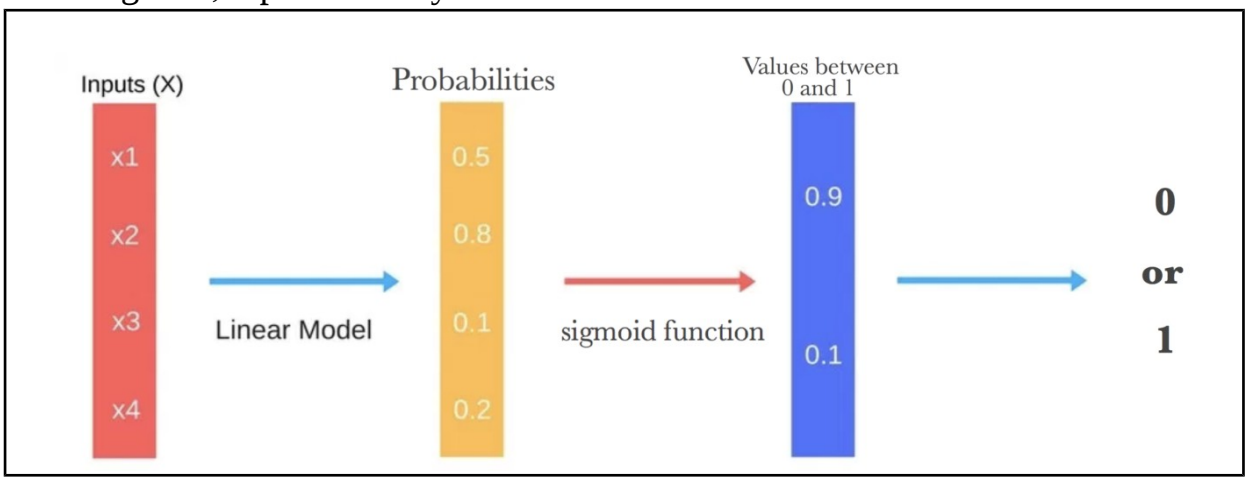




***Note that in this project's case, "Outpt: the object's position" means classifying the data into Benign or Malignant cases.

The above diagram(s) show how the Linear Regression Algorithm will function, the basic steps involved. The only difference in the standard approach shown above and my project implementation is the use of validation data set to figure out the hyper parameters that allow optimal performance.

The figure below explains how the logistic regression algorithm classifies the data into two categories, represented by 0 and 1.



2. Dataset definition:

Wisconsin Diagnostic Breast Cancer (WDBC) dataset is be used for training, validation and testing. The dataset contains 569 instances with 32 attributes (or features). The features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. The goal is for the model to learn how to predict which cases from the given unknown datasets (validation data and testing data) fall under Malignant or Benign categories. Some of the features from the dataset related to the FNA are: radius, texture, perimeter, area, smoothness, compactness, concavity, symmetry etc.

3. Dataset Partitioning:

Machine Learning deals with three partitions of a given data set:

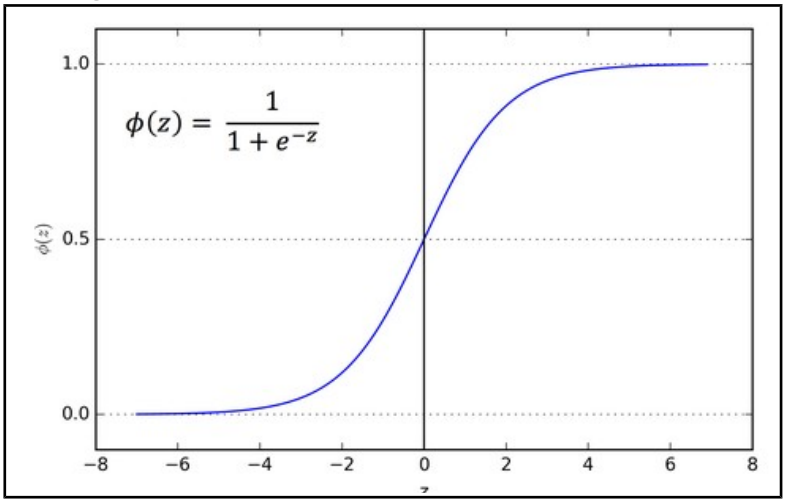
- **Training Data:** Used for learning and hence making predictions about the validation data and testing data. In my program, I have allocated 80% of the original WDBC dataset to the Training Data partition. (80% = 455 out of 569 rows of data)
- **Validation Data:** Used for tweaking the hyper parameters (hyper parameters are those parameters that vary and in turn make the predictions vary along with them, in this case – number of iterations and learning rate are used as the hyper parameters). Validation data gives us the estimation as to how well our model is performing and based on the performance we can enter the 'Testing' phase. In my program, I have allocated 10% of the original WDBC dataset to the Training Data partition. (10% = 57 out of 569 rows)
- **Testing Data:** Used for depicting how accurate our model is and how well it performs. Predictions made about the Testing Data show how well our model has been trained. I have allocated 10% of the original WDBC dataset to the Training Data partition. (10% = 57 out of 569 rows)

4. Training using Logistic Regression:

Training refers to the process of applying the Logistic Regression Algorithm to the Dataset to obtain a model that can efficiently predict whether a given case would fall into the Benign or Malign category based on the knowledge the model has gained so far from the dataset. 'Accuracy' is the key concept that needs to be focused on, and when the predictions are made for the testing data, the accuracy has to be as high as possible.

The Sigmoid function plays a crucial role in the functioning of the algorithm, it essentially maps the predicted values to probabilities.

The sigmoid function can be calculated as follows:



The algorithm used for implementing the core concept of Logistic regression is shown in the figure below:

jupyter main Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

3. Train using Logistic Regression:

In [4]:

```
#Transforming values for computation
X_train,Y_train=X_train.T,Y_train.reshape(1,Y_train.shape[0])
X_test,Y_test=X_test.T,Y_test.reshape(1,Y_test.shape[0])
X_validate,Y_validate=X_validate.T,Y_validate.reshape(1,Y_validate.shape[0])

def sigmoid(z): #Sigmoid Function
    return 1 / (1 + np.exp(-z))

def func2_regresses(X_value_matrix,epochs,learningrate):
    global w,b, losstrack
    losstrack = [] #Tracking the way the cost function drops in value
    m = X_train.shape[1] #Number of Training instances
    w = np.random.randn(X_train.shape[0], 1)*0.01 #Weight
    b = 0 #Bias

    for epoch in range(epochs):
        z = np.dot(w.T, X_train) + b
        p = sigmoid(z)
        cost = -np.sum(np.multiply(np.log(p), Y_train) + np.multiply((1 - Y_train), np.log(1 - p)))/m
        losstrack.append(cost)
        dz = p-Y_train
        dw = (1 / m) * np.dot(X_train, dz.T)
        db = (1 / m) * np.sum(dz)
        w = w - learningrate * dw
        b = b - learningrate * db

def func1_predicts(X_value_matrix):
    z = np.dot(w.T, X_value_matrix) + b
    p = sigmoid(z)
    p_new = (p>0.5).astype(int)
    return p_new

func2_regresses(X_train,epochs = 20000,learningrate=0.1)

predicted_Y_train=(func1_predicts(X_train))#Training
predicted_Y_validate=(func1_predicts(X_validate))#Predicting
predicted_Y_test=(func1_predicts(X_test))#Predicting
```

Another thing to be noted in the algorithm used is the dependency that the performance of the algorithm would have on the parameters - ‘epochs’ i.e. the number of iterations and ‘learningrate’ i.e. the Learning Rate, these are the ‘hyper parameters’ that will be discussed later and will be tweaked to obtain better performance/accuracy.

5. Tuning Hyper Parameters:

As described earlier, the hyper parameters this dataset is concerned with are ‘epochs’ i.e. the number of iterations and ‘learningrate’ i.e. the Learning Rate. One could clearly understand that, as the iterations and the learning rate keep growing, the accuracy too, shall grow. But it is not that simple, performing huge number of iterations is not at all feasible in datasets that have sizes in the range of terabytes, and the learning rate cannot go beyond a certain extent as well.

The values of my hyper parameters along the accuracy, precision, recall values, confusion matrix, and classification report associated with them is represented in detail below: ***(The data below is the output generated by my code in Jupyter Notebook submitted along with this report, Ref. Section #5 in the code)***

```
epoch = 20000 and learningrate = 0.1
Confusion Matrix - Training data set
[[262   7]
 [ 39 147]]
Train_Accuracy =  0.8989010989010989
Train_Precision =  0.9739776951672863
Train_Recall =  0.8704318936877077
Classification Report - Training data set
              precision    recall  f1-score   support

           0       0.87      0.97      0.92       269
           1       0.95      0.79      0.86       186

   micro avg       0.90      0.90      0.90       455
   macro avg       0.91      0.88      0.89       455
weighted avg       0.90      0.90      0.90       455
```

```
Confusion Matrix - Validation data set
[[42  1]
 [ 2 12]]
Validate_Accuracy =  0.9473684210526315
Validate_Precision =  0.9767441860465116
Validate_Recall =  0.9545454545454546
Classification Report - Validation data set
              precision    recall  f1-score   support

           0       0.95      0.98      0.97        43
           1       0.92      0.86      0.89        14

   micro avg       0.95      0.95      0.95        57
   macro avg       0.94      0.92      0.93        57
weighted avg       0.95      0.95      0.95        57
```

---

epochs = 35000 & learningrate = 0.05

```
Confusion Matrix - Training data set
[[263   6]
 [ 40 146]]
Train_Accuracy =  0.8989010989010989
Train_Precision =  0.9776951672862454
Train_Recall =  0.8679867986798679
Classification Report - Training data set
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       269
           1       0.96      0.78      0.86       186

   micro avg       0.90      0.90      0.90       455
   macro avg       0.91      0.88      0.89       455
weighted avg       0.91      0.90      0.90       455

Confusion Matrix - Validation data set
[[42  1]
 [ 2 12]]
Validate_Accuracy =  0.9473684210526315
Validate_Precision =  0.9767441860465116
Validate_Recall =  0.9545454545454546
Classification Report - Validation data set
              precision    recall  f1-score   support

           0       0.95      0.98      0.97        43
           1       0.92      0.86      0.89        14

   micro avg       0.95      0.95      0.95        57
   macro avg       0.94      0.92      0.93        57
weighted avg       0.95      0.95      0.95        57
```

---

epochs = 35000 & learningrate = 0.1

Confusion Matrix - Training data set

```
[[261   8]
 [ 35 151]]
```

Train_Accuracy = 0.9054945054945055

Train_Precision = 0.9702602230483272

Train_Recall = 0.8817567567567568

Classification Report - Training data set

	precision	recall	f1-score	support
0	0.88	0.97	0.92	269
1	0.95	0.81	0.88	186
micro avg	0.91	0.91	0.91	455
macro avg	0.92	0.89	0.90	455
weighted avg	0.91	0.91	0.90	455

Confusion Matrix - Validation data set

```
[[40  3]
 [ 2 12]]
```

Validate_Accuracy = 0.9122807017543859

Validate_Precision = 0.9302325581395349

Validate_Recall = 0.9523809523809523

Classification Report - Validation data set

	precision	recall	f1-score	support
0	0.95	0.93	0.94	43
1	0.80	0.86	0.83	14
micro avg	0.91	0.91	0.91	57
macro avg	0.88	0.89	0.88	57
weighted avg	0.91	0.91	0.91	57

---

epochs = 40000 & learningrate = 0.05

Confusion Matrix - Training data set

```
[[262   7]
 [ 39 147]]
```

Train_Accuracy = 0.8989010989010989

Train_Precision = 0.9739776951672863

Train_Recall = 0.8704318936877077

Classification Report - Training data set

	precision	recall	f1-score	support
0	0.87	0.97	0.92	269
1	0.95	0.79	0.86	186
micro avg	0.90	0.90	0.90	455
macro avg	0.91	0.88	0.89	455
weighted avg	0.90	0.90	0.90	455

```
Confusion Matrix - Validation data set
[[42  1]
 [ 2 12]]
Validate_Accuracy =  0.9473684210526315
Validate_Precision =  0.9767441860465116
Validate_Recall =  0.9545454545454546
Classification Report - Validation data set
              precision    recall  f1-score   support

           0       0.95      0.98      0.97        43
           1       0.92      0.86      0.89        14

   micro avg       0.95      0.95      0.95        57
   macro avg       0.94      0.92      0.93        57
weighted avg       0.95      0.95      0.95        57
```

---

epochs = 40000 & learningrate = 0.1

```
Confusion Matrix - Training data set
[[261   8]
 [ 34 152]]
Train_Accuracy =  0.9076923076923077
Train_Precision =  0.9702602230483272
Train_Recall =  0.8847457627118644
Classification Report - Training data set
              precision    recall  f1-score   support

           0       0.88      0.97      0.93       269
           1       0.95      0.82      0.88       186

   micro avg       0.91      0.91      0.91       455
   macro avg       0.92      0.89      0.90       455
weighted avg       0.91      0.91      0.91       455

Confusion Matrix - Validation data set
[[40  3]
 [ 2 12]]
Validate_Accuracy =  0.9122807017543859
Validate_Precision =  0.9302325581395349
Validate_Recall =  0.9523809523809523
Classification Report - Validation data set
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        43
           1       0.80      0.86      0.83        14

   micro avg       0.91      0.91      0.91        57
   macro avg       0.88      0.89      0.88        57
weighted avg       0.91      0.91      0.91        57
```

---

6. Testing the machine learning scheme on the testing set:

The whole point of creating a ‘Testing Data’ partition is to have an ‘un-tampered’ data set on which one could apply the model to predict the classification outcome, and then check whether the predicted values match with the actual values. This comparison between the predicted values and the actual values give us the accuracy of the model.

To reach the ‘best’ and most accurate predictions on the Testing data, one has to fix the hyper parameters that lead us to those best predictions.

As one can easily deduce, the **‘best’ hyper parameters**, **i.e. the ones that give the most accurate *(reasonably and believably accurate)* result is** :

```
BEST CASE SCENARIO:
epochs = 40000 & learningrate = 0.1
```

```
Confusion Matrix - Training data set
[[261   8]
 [ 34 152]]
Train_Accuracy =  0.9076923076923077
Train_Precision =  0.9702602230483272
Train_Recall =  0.8847457627118644
Classification Report - Training data set
              precision    recall  f1-score   support

           0       0.88      0.97      0.93       269
           1       0.95      0.82      0.88       186

   micro avg       0.91      0.91      0.91       455
   macro avg       0.92      0.89      0.90       455
weighted avg       0.91      0.91      0.91       455
```

```
Confusion Matrix - Testing Data set
[[42  3]
 [ 4  8]]
Test_Accuracy =  0.8771929824561403
Test_Precision =  0.9333333333333333
Test_Recall =  0.9130434782608695
Classification Report - Testing Data set
              precision    recall  f1-score   support

           0       0.91      0.93      0.92        45
           1       0.73      0.67      0.70        12

   micro avg       0.88      0.88      0.88        57
   macro avg       0.82      0.80      0.81        57
weighted avg       0.87      0.88      0.88        57
```

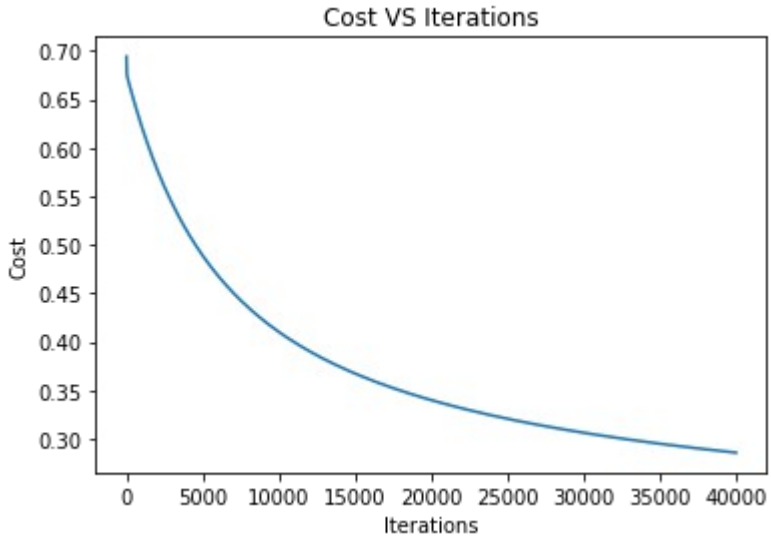
```
Confusion Matrix - Validation data set
[[40  3]
 [ 2 12]]
Validate_Accuracy =  0.9122807017543859
Validate_Precision =  0.9302325581395349
Validate_Recall =  0.9523809523809523
Classification Report - Validation data set
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        43
           1       0.80      0.86      0.83        14

   micro avg       0.91      0.91      0.91        57
   macro avg       0.88      0.89      0.88        57
weighted avg       0.91      0.91      0.91        57
```

Along the process of reaching the 'best' accuracy for predicted values, I plotted a couple of 'Iteration VS Cost' Plots to show the effective functioning of my Learning model that iterated across the *'func2_regresses'* function and achieved a reduced value of **'cost'** with each iteration. **'Cost'** can also be called **'Loss'** and it is a parameter of the Logistic. Regression that must reduce with every iteration to indicate the proper functioning of the algorithm.

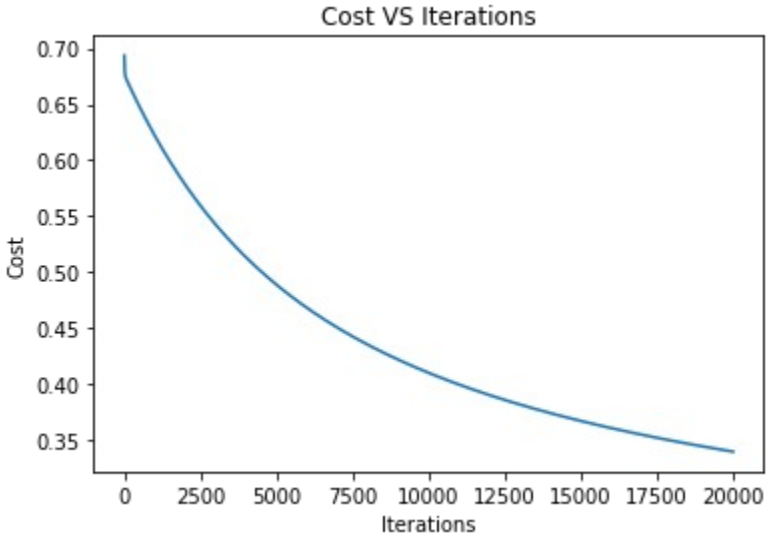
Logistic regression cost function

For logistic regression, the Cost function is defined as:

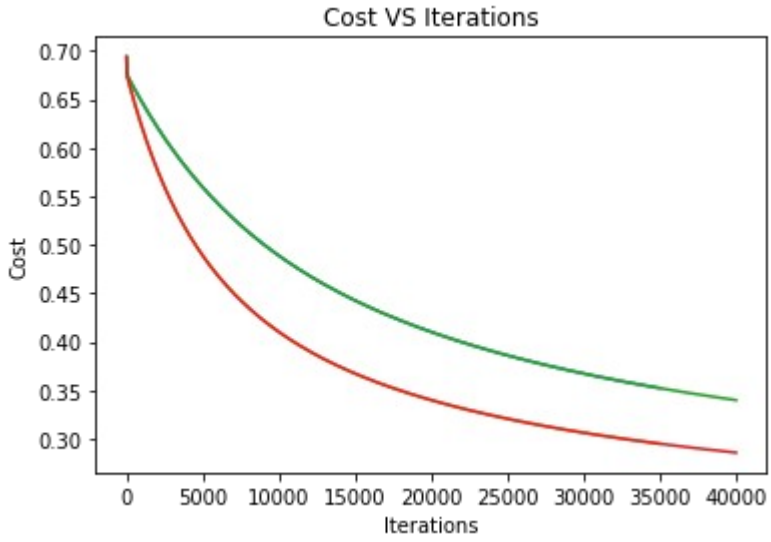
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if y} = 1 \\ -\log(1 - h_\theta(x)) & \text{if y} = 0 \end{cases}$$



BEST CASE SCENARIO:
epochs = 40000
learningrate = 0.1



epoch=20000
learningrate=0.1



for tuples in:
epoch = [35000,40000]
learningrate = [0.05,0.1]

7. Conclusion:

In this project I performed the following steps: Feature extraction, Data Partitioning, Training using Logistic Regression, Tuning Hyper Parameters, Testing the Machine Learning scheme on Testing set. This project has led to a better understanding of Logistic Regression as an algorithm to perform classification on a broad dataset such as the WDBC dataset. The final predicted values with optimal hyper parameters are - Train_Accuracy = 0.9076923076923077 , Test_Accuracy = 0.8771929824561403 , Validate_Accuracy = 0.9122807017543859.

8. References:

[1] https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148
[2] https://machinelearningmastery.com/logistic-regression-for-machine-learning/
[3] https://medium.com/@martinpella/logistic-regression-from-scratch-in-python-124c5636b8ac
[4] https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#sigmoid-activation
[5] https://www.internalpointers.com/post/cost-function-logistic-regression
[6] https://www.semanticscholar.org/paper/Logistic-regression-based-device-free-localization-Lei-Lv/45cd2c6863530f150522a1bf81f51e5d19db93cd/figure/1