Logistic Regression

Soumita Das
University at Buffalo
UBIT Name: soumitad
UB Person Number: 50320170
soumitad@buffalo.edu

Abstract

One of the most common classification algorithm is regression. Regression can be linear or multivariate depending on the number of features available. Regression maps the input vector to an output using a polynomial or basis function. Logistic regression is useful in determining the class to which the input vector belongs. It is called binary logistic regression when the number of classes is dichotomous(binary). In this report binary logistic regression algorithm is employed to determine the malignancy status of the patients in the Wisconsin Diagnostic Breast Cancer (WDBC) dataset.

1 Introduction

Machine learning problems can be of two types, *supervised* and *unsupervised*. Classification that depends on the attributes and target variables of the training set, and then go on to predict class labels for unknown test samples is a category of *supervised* learning. On the other hand clustering techniques which are *unsupervised* learning groups input samples based on the homogeneity among their features.

Regression is a type of classification algorithm or supervised learning. Linear regression predicts continuous values while logistic regression predicts only the outcome. Logistic regression requires training to tune its parameters(weights) and then predicts the outcome of the test cases using a sigmoid activation function. It may be of three categories: *Binary*, where the input vector either belongs to class 0 or 1, *Multinomial*, where the input vector may belong to more than two classes, and *Ordinate* where multiple categories to which the input vector belongs is ordered. The algorithm is discussed in detail in the following section.

2 Algorithm

For a given data set with n samples and d features, let $X_i = [x_1, x_2, ...x_d]$ denote each input sample where i ranges from $1ton$. A set of parameters(or weights) including the bias $w = [w_0, w_1, w_2....w_d]$ is tuned based on the input training set. For a test sample the hypothesis is calculated as $h(X) = w_0 + w_1x_1 + w_2x_2....w_dx_d$. The hypothesis is then fed into the sigmoid activation function which returns a probability value between 0 to 1. Based on the value of the sigmoid function, the test sample is classified using a decision boundary(here 0.5).
Figure 1 shows the neural architecture of logistic regression, the activation function is sigmoid.

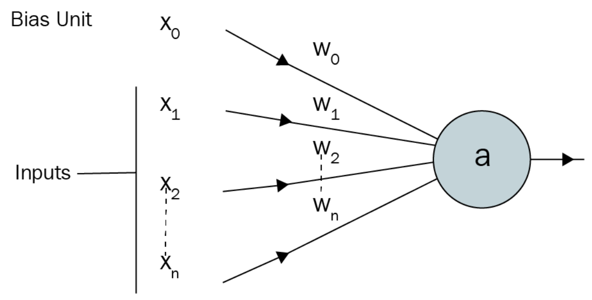

Figure 1: Neural architecture of logistic regression

2.1 The Sigmoid Activation Function

Since the hypothesis may take an infinite range of values, the sigmoid function helps to classify the test data and set a decision boundary. The sigmoid activation function used in logistic regression is:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{1}$$

$\sigma(z)$ is always bounded between 0 to 1. The following figure shows a graph of the function.

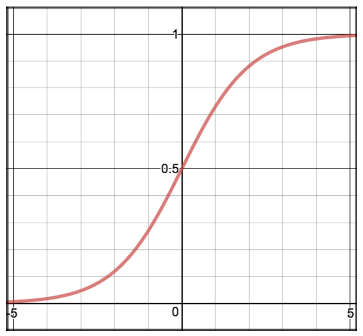

Figure 2: Plot of Sigmoid Activation Function

This helps us to create a decision boundary. If the output of the sigmoid function is greater than 0.5 the test sample is assigned to class 1, and if the output is less than 0.5 the test sample is assigned to class 0.

2.2 Loss Function

Instead of the Squared Error here we use the Cross Entropy for error calculation. Let the output from the activation function be a_i and the actual class label of a sample be y_i for the i_{th} sample in the training data then the Loss function will be calculated as :

$$L = \frac{\sum_{i=1}^{m} -(y_i log(a_i) + (1-y_i) log(1-a_i))}{m} \tag{2}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}(y_i log(\sigma(w^T X_i + b)) + (1-y_i) log(1-\sigma(w^T X_i + b))) \tag{3}$$

where b is the bias. Now let $w^T X_i + b$ be z_i, then the equation reduces to:

$$= -\frac{1}{m}\sum_{i=1}^{m}(y_i log(\sigma(z_i)) + (1-y_i) log(1-\sigma(z_i))) \tag{4}$$

where m is the number of training samples.

2.3 Gradient Descent

The weights and the bias are updated based on the gradient descent. The following formula is used to update the weights:

$$w_{new} = w_{old} - \eta \Delta w_{old} \tag{5}$$

where η is the learning rate
The aim is to minimize the loss function. Using gradient descent we aim to find the the global minimum of the loss function. η or the learning rate must be taken care of. If taken too large it may oscillate over the global minimum, while an extremely small value may result in getting stuck at local minimum.
Now, here the weights are updated for each training sample, thus the Loss Function L reduces to:

$$L = -(y_i log(\sigma(z_i)) + (1 - y_i) log(1 - \sigma(z_i))) \tag{6}$$

So,

$$\begin{aligned}
\Delta w_i = \frac{\partial L}{\partial w_i} &= -\frac{\partial}{\partial w_i}(y_i log(\sigma(z_i)) + (1 - y_i) log(1 - \sigma(z_i))) \\
&= -(y_i \frac{1}{\sigma(z_i)} \frac{\partial}{\partial w_i} \sigma(z_i) + (1 - y_i) \frac{1}{1 - \sigma(z_i)} \frac{\partial}{\partial w_i}(1 - \sigma(z_i))) \\
&= -y_i \frac{1}{\sigma(z_i)} \sigma(z_i)(1 - \sigma(z_i) \frac{\partial}{\partial w_i}(z_i) + \\
&(1 - y_i) \frac{1}{1 - \sigma(z_i)} \sigma(z_i)(1 - \sigma(z_i)) \frac{\partial}{\partial w_i}(-(z_i)) \\
&= -(y_i(1 - \sigma(z_i))x_i + (1 - y_i)\sigma(z_i)(-x_i)) \\
&= -(y_i - y_i\sigma(z_i) - \sigma(z_i) + y_i\sigma(z_i))x_i \\
&= -(y_i - \sigma(z_i))x_i
\end{aligned} \tag{7}$$

Thus for the i_{th} sample X_i, the weights are modified as:

$$w_{new} = w_{old} - \eta(\sigma(z_i) - y_i)x_i \tag{8}$$

2.4 Pseudo code

```
Step 1. Load the data set.($n$ samples and $d$ features)
Step 2. Normalize the data. (Pre-processing)
Step 3. Split the data into Training(80%), Testing(10%) and Validation(10%)
Step 4. Assign $w = [w_0, w_1, ...w_d]$ randomly.
Step 5. Start training the Logistic Regression Model.
        while(epochs>0):
            for all the $m$ samples in the Training set:
                Append [1] to input sample $X = [x_1, x_2...x_d]$ to accommodate the bias ($w_0$)
                Calculate the dot product $w^T X$
                Map $w^T X$ between 0 to 1 using the sigmoid activation function.
                Adjust parameters($w$) using Gradient Descent.
            Calculate Training and Validation Error
Step 6. Test the model on the Testing data.
```

3 Experimental Setup

3.1 Data set

Wisconsin Diagnostic Breast Cancer (WDBC) dataset will be used for training, validation and testing. The dataset contains 569 instances with 32 attributes. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image in the 3-dimensional space. Features are computed from the following major 10 features:
1. radius (mean of distances from center to points on the perimeter)

2. texture (standard deviation of gray-scale values)
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter2/area 1.0)
7. concavity (severity of concave portions of the contour)
8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension

3.2 Pre-processing

As a part of pre-processing, we first normalise the data set. Since each feature has different range of values, normalisation is required, else some features may have more weightage on the outcome(prediction) of the input sample even though they should be treated equal. Using the following normalisation equation all the features in the data set are mapped to a range of 0 to 1.

$$norm\_featurevalue = \frac{org\_feauturevalue - min_{feauture}}{max_{feature} - min_{feature}} \tag{9}$$

where $org\_feauturevalue$ is the un-normalised feature value, and the $max_{feature}$ and $min_{feature}$ are the maximum and the minimum of the corresponding feature list.
This process is repeated for all the 30 features in the dataset excluding(ID and Diagnosis(M/B)).
Following this, we split the data set randomly into 80% training data, 10% testing data and 10% validation data.

3.3 Evaluation Metrics

To calculate the effectiveness of the algorithm, we use accuracy, precision and recall.

$$Accuracy = \frac{t_p + t_n}{t_p + f_p + t_n + f_n} \tag{10}$$

$$Precision = \frac{t_p}{t_p + f_p} \tag{11}$$

$$Recall = \frac{t_p}{t_p + f_n} \tag{12}$$

t_p or true positive is the number of times the algorithm could correctly guess the Malignancy status. t_n or true negative is the number of times the algorithm could correctly guess the Non-Malignant or Benign status. f_n or false negative is the number of times the algorithm could not correctly guess the Malignancy status. f_p or false positive is the number of times the algorithm wrongly assigned a Malignancy status.

3.4 Results and Discussions

To verify the effectiveness of the algorithm we use the evaluation metrics discussed in the previous Section. The number of epochs have been varied and the accuracy, precision and recall has been observed as in Figures 3,4 and 5 respectively. For a small batch of 100 epochs it is observed that the accuracy,precision and recall values are less than that of a batch of 1000 epochs.

For the next part we vary the validation error and the training error for every epoch. Both validation and training error are calculated as cross entropy error. As expected, both the values are quite high for the initial few epochs and gradually decreases to 0 as the parameters are tuned. In Figures 6 this phenomenon is observed for each of the 500 epochs.

As we know that the learning rate η has a major impact on finding the global minimum of the cost function. The value of η is varied from a very small value of 0.001 to a maximum value of 5. The nature of the graphs in Figure 7 to Figure 13 show how the validation and the training errors vary with respect to epochs. We can note that for a very small learning rate the curve is not skewed at all. While for high learning rate, for this case above 0.5 that validation error starts increasing sharply. Depending on the plots we choose the optimum value of η. In our case the η has been set to 0.01.

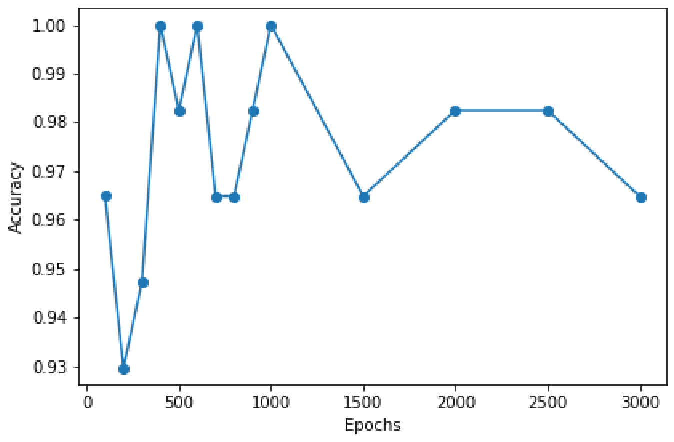

Figure 3: Accuracy vs Number of Epochs

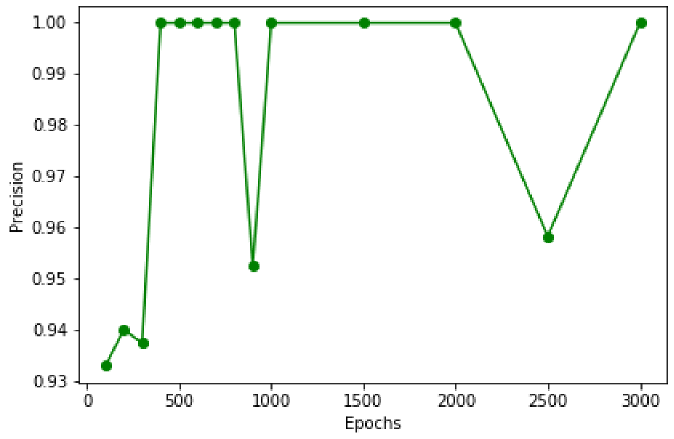

Figure 4: Precision vs Number of Epochs

4 Conclusion and Future Work

In this project Logistic Regression was implemented using Python. It was a Binary Logistic Regression, since we had to classify the input samples to Malignant(M) or Benign(B) status. Cross entropy error was used as the cost function. Extensive experiments on the Wisconsin Diagnostic Breast Cancer (WDBC) dataset show how the prediction accuracy increases with the number of epochs. Results also show as the random initialized parameters(weights) are tuned, the training and validation loss decreases significantly.
In future, I plan to investigate the attribute importance and perform dimension reduction accordingly. Also, I plan to investigate overfitting of the model, and penalize the weights accordingly using regularization.

5 References

[1] Pattern Recognition and Machine Learning by Christopher M. Bishop
[2]https://www.openml.org/d/1510 Author: William H. Wolberg, W. Nick Street, Olvi L. UCI Breast Cancer Wisconsin (Diagnostic) Data Set (WDBC)
[3] Logistic Regression by Andrew Ng at http://cs229.stanford.edu/notes/cs229-notes1.pdf

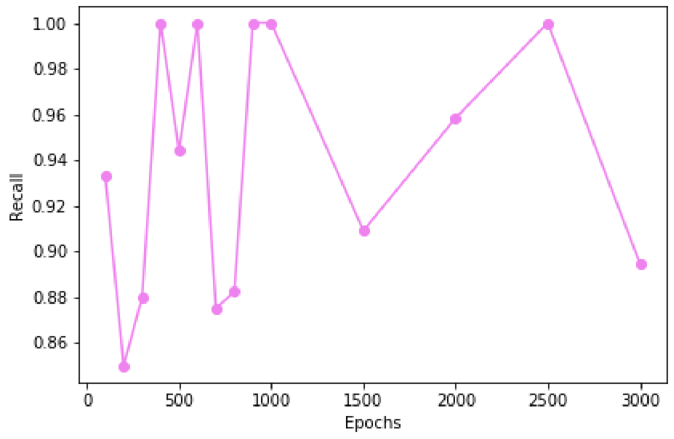

Figure 5: Recall vs Number of Epochs

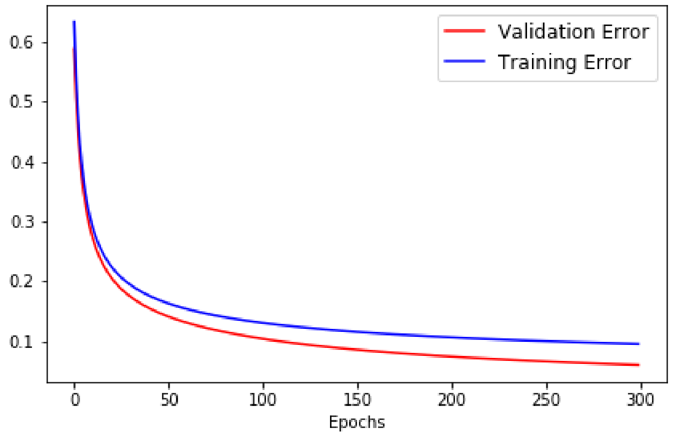

Figure 6: Plot of Training and validation Loss for 500 Epochs

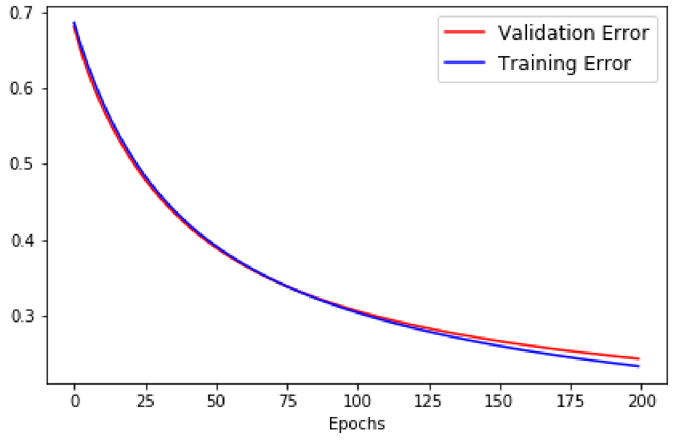

Figure 7: Training and validation Loss for $\eta = 0.001$

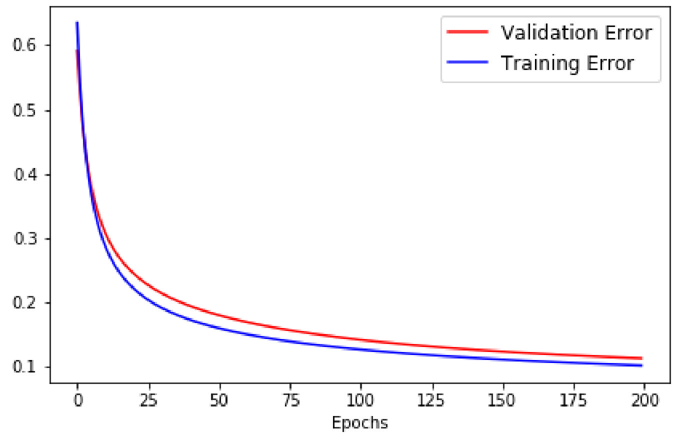

Figure 8: Training and validation Loss for $\eta = 0.01$

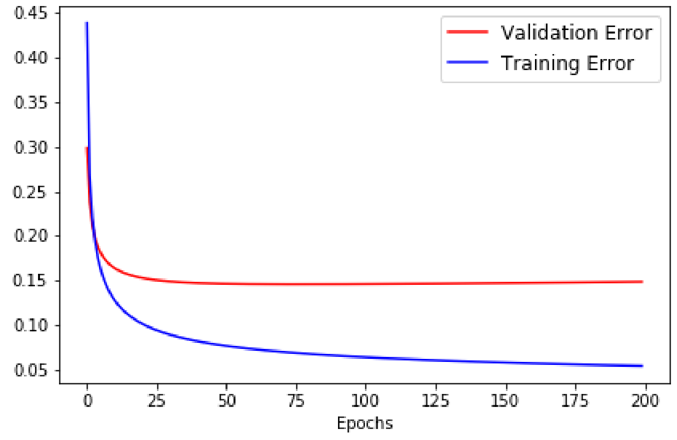

Figure 9: Training and validation Loss for $\eta = 0.1$

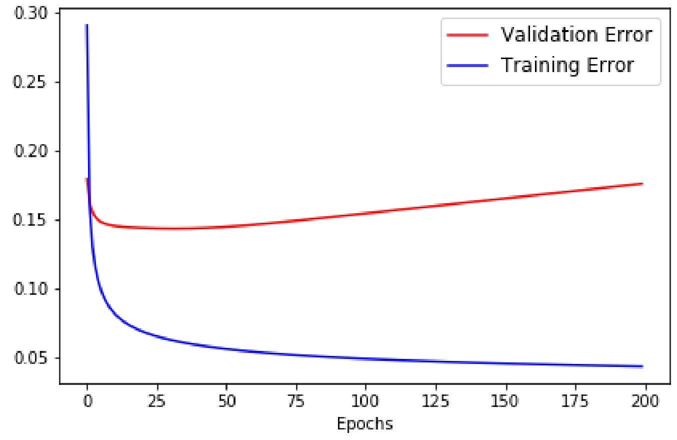

Figure 10: Training and validation Loss for $\eta = 0.5$

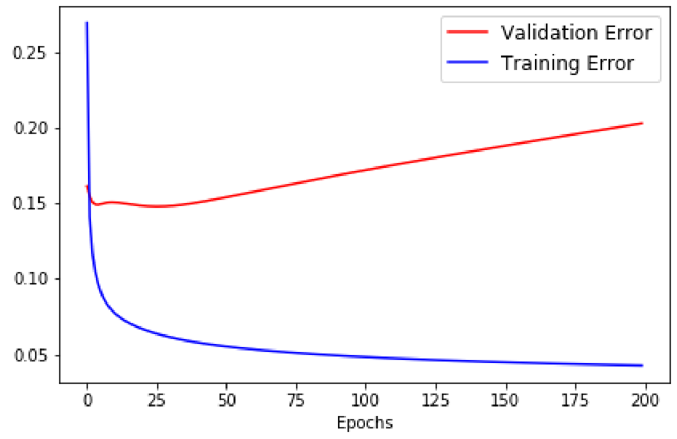

Figure 11: Training and validation Loss for $\eta = 1$

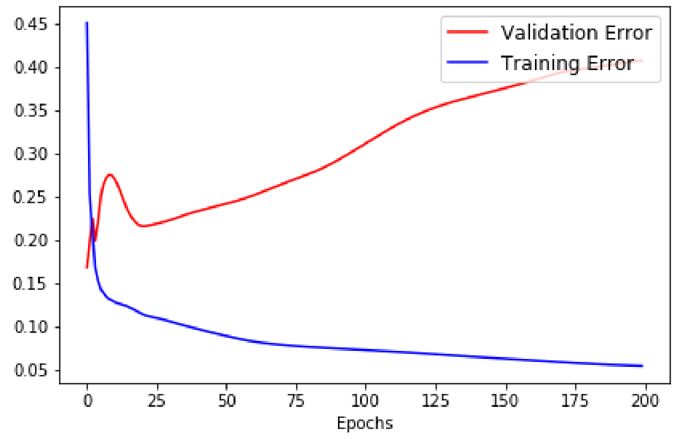

Figure 12: Training and validation Loss for $\eta = 3$

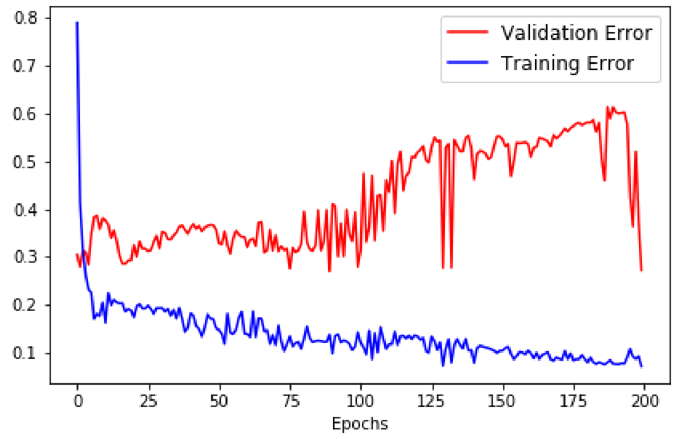

Figure 13: Training and validation Loss for $\eta = 5$