Logistic Regression for Two Class Classification Problem in Python

Prudhveer Reddy Kankar
50320121
prudhvee@*buffalo.edu*

Abstract

In this project we use the Logistic Regression, a machine learning algorithm to train and test a classification problem. A cancer dataset (Wisconsin Diagnostic Breast Cancer) which tells whether the cancer is benign or malignant based on a number of features is used as a classification problem that needs to be solved. This project report provides a brief explanation of how to use a logistic regression algorithm for solving a two class classification problem.

1 Introduction

There are basically two types of problems i.e.. Regression Problem and Classification Problem. Regression problems basically deals with predicting continuous valued outputs , where as classification problems deal with discrete valued outputs (0,1,2 etc). Linear regression is one of the popular algorithms that is used to solve the classification problems. In the below subsections, we describe briefly all the important concepts of Linear Regression.

1.1 Linear vs Logistic Regression

There are two main types of regression algorithms. They are Linear regression and Logistic regression. The key difference between both these algorithms is their implementation process and mainly the type of problems that they solve. Linear regression deals with regression problems where as logistic regression deals with classification problems. Though we can use linear regression to solve classification problems by using a threshold output classifier (by setting y=1 if h(x) >= 0.5……), it is not at all recommended. Because there might be cases where the predictions made using linear regression exceed 0 or 1 ([1]depending on the number of classes).Hence we use logistic regression.

In logistic regression we apply a sigmoid function to the hypothesis which makes sure that the predictions made by the algorithm stays between 0 or 1 (again depending on the number of classes, if it is a three case then we have to consider 2 also; similarly 3 , 4, 5 .. depending on the multi class problem given). The sigmoid function and the output graph is given in the below figure.

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

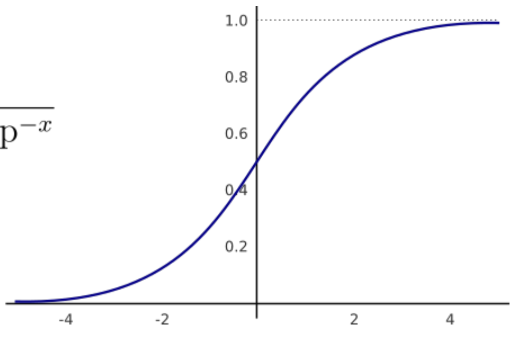


Figure 1: Sigmoid function and Logistic Regression output value graph.

1

As we can see from the above graph the output range of the predictions is limited to values between 0 and 1. This is the example of two class problem. We then set the threshold values (predicted value>=0.5 or predicted value < 0.5) to predict the to which class the prediction belongs to.

1.1.1 Cost function

Linear regression In Logistic re[2]gression we use a log based cost function as we need the function to be convex. Because for the gradient descent to compute global minimum, it is important for the curve to be convex. The cost function of logistic regression is shown in the below figure.

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

The above formula can be simplified to the below form:

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

Where depending on the y value the cost functions conditions will remain the same.

1.1.2 Gradient descent

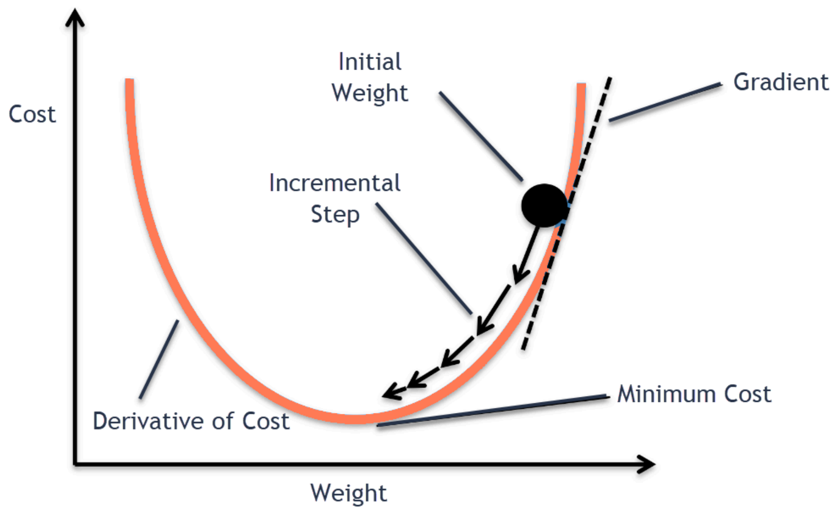


Figure 2: Gradient Descent.

We use Gradient descent finding the minimum of a function. In our machine learning algorithm, we use gradient descent to continuously update the parameters of our model. The formula is given by

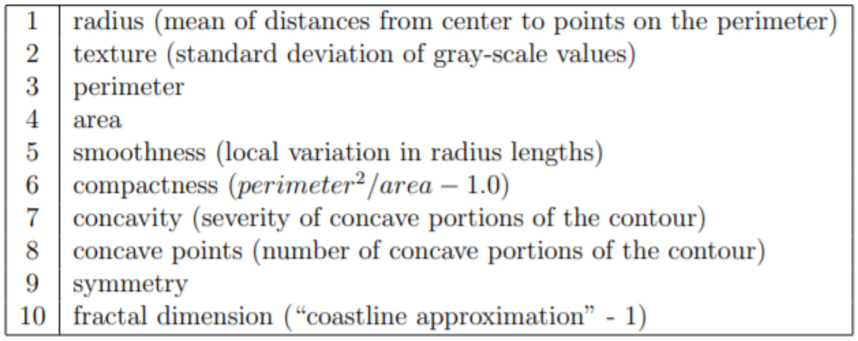
$$\begin{array}{l} \text{repeat until convergence } \{ \\ \quad \theta_j := \theta_j - \alpha \dfrac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \\ \} \end{array}$$

2

Where j can be 0 and 1. Alpha is the learning rate. We have the select alpha in such a way that it has to find out the global minimum without overshooting or taking more that average time. In our problem we update weights and bias.

2 Dataset

The dataset provided to us is the Wisconsin Diagnostic Breast Cancer (WDBC) dataset which is of type two class classification problem. The data set tells whether the cancer is benign or malignant based on the features in 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. Computed features describes the following characteristics of the cell nuclei present in the image:



1	radius (mean of distances from center to points on the perimeter)
2	texture (standard deviation of gray-scale values)
3	perimeter
4	area
5	smoothness (local variation in radius lengths)
6	compactness ($perimeter^2/area - 1.0$)
7	concavity (severity of concave portions of the contour)
8	concave points (number of concave portions of the contour)
9	symmetry
10	fractal dimension ("coastline approximation" - 1)

Figure 3: Dataset

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.

3 Preprocessing

Reading:

I used the pandas library to input the data from the local machine by using inbuilt readcsv function. The first column in the data consists of ID's which are of no use for computing the outputs, hence they are not considered. Instead of dropping the values I simply did not assign the column ID by using loc function while aligning values to either X or Y.

The Benign and Malignant values are set as B and M respectively in the dataset. I have converted B and M to 0 and 1 respectively by using the replace function.

Partitioning:

The entire data is partitioned into three parts: training , validation and test. They contain 80%, 10% and 10% of the total data respectively. This was done using train_test_split function from sklearn.model_selection library.

Scaling:

I used the scikitlear[3]n library present in the standard scaler for normalizing the data.This helps in removing the mean and scaling to unit variance.

3

4 Architecture

The overall architecture can be visualized as shown in the below diagram.

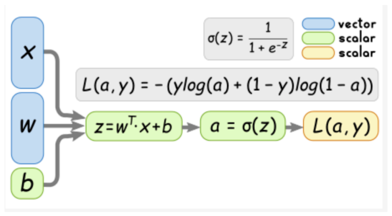

$\sigma(z) = \frac{1}{1+e^{-z}}$
vector
scalar
scalar
x
w
b
$L(a,y) = -(y\log(a) + (1-y)\log(1-a))$
$z = w^T \cdot x + b$
$a = \sigma(z)$
$L(a,y)$

Figure 5: Logistic Regression computational graph

Where L is the loss function

The architecture of Logistic regression algorithm used is this project is explain briefly in a stepwise manner:

Step 1. Preprocessing the Data:
The dataset given has to be preprocessed as explained in section 3. The data has to be divided so that we get test data and train data on which the testing and training operations can be performed.

Step 2. Sigmoid Function:
The data is converted into a matrix form with weights and X (column values) as shown in the below figure:

$$h_\theta(x) = \quad \theta^T x$$

The sigmoid function is now applied to the above formula. The formula for sigmoid is given below:

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

The sigmoid function makes sure that the output is in the range of 0 and 1 only.

Step 3. Cost Function:
Linear regression In Logistic regression we use a log based cost function as we need the function to be convex. Because for the gradient descent to compute global minimum, it is important for the curve to be convex. The cost function of logistic regression is shown in the below figure.

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

The above formula can be simp[4]lified to the below form:

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

Where depending on the y value (0 and 1) the cost functions conditions will remain the same.
The final cost function for logistic regression can be calculated using the below formula :

4

Logistic regression cost function

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

Step 4. Gradient Descent:
We use gradient descent method to mini[5]mize the cost function. The weights and bias are computed by the below formula and are simultaneously updated.

Want $\min_\theta J(\theta)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$\}$ (simultaneously update all θ_j)

Similarly, for all parameters

$$\boxed{w_i = w_i - \alpha \frac{\partial L}{\partial w_i}}$$ $i = 1, 2, \cdots, m$

m = no of parameters

$$\boxed{b = b - \alpha \frac{\partial L}{\partial b}}$$

Where, $\frac{\partial L}{\partial b} = (a - y)$

Alpha is the learning rate. We have to choose the learning rate in such a way that it will neither cause overshooting condition nor take large time to compute the global minimum.

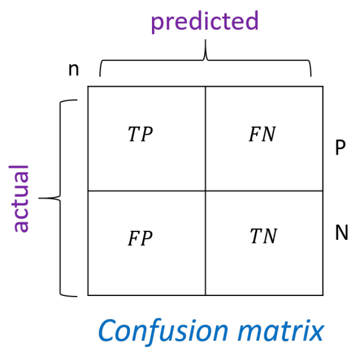
Step 5. Assigning Threshold Values and getting the outputs:

An epoch is defined as one iteration through the dataset. For each iteration, we update the weights and bias.We also store the costs and predictions for each epoch in the form of arrays.
We then calculate the predictions using the weights and the biases we got.
The threshold values are set in such a way that
If (sigmoid(h(x)) >= 0.5) then y is set to 1.
If (sigmoid(h(x)) <0.5) then y is set to 0.
In this way we get the output in the form of 0’s and 1’s.

Step 6. Confusion Matrix:

A confusion matrix is also known as an error matrix. It is a visualization table that allows us to compute the performance of an algorithm.

5



6

Figure 4: Confusion Matrix

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.We generate a confusion matrix to the test data and compute accuracy, precision and recall by using the below formulas.

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

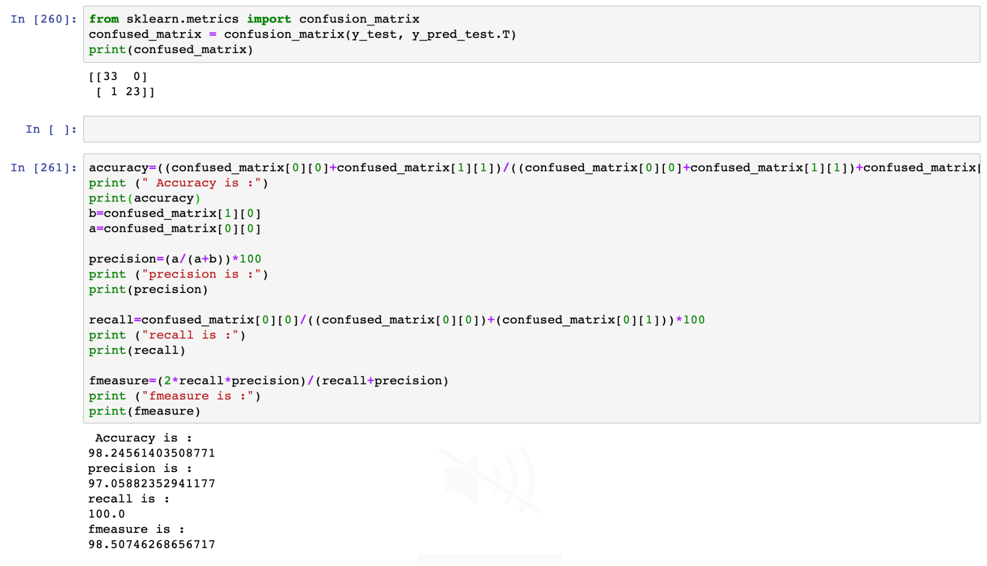
$$Precision = \frac{TP}{TP + FP}$$

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

6

5 Results

After training the model, we apply the model on the testing data and check the performance of the model using various metrics.



```
In [260]: from sklearn.metrics import confusion_matrix
          confused_matrix = confusion_matrix(y_test, y_pred_test.T)
          print(confused_matrix)

          [[33  0]
           [ 1 23]]

In [ ]:

In [261]: accuracy=((confused_matrix[0][0]+confused_matrix[1][1])/((confused_matrix[0][0]+confused_matrix[1][1])+confused_matrix
          print (" Accuracy is :")
          print(accuracy)
          b=confused_matrix[1][0]
          a=confused_matrix[0][0]

          precision=(a/(a+b))*100
          print ("precision is :")
          print(precision)

          recall=confused_matrix[0][0]/((confused_matrix[0][0])+(confused_matrix[0][1]))*100
          print ("recall is :")
          print(recall)

          fmeasure=(2*recall*precision)/(recall+precision)
          print ("fmeasure is :")
          print(fmeasure)

           Accuracy is :
          98.24561403508771
          precision is :
          97.05882352941177
          recall is :
          100.0
          fmeasure is :
          98.50746268656717
```

Figure 5: Performance Metrics Calculation.

I have used confusion matrix to compute the performance metrics.

<u>**Training Data Graphs:**</u>

The cost function vs number of iterations graph for the number of iterations for training data is shown below:

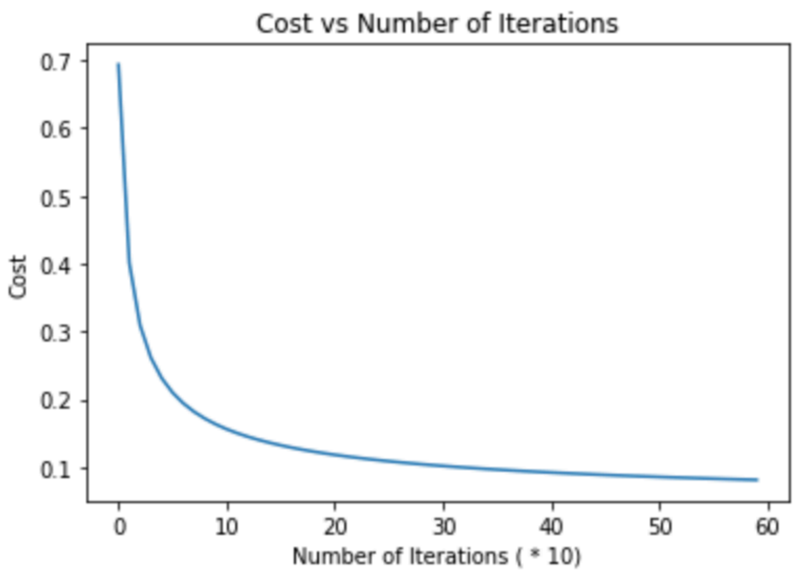


Figure 6: Cost vs Number of Iterations

The cost function vs number of iterations graph for the number of iterations for training data is shown below:

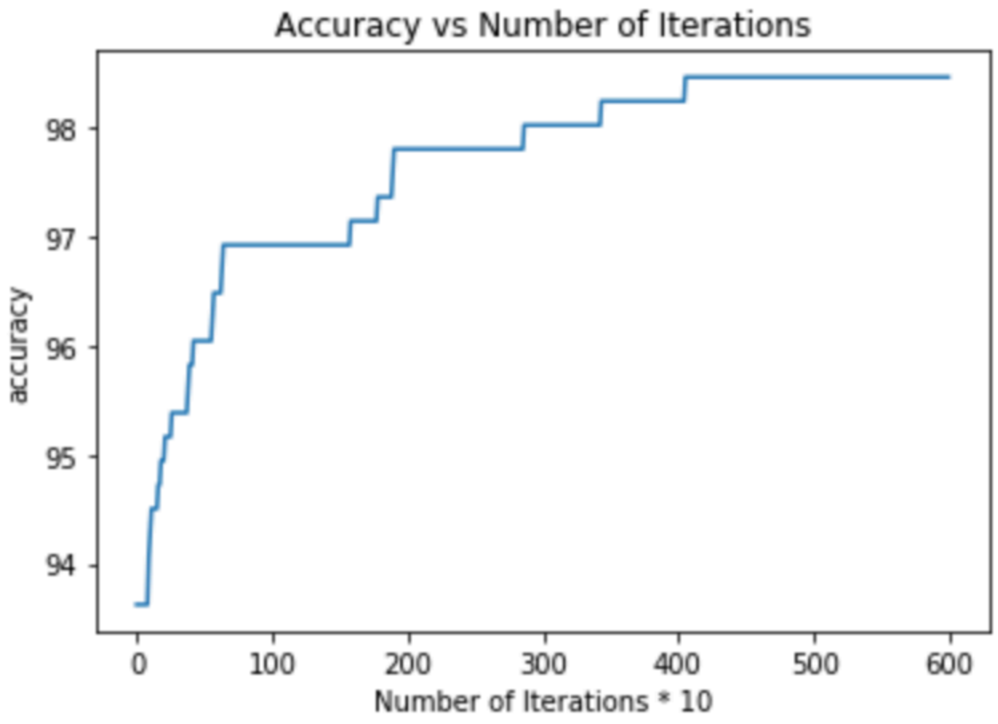


Figure 7: Accuracy vs Number of Iterations

Validation Data Graphs:
The below are the graphs for validation sets for 600 epochs.
1) For learning rate alpha = 0.03 , the cost after 590 iteration is 0.055893. The graph is shown below :

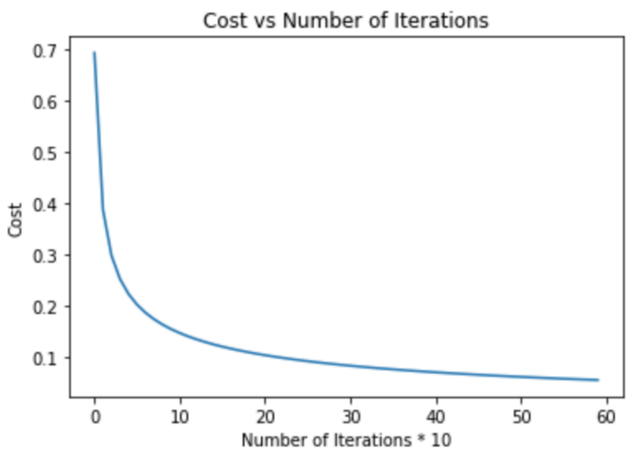


Figure 8: Graph for Alpha = 0.03.

2) For learning rate alpha = 0.04 , the cost after 590 iteration is 0.046295. The graph is shown below :

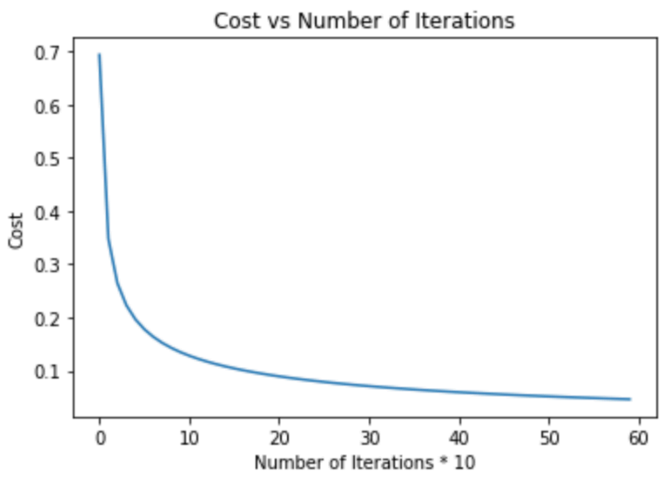


Figure 9: Graph for Alpha = 0.04.

3) For learning rate alpha = 0.06 , the cost after 590 iteration is 0.034903. The graph is shown below :[8]

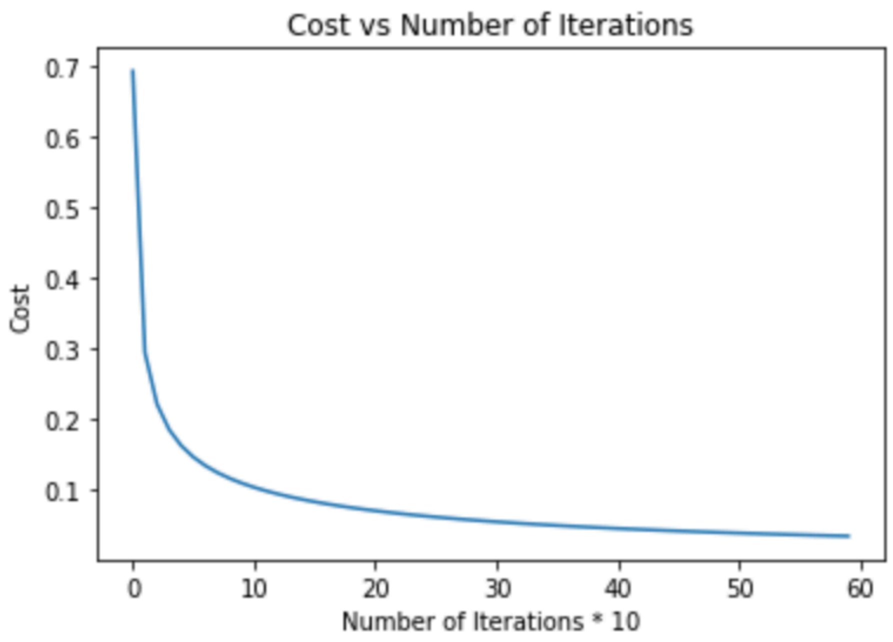


Figure 10: Graph for Alpha = 0.06.

8

6 Conclusions

Successfully completed training, validation and testing on the given Wisconsin Diagnostic Breast Cancer cancer dataset. I have received the following values for evaluation metrics.

Accuracy : 98.24561403508771

Precision : 97.05882352941177

Recall is : 100.0

Fmeasure is : 98.50746268656717

7 References

[1] Logistic Regression from Scratch https://towardsdatascience.com/logistic-regression-from-very-scratch-ea914961f320

[2] The cost function of logistic regression https://www.internalpointers.com/post/cost-function-logistic-regression

[3] Confusion Matrix https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

[4] Logistic Regression overview https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc.

[5] Derivatives for logistic regression-step by step http://ronny.rest/blog/post_2017_08_12_logistic_regression_derivative/

[6] Gradient Descent https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

[7] sklearn.metrics.confusion_matrix https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html