Logistic Regression using Python

Priya Rao
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
prao4@buffalo.edu

Abstract

This paper describes a two-class classification performed by means of logistic regression over the Wisconsin Diagnostic Breast Cancer (WDBC) dataset which consists of 569 instances and 32 attributes. As part of these findings, the effects on the loss function and accuracy upon manipulating the hyperparameters and applying different techniques of normalization and/or standardization have also been compared.

1 Introduction

The problem of two class classification or binary classification albeit a seemingly trivial one has constantly been at the crux of machine learning – to classify if a document is fraudulent or not, to approve a candidate's loan request or not, etc. Several methods are used to address the classification problem including that of Decision Trees, Random Forests, Bayesian Networks, SVMs, etc. This report covers two-class classification using logistic regression.

1.1 Forward Pass

The method of logistic regression uses a logistic function to model a binary dependent variable. Although linear regression is also a binary classifier, it predicts continuous values unlike logistic regression which predicts discrete values. This is because the logistic regression model uses a sigmoid function which produces probabilistic scores that help in classifying the data as 1 or 0. The sigmoid function (Equation (2) in the image below) squashes the data giving an output between values 0 and 1.

The sigmoid function takes a hypothesis function as the input parameter which is calculated using Equation (1). Here Θ is a vector of weights of dimensionality 1 X M where M is the total number of feature vectors (in this case, M = 30 since we have 30 feature vectors), X is the dataset in consideration and b is the bias which is a scalar value. The loss function (Equation (3)) in turn is a summation of cross entropy cost function for both y = 0 and y = 1. Taking log for both cost functions assists during the calculation of the gradient. The above equations combined can demonstrate the loss at each epoch and hence comprise of the forward pass which is performed for both the training and the validation dataset.

1.1 Forward pass equations (applied on both training and validation sets):

$$z = \theta^T X + b \tag{1}$$

$$a = \frac{1}{1 + e^{-z}} \tag{2}$$

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)}) \tag{3}$$

1.2 Backward Pass

In order to update the model, the weights and the bias need to be updated as well. This is done using gradient descent. Gradient descent allows for the losses to be minimized by converging to the local minimum based on the learning rate. This is done by finding the partial derivative of the loss function. The formula for gradient descent is given by equations (6) and (7), first for the weights and second for the bias. The weights and bias are updated accordingly (equations (4) and (5)) and the process of the forward pass is repeated for the next epoch.

1.2 Backward pass equations (using training data):

$$\theta_i = \theta_i - \eta \frac{\delta}{\delta\theta} J(\theta) \tag{4}$$

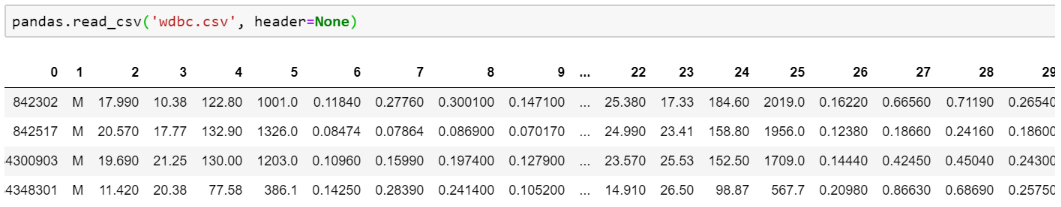
$$b = b - \eta \frac{\delta}{\delta b} J(\theta) \tag{5}$$

$$\frac{\delta}{\delta\theta} J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} (y^{(i)} - a^{(i)}) X_i \tag{6}$$

$$\frac{\delta}{\delta b} J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} (y^{(i)} - a^{(i)}) \tag{7}$$

2 Dataset Definition

The dataset used for this experiment was the Wisconsin Diagnostic Breast Cancer (WDBC) dataset which consisted of 569 instances and 32 attributes as seen below:

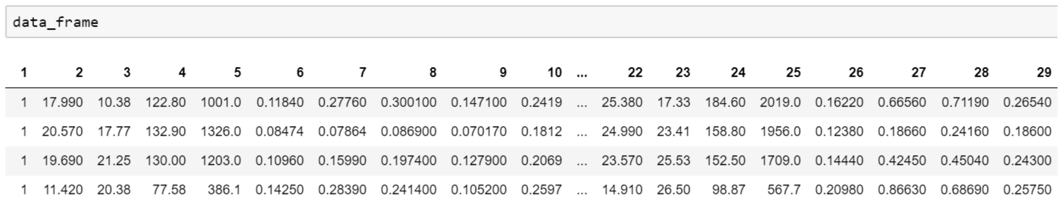


```
pandas.read_csv('wdbc.csv', header=None)
```

0	1	2	3	4	5	6	7	8	9	...	22	23	24	25	26	27	28	29
842302	M	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.300100	0.147100	...	25.380	17.33	184.60	2019.0	0.16220	0.66560	0.71190	0.26540
842517	M	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.086900	0.070170	...	24.990	23.41	158.80	1956.0	0.12380	0.18660	0.24160	0.18600
4300903	M	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.197400	0.127900	...	23.570	25.53	152.50	1709.0	0.14440	0.42450	0.45040	0.24300
4348301	M	11.420	20.38	77.58	386.1	0.14250	0.28390	0.241400	0.105200	...	14.910	26.50	98.87	567.7	0.20980	0.86630	0.68690	0.25750

These attributes or features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. The computed features describe characteristics of radius, texture, perimeter, area, smoothness, compactness, etc. of the cell nuclei included in the image.
Among these attributes, the first attribute was that of Patient ID and the second attribute was the target vector which determined for each Patient ID if the cancer was Malignant (M) or

Benign (B). The Patient ID column was dropped since this would not affect the decision boundary through the course of the experiment and the Malignant/Benign feature vector was replaced with 1/0 respectively for classification purposes, yielding the below dataset:



```
data_frame
```

1	2	3	4	5	6	7	8	9	10	...	22	23	24	25	26	27	28	29
1	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.300100	0.147100	0.2419	...	25.380	17.33	184.60	2019.0	0.16220	0.66560	0.71190	0.26540
1	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.086900	0.070170	0.1812	...	24.990	23.41	158.80	1956.0	0.12380	0.18660	0.24160	0.18600
1	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.197400	0.127900	0.2069	...	23.570	25.53	152.50	1709.0	0.14440	0.42450	0.45040	0.24300
1	11.420	20.38	77.58	386.1	0.14250	0.28390	0.241400	0.105200	0.2597	...	14.910	26.50	98.87	567.7	0.20980	0.86630	0.68690	0.25750

This dataset was then further used in preprocessing to obtain a more normalized distribution of data.

3 Preprocessing

As part of preprocessing of the dataset, firstly the dataset was split into training, validation and testing datasets using **numpy.split** and then a normalization or a standardization technique was applied on the same using the **sklearns** library. The training, validation and test data were split in the ratio 80:10:10 yielding matrices of dimensionality 455x30, 57x30 and 57x30 each with a random seed (the data was split in a random fashion). By normalization, the feature vectors are rescaled to a [0,1] range whereas through standardization, the feature vectors are altered to achieve zero mean and unit variance. Although there are several techniques of both normalization and standardization that can be applied, this experiment explores the following techniques:

1. MinMaxScaler
2. StandardScaler

3.1 Min Max Scaler

MinMaxScaler is a data preprocessing library provided by sklearns which is achieved by subtracting the feature value from the min value and dividing by the difference between the max and the min value of that feature. This gives an output between 0 and 1. The formula is as indicated below:

$$v = \frac{v - min}{max - min}$$

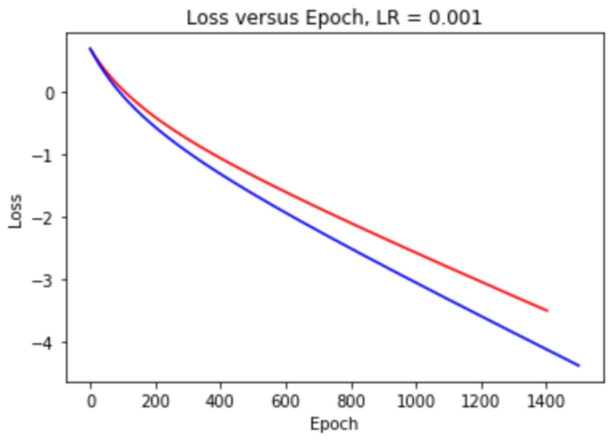
This assures that the distribution is present within the [0,1] range without distorting the data distribution. With this technique, the highest result metrics were obtained (as displayed in the Results section).

3.2 Standard Scaler

StandardScaler is a data processing library provided by sklearns which subtracts the mean from the feature value, divided by the variance. The formula is as indicated as below:

$$v = \frac{v - \mu}{\sigma}$$

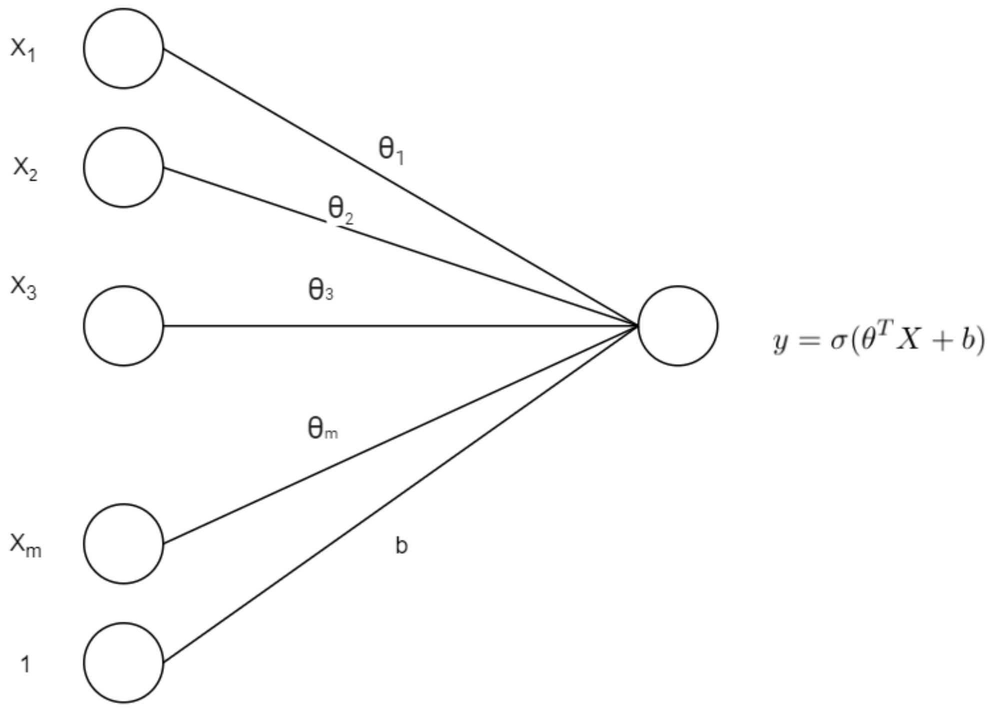
This method gives a normal probabilistic distribution of data (Gaussian distribution or bell curve). However, using this method, the below graph for loss versus epochs was obtained:



The accuracy alone using the above method was approximately 71%. This indicates that upon application of StandardScaler, the data could have been distorted which resulted in a poor metric.

4 Architecture

The generic computational model for logistic regression can be depicted as below where X_1, X_2, … X_m are the feature vectors (in this case m = 30 since there are 30 feature vectors provided), b is the bias and Θ_1, Θ_2, … Θ_m are the weights used corresponding to each of the feature vectors. Combined, these form the input parameters to the sigmoid function, which based on the probability i.e., if y >= 0.5, the data point is predicted as 1 and if y < 0.5, it is predicted as 0. The decision boundary would therefore be at y = 0.5 as depicted in the second diagram.



Combined, these form the input parameters to the sigmoid function (the red line indicated), which based on the probability i.e., if y >= 0.5, the data point is predicted as 1 and if y < 0.5, it is predicted as 0. The decision boundary (the blue line indicated) would therefore be at y = 0.5 as depicted in the diagram below.

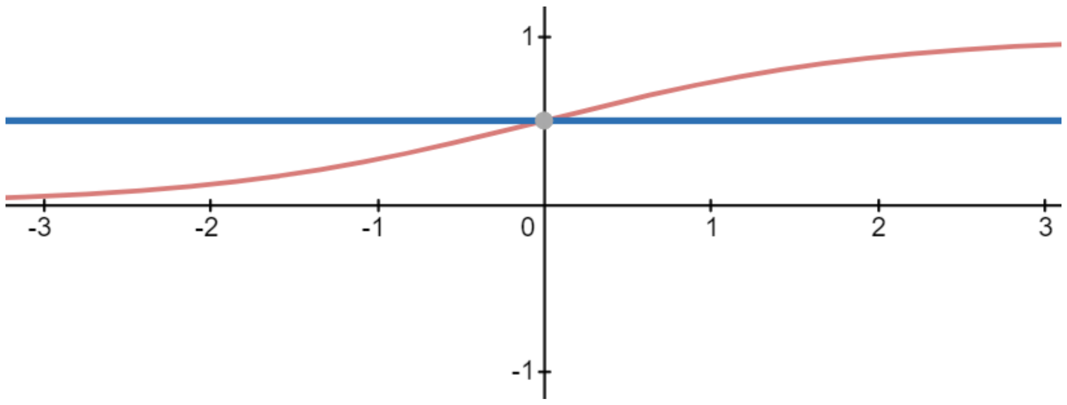


Figure 1: Sample Figure Caption

5 Results

The results of the experiment were drawn based on tuning of both hyperparameters as well as using different standardization and/or normalization techniques. The loss function was plotted against the epoch for both the validation dataset (in blue) and the training dataset (in red). The result metrics was calculated in terms of accuracy, precision, recall and F1 score. Their respective formulas are:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

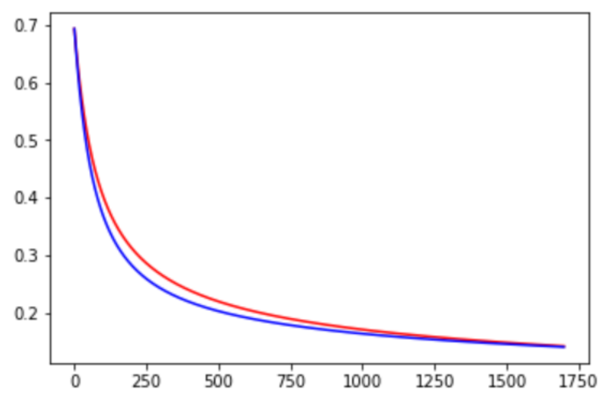
$$Precision = \frac{TP}{TP + FP}$$

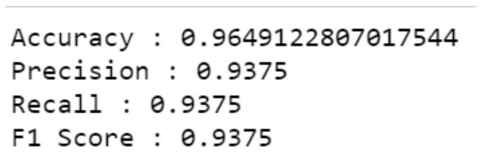
$$Recall = \frac{TP}{TP + FN}$$

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

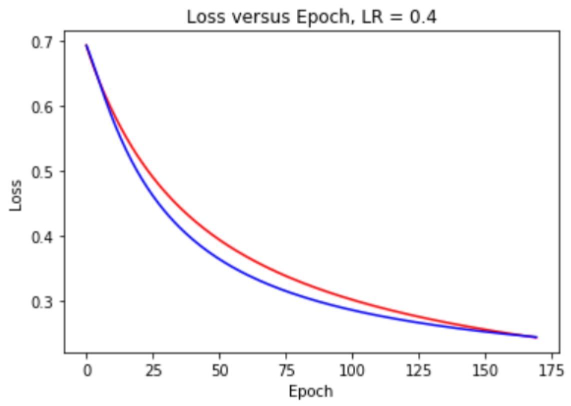
- Accuracy provides the ratio of correctly labelled result out of all the predicted data.
- Precision provides the ratio of the correctly labelled positive result out of all the positive results.
- Recall gives the ratio of the correctly labelled positive result out of all the results.
- F1 Score gives the weighted average of precision and recall.

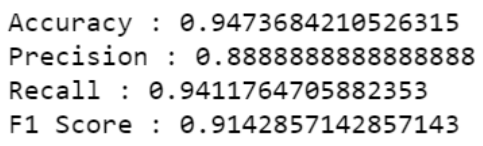
The results also fluctuated based on the randomness of the data set. For example, for a learning rate of 0.2 and at ~1750 epochs (point of convergence between the training and validation curve), the below loss versus epoch graph followed by its metrics were obtained:



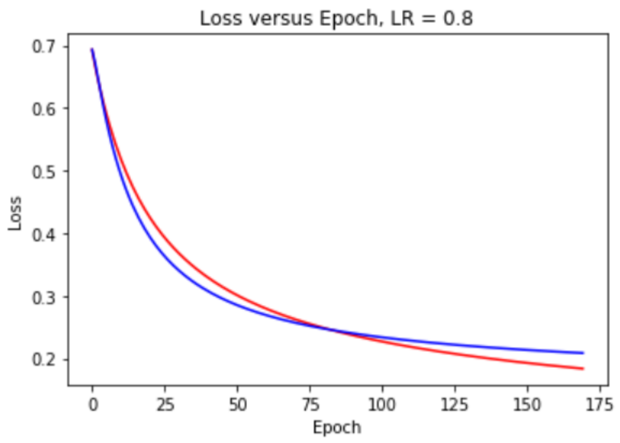

```
Accuracy : 0.9649122807017544
Precision : 0.9375
Recall : 0.9375
F1 Score : 0.9375
```

But after shuffling the data once more, and applying a learning rate of 0.4, the graphs converge at ~175 epochs with the metrics as indicated below:

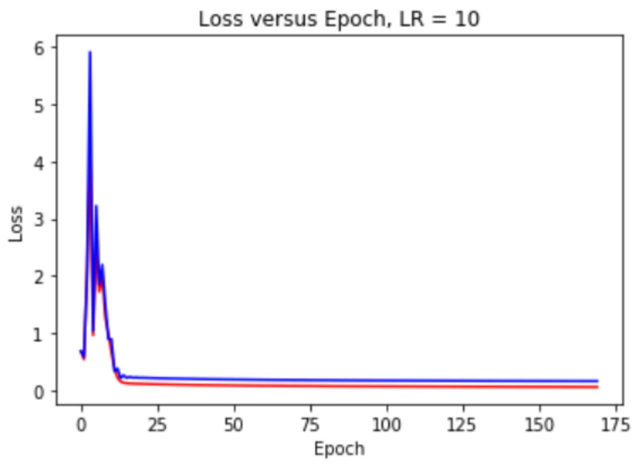



```
Accuracy : 0.9473684210526315
Precision : 0.8888888888888888
Recall : 0.9411764705882353
F1 Score : 0.9142857142857143
```

Upon increasing the learning rate to 0.8 for the same number of epochs, we get the below graph:

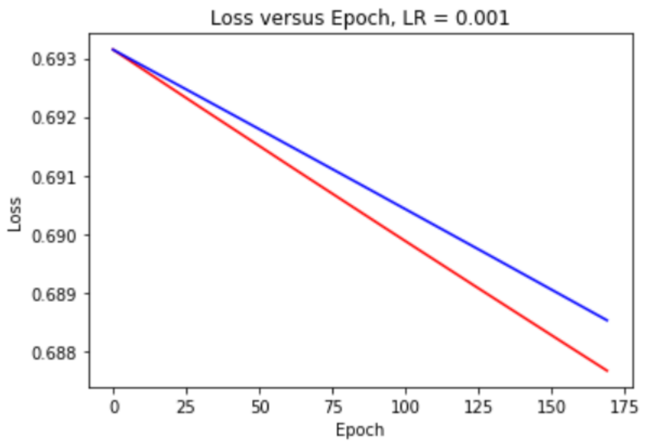


The region past the point of convergence indicates overfitting of the training dataset and would hence not be ideal to generate the metrics result past this point. If we further increase the learning rate to 10, we get:



This graph occurs because due a large learning rate, the gradient descent fails to find the local minimum and would be rapidly changing in order to converge to this/another local minimum. This does not benefit the purpose of logistic regression. For this very reason, it is beneficial to take a learning rate just big enough to converge at an apt pace at the local minimum.

On the other hand, if we take a very low learning rate of say 0.001, for the same number of epochs we get:



This indicates that using this learning rate, it is not converging fast enough to reach the local minimum of the lowest cost. Hence either the learning rate must be increased, or the number of epochs must be increased, or both in order to converge to the local minimum faster.

6 Conclusion

Through this logistic regression experiment, the results are altered based on the fluctuations of the hyperparameters (learning rate and epoch). Using the MinMaxScalar normalization technique, an accuracy of 96.49% is achieved on the WDBC dataset with an F1 Score of 0.9375. Since in this case the F1 Score is high (almost 94%), it is an indicator of a high precision and recall which in turn is a good indicator of the classifier.

References

[1] Wikipedia - Logistic Regression https://en.wikipedia.org/wiki/Logistic_regression

[2] Christopher M. Bishop ©2006 Pattern Recognition and Machine Learning. Springer-Verlag Berlin, Heidelberg