Machine Learning
January, 2025
Application of Logistic Regression from Scratch

Khalid Mammadov

Part 1: Loading data

In this part, I downloaded the data with pandas and copied the data so that I can work on copied data instead of original onw. The reason behind it because in case of performing something wring on data, I can just go back and copy the original one instead of loading it again and again. Here is how final dataset looks:

Out[75]:

	exam_1	exam_2	admitted
0	34.623660	78.024693	0
1	30.286711	43.894998	0
2	35.847409	72.902198	0
3	60.182599	86.308552	1
4	79.032736	75.344376	1

After loading the data, I put first two cloumns (features) into X array and last column (label) into y array. Then, by using MinMaxScaler from sklearn, I scale my values. Here is the formula for this scaling (https://en.wikipedia.org/wiki/Feature_scaling#Rescaling_(min-max_normalization)):

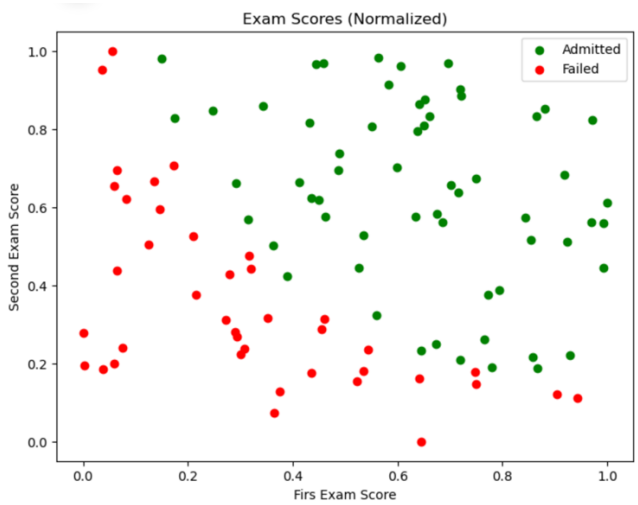
$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

After scaling, my features look like:

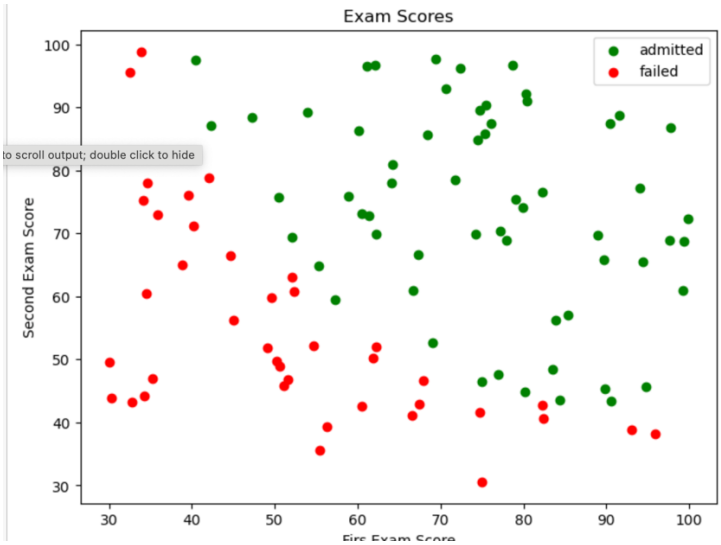
```
In [142]: X_scaled

Out[142]: array([[0.06542784, 0.69465488],
                 [0.00326632, 0.19470455],
                 [0.08296784, 0.61961779],
                 [0.43176427, 0.81600135],
                 [0.7019434 , 0.65539214],
                 [0.2153456 , 0.37665959],
                 [0.44500891, 0.96545859],
                 [0.64449684, 0.23365526],
                 [0.65989108, 0.83229079],
                 [0.77934283, 0.18940757],
```

Part 2: Visualization

In this part, I need to visualize first exam score vs second exam score (I used matplotlib). Here is how my scatter plot looks:



In addition, I also added visulaization for not normalized data. Here, how it looks:



From here, we see that a line (decision boundry) can separate this two sets. Even though it will not be perfect, it will be good one. Let us now go to implementation part to see.

Part 3: Implementation of Logistic Regression from scratch

To implement logistic regression, first we need to define a function to calculate sigmoid. Here is the formula:

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

Sigmoid give us the probability of a positive label, and then we check; if this probability is >= 0.5, we classify it as 1, otherwise 0.

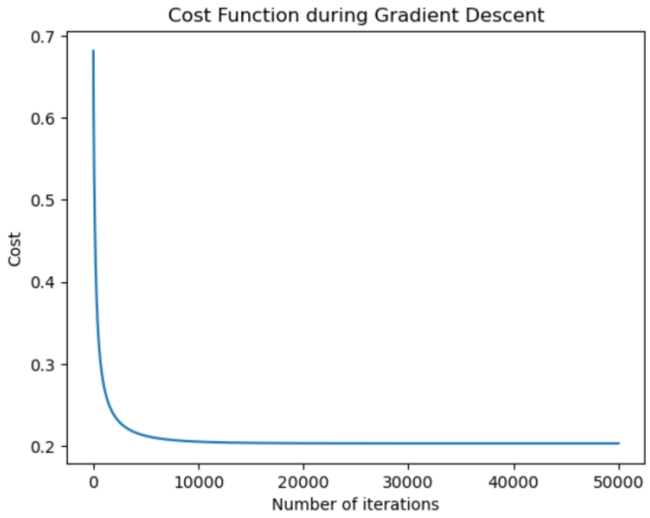
Then, we define a function to compute the cost function. Here is the formula to compute it:

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

Next step is to perfom gradient descent. For it, we define a gradient descent function. First, we define 'm' which equals to the number of instances. Then, I create a list to record cost_history so that I can plot it. When it comes to funtion, it makes prediction with sigmoid, calculates gradients based on these predictions and update thetas. It is repeated several times (in our case it is 50000). At the end of the function, I calculate the the cost and append it to the list. It returns best thetas and cost hisgtroy list.

To start to train, I create a matrix consists of 1s as a first column, and features in the second and third colmns. Then, we initialize thetas. As we have two features, we need three weigths. I initialized all to 0s. Next, hyperparameters, learning rate (0.1) and number of iterations (50000), are assigned. After all, we can train the model by calling the function created (gradient_descent). My thetas based on gradient descent is: [-11.97 13.48 12.86].

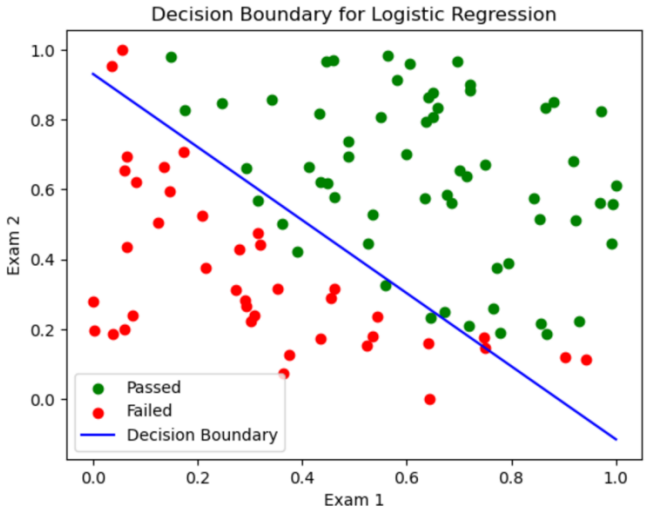
After implementation, I plot the cost function to see how it changes. From the graph, we see that it decreases then, stays the same when it is about 0.2.



When it comes to decision boundary, first I created array for independent values. It is exam_1 values. Then, by using this equation:

$$\text{theta\_0} + \text{theta\_1} * x1 + \text{theta\_2} * x2 = 0$$

we calculate x2 which is exam_2 scores in our case. Then, we lost the graph:



This line can distinguish red and green data points which was actually our main aim.

To check how my predictions are, I checked two cases. First case:

- I calculated the predictions on the whole training dataset to calculate the accuracy. For this purpose, I have predict_train function, which takes all features of train dataset and optimal thetas, calculate sigmoid and classify it based on threshold 0.5. To calculate accuracy, I use accuracy_score() from sklearn. Accuracy of training dataset is 89%.

And, the second case:

- I calculated the predictions on two new data points that model has never seen. For this purpose, I have predict_tests function, which takes optimal thetas and features of these two data points, scale them, add bias (1s), calculate sigmoid and classify it based on threshold 0.5. When I made the prediction, here is the result → [55 70]: 1, [40 60]: 0. Both predictions are the same with true labels.

Part 3: Logistic Regression using Library

To compare my results, I also use library from sklearn.linear_model to calculate logistic regression. First, I again downloaded the dataset, split features and labels, scale features (I did it for having the whole code in one cell). Then, I fit the model with features and label; then, predict the values by using “predict” function of the model. Again, I check train dataset → the accuracy is 93% but mine was 83%. Then, I make the predictions on the same data points and get the same predictions with mine. The difference between gradient descent and library accuracies might be because of:

- Hyperparameters I initiliazed
- Library uses L2 regularization by default. In addition to prevent overfitting, regularization can improve the model's performance, especially on small (which is the case in our dataset) datasets.
- Library may use different ways to compute the sigmoid or cost function more efficiently