Logistic Regression — Detailed Overview

Saishruthi Swaminathan Follow
Mar 15, 2018 · 5 min read

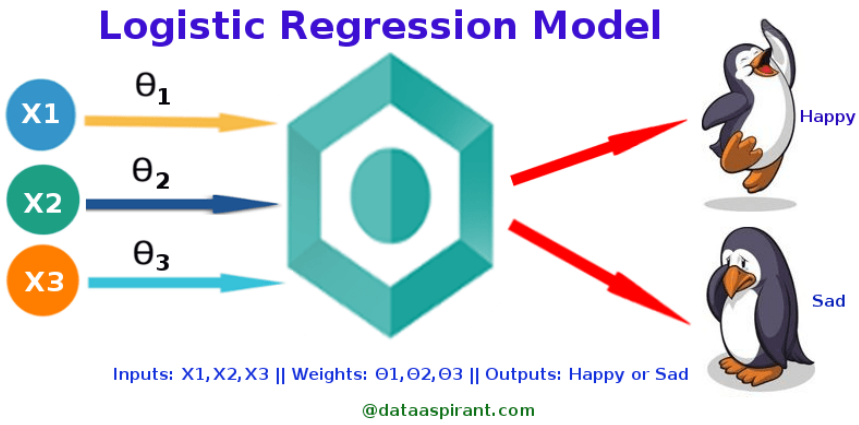

Figure 1: Logistic Regression Model (Source:http://dataaspirant.com/2017/03/02/how-logistic-regression-model-works/)

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.

Simple Logistic Regression

(Full Source code: https://github.com/SSaishruthi/LogisticRegression_Vectorized_Implementation/blob/master/Logistic_Regression.ipynb)

Model

Output = 0 or 1

Hypothesis => Z = WX + B

hΘ(x) = sigmoid (Z)

Sigmoid Function

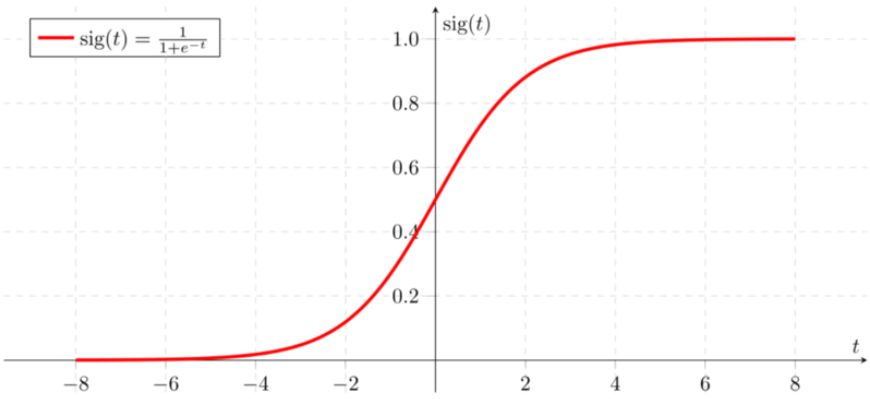

Figure 2: Sigmoid Activation Function

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

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 = [x0 x1] = [1 IP-Address]

Based on the x1 value, let's say we obtained the estimated probability to be 0.8. This tells that there is 80% chance that an email will be spam.

Mathematically this can be written as,

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

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

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

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

Figure 3: Mathematical Representation

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.

Types of Logistic Regression

1. Binary Logistic Regression

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

2. Multinomial Logistic Regression

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

3. Ordinal Logistic Regression

Three or more categories with ordering. Example: Movie rating from 1 to 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 predicted_value $\geq$ 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.

Cost Function

$$\text{Cost}(h_\Theta(x), Y(\text{actual})) = -\log(h_\Theta(x)) \text{ if } y=1$$
$$-\log(1-h_\Theta(x)) \text{ if } y=0$$

Figure 4: Cost Function of Logistic Regression

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 (theta). Gradient descent will converge into global minimum only if the function is convex.

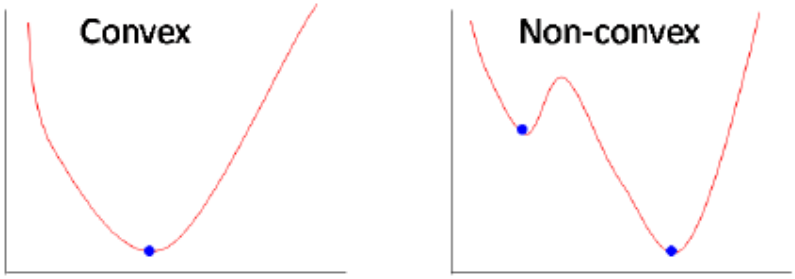

Figure 5: Convex and non-convex cost function

Cost function explanation

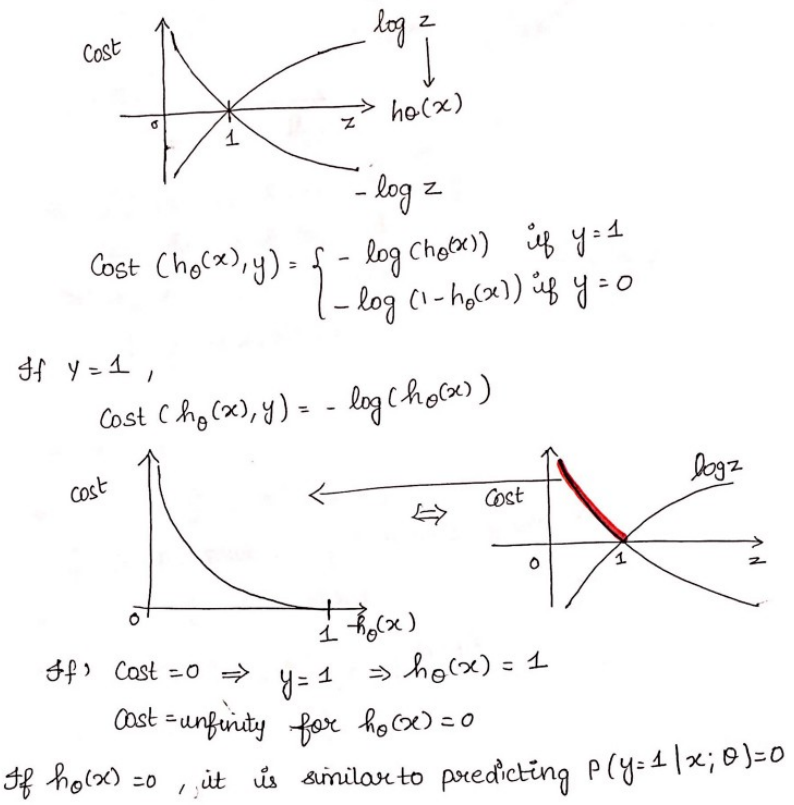

Figure 6: Cost Function part 1

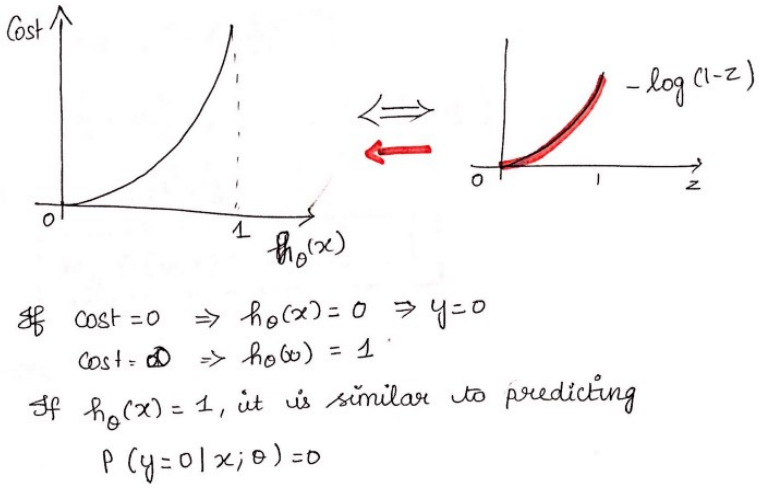

Figure 7: Cost Function part 2

Simplified cost function

$Cost(h_\Theta(x), y) = -y \log(h_\Theta(x)) - (1-y) \log(1- h_\Theta(x))$

If y = 1, (1-y) term will become zero, therefore – log ($h_\Theta(x)$) alone will be present

If y = 0, (y) term will become zero, therefore – log (1- $h_\Theta(x)$) alone will be present

Figure 8: Simplified Cost Function

Why this cost function?

Let us consider,

* $\hat{y} = P(y=1|x)$

$\hat{y}$ is the probability that y = 1, given x

* $1-\hat{y} = P(y=0|x)$

* $P(y|x) = \hat{y}^{y} \cdot (1-\hat{y})^{(1-y)}$

If $y=1 \Rightarrow P(y|x) = \hat{y}$

Figure 9: Maximum Likelihood Explanation part-1

$\Rightarrow \log(\hat{y}^{y} \cdot (1-\hat{y})^{(1-y)})$

$\Rightarrow y \log \hat{y} + (1-y) \log (1-\hat{y})$

$\Rightarrow -L(\hat{y}, y)$

$\log P(y|x) = -L(\hat{y}, y)$

Figure 10: Maximum Likelihood Explanation part-2

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.

Deriving the formula for Gradient Descent Algorithm

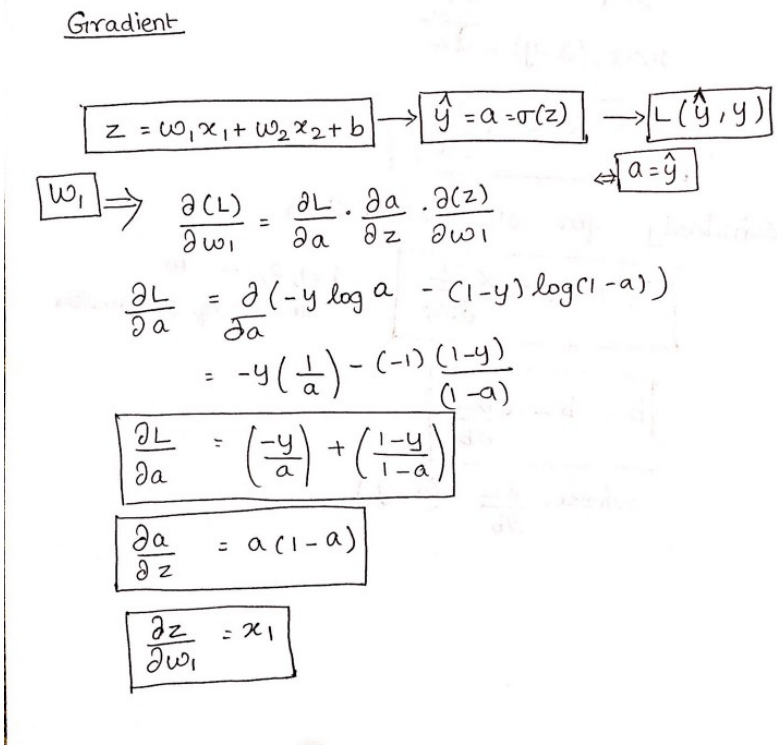

Figure 11: Gradient Descent Algorithm part 1

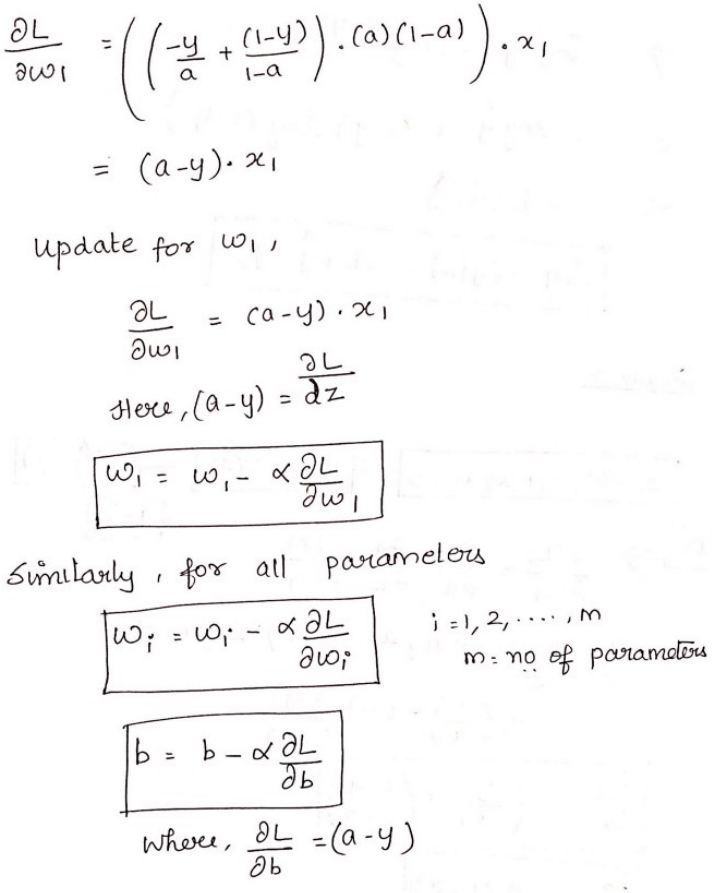

Figure 12: Gradient Descent part 2

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(1-final_result)))))
    #

    #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
```

Cost vs Number_of_Iterations

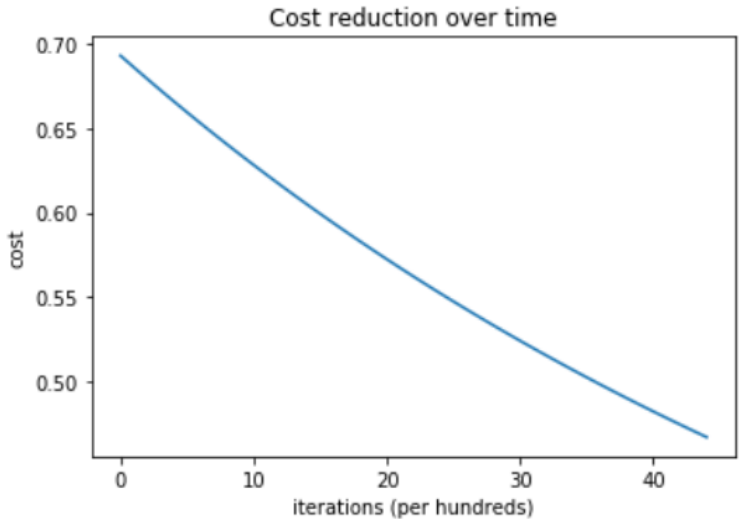

Figure 13: Cost Reduction

Train and test accuracy of the system is 100 %

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

This is an educational post and inspired from Prof. Andrew Ng's deep learning course.

Full code : https://github.com/SSaishruthi/LogisticRegression_Vectorized_Implementation/blob/master/Logistic_Regression.ipynb