International Journal of Research in Engineering, Science and Management
Volume-3, Issue-2, February-2020
www.ijresm.com | ISSN (Online): 2581-5792

Defining Cost Function for Optimization of Machine Learning Models

Arjun Pandey
Student, Department of Computer Science, Jaipur, india

***Abstract*: This paper explores the intuition, reliability and relevance of the cost function while developing machine learning models and how to utilise this function with the gradient descent algorithm. The Cost Function is a method through which I assess the reliability of a machine learning model. In this paper I will discuss the meaning and logic behind the cost function and hence illustrate the intuitions and derivations behind the cost functions fitting various machine learning models. The usage of the cost function is central in deriving a proper and reliable model hence this paper verifies the prominence and approach to various models fitting various situations.**

***Keywords*: Cost Function, Optimization, Machine learning models.**

1. Introduction

The following paper will discuss the methodology and derivation of the cost function through the graphical and mathematical approaches. I will take this further by combining these cost function approaches to the gradient descent algorithm to finally get the best possible machine learning model. When discussing regression models and their cost functions I will also be looking to define our equation using the Bias-Variance Tradeoff.

A. Define the Cost Function

The cost function is a mathematical model that minimizes the difference between the predicted and real values. However, the word "minimization" here refers to bringing our error rate as close as possible to 0, a negative cost function value is in fact equally harmful as a high positive one.

B. Define the Loss Function

The Loss function is in fact similar to the cost function. However, it is calculated specifically for each example set. So for a dataset the loss function calculates the possible error for the Ith example not all n examples in a dataset.

C. Define Expectation

Mathematical Expectation is the summation of all products of the probability of a certain value occurring and the actual occurred value in real-time collection.

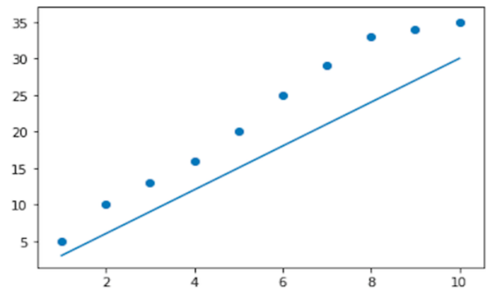
$$E(x) = [x_1p_1 + x_2p_2 + x_3p_3 ... + x_np_n] \quad (1)$$

The expectation function will be a key link in this paper hence, I will take an example to explain it. Suppose I have the following data set:

$$X = [3,4,5,2,1,8,6] \quad (2)$$

Here the value of any particular number occurring when picking a number is 1/7. Hence expectation becomes:

$$E(x) = 3 * 1/7 + 4 * 1/7 ... + 6/7 \quad (3)$$



Suppose I am predicting house prices. While using linear regression my output comes as shown above. This is a classic example where the cost function plays a highly important role. Majority of the data points are way off than what they are supposed to be and hence I need to first calculate the very aggregate deviation of all the points or in mathematical terms the cost function. Now let’s get started by deriving a function through which you can best visualize the penalty this model should get for being off at the perfect combination.

This is a simple linear regression equation:

$$Y = m + nx \quad (4)$$

For any regression function, or any machine learning model the very purpose is to achieve accuracy and supply a reliable mode. On parallel lines, I want the values of M and N to be the best fit to this particular model and dataset. When creating a model using a training set of say n examples I for sure know for n examples of X the exact value of Y. Hence what I am essentially doing is minimizing over M and N, or minimizing the discrepancy between the model and the actual values. For simplicity I can assume that m = 0. The above assumption leads us to this equation,

$$J = Y(pred) - Y \tag{5}$$

As desired by us the above function should be as close to 0 as possible. As mentioned above, this value can be both negative and positive so as to remove sign discrepancies and the value differences I take a squared function resulting in this:

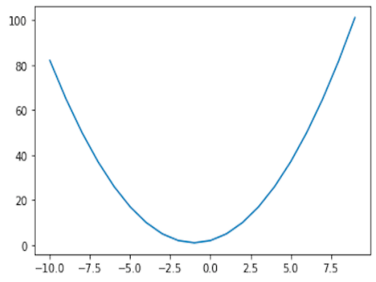
$$(Y(pred) - y)^2 \tag{6}$$

Now these values are specific to some training example in our dataset hence I have finally arrived at our 'Loss Function', a calculation of the error rate for a specific data example.

Now one value is not a good estimator for the error scope in our entire function hence the most logical way to find the 'error rate' is by finding an average. This logic results in our the following equation:

$$\frac{1}{n}\sum_{i=1}^{n} (y_{predicted-y})^2 \tag{7}$$

If I repeat the process again and again I get a graph like this, which is plotted as the Cost versus the n variable (remember I kept m = 0):



So I now end up with a cumulative average error rate of our function and I finally have our cost function.

2. Introducing the Gradient Descent Algorithm

Since I have finally defined our cost function, now one very crucial question arises: How do I use our cost function to minimize the error. The above graph shows how there can be infinite values of m and n for the linear regression but only one minima. To compute and find this minima the gradient descent algorithm is used. An important thing to note is, unlike the cost function, the gradient descent is a more generalized algorithm which can be utilized to find the minimum of several possible cost functions. In essence gradient descent, as the name suggests, finds the point where the gradient is minimum. The above graph is a parabola and its minimum its derivative is 0, meaning gradient is minimum. Gradient descent can be visualized as well. Consider yourself on a rocky mountainous terrain wherein you must take a step by step route to the valley. This is the essence of the gradient descent algorithm and it is defined as follows:

$$\theta_j \coloneqq \theta_j + a\frac{\partial}{\partial\theta_j}J(\theta,\theta_1) \tag{8}$$

The first thing to notice is that I don't have an equals to function but an assignment operator present in our equation. The operator indicates that the θ on the left will be assigned the value computed on the right. The assisgnment operator itself hints at the fact that value will be repeatedly change. So whenever the gradient descent algorithm is implemented it is calculated until convergence of the J(θ,θ_1) function, meaning the cost function. As explained above, I want to reach the minima of the cost and hence the gradient descent algorithm plays a central role in doing so. Now, the a is also defined as the learning rate. So in the above graphs I can see that I need to take a step by step approach to reach the minima of a convex graph. The learning rate is essentially the 'step size', or how much further the algorithm goes before defining the value again. An important thing to note is the fact that I can define the learning rate ourselves, so I can decide the step size our algorithm will take. Now, the function also includes the first derivative and the obvious reason is that this part defined the slope of the graph. Further, the first derivative also lets us know the direction of the function, whether it has a positive or a negative slope.

Now let's discuss the implementation of this algorithm. So since I have a recursive function I can start from any value knowing that I will eventually land at the minima. Hence I initialize m and n in the linear regression function to 0 or very close to 0. The values are then plugged in to the algorithm to find new values of m and n.

I start by plugging in our cost function

$$\theta_j := \theta_j + a\frac{\partial}{\partial\theta_j}J(\theta,\theta_1) \tag{9}$$

$$\frac{\partial}{\partial\theta_j}J(\theta,\theta_1) = \frac{\partial}{\partial\theta_j}\frac{1}{n}\sum_{i=1}^{n} (y_{predicted-y})^2 \tag{10}$$

First I find the partial derivative with respect to one variable i.e. m

$$= \frac{1}{m}\sum_{i=1}^{m} 2(y_{pred} - (mx_i + c))(-x_i) \tag{11}$$

$$= \frac{-2}{m}\sum_{i=1}^{m} (Y_{pred} - y)(-x_i) \tag{12}$$

Now I repeat the process with the other variable:

$$= \frac{-2}{n}\sum_{i=1}^{m} (y_{pred} - y) \tag{13}$$

Now since I have converging function for both variables I just need to recursively update the variable values till the Loss function is either 0 or minimal.

$$m = m - a * D_m \tag{14}$$

$$n = n - a * D_n \tag{15}$$

Here D is the derivative of the function and a is the learning rate of the function. Hence I finally have a summarized way of how to implement the cost function to find the optimal pair of the regression. The cost function implemented with the gradient descent algorithm gives us our best fit model.

3. Cost Function with Logistic Regression

Logistic Regression is a classification algorithm meaning the output will be a set labelled value. The intuition behind classification is the prediction of the possibility of the function giving a particular output. The following is a basic linear function:

$$y = w + nx \tag{16}$$

In logistic regression, I define a particular sigmoid function that analyzes the value of Y corresponding to X and labels it in this case as 0 or 1. The sigmoid function converts a model to classify values as High or Low which in this case corresponds to 0 or 1. The sigmoid function mathematically is defined like this:

$$Sig = \frac{1}{1+e^{-y}} \tag{17}$$

A. *Deriving the formula for logistic regression*

Before starting I lets get a background on logistic regression and how I can use the sigmoid function for our linear regression equation to get

$$Sig = \frac{1}{1+e^{-y}} \tag{18}$$

Sigmoid will now be represented by p

$$p = \frac{1}{1+e^{-m-nx}} \tag{19}$$

I now want a value for m+nx

$$\frac{1}{p} - 1 = e^{-m-nx} \tag{20}$$

The inverse of e, natural logs are introduced

$$ln(\frac{1-p}{p}) = -m - nx \tag{21}$$

Basic Log properties result in this equation

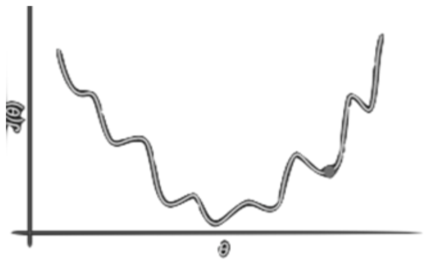
$$ln(\frac{p}{1-p})^{-}1 = m + nx \tag{22}$$

$$ln(\frac{p}{1-p}) = m + nx \tag{23}$$

The natural logarithm acts as the bridge between the real value and the probability in logistic regression. So by now you should have a brief idea about how logistic regression works and how it is derived.

B. *Cost Function with Logistic regression*

Coming back to the cost function with logistic regression. The following is the cost function for logistic regression. Now, why do I not use the same cost function which I derived for the regression model to help optimize our function in this case. Although it is very much possible, the problems arise when I combine the cost function with the gradient descent algorithm. You might remember that our gradient descent algorithm mapped out a perfectly convex function that further allowed us to find the near perfect parameters to our function. The convex function charted one minima that gave an ideal answer and combination to our cost function. However, when I try to plugin the gradient descent algorithm the output comes as the following:



As clearly seen, the function is subject to multiple local minima's and if our algorithm might end up at a different minima if the function doesn't end up being neat convex. Hence, I can conclude that there is need for a cost function that outputs a perfect neat convex.

$$-yln(h_\theta(x)) - (1-y)ln(h_\theta(x)) \tag{24}$$

First of all h $_\theta$(x) equals g(m + nx) where g is the sigmoid function. So I will begin by giving a brief background and derivation of these formulas. As I can see above the two possible output for our algorithm is 0 and 1. Hence, I define the cost function individually first for these outputs, note the following equations:

for y = 1

$$-ln(h_\theta(x)) \tag{25}$$

for y = 0

$$-ln(1 - h_\theta(x)) \tag{26}$$

Since I found a minimizing function for linear regression, I also decided to develop a minimization function for logistic regression and this also indicated by the negative sign in the logarithms. The logarithms are used as they are monotone increasing functions, what this means is that they have an increasing value of y for all real and increasing values of x. Another interesting property about this function is that it is a one-to-one function, meaning for the same x interval the value of y will never occur again. Lastly, I also use logarithms because instead of finding large value products logarithms find sums of the same. Without the logarithms I would have a product form of this function, hence the logarithms calculate the minimum of the function using the logarithmic sum property.

C. *Testing our Equation*

Consider the following dataset:

Y_actual	Y_pred
1	0.9
1	0.2

The MSE for both these values is 0.01 and 0.64 respectively. Remember, I initially took a squared function to penalize large value deviations. So for large off classifications I plan to do the same. The Log Loss for both these functions is 0.1504 and 1.61 respectively. Clearly the log loss functions calls for between optimization by designing larger penalizations. Hence, I can finally confirms that the cross entropy/log loss function is the best fit for classification problems.

4. Decomposition of the error function with bias and variance

Finally, I have an in-depth understanding of the cost models in both regression and classification models. Now, I will get into defining our cost function with another method: the bias-variance tradeoff.

A. *Define Bias*

Bias is the difference between the predicted value of the model and the correct value which I are trying to predict. The bias variance tradeoff measures the 'flexibility' of our function, in other words how will our function perform when subject to a general approach It is defined with the Expectation function as follows:

$$E(x) - x = Bias \qquad (27)$$

B. *Define Variance*

Variance is the spread of our data over a given data point. Variance is defined as the spread of our function over its average value. It is mathematically defined as the average squared differences from the mean. Now having defined Bias and Variance, where is the tradeoff? High bias means that are model is off from the expected function hence is not able to fit the general trend of our model

C. *Bias-Variance Tradeoff*

I will be discussing this model in terms MSE with linear regression. I defined MSE as the average deviation over a range of values, however I knot that what remains most important is the value of MSE over our test set or when it needs to predict values over an unknown dataset. This 'Test MSE' is defined as:

$$Test\ MSE = E[(y_i - f(x_i))^2] \qquad (28)$$

Here I take the values across all new/test pairs in the dataset. The sum function is itself integrate in our expectation hence this becomes our test MSE. While making some models, I came across a strange yet common observation. The MSE over my training set was over 10 points lesser than that of test MSE. When I used a model with high bias, I found out that the I had an 'over fitted' model. A model whose training MSE lay around 3-4 points but increased by a multiple of over 3.5x for similar calculations in the test set. Similarly, for a model with high variance, the model was under fitted and was not able to identify the pattern in my regression. For an optimal model, I knew that a function with the least test MSE having a balance between its bias and variance. I defined the equation as follows

$$Bias = E(x) - x \qquad (29)$$

$$Variance = E[(E(x) - x)^2] \qquad (30)$$

Now lets start dissecting the formula. The cost is:

$$S = (y_{pred} - y)^2 \qquad (31)$$

It can also be written like this:

$$S = (y_{pred} - E(x) + E(X) - y)^2 \qquad (32)$$

$$= (y - E(y_i))^2 + (E(y_i) - y)^2 + 2(y - E(y_i))(E(y_i) - y) \qquad (33)$$

Now plugging the expectation function to both sides:

$$= E([y_i - y]^2) = (y - E(y_i))^2 + E[(E(y_i) - y)^2] \qquad (34)$$

Now carefully notice this is in fact:

$$= Bias^2 + Variance \qquad (35)$$

However, I missed out on the 2ab term in this perfect square. Below is the computation behind that term,

$$= 2E((y - E(y_i))(E(y_i) - y_i)) \qquad (36)$$

$$= 2((y - E(y_i))E[E(y_i) - y_i]) \qquad (37)$$

$$= 2((y - E(y_i))E(E(y_i)) - E(y_i)]) \qquad (38)$$

As per the properties of Expectation E(E(x)) = E(x)

$$2((y - E(y_i))[E(y_i)) - E(y_i)]) = 0 \qquad (39)$$

So now I can clearly see that the MSE can be decomposed into the variance and bias of that particular function. As clearly seen, Bias and Variance are inversely related hence I need to find the ideal values which minimized both these values. Although there won't be an ideal '0' value because in real test cases the 'Irreducible Error' will always be present but this gives a brief intro into how Bias-Variance tradeoff works with the Mean Squared Error of a function

5. Conclusion

With that I have an in-depth explanation of the cost function for linear regression, logistic regression as Ill as the gradient descent algorithm. Further, by now you should be accustomed with the working of the bias-variance tradeoff. The overall objective of the paper was to give a brief analysis into the working, intuition and derivation behind the cost function. Now I can use these functions to build optimal models for our machine learning problems.

References

[1] www.coursera.org/learn/machine-learning/supplement/9SEeJ/cost-function-intuition-ii.
[2] "Bias-Variance." Info Ed-Uk, www.inf.ed.ac.uk.
[3] Shahram Shahbaz Panahi. "Mean Squared Error." Science Direct 2018.