CLASSIFICATION USING LOGISTIC REGRESSION

Mohammed Mahaboob Khan
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
mkhan32@buffalo.edu

Abstract

The task of this project was to perform classification using machine learning. The problem given to was a two-class problem. The features used for classification were pre-computed from images of a fine needle aspirate (FNA) of a breast mass. My task was to classify suspected FNA cells to Benign (class 0) or Malignant (class1) using logistic regression as the classifier.

1 Introduction

At a high level, these machine learning algorithms can be classified into two groups based on the way they "learn" about data to make predictions: supervised and unsupervised learning. The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (X) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output Y = f(X). The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

Logistic regression is a supervised machine learning technique. Logistic regression is named for the function used at the core of the method, the logistic function. Also called the sigmoid function, it is an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

2 Dataset

The dataset in use is the Wisconsin Diagnostic Breast Cancer (WDBC) dataset. The WDBC 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.

For this project, I have partitioned the dataset into training set, validation set and testing set in the ratio 8:1:1. Hence, I have used 455 instances for training, 57 instances for validation and 57 instances for testing.

- **Training data**:

I have used 455 instances from the WDBC dataset to train my model using Gradient Descent for logistic regression.
Hyperparameters used: Learning Rate(lr) and epoch (one complete presentation of the data set to be learned)

- **Validation data**:

I have used 57 instances from the WDBC dataset to validate my model. Validation also helped me tune my hyperparameters.

- **Testing data**:

I have used 57 instances from the WDBC dataset to test my model and evaluate its performance parameters (accuracy, precision, recall, F1 Score).

3 Pre-processing

The WDBC dataset required some pre-processing before feeding it to the model for training. I have mapped the class Benign(B) to 0 and the class Malignant(M) to 1. This was necessary as it helped ease the task of comparing the predicted value and also compute the performance parameters of the model namely, accuracy, precision, recall and F1 score. The other pre-processing tasks that I performed are,

- Split the dataset into training, validation and test data
- Normalize the data

To summarize, the pre-processing tasks performed are,

- Drop the first column that contains IDs
- Map the 'M' and 'B' values to 1 and 0
- Split the WDBC dataset into training, validation and test set in the ratio 8:1:1
- Normalize the data before feeding it to the model

4 Architecture

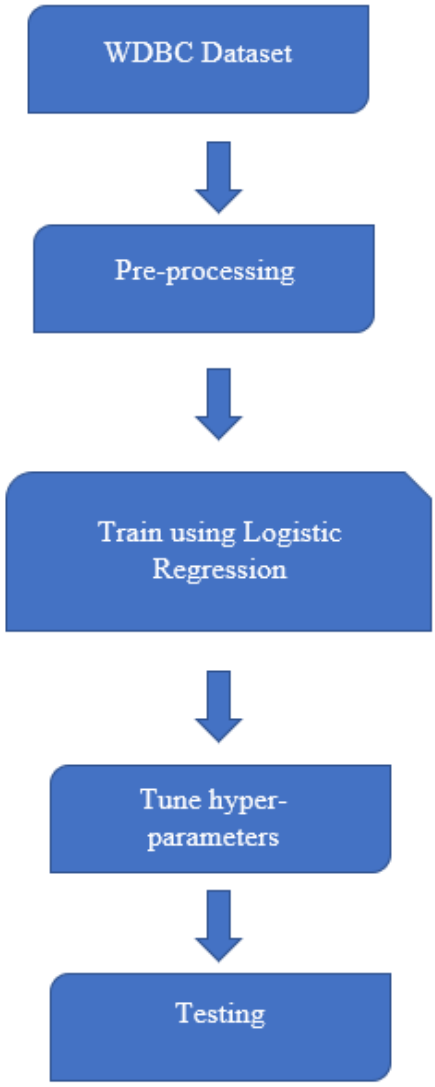


Figure 1 – Architecture

- **Logistic Regression:**

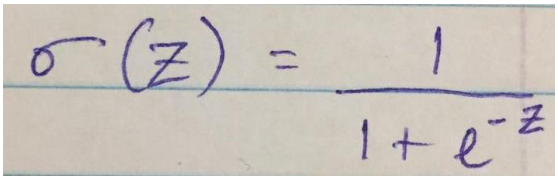


Figure 2 – Sigmoid Function

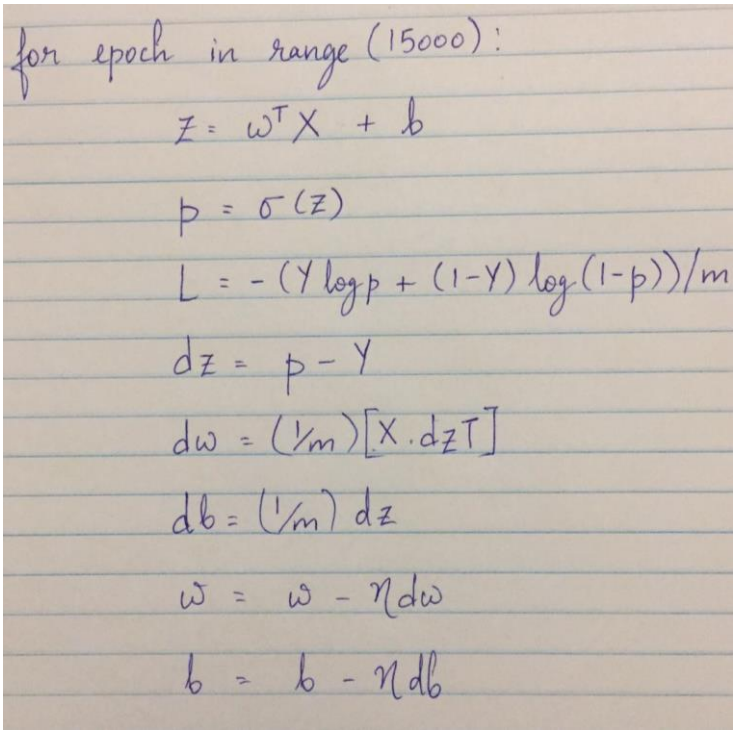


Figure 3 – Logistic Regression

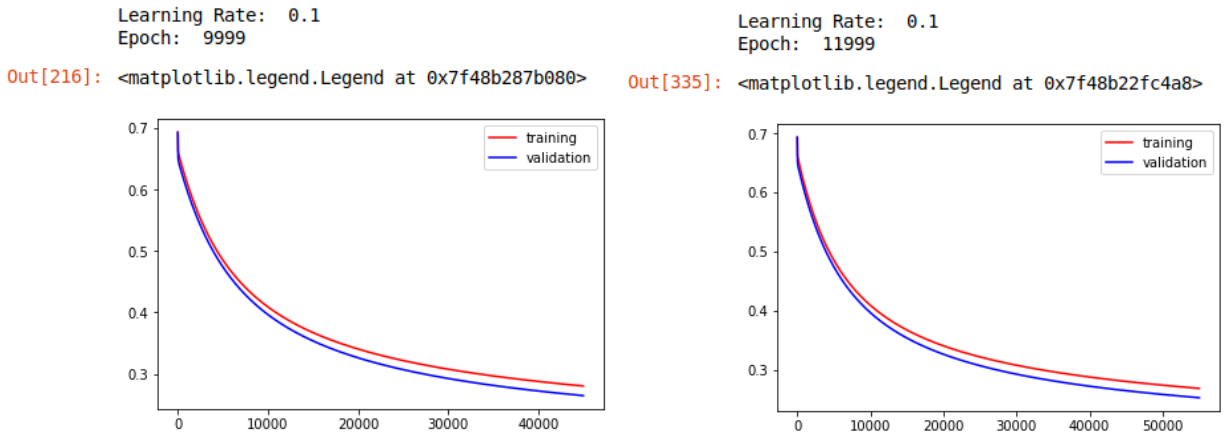
5 Results

By using the validation data, I tune my hyper-parameters and choose them to be,
Learning rate = 0.1
Epoch = 15000

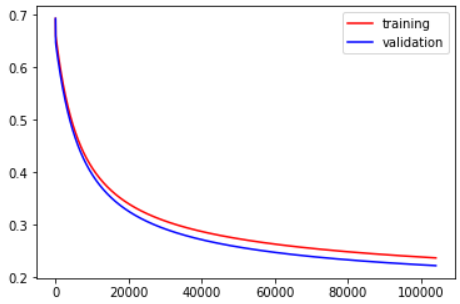
- **Choosing Epoch:**

Epoch is the hyper-parameter that defines the number times that the learning algorithm will work through the entire training dataset.

The screenshots below show different epoch settings.



```
Learning Rate:  0.1
Epoch:  14999
Out[236]: <matplotlib.legend.Legend at 0x7f48b244af60>
```




```
Learning Rate:  0.1
Epoch:  24999
Out[285]: <matplotlib.legend.Legend at 0x7f48b20c6978>
```

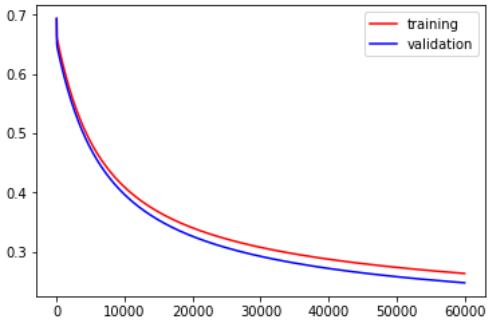


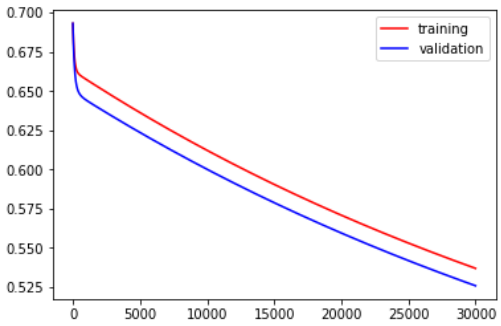
Figure 3 – Choosing Epoch

- **Choosing Learning Rate:**

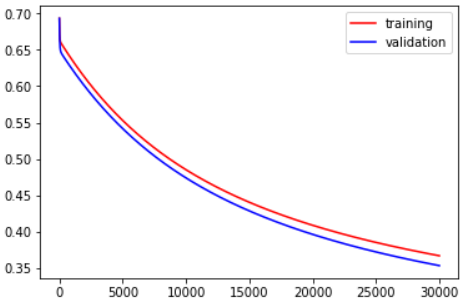
Learning rate is a hyperparameter that controls how much we are adjusting the weights of our network with respect to the loss gradient. The lower the value, the slower we travel along the downward slope. While this might be a good idea in terms of making sure that we do not miss any local minima, it could also mean that we'll be taking a long time to converge.

The screenshots below show different learning rate settings

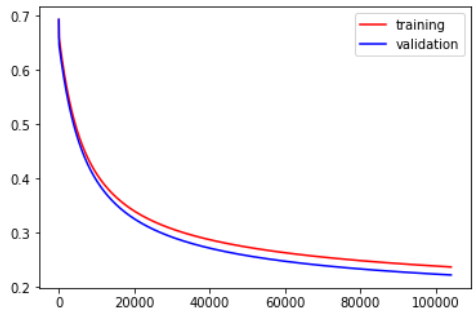
```
Learning Rate:  0.01
Epoch:  14999
Out[404]: <matplotlib.legend.Legend at 0x7f48b1b25fd0>
```




```
Learning Rate:  0.05
Epoch:  14999
Out[399]: <matplotlib.legend.Legend at 0x7f48b1c4b9e8>
```



```
Learning Rate:  0.1
Epoch:  14999
Out[236]: <matplotlib.legend.Legend at 0x7f48b244af60>
```




```
Learning Rate:  0.9
Epoch:  14999
Out[390]: <matplotlib.legend.Legend at 0x7f48b1d11780>
```

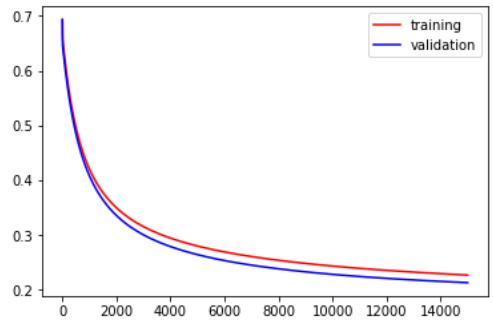


Figure 4 – Choosing Learning Rate

- **Performance Measurement:**

I compute a confusion matrix that gives me a count of True Positives, True Negatives, False Positives and False Negatives.

```
[[37  0]
 [ 2 18]]
```

```
In [417]: TP = conf_matrix[0][0]
          FP = conf_matrix[1][0]
          FN = conf_matrix[0][1]
          TN = conf_matrix[1][1]
```

Figure 5 – Confusion Matrix

Using the above, I calculate Accuracy, Precision, Recall and F1 Score for my model defined as follows,

```
In [418]: #calculate accuracy, recall and precision
          #Accuracy = ratio of correctly predicted observation to the total observations
          #Precision = ratio of correctly predicted positive observations to the total predicted positive observations
          #Recall = ratio of correctly predicted positive observations to the all observations in actual class
          #F1 Score = the weighted average of Precision and Recall
          Accuracy = (TP+TN)/(TP+TN+FP+FN)
          Precision = TP/(TP+FP)
          Recall = TP/(TP+FN)
          F1 = 2*(Recall * Precision) / (Recall + Precision)
```

Figure 6 – Performance Measures

The resulting model has the following performance parameters as calculated from the above formulae,

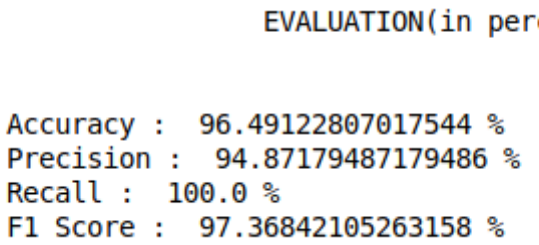


Figure 7 – Results

6 Conclusion

The end product of this project is a classification model based on Logistic Regression that has an accuracy of 96.49%. This project has helped me apply my existing knowledge of classification to practical use and gain deeper insights about classification problems. It has bolstered my understanding of Logistic Regression and Gradient Descent. I hope to test my model on other datasets in the future.

Acknowledgement

I am extremely grateful to Professor Sargur Srihari for explaining all the necessary concepts of Classification and Logistic Regression required to complete this project. I would also like to thank Mihir Hemant Chauhan, the senior Teaching Assistant for clarifying my doubts and making concepts clear in his recitation.