# Machine Learning(CS60050) Mini Project-2


**Project Code - SNRG**

**Project Title - sine-Curve Fitting using Standard Linear and Logistic Regression**

**Group Number – 23**

**Roll Numbers – 20CS91F02 (Prasanta Dutta)**

**20CS91R02 (Abhishek Kumar)**

**20CS91P02 (Deepak Mewada)**


*Note : Here we have discussed problem descriptions in brief and shown corresponding outputs. For more details kindly refer to our program file (.ipynb).*

# TASK 1: Synthetic Data Generation and Curve Fitting

**Task (1.a)** was to generate a synthetic dataset **[X,Y]** and to add noise in it**.** The following plot shows the generated data.

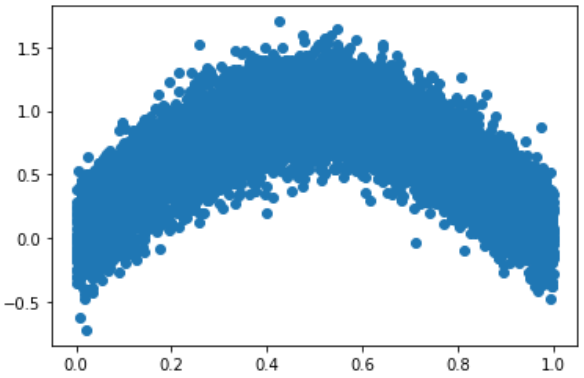


**Figure 1**

**Task (1.b)** is to plot the Histogram of **500** points of **X** which is shown in the figure below-

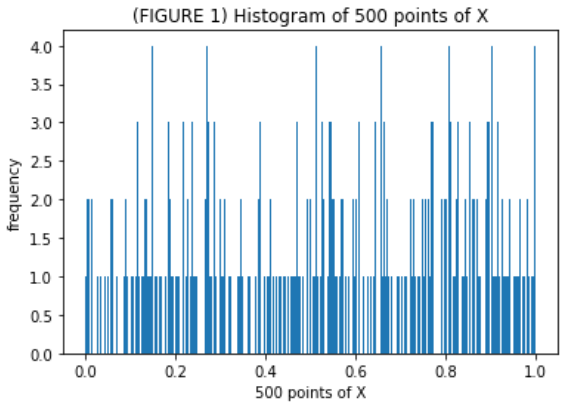


**Figure 2**

**Task (1.c)** is to split the dataset into **train (75%)** and **test (25%)** dataset

*Gradient-Descent (GD)*

*The main attraction of this program is Gradient-Descent. So we have given it special attention. As our team worked in parallel. Hence we have implemented two different GDs. One for Linear Regression & another for logistic regression.*

***Here by GD, we are denoting GD for linear regression only.***

*This GD can work as Batch GD , Mini-batch GD and Stochastic GD based on batch size. GD function takes X & Y and returns computed weights and the train RMSE. Though inside this function we have some tuning parameters which will be discussed below -*

- *LR : learning rate for GD. LR may be fixed or can vary per epoch based on the value of LR_dividend.*
- *LR_dividend : LR_dividend=1 => static LR and LR_dividend ≠ 1 => dynamic LR.*
- *no_epochs : Max. number of iteration allowed*
- *min_allowed_error : Precision of accuracy by the algorithm*
- *batch_size : The length of data to be processed per epoch. batch_size = 1 => Stochastic GD, batch_size = complete datasize => Batch gradient descent and batch_size = (1 < x < complete datasize) => Mini-batch gradient descent.*
- *w0 : Initial intercept*
- *wi, where i ≠ 0 : Slopes*

**Task (1.d)** is to find, train and test **RMSE** errors based on **Gradient-Descent**, which are as follows -

***Train RMSE error : 0.2029295975400735***

***Test RMSE error : 0.20740536068813914***

**Task (1.e)** is to plot the actual points and the predicted points. For that we have attached two figures below (on **train data** and on **test data**).

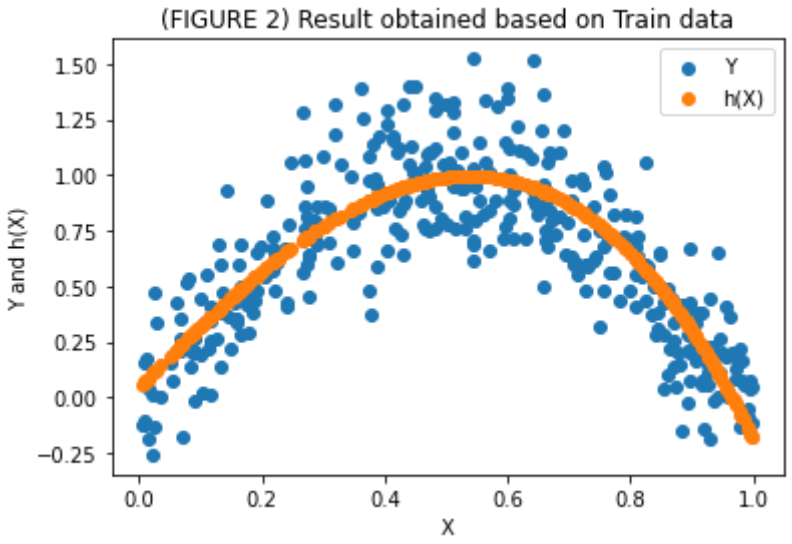


**Figure 3**

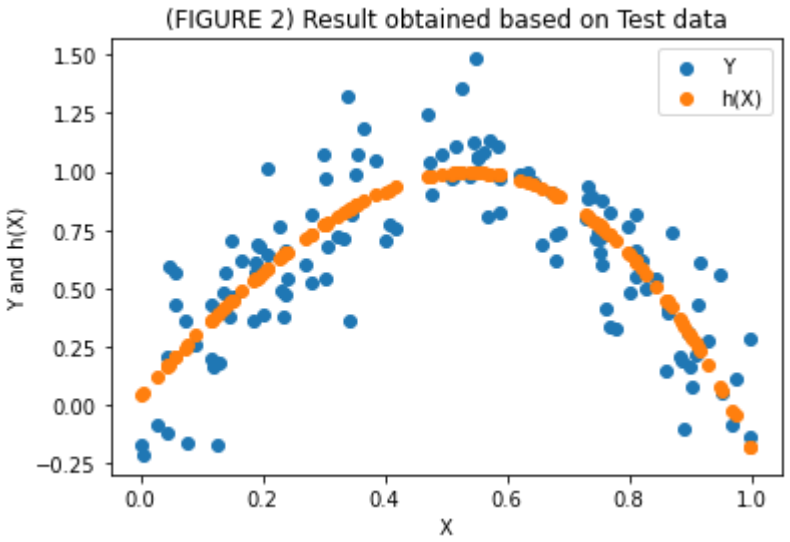


**Figure 4**

# TASK 2: Variation of Loss/Error with m

Now task 2 is all about checking the variation in error with respect to different sample sizes,**m**.

**Task (2.a), (2.b)** & **(2.c)** are some basic work, which can be easily seen in the python notebook attached.

**Task (2.d)** is to plot the RMSE values with respect to different sample sizes, the results are shown below -

***Variation of RMSE with m***

***----------***

***10 0.23760540365917276***

***20 0.19199270796176435***

***50 0.19431960586556768***

***100 0.18784171187046209***

***200 0.20516217833884792***

***500 0.20404362999021475***

**Task (2.e)** Figure 5 represents the plot of **RMSE** vs **m**

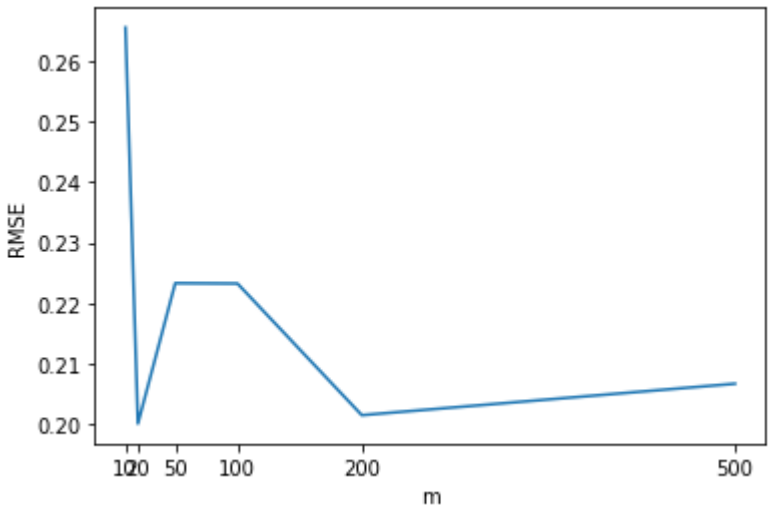


**Figure 5: Variation of RMSE w.r.t to m**

**Justification for variation in error:** As initially sample size was small, our predicted curve didn't find out the actual distribution of data, hence couldn't fit it properly.Thus we can see some fluctuation of RMSE. Gradually as the sample size increases, our predicted curve gets the actual distribution of data, hence fit it properly. Thus the error stabilized.

# TASK 3: Logistic Regression

Here we are given two input columns X1 and X2, and output column as label Y, on which we need to perform Logistic regression and perform Gradient Descent using an appropriate cost function.

As given, the sigmoid function is -

$$h(x) = sigmoid(Z) = \frac{1}{(1 + e^{-Z})}$$

where Z = $w_0 + w_1 x_1 + + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2$

The loss function can hence be calculated as

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^{m} [y_i \cdot log\hat{y}_i + (1 - y_i) \cdot log(1 - \hat{y}_i)]$$

Here we used the cross-entropy cost function.

Here we plot the data points so to visualise the distribution to get an idea about decision boundary, which can be seen below in figure 6.

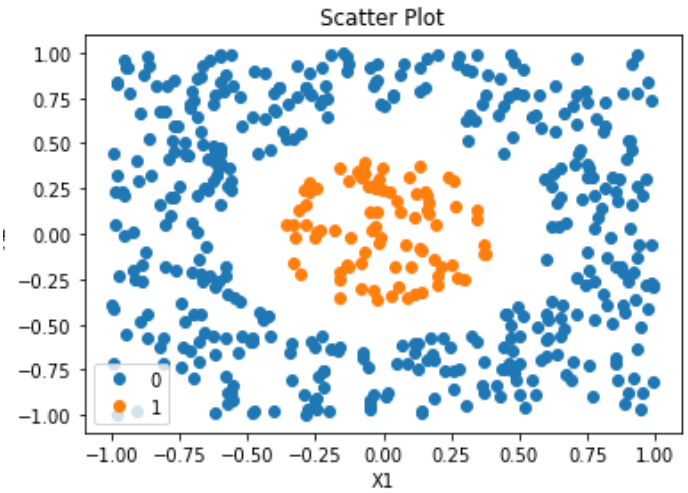


**Figure 6**

After performing the logistic regression, we got trained values of weight parameters.

3(a) Now we use those trained values of weight to predict values of Y's & we found **Test Accuracy** and **Train accuracy**. Which are as follows:

```
Accuracy of Train Data =          100.00 %

Accuracy of Test Data =          100.00 %
```

We performed **parameter tunning** to get the best possible accuracy, here best possible accuracy we got is 100% for both datasets. This accuracy we achieved by keeping the parameter like *learning rate*, *num of iteration* at their optimal value which we got after performing **parameter tunning**. These are as follows:

```
learning_rate=0.8

num_iterations=500
```

We also plotted the following graphs to enhance the ease of understanding of gradient descent working :

(i) Iteration vs cost

(ii) Iteration vs Accuracy

these can be seen in Figure 7 and Figure 8

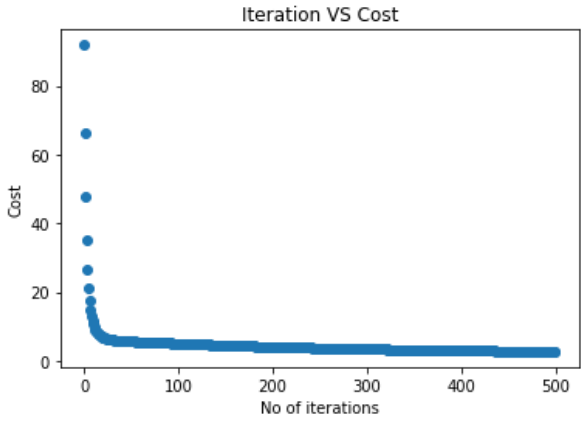


**Figure 7**

**Plot obtained for cost against different values of number of Iterations of GD**

**Interpretation of plot** : Here we can see that the cost is very high initially but as the number of iterations of GD increases the cost value decreases significantly. This happens because we are performing Gradient Descent. And in every new iteration the gradient descent will minimise cost according to learning rate. And finally we will save the Weight value which give us the lowest cost.

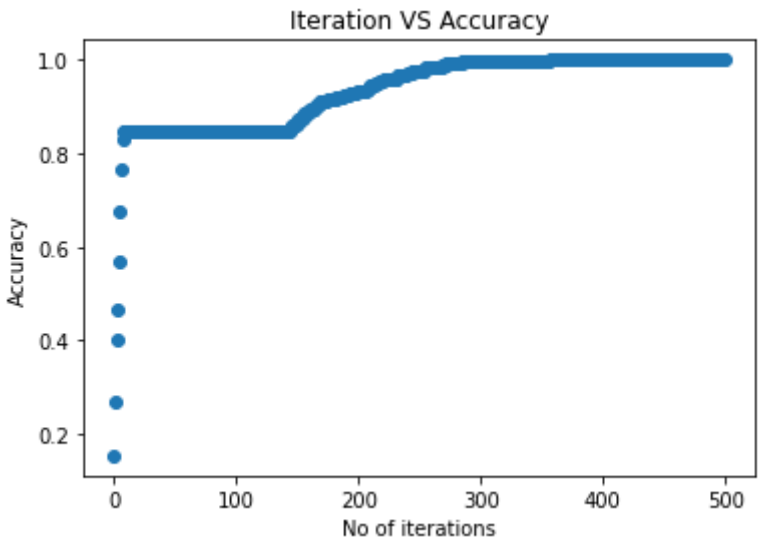


**Figure 8**

**PLot for Accuracy of Train data against Number of Iterations.**

**Interpretation of Plot :** Here we can see that the accuracy is lowest initially but as we increase the number of iterations of Gradient Descent the accuracy suits up to high values. It happens because as the iteration increases, gradients move towards a lower value of cost functions so the cost decreases and hence the accuracy increases.

---

For **(Task 3.b)** we need to plot Y vs X by plotting the points of X1, X2 and use Y values to show points in different colours based on value of Y.

Now on the same figure we plotted the points of curve Z vs X to show how well curve Z divides the points of X (FIGURE 4)

**Figure 9** shows the decision boundary for whole data.

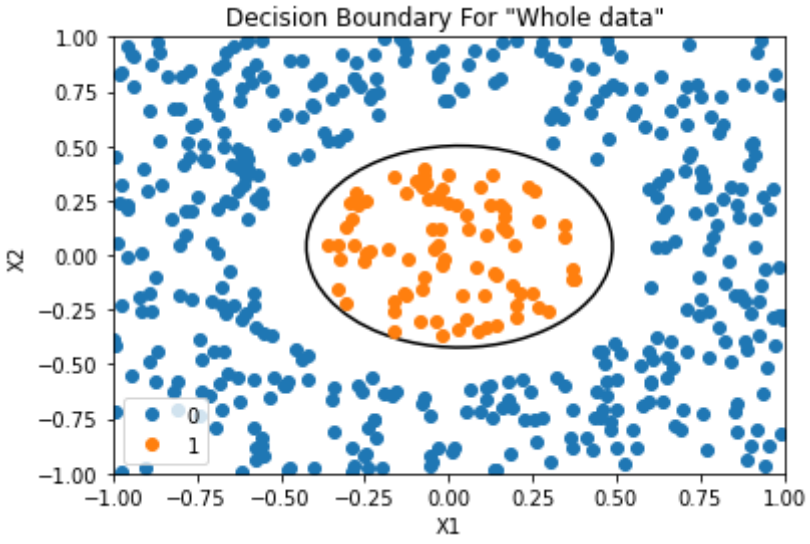


**Figure 9: Decision Boundary for whole dataset**

Figure 10 and Figure 11 shows the decision boundary on Train and Test data respectively.

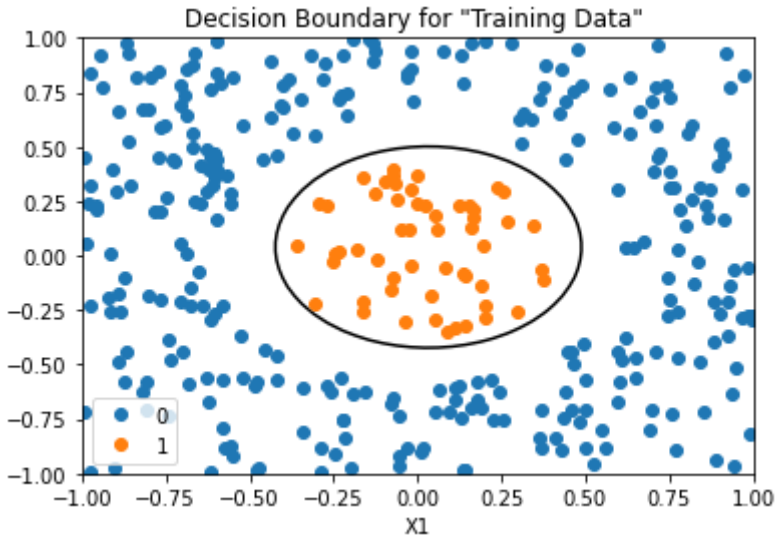


**Figure 10: Train Data**

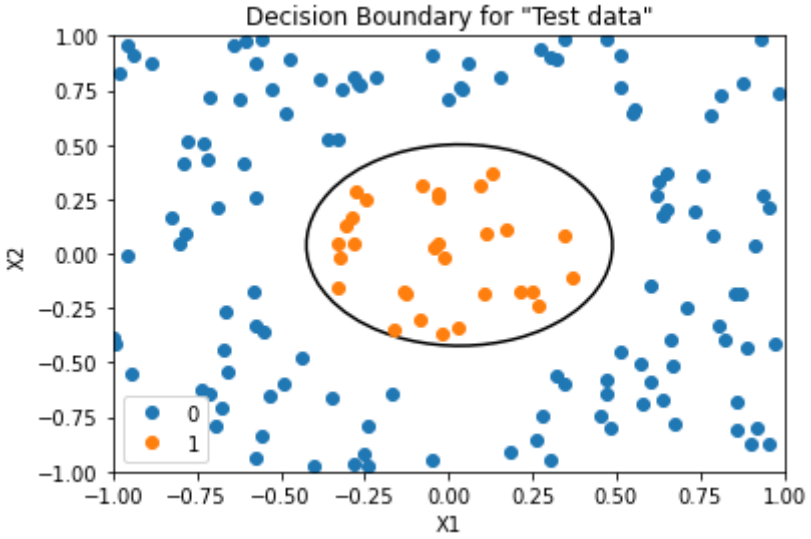


**Figure 11: Test Data**

**Decision Boundary for Test Data**

---

**That's it from Team-23.**

**Thank You :)**

---