

Report
on

Logistic Regression
04-Jan-2019

Submitted by

Love Prakash



DronaMaps Private Limited

Contents

1	Introduction:	1
2	Simple Logistic Regression:	1
2.1	Model	1
2.2	Sigmoid Function	2
3	Analysis of the hypothesis	2
4	Types of Logistic Regression:	3
4.1	Binary Logistic Regression	3
4.2	Multinomial Logistic Regression	3
4.3	Ordinal Logistic Regression	3
5	Decision Boundary	3
6	Cost Function	3
7	Why this Cost Function:	5
8	Deriving the formula for Gradient Descent Algorithm	6
9	Python Implementation:	7
10	Cost vs Number of Iterations:	8

1 Introduction:

Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical.

For example,

To predict whether an email is spam (1) or (0) Whether the tumor is malignant (1) or not (0) Consider a scenario where we need to classify whether an email is spam or not. If[?] we use linear regression for this problem, there is a need for setting up a threshold based on which classification can be done. Say if the actual class is malignant, predicted continuous value 0.4 and the threshold value is 0.5, the data point will be classified as not malignant which can lead to serious consequence in real time.

From this example, it can be inferred that linear regression is not suitable for classification problem. Linear regression is unbounded, and this brings logistic regression into picture. Their value strictly ranges from 0 to 1.

2 Simple Logistic Regression:

2.1 Model

Output = 0 or 1

Hypothesis $\Rightarrow Z = WX + B$

$h(x) = \text{sigmoid}(Z)$

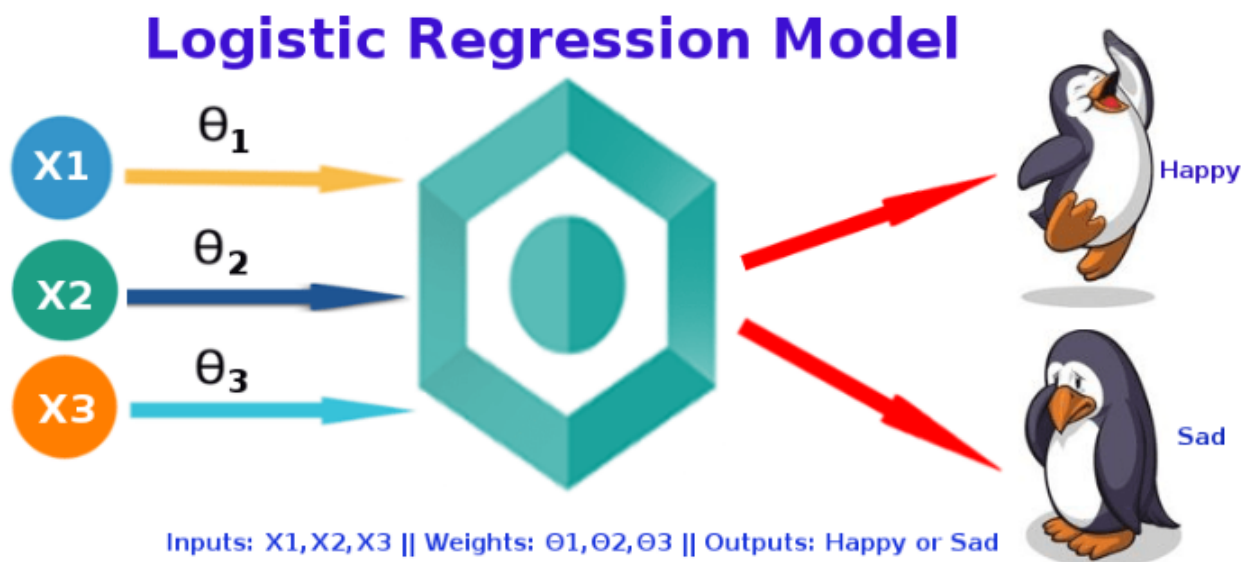


Figure 1: Logistic Regression Model

2.2 Sigmoid Function

If 'Z' goes to infinity, Y(predicted) will become 1 and if 'Z' goes to negative infinity, Y(predicted) will become 0.

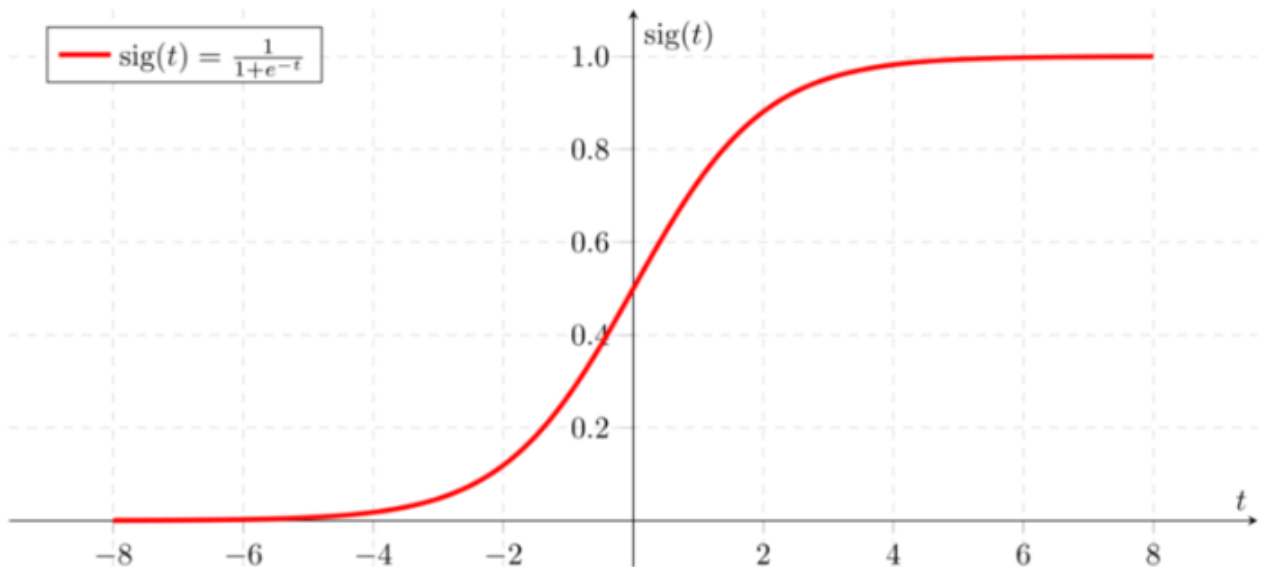


Figure 2: Sigmoid Activation Function

3 Analysis of the hypothesis

The output from the hypothesis is the estimated probability. This is used to infer how confident can predicted value be actual value when given an input X. Consider the below example,

$X = [x_0 \ x_1] = [1 \text{ IP-Address}]$

Based on the x_1 value, let's say we obtained the estimated probability to be 0.8. This tells that there is 80 percent chance that an email will be spam.

$$h_{\theta}(x) = P(Y=1|X; \theta)$$

Probability that $Y=1$ given X which is parameterized by ' θ '.

$$P(Y=1|X; \theta) + P(Y=0|X; \theta) = 1$$

$$P(Y=0|X; \theta) = 1 - P(Y=1|X; \theta)$$

Figure 3: Exact Features

This justifies the name ‘logistic regression’. Data is fit into linear regression model, which then be acted upon by a logistic function predicting the target categorical dependent variable.

4 Types of Logistic Regression:

4.1 Binary Logistic Regression

The categorical response has only two 2 possible outcomes. Example: Spam or Not

4.2 Multinomial Logistic Regression

Three or more categories without ordering. Example: Predicting which food is preferred more (Veg, Non-Veg, Vegan)

4.3 Ordinal Logistic Regression

Three or more categories with ordering. Example: Movie rating from 1 to 5

5 Decision Boundary

To predict which class a data belongs, a threshold can be set. Based upon this threshold, the obtained estimated probability is classified into classes.

Say, if $\text{predicted_value} > 0.5$, then *classify email as spam else as not spam*.

Decision boundary can be linear or non-linear. Polynomial order can be increased to get complex decision boundary.

6 Cost Function

Why cost function which has been used for linear can not be used for logistic?

Linear regression uses mean squared error as its cost function. If this is used for logistic regression, then it will be a non-convex function of parameters (θ). Gradient descent will converge into global minimum only if the function is convex.

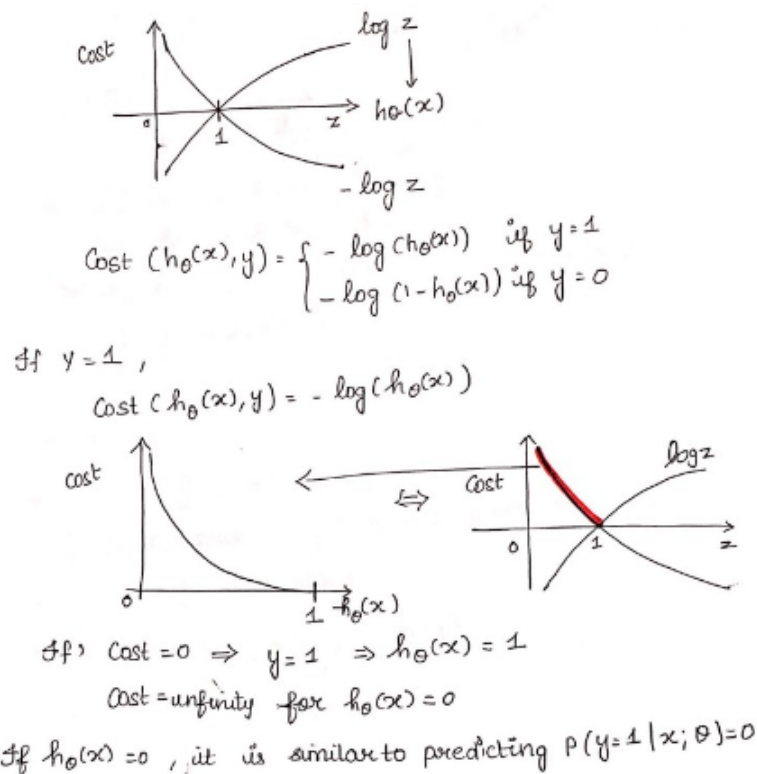


Figure 4: Cost function Part1

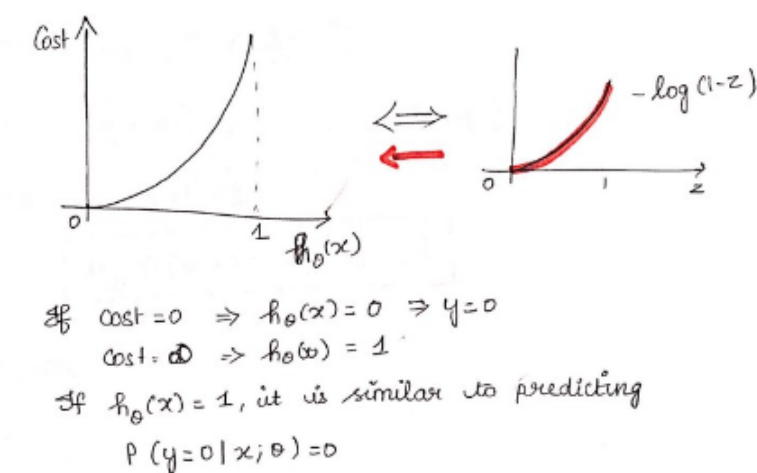


Figure 5: Cost function Part2

Let us consider,

$$\begin{aligned}
 * \quad \hat{y} &= P(y=1|x) \\
 \hat{y} &\text{ is the probability that } y=1, \text{ given } x \\
 * \quad 1-\hat{y} &= P(y=0|x) \\
 * \quad P(y|x) &= \hat{y}^y \cdot (1-\hat{y})^{(1-y)} \\
 \text{If } y=1 &\Rightarrow P(y|x) = \hat{y}
 \end{aligned}$$

Figure 6: S
implified Cost Function

7 Why this Cost Function:

$$\begin{aligned}
 &\Rightarrow \log(\hat{y}^y \cdot (1-\hat{y})^{(1-y)}) \\
 &\Rightarrow y \log \hat{y} + (1-y) \log (1-\hat{y}) \\
 &\Rightarrow -L(\hat{y}, y) \\
 &\boxed{\log P(y|x) = -L(\hat{y}, y)}
 \end{aligned}$$

Figure 7: M
aximum likelihood explanation part2

figure This negative function is because when we train, we need to maximize the probability by minimizing loss function. Decreasing the cost will increase the maximum likelihood assuming that samples are drawn from an identically independent distribution.

$$\frac{\partial L}{\partial w_1} = \left(\left(\frac{-y}{a} + \frac{(1-y)}{1-a} \right) \cdot (a)(1-a) \right) \cdot x_1$$

$$= (a-y) \cdot x_1$$

update for w_1 ,

$$\frac{\partial L}{\partial w_1} = (a-y) \cdot x_1$$

$$\text{here, } (a-y) = \frac{\partial L}{\partial z}$$

$$w_1 = w_1 - \alpha \frac{\partial L}{\partial w_1}$$

Similarly, for all parameters

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

$m = \text{no. of parameters}$

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

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

Figure 9: Gradient Descent Part2

9 Python Implementation:

```
def weightInitialization(n_features):
    w = np.zeros((1, n_features))
    b = 0
    return w, b

def sigmoid_activation(result):
    final_result = 1/(1+np.exp(-result))
    return final_result

def model_optimize(w, b, X, Y):
    m = X.shape[0]

    #Prediction
    final_result = sigmoid_activation(np.dot(w, X.T) + b)
    Y_T = Y.T
    cost = (-1/m) * (np.sum((Y_T * np.log(final_result)) + ((1-Y_T) * (np.log
```

```
#

#Gradient calculation
dw = (1/m)*(np.dot(X.T, (final_result-Y.T).T))
db = (1/m)*(np.sum(final_result-Y.T))

grads = {"dw": dw, "db": db}

return grads, cost
def model_predict(w, b, X, Y, learning_rate, no_iterations):
    costs = []
    for i in range(no_iterations):
        #
        grads, cost = model_optimize(w,b,X,Y)
        #
        dw = grads["dw"]
        db = grads["db"]
        #weight update
        w = w - (learning_rate * (dw.T))
        b = b - (learning_rate * db)
        #

    if (i % 100 == 0):
        costs.append(cost)
    #print("Cost after %i iteration is %f" %(i, cost))

#final parameters
coeff = {"w": w, "b": b}
gradient = {"dw": dw, "db": db}

return coeff, gradient, costs
def predict(final_pred, m):
    y_pred = np.zeros((1,m))
    for i in range(final_pred.shape[1]):
        if final_pred[0][i] > 0.5:
            y_pred[0][i] = 1
    return y_pred
\newpage
```

10 Cost vs Number of Iterations:

Train and test accuracy of the system is 100 percent.

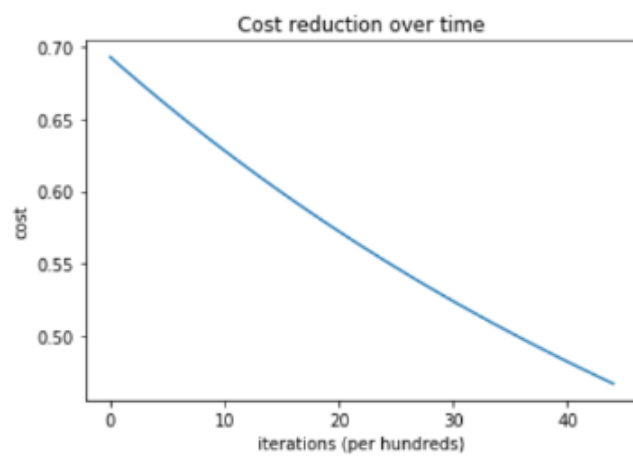


Figure 10: Cost Reduction

This implementation is for binary logistic regression. For data with more than 2 classes, softmax regression has to be used.