ML Project 1: Classification using Logistic Regression

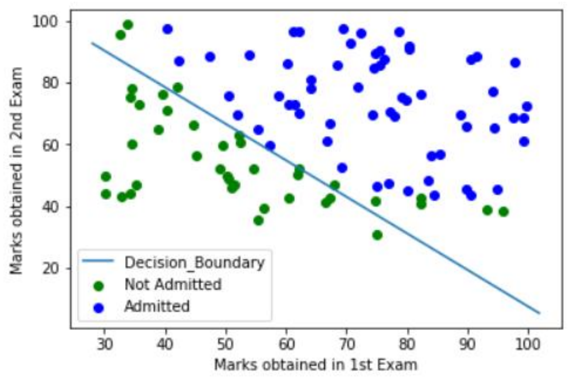
ANUJ SHAH
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
anujyoge@buffalo.edu

Abstract

Here, I present an efficient approach to solve the two-class problem of classification using machine learning which lets humans off the hook of this tedious work, saving their precious time to concentrate on other trivial tasks. A dataset of Wisconsin Diagnostic Breast Cancer is used which consists of 30 features on the basis of which a fine needle aspirate (FNA) can be classified as malignant or benign. To resolve this task in an efficient and time saving manner I propose the method of Logistic Regression which expedites the entire process into a matter of seconds.

1 Introduction

The process of scrutinizing a report may not consume a lot of time but in some situations even those minute minutes make a huge difference. This task as simplified and expedited by a machine which is able to analyze and provide results within matter of seconds for a number of patients. We use the method of Logistic regression for this prediction. Logistic Regression is just another technique got from statistic and is similar to linear regression. It is a binary classification method. For instance, a mail, is it spam (class 0) or not spam (Class 1)? It predicts and we need to optimize this prediction. Let's take an example to understand this two-class classification problem better.



It's clear from the graph that we have made a boundary between the two classes so that the next time if any new point is being plotted one will know which class it will belong to. In this example the points above the boundary will be put into the class of 'admitted' and below will be classified as 'not admitted.'

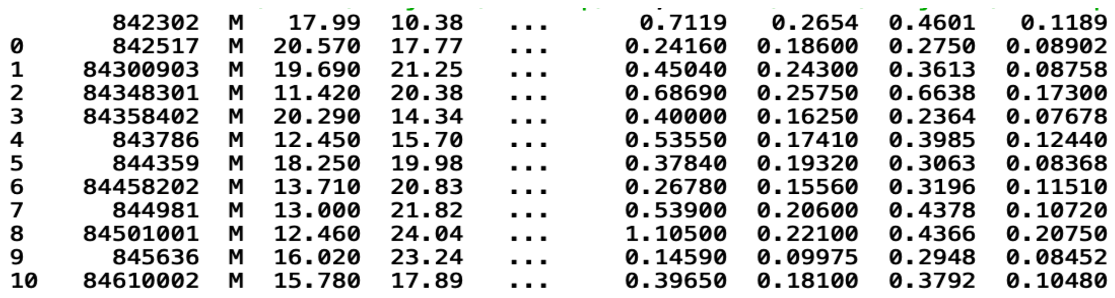
To achieve this goal, we have made use of gradient descent function which helps the machine to learn faster. The entire method will be explained in detail further in the coming sections.

3 Dataset Definition

We have been provided the Wisconsin Diagnostic Breast Cancer dataset. Data is the primary bridge for our predictions to match the true values. This dataset is in comma separated values format and consists of 32 columns and 569 rows. These rows represent the observations/ samples. The computed features describe the following characteristics viz; radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry and fractal dimension. Operations were performed on these characteristics to obtain the 30 features.

DATASET COLUMN NO	DESCRIPTION
1	ID number
2	Target values or resultant values
3-32	Values of the features

A part of the original data looks like this:



```
        842302  M   17.99  10.38   ...     0.7119   0.2654  0.4601   0.1189
0       842517  M  20.570  17.77   ...    0.24160  0.18600  0.2750  0.08902
1     84300903  M  19.690  21.25   ...    0.45040  0.24300  0.3613  0.08758
2     84348301  M  11.420  20.38   ...    0.68690  0.25750  0.6638  0.17300
3     84358402  M  20.290  14.34   ...    0.40000  0.16250  0.2364  0.07678
4       843786  M  12.450  15.70   ...    0.53550  0.17410  0.3985  0.12440
5       844359  M  18.250  19.98   ...    0.37840  0.19320  0.3063  0.08368
6     84458202  M  13.710  20.83   ...    0.26780  0.15560  0.3196  0.11510
7       844981  M  13.000  21.82   ...    0.53900  0.20600  0.4378  0.10720
8     84501001  M  12.460  24.04   ...    1.10500  0.22100  0.4366  0.20750
9       845636  M  16.020  23.24   ...    0.14590  0.09975  0.2948  0.08452
10    84610002  M  15.780  17.89   ...    0.39650  0.18100  0.3792  0.10480
```

4 Pre-Processing

The data received by us needs to be pre-processed before it can be used to receive better results. I have processed the data as follows:

- Added Headers to the data frame for convenience of functions such as delete.
- Drop the 1st column, (ID column) since it wasn't required.
- Mapped M to 1 and B to 0 in the target column.
- Divided the target vector and from the features
- Split the data into Train, Validate and Test.
- Normalize the data so that all the data comes into one range.

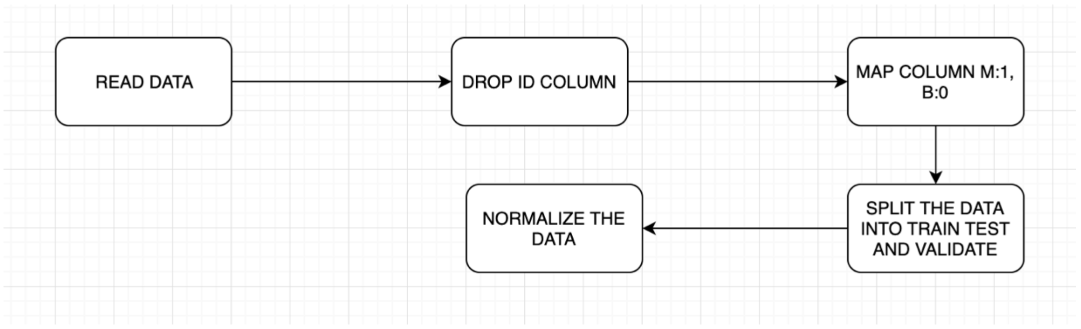
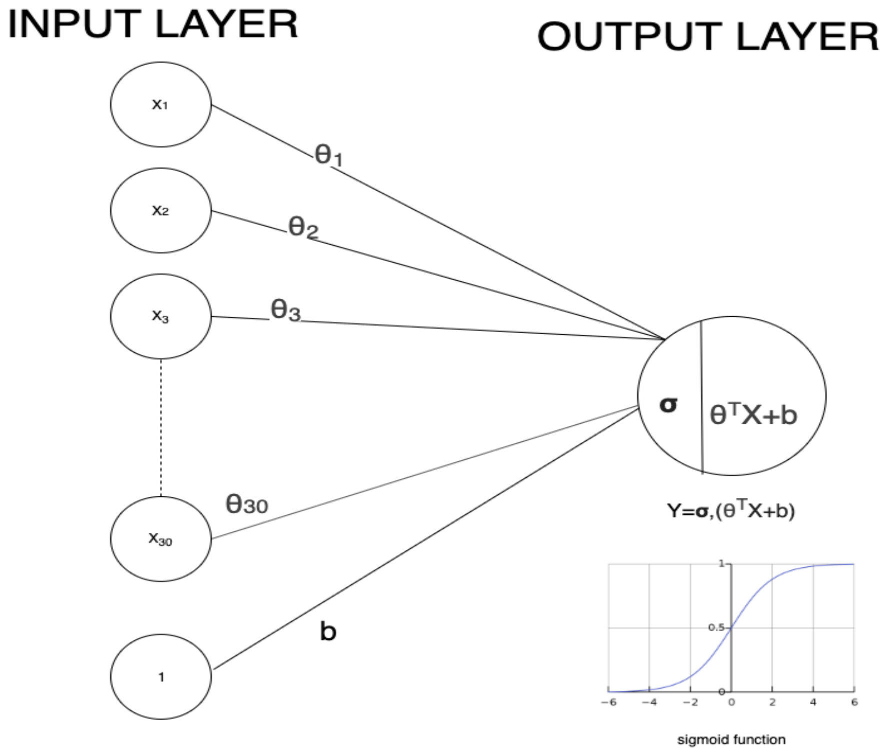


Figure: Flowchart for Pre-Processing

5 Model

5.1 Model Architecture

Given below is the architecture of our model used.



As seen from the figure above, there are 30 inputs and 1 output. These 30 inputs are because there are 30 features in our dataset which we use to get the output. Each input has a weight associated with it. This is represented by θ in our architecture. Our model consists of another input which is the bias and is represented by b. It has denoted by 1 and not any variable cause there can be only 1 bias. As seen from the equation of Y (from the diagram) to get Y(output) we pass the sigmoid function over our equation. The sigmoid function basically gets all the values between 0 and 1. Since our model is probability based, we use this function to limit the values between 0 and 1. Now to get the final output, that is Y, we divide the probabilities into 2 halves at 0.5. If the probability is 0.5 and above then it is class 1 (Malignant in our case) and for the probabilities below 0.5 it is class 0(Benign in our case)

5.1 Methodology

We know that we need to classify the problem on the basis of the features provided to us. For this we have a number of inputs which are the features, on the basis of this we will be able to predict the class. The genesis equation for this will be

$$z = \boldsymbol{w} \cdot \boldsymbol{x} + \mathrm{b}$$

Now for the predictions we pass this genesis function into the sigmoid function. The sigmoid function is used since the probabilities need to be between 0 and 1 thus this function is used.

Sigmoid function can be defined as:

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

Now we need to calculate the Loss and cost function. Let's define y as the true label (y = 0 or y = 1) and yP as the predicted output (or the probability that y = 1 given inputs w and x). Therefore, the probability that y = 0 given inputs w and x is (1 - yP), as shown below.

$$P\ (y = 1|\boldsymbol{w}, \boldsymbol{x}) = \hat{y}$$
$$P\ (y = 0|\boldsymbol{w}, \boldsymbol{x}) = 1 - \hat{y}$$

After doing some derivations based on the equations above, we can define the logistic loss function to be:

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$$

This function is known as the Cross-Entropy function. The goal is to minimize the error between the predicted and the desired output, so as to arrive to an optimum solution.

Using this function and gradient descent we minimize the error and keep on updating values of the weights which will help gain better accuracy of our prediction and take us one step closer to our goal.

To understand this process let's take an example of rolling a ball down the bowl-shaped function it would settle at the bottom; similarly, to find the minimum of the cost function, we need to get to the lowest point. To reach the lowest point we need to keep updating the weights and bias for which we use the following formulas:

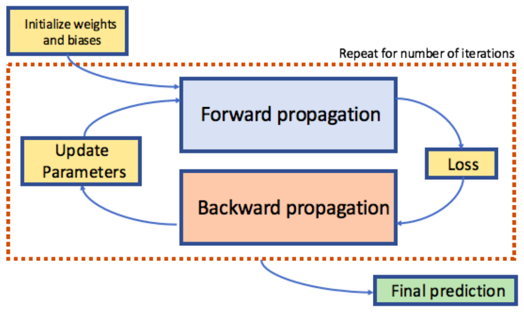
Alpha(symbol after w-) is the learning rate

$$w = w - \alpha\frac{\partial J(w,b)}{\partial w}$$

$$b = b - \alpha\frac{\partial J(w,b)}{\partial b}$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[(y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})]$$

The process of training the model by checking with respect to the weights in the equation is the forward pass and the part when it updates the weights and sends them back for the new checking is the backward propagation part. After the end of the process we get the final values of weights and bias through which we can predict the final answer.



In this entire process the hyper-parameters such as epochs and learning rate need to be tuned to get the best possible results. Utmost care need to be taken, since if the learning rate is too high the it might overshoot and if it too low it might take too much of time. Thus reducing the overall efficiency of the model and deteriorating it.

6 Results of Experimentation

The task that we are performing here is to train our model to predict whether the FNA cell are malignant or benign for any given set of data. To receive the best results, I had to tune the hyper parameters a number of times. Here are the results of what happens when I changed learning rate (LR) to optimal, very high and very low with respect to this particular problem.

When the LR is optimum (0.016)

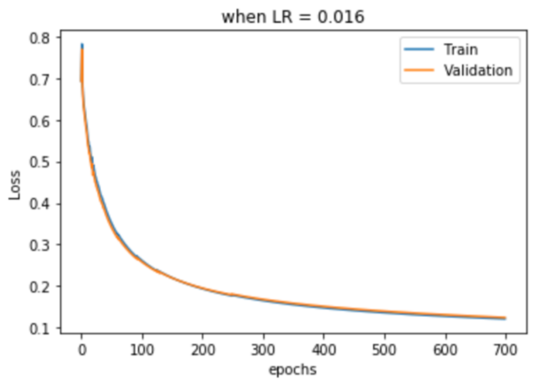


Figure: Loss Function Graph

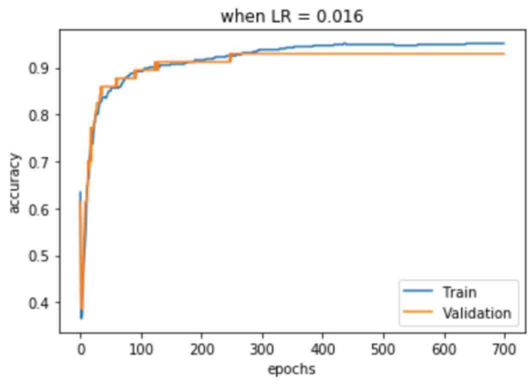


Figure: Accuracy Function graph

From the graph above we can see that in merely 100 epochs the accuracy hits 90% and the total loss becomes as low as 0.2 which shows the efficiency of the model.

When the LR is very low (0.001)

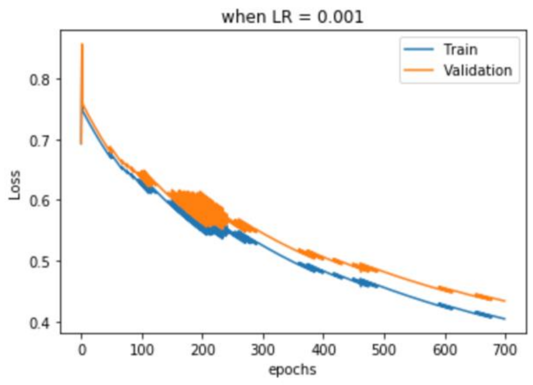


Figure: Loss Function Graph.

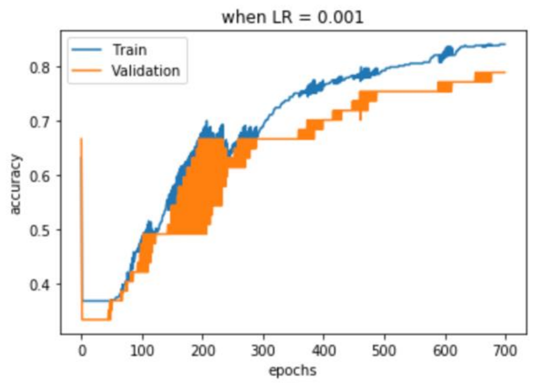


Figure: Accuracy Function graph

When the LR is set at 0.001 we can infer from the graph that there are many fluctuations in the accuracy as well as loss. This shows that our model is having difficulty reaching its final result. To The total cost should reduce as fast as possible in least amount of time (epoch) making sure that overshooting doesn't take place. Here are model is taking too many epochs to reach an accuracy of only 80% which can be made better.

<u>When the LR is very high (1)</u>

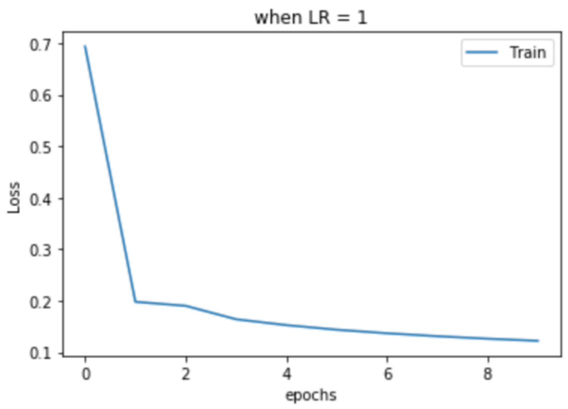
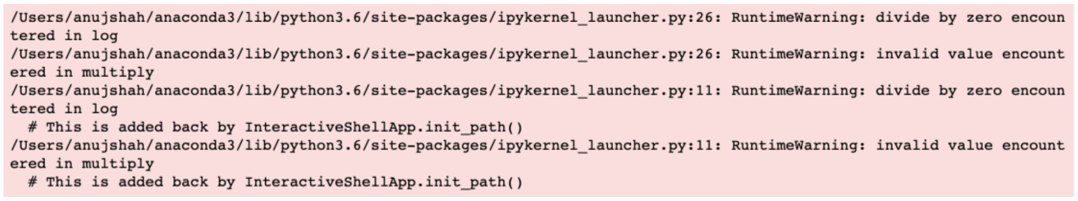


Figure: Loss function Graph

See carefully, in this case the number of epochs is only 8!
An error occurs post that, which is 'divide by 0 encountered in log' error. This is mainly cause log part of the Cross-Entropy function reaches a point after which it gives log(0), which cannot be defined.



```
/Users/anujshah/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:26: RuntimeWarning: divide by zero encoun
tered in log
/Users/anujshah/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:26: RuntimeWarning: invalid value encount
ered in multiply
/Users/anujshah/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:11: RuntimeWarning: divide by zero encoun
tered in log
  # This is added back by InteractiveShellApp.init_path()
/Users/anujshah/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:11: RuntimeWarning: invalid value encount
ered in multiply
  # This is added back by InteractiveShellApp.init_path()
```

Another thing observed while using scikit-learn functions is that due to different normalization functions the learning rate varies vastly. When we use normalize() as the function for normalization of data the learning rate needs to be very high for the model to be trained efficiently. This is mainly because the normalize function rescales each sample to have a unit form. In other words, the normalizer does not remove the mean and scale by deviation but scales the whole row to unit norm. When we use the standard scaler for the process of normalizing data, we receive the results as above. The standard scaler method assumes that your data is normally distributed within each feature and may behave badly if it is not so. Here, the normalization is done by subtracting the mean and then scaling to unit variance.

I have tried splitting the data's in different combinations and have observed that the lowest accuracy falls to 87% and the highest is 80%. The above represented graphs are for a particular set.

7 Conclusion

The model has been trained, validated and tested with an overall accuracy of 98.24 % during the testing phase. The architecture is based on logistic regression and also makes use of gradient descent function. The results of our experiments which have been performed are shown above which demonstrates the power and efficiency of our model with respect to test data.

Acknowledgments

We are extremely grateful to Professor Sargur Srihari for elucidating all the necessary concepts related to machine learning. My special gratitude to the teaching assistants Mihir Chauhan, Tiehang Duan and Yan Shen to remove my ambiguity in a few concepts.

References

[1] Pattern Recognition and Machine Learning by Christopher M. Bishop

[2] Lecture notes

[3] https://scikit-learn.org