CSE-574 Project 1- Logistic Regression

Venugopal Shah
Masters Student
Department of Computer Science
University at Buffalo
vshah3@buffalo.edu
Person number: 50291126

Abstract

The task of this project is to build and implement Logistic Regression from scratch for a two-class problem. The code is written in Python and the problem is to use logistic regression as a classifier on the Wisconsin Diagnostic Breast Cancer dataset to classify FNA cells as either Benign or Malignant.

1 Introduction

1.1 Logistic Regression

Logistic Regression is a regression model that is used for classification problems. It models probabilities for questions that have two possible outcomes. Since the linear regression model works well for regression but not classification, logistic regression is preferred for 2-class problems. The input variables are continuous variables.

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_l x_l$$

Fig.1 Logistic Regression Equation

1.2 Gradient Descent

The aim of Gradient Descent is to find the optimize a given function by looking for optimal parameters. For logistic regression, it is done by minimizing the loss function to find the optimal 'theta'.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^i \log(h_\theta(x^i)) + (1 - y^i)\log(1 - h_\theta(x^i))$$

Fig.2 Logistic Regression Loss Function where m is the number of training examples

Gradient descent is an iterative optimization algorithm which finds the minimum of a function. in stochastic gradient descent algorithm's iteration, the gradient is computed against a single randomly chosen training example. This way, stochastic gradient descent is better because it requires only one example in each step. On the other hand, the learning rate has to be chosen carefully.

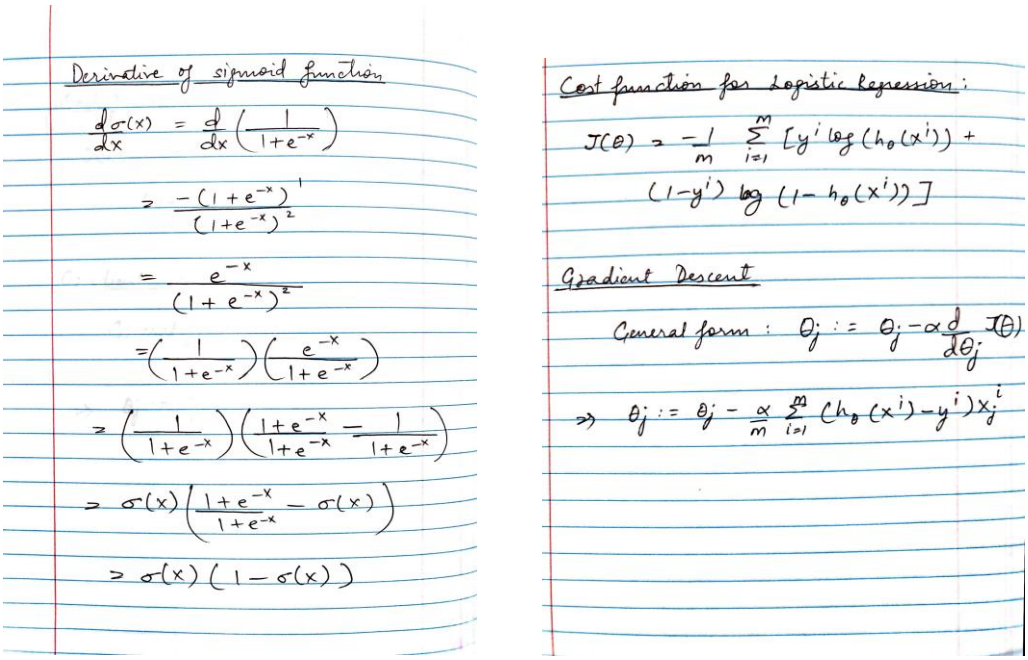

Fig.3 Math behind the derivation of sigmoid function and general form of the Gradient descent along with the cost function for Logistic Regression

2 Setup

2.1 Pre-processing data

First let us learn more about the dataset:

- The dataset contains 569 instances with 32 attributes (ID, diagnosis (B/M), 30 real-valued input features).
- Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass.
- The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features.
- 80% of this dataset was partitioned for training, while the remaining 20% was split into validation and training.
- The actual readings in the data, namely 'M' for Malignant and 'B' for Benign were mapped to 1 and 0. So now, 1 represents M and B represents 0.
- We also normalize the dataset, except for the results. This is to ensure all the values lie within a range which makes our calculations avoid divide by zero errors.
- The id column was dropped from the dataset.
- Each set (training, validation and test) was split into two matrices, X and Y. X contains all the input features while Y simply contains the results. We then attempt to find out y, a set of predictions from our model.

14	1	6.388376	8.725059	...	15.175258	8.007096	11.551882
15	1	7.155095	12.059520	...	11.766323	10.459294	10.371245
16	1	7.287614	7.047683	...	11.058419	5.771733	3.557654
17	1	8.660135	7.419682	...	14.247423	8.440765	7.760724
18	1	12.143499	8.413933	...	16.412371	4.742756	2.769251
19	0	6.208529	3.145079	...	8.852234	5.566726	2.302243
20	0	5.773108	4.058167	...	5.005498	6.382811	3.514364
21	0	2.388187	1.846466	...	4.279725	3.489060	2.976518
22	1	7.912348	3.077443	...	16.446735	12.229450	5.827102
23	1	13.421364	9.015894	...	13.807560	4.955648	2.652499
24	1	9.152350	7.893135	...	14.398625	8.074118	5.325987
25	1	9.616167	4.524856	...	17.525773	9.860043	6.671914

Fig4. Dataset after M and B have been mapped to 1 and 0 and all the other values have been normalized to a range of 20.

2.2 Model

Let us now see how our model works:

- We train our model using the training dataset. The other two hyper-parameters for our model is the number of iterations the model runs for and the learning rate.
- All the hit and trial experiments were run on the validation set to see which parameters result in the best accuracy.
- The experiments on the validation set was tried for different iterations and different learning rates, noting the training and validation loss and also the training and validation accuracies.

  Different learning rates experimented with:
  `[0.00001, 0.00002, 0.00003, 0.00004, 0.00005, 0.00006, 0.00007]`

  Different epochs experimented with:
  `[20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000, 200000]`

- The hyper-parameters for the best results was then noted and used for running the test set.
- The results on the test set were then used to calculate accuracy, precision and recall by finding out the True Positives, True Negatives, False Positives and False Negatives (TP, TN, FP and FN).

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

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

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

Fig.5 Formulas used to calculate Accuracy, Precision and Recall

Now let us move on to the results where we will analyze and infer from our findings.

3 Results

3.1 Learning Rate

Learning rate is a very vital hyper-parameter that has to be chosen really carefully. A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small can cause the process to get stuck.

The following graph shows how the learning rate lowers the training loss. But there will eventually come a time where the loss will start increasing again.

NOTE: For each of these tests, the epochs were set at 100000

Fig.6 Learning Rate was increased and a drop in training loss was observed

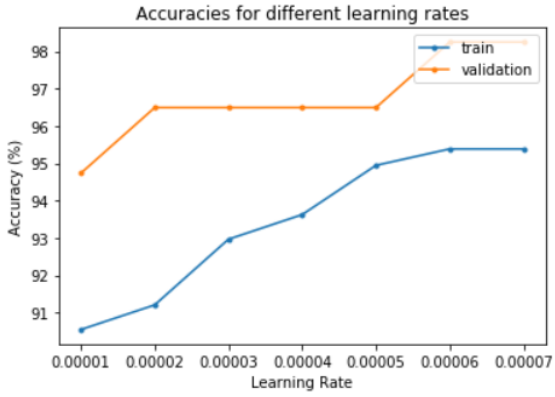

Fig.7 Increasing learning rate increased training and validation accuracies but then overfitting was observed after lr > 0.00006

3.2 Epoch

As the number of epochs increases, more number of times the weights are changed in the neural network and the curve goes from under-fitting to optimal to overfitting curve. As you can see from following graph, the number of epochs increase and reduce the loss. But again,

a very high number of epoch can lead to overfitting.

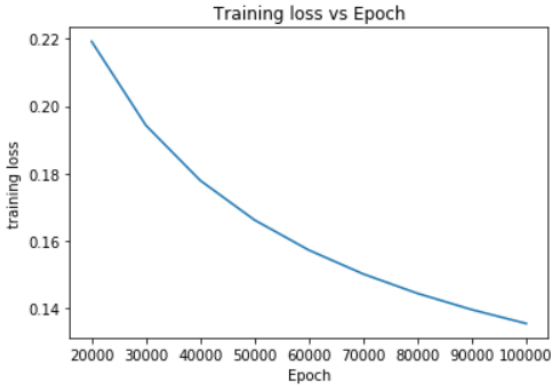

Fig.8 With increasing Epoch the training loss goes down before overfitting happens

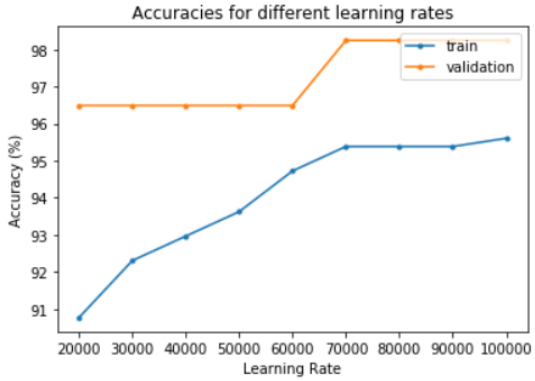

Fig.9 Both training and validation accuracies increase with increasing epoch

3.3 Test

Let us first see the different accuracies calculated for training and validation data sets.

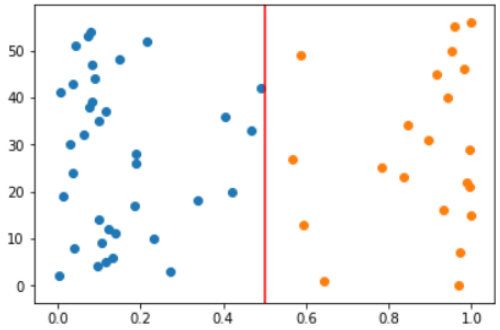

Fig.10 A visualization of the final predictions on the test data

From our experiments, we decide to keep our learning rate as 0.00008 and the epoch as 100000 for which the best results on test will be observed.

Train accuracy: 95.6%

Validation accuracy: 98.2%

Test accuracy: 92.9%

TP: 21 **FP:** 1 **TN:** 32 **FN:** 3

3.4 Accuracy, Precision and Recall

For the test set, the following were calculated:

Accuracy: 53/57 = 92.9%

Precision: 21/22 = 95.4%

Recall: 21/23 = 87.5%

References

https://mccormickml.com/2014/03/04/gradient-descent-derivation/

https://medium.com/data-science-group-iitr/logistic-regression-simplified-9b4efe801389

https://medium.com/@martinpella/logistic-regression-from-scratch-in-python-124c5636b8ac

Fig. 1 https://towardsdatascience.com/gradient-descent-demystified-bc30b26e432a

Fig. 2 https://towardsdatascience.com/gradient-descent-demystified-bc30b26e432a

Prof. Srihari's lectures, slides and assistance from the TAa was also essential in completing this project.