CSE 474/574: Introduction to Machine Learning (Fall 2019)
Solving Classification Problem using Logistic Regression

Sonali Naidu
Department of Computer Science
University at Buffalo
Buffalo, NY – 14226
sonaliye@buffalo.edu

Abstract

In this project I have worked on the classification problem using Machine Learning. It is a two class Logistic Regression problem and the features are pre-computed from images of a fine needle aspirate (FNA) of a breast mass. The dataset used is the Wisconsin Diagnostic Breast Cancer (wdbc.dataset). Here I have used logistic regression as a classifier to classify suspected FNA cells to Benign (class 0) or Malignant (class 1).

Introduction

Logistic Regression is a supervised Machine Learning algorithm that is used in binary classification. It is a classification algorithm which is used to predict the probability of categorical dependent variable. The dependent variable contains data coded as 1(success) and 0(failure) i.e., the model predicts P(Y=1) as a function of X. One of the limitations of Logistic Regression is that it can only classify the data into two distinct classes. In other words, it fits a line to the dataset and returns the probability of new sample belonging to one of the two classes based on location with respect to the line.

Algorithm:

The Logistic Regression is built to predict whether a person has Malignant or Benign tumor given certain variables(in our case it would be the 30 features from our dataset). We build a model that outputs the probability (a number between 0 and 1) to show whether the tumor is Malignant with a value of 1 or Benign with a value of 0.

Dataset

The dataset used in this problem is Wisconsin Diagnostic Breast Cancer (WDBC) which will be used for training, validation and testing. This dataset contains 569 instances with 32 attributes (ID, diagnosis (B/M), 30 real-valued input features). The features are pre-computed from images of a fine needle aspirate (FNA) of a breast mass and they describe characteristics of the cell nuclei present in the image. Radius, texture, perimeter, area, smoothness, compactness, symmetry, concave points are few of the features from the dataset. The entire dataset is divided as follows 80 percent for training and 10 percent each for validation and testing purpose.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split( x, y, test_size = 0.2)
x_test, x_val, y_test, y_val = train_test_split( x_test, y_test, test_size = 0.5)
```

Below is the size of the train, test and validation set after splitting the data.

```
x train:  (31, 455)
x test:  (31, 57)
y train:  (455,)
y test:  (57,)
x Val:  (31, 57)
y Val:  (57,)
```

Preprocessing

Normalization

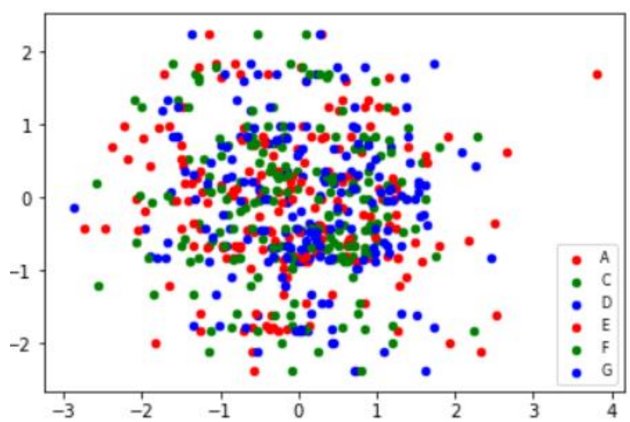
We start with data preparation for the Machine Learning algorithm by normalizing the dataset. Applying normalization to dataset changes the values of numeric columns in the dataset to a common scale without affecting the differences in the range of values.

```
#Normalizing the feature values
x = (X - np.min(X))/(np.max(X) - np.min(X)).values
print(x.shape)
print(x)
```

```
(569, 31)
          A         C         D         E         F         G         H  \
0  0.000915  0.521037  0.022658  0.545989  0.363733  0.593753  0.792037
1  0.000915  0.643144  0.272574  0.615783  0.501591  0.289880  0.181768
2  0.092495  0.601496  0.390260  0.595743  0.449417  0.514309  0.431017
3  0.092547  0.210090  0.360839  0.233501  0.102906  0.811321  0.811361
4  0.092559  0.629893  0.156578  0.630986  0.489290  0.430351  0.347893
5  0.000916  0.258839  0.202570  0.267984  0.141506  0.678613  0.461996
6  0.000917  0.533343  0.347311  0.523875  0.380276  0.379164  0.274891
```

Data Exploration

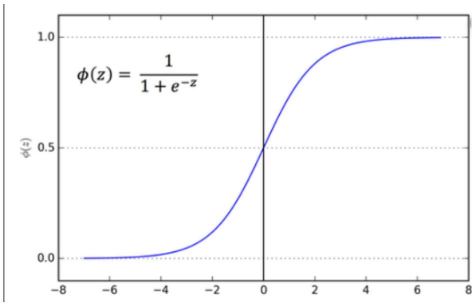
Let's have a look at below scatter plot showing how the current data is scattered after normalization. The below plot represents 6 of the 30 features from the dataset(only 6 of 30 features were chosen to visualize).



Sigmoid Function

The key to understand classification using logistic regression is that the logistic regression curve can only take values between 0 and 1 and this is achieved by applying the sigmoid function on the feature dataset. The sigmoid function also known as the logistic function takes in any values and outputs it to be between 0 and 1.

The main point to notice here is that whatever value is put into the sigmoid function we will always get a value between 0 and 1 and when z tends to negative infinity, the probability approaches zero.
Here, z = mx+b



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

Sigmoid Function

Data Split:

The dataset is split in divisions of 80 percent for training, 10 percent for testing and 10 percent for validation. Here is the code snippet.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split( x, y, test_size = 0.2)
x_test, x_val, y_test, y_val = train_test_split( x_test, y_test, test_size = 0.5)

x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T
x_val = x_val.T
y_val = y_val.T

print("x train: ", x_train.shape)
print("x test: ", x_test.shape)
print("y train: ", y_train.shape)
print("y test: ", y_test.shape)
print("x Val: ", x_val.shape)
print("y Val: ", y_val.shape)
```

```
x train:  (31, 455)
x test:  (31, 57)
y train:  (455,)
y test:  (57,)
x Val:  (31, 57)
y Val:  (57,)
```

X contains the features and Y contains target variable.

x_train – Training Feature set

y_train – Training Label

x_test - Testing Feature set

y_test – Testing Label

x_val - Validation Feature set

y_val – Validation Label

Training and Predicting

- Let's use the logistic regression to train the model. We use the sigmoid function to describe the probability that the sample belongs to one of the two classes.
- We start with initializing weights and bias for the sigmoid function.
- When implementing logistic regression, first step is to learn parameters w and b so that z is approximately equal to the test target. In order to learn the parameters w and b, we need to define a cost function which we would use to train the logistic regression model. This can be

achieved by updating the weight and bias term after every iteration. A cost function estimates how good or bad our model is in predicting the known output in general.

- So, it important to minimize the cost function for minimizing errors across the training data set to find w and b. We achieve the value of the parameters using gradient descent technique.
- Once the model has been trained we go ahead with predicting the x_test values where a sigmoid value less than 0.5 is categorized to class 0(Benign) and value greater then 0.5 is categorized to class 1(Malignant).

```
for i in range(p.shape[1]):
    if p[0, i]<= 0.5:
        Y_prediction[0, i] = 0
    else:
        Y_prediction[0, i] = 1

return Y_prediction
```

- Before we predict the values on the test dataset, we tune the hyperparameters such as the learning rate and number of iterations on the validation set. The logistic regression model in my case achieved highest accuracy with a learning rate of 0.15 and 100 iterations.

Performance Measure

To evaluate the performance of a classification model we use the confusion matrix to calculate the accuracy, recall and precision.

```
print("Recall:",TP/(TP+FN))
print("Accuracy:",(TP+TN)/(TP+TN+FP+FN))
print("Precision:",(TP)/(TP+FP))
return (TP+TN)/(TP+TN+FP+FN)
```

```
Recall: 0.9565217391304348
Accuracy: 0.9649122807017544
Precision: 0.9565217391304348
```

Recall : 95.65%
Accuracy: 96.49%
Recall: 95.65%

Results

Cost Function vs Number of Iterations on Validation Set

```
logistic_regression_val(x_train, y_train, x_val, y_val, learning_rate = 0.4, num_iterations = 150)
```

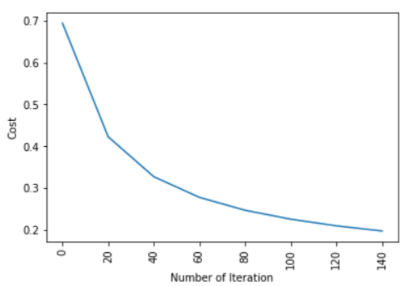
```
Cost after  0 iterations -  0.693728
Cost after  20 iterations -  0.516369
Cost after  40 iterations -  0.423702
Cost after  60 iterations -  0.366653
```

Accuracy: 87.71%
Learning rate: 0.4
Number of Iterations: 150

```
logistic_regression_val(x_train, y_train, x_val, y_val, learning_rate = 0.8, num_iterations = 150)
```

```
Cost after  0 iterations -  0.693728
Cost after  20 iterations -  0.422652
Cost after  40 iterations -  0.327191
Cost after  60 iterations -  0.277836
Cost after  80 iterations -  0.247081
Cost after  100 iterations -  0.225739
Cost after  120 iterations -  0.209866
Cost after  140 iterations -  0.197476
```

Accuracy: 89.47%
Learning rate: 0.8
Number of Iterations: 150

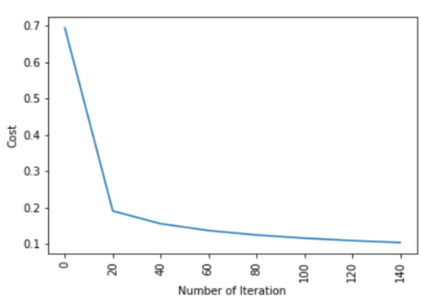



```
train accuracy: 95.38461538461539 %
test accuracy: 89.47368421052632 %
```

```
logistic_regression_val(x_train, y_train, x_val, y_val, learning_rate = 4.5, num_iterations = 150)
```

```
Cost after  0 iterations -  0.693728
Cost after  20 iterations -  0.190562
Cost after  40 iterations -  0.155554
Cost after  60 iterations -  0.136788
Cost after  80 iterations -  0.124558
Cost after  100 iterations -  0.115762
Cost after  120 iterations -  0.109048
Cost after  140 iterations -  0.103711
```

Here, we can see that with
Learning rate = 4.5 and
Number of Iterations= 150
we get the highest accuracy

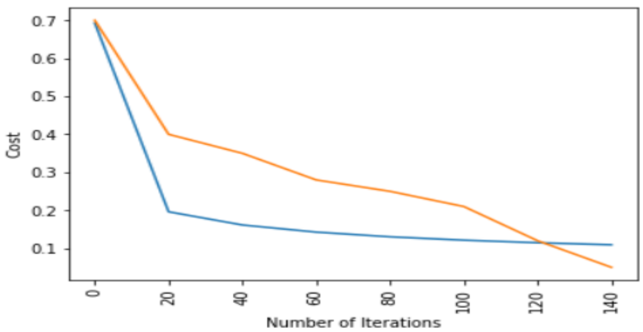



```
train accuracy: 97.8021978021978 %
test accuracy: 92.98245614035088 %
```

Cost Function vs Number of Iterations on Validation Set

```
y_pred_test = logistic_regression(x_train, y_train, x_test, y_test, learning_rate = 4.5, num_iterations = 150)
```

```
Cost after  0 iterations is  0.692903
Cost after  20 iterations is  0.196138
Cost after  40 iterations is  0.161436
Cost after  60 iterations is  0.142640
Cost after  80 iterations is  0.130326
Cost after  100 iterations is  0.121458
Cost after  120 iterations is  0.114693
Cost after  140 iterations is  0.109327
```




```
train accuracy is : 97.8021978021978 %
test accuracy is : 96.49122807017544 %
```

As we can see, we get an accuracy of 96.49% with a
Learning rate of 4.5 and
Number of Iterations of 150

Conclusion

Logistic Regression was used in this problem as we had a binary classification problem to predict whether a tumor is Malignant or Benign. This was achieved by using a sigmoid function that outputs the value between the range 0 and 1. Gradient descent was used to measure the values of the parameter and update it after every iteration. The hyper parameter in this algorithm i.e., learning rate and number of iterations were tuned using the validation get and got the highest accurate rate at (learning rate= 4.5 and number of iterations=150).
Finally, we got the accuracy of 91.22% on the test set after training the model using the hyperparameters from the validation set.

References:

https://www.youtube.com/watch?v=zM4VZR0px8E&t=605s
https://towardsdatascience.com/logistic-regression-python-7c451928efee
https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8
https://www.geeksforgeeks.org/
https://medium.com/technology-nineleaps/logistic-regression-gradient-descent-optimization-part-1-ed320325a67e