L7 Cost functions

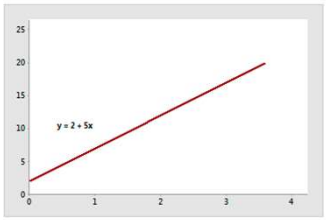
https://intellipaat.com/blog/cost-function-in-machine-learning/

Types of slope

- Linear relationship can be represented by the equation $y = b_0 + b_1x$, where b_0 is the y-intercept and b_1 is the slope.
- if the slope is positive, y increases as x increases, and the function runs "uphill" (going left to right).
- If the slope is negative, y decreases as x increases and the function runs downhill.
- If the slope is zero, y does not change, thus is constant—a horizontal line.
- Vertical lines are problematic in that there is no change in x.
  - Thus our formula is undefined due to division by zero.
  - Some will term this condition infinite slope
  - but we can not tell if it is positive or negative infinity
  - Hence the rather confusing term no slope is also in common usage for this situation.

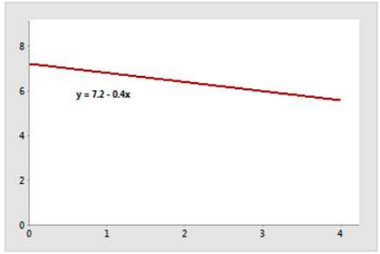
Types of slope - positive slope

The slope is positive 5. When x increases by 1, y increases by 5. The y-intercept is 2.



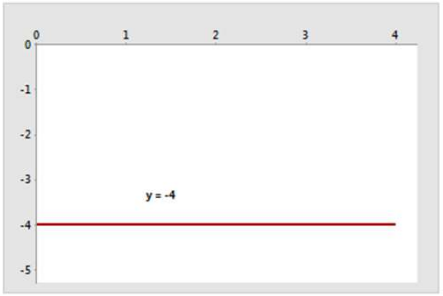
Types of slope - negative slope

The slope is negative 0.4. When x increases by 1, y decreases by 0.4. The y-intercept is 7.2.



Types of slope - zero slope

- The slope is 0. When x increases by 1, y neither increases or decreases. The y-intercept is -4
- A condition in which the variable at the y-axis remains the same with the change in the variable at the x-axis is known as the slope of zero.
- A slope of zero neither moves into the upward or downward direction. It moves only to the leftward or rightward directions.



Cost function vs loss function

- loss functions are a measurement of how good our model is in terms of predicting the expected outcome.
- The loss function is directly related to the predictions of the model
- If loss function value is low, model will provide good results.
- The loss function (or rather, the cost function) to evaluate the model performance needs to be minimized to improve its performance.
- The cost function and loss function refer to the same context (i.e. the training process to minimize the error between the actual and predicted outcome).
- Calculate the cost function as the average of all loss function values
- Calculate the loss function for each sample output compared to its actual value.

Visualization of cost function for linear regression

Visualization of cost function for linear regression - Example

- While dealing with Linear Regression we can have multiple lines for different values of slopes and intercepts.
- Which of those lines actually represents the right relationship between the X and Y ?
- To find that we can use the Mean Squared Error or MSE as the parameter.
- For linear regression, MSE is nothing but the Cost Function.

- Mean Squared Error is the sum of the squared differences between the prediction and true value.
- And the output is a single number representing the cost.
- So the line with the minimum cost function or MSE represents the relationship between X and Y in the best possible manner.
- Once we have the slope and intercept of the line which gives the least error, we can use that line to predict Y

Visualization of cost function for linear regression - Example

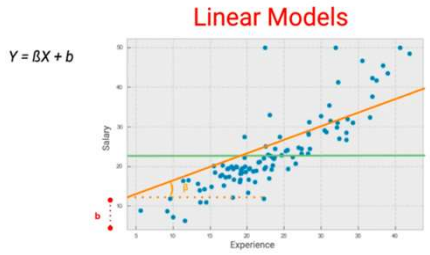
- Problem statement: Calculate the errors/cost for various lines and then find the cost function for prediction
- How to calculate the error for various lines and how to find the optimum line
- Line represented by two parameters- slope(β= B1) and intercept(b = B0).
- What is the error for various values of β and B ?
- How do we find the most optimum values of these two parameters?
- For the line, calculate the errors(also known as cost or loss) which this line would have from the underlying data point to find the line which gives the least error.

Visualization of cost function for linear regression - Example

	salary	experience
0	1.7	1.2
1	2.4	1.5
2	2.3	1.9
3	3.1	2.2
4	3.7	2.4
5	4.2	2.5
6	4.4	2.8
7	6.1	3.1
8	5.4	3.3
9	5.7	3.7
10	6.4	4.2
11	6.2	4.4

Create a data set for Experience and Salary. Experience in x-axis and salary in y axis

Visualization of cost function for linear regression - Example



Visualization of cost function for linear regression - Example

- Importing Libraries
- import the necessary libraries - matplotlib, pandas and scikit learn to calculate the error:
- #import the libraries
- import matplotlib.pyplot as plt
- import pandas as pd
- from sklearn.metrics import mean_squared_error as mse

Visualization of cost function for linear regression - Example

- Creating sample Data -Create a list and then convert to a Pandas Dataframe using pd.DataFrame()

```
# creating the sample dataset
experience = [1.2,1.5,1.9,2.2,2.4,2.5,2.8,3.1,3.3,3.7,4.2,4.4]
salary = [1.7,2.4,2.3,3.1,3.7,4.2,4.4,6.1,5.4,5.7,6.4,6.2]
data = pd.DataFrame({ "salary" : salary, "experience" : experience })
print(data)
print(data.head()) #the first five rows of our dataset
```

Visualization of cost function for linear regression - Example

Plotting the data

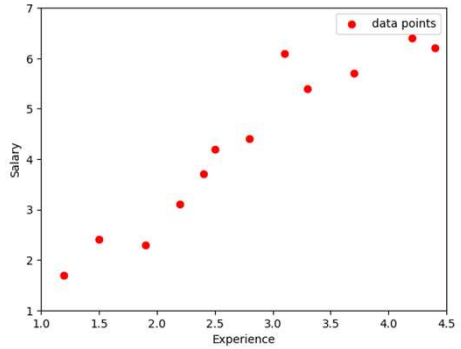
Explore the dataset by exploring the relationship between salary and experience using Matplotlib:

```
# plotting the data

plt.scatter(data.experience, data.salary, color = 'red', label = 'data points')
plt.xlim(1,4.5)
plt.ylim(1,7)
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.legend()
```

Visualization of cost function for linear regression - Example

```
[16]: <matplotlib.legend.Legend at 0x1aa49131510>
```



Visualization of cost function for linear regression - Example

- Starting the Line using small values of parameters(beta and b)
- We can see a linear relationship between experience and salary.
- So now start by plotting the lines by using various values of β and b

- To start with β = 0.1 and b = 1.1
- create a line with these two parameters and plot it over the scatter plot.

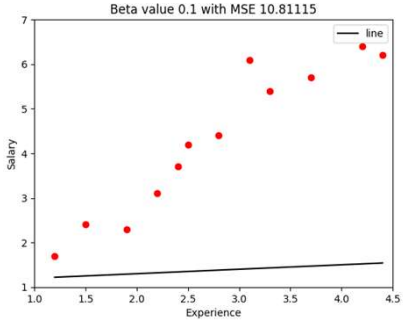
Visualization of cost function for linear regression - Example

```
# making lines for different Values of Beta 0.1, 0.8, 1.5
beta = 0.1
#beta = 1.5
#beta = 0.8

# keeping intercept constant
b = 1.1
# to store predicted points
line1 = []
# generating predictions for every data point
for i in range(len(data)):
   line1.append(data.experience[i]*beta + b)
```

```
# Plotting the line
plt.scatter(data.experience, data.salary, color = 'red')
plt.plot(data.experience, line1, color = 'black', label = 'line')
plt.xlim(1,4.5)
plt.ylim(1,7)
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.legend()
MSE = mse(data.salary, line1)
plt.title("Beta value "+str(beta)+" with MSE "+ str(MSE))
MSE = mse(data.salary, line1)
```

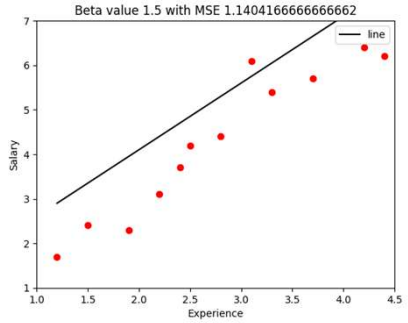
Visualization of cost function for linear regression - Example



Visualization of cost function for linear regression - Example

- The line for beta = 0.1 and b = 1.1
- MSE for this line = 10.81115
- The slope of the line is very less
- Try a higher slope and change beta = 1.5
- MSE= 1.1404166666666662

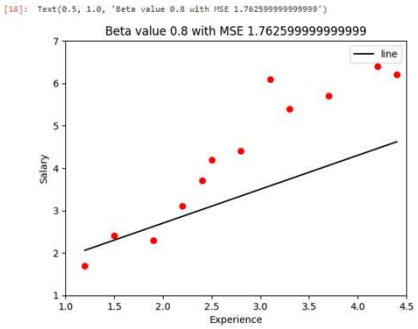
Visualization of cost function for linear regression - Example



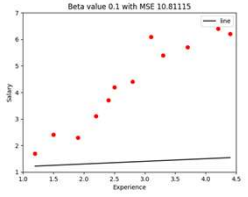
Visualization of cost function for linear regression - Example

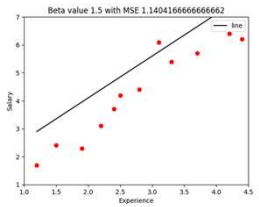
- We can see a better slope but more than what we actually want.
- So the right value might be somewhere in between 0.1 and1.5
- so try beta = 0.8
- MSE= 1.762599999999999

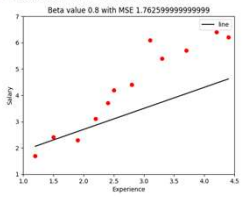
Visualization of cost function for linear regression - Example



Visualization of cost function for linear regression - Example







b = 1.1
The line for beta = 0.1, MSE = 10.81115
beta = 1.5, MSE= 1.1404166666666662
beta = 0.8, MSE= 1.762599999999999

Visualization of cost function for linear regression - Example

- We have used 3 different lines with each of these having different MSE
- Now we can check for various values of beta and see the relationship between beta and mean squared error(MSE) for a fixed value intercept i.e b.
- So b is same b = 1.1

Visualization of cost function for linear regression - Example

- Computing Cost Function over a range of values of Beta
- Create a function by name Error
- For a given value beta, it finds the MSE for these data points.
- Here b is fixed and we are spanning with different values of Beta.

Visualization of cost function for linear regression - Example

```
def Error(Beta, data):
# b is constant
   b = 1.1
   salary = []
   experience = data.experience
# Loop to calculate predict salary variables
   for i in range(len(data.experience)):
      tmp = data.experience[i] * Beta + b
      salary.append(tmp)
    MSE = mse(data.salary, salary)
    return MSE
```

Visualization of cost function for linear regression - Example

- Try with all values of Beta between 0 to 1.5 with an increment of 0.01 and then append everything to a list:

- # Range of slopes from 0 to 1.5 with increment of 0.01

```
slope = [i/100 for i in range(0,150)]
Cost = []
for i in slope:
   cost = Error( Beta = i, data = data)
   Cost.append(cost)
```

Visualization of cost function for linear regression - Example

Visualizing cost with respect to Beta

- When this is done, just convert this into this is the Data Frame:

- # Arranging in DataFrame

```
Cost_table = pd.DataFrame({
'Beta' : slope,
'Cost' : Cost
})
```

- Cost_table.head()

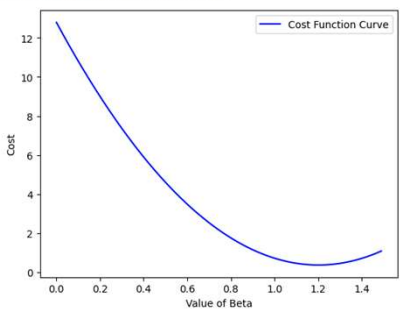
Visualization of cost function for linear regression - Example

- # plotting the cost values corresponding to every value of Beta

- plt.plot(Cost_table.Beta, Cost_table.Cost, color = 'blue', label = 'Cost Function Curve')
- plt.xlabel('Value of Beta')
- plt.ylabel('Cost')
- plt.legend()

Visualization of cost function for linear regression - Example

```
[17]: <matplotlib.legend.Legend at 0x1aa4937d090>
```



	Beta	Cost
0	0.00	12.791667
1	0.01	12.585876
2	0.02	12.381806
3	0.03	12.179455
4	0.04	11.978824

	Beta	Cost
0	0.00	12.791667
1	0.01	12.585876
2	0.02	12.381806
3	0.03	12.179455
4	0.04	11.978824
..	...	...
145	1.45	0.905412
146	1.46	0.948974
147	1.47	0.994255
148	1.48	1.041256
149	1.49	1.089976

[150 rows x 2 columns]

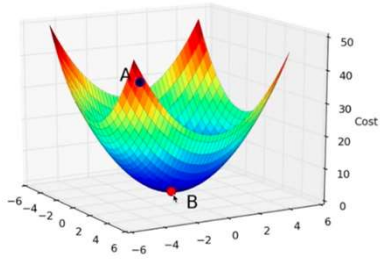
Visualization of cost function for linear regression - Example

	Beta	Cost
0	0.00	12.791667
1	0.01	12.585876
2	0.02	12.381806
3	0.03	12.179455
4	0.04	11.978824

- Value of cost at starting value of beta at 0= 12.791667
- After that the error goes down with the increasing value of Beta, reaches a minimum, and then it starts increasing.

- Given that we know this relationship, what are the values of beta and b for which we can find out this particular location where the cost is minimum.
- What happens when we have two parameters
- Now we assumed b = 0.1
- We can change b as well as beta to get 3D plot
- But we cannot do this iteratively in the way we did so far.
- So to find this minimum value use Gradient Descent technique.

Visualization of cost function for linear regression – 3D plot



Cost function for stochastic Gradient Descent

MSE

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

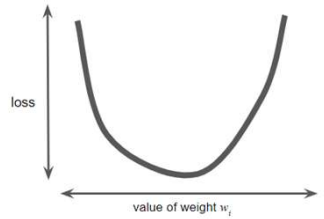
MSE loss function as the average of squared differences between the actual and the predicted value.
It is the most commonly used regression loss function.

Cost function

- Optimization algorithms are algorithms that are designed to converge to a solution.
- The solution here could be a local minimum or the global minimum by minimizing a cost function say 'L'
- The cost function is a measure of how good our model is at making predictions.
- The shape of a cost function defines our goal of optimization.
- If the shape of the cost function is a convex function, our goal is to find the only minimum.
- This is relatively simpler because there is no local minimum and we just need to converge to the global minimum.
- If the shape of the cost function is not a convex function, our goal is to find the lowest possible value in the neighborhood.

Loss function

- To calculate the loss for all possible values of weight B1 (w1)
- For the regression problems, the resulting plot of loss vs. B1 will always be convex (bowl-shaped)



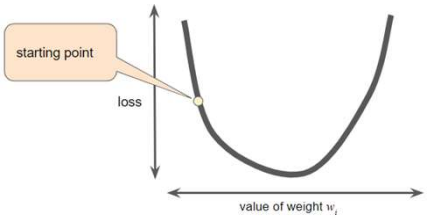
Loss function – Gradient Descent

- Convex problems have only one minimum;
- that is, only one place where the slope is exactly 0.
- That minimum is where the loss function converges.
- Calculating the loss function for every value of B1 over the entire data set would be an inefficient way of finding the convergence point.
- a better mechanism popular in machine learning is called gradient descent.
- The first stage in gradient descent is to pick a starting value (a starting point) to set to 0 or pick a random value.

Loss function – Gradient Descent

- The gradient descent algorithm then calculates the gradient of the loss curve at the starting point.
- the gradient of the loss is equal to the derivative (slope) of the curve
- When there are multiple weights, the gradient is a vector of partial derivatives with respect to the weights.

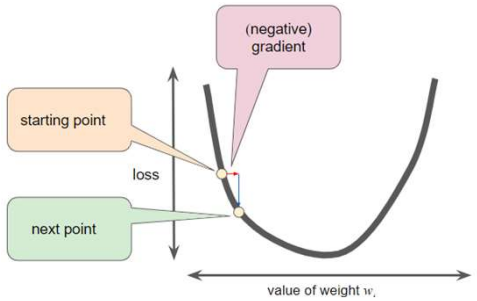
Loss function – Gradient Descent



Loss function – Gradient Descent

- the negative of the gradient moves in the direction of maximum decrease in height.
- In other words, the negative of the gradient vector points into the valley.
- The gradient descent algorithm takes a step in the direction of the negative gradient in order to reduce loss as quickly as possible.

Loss function – Gradient Descent



To determine the next point along the loss function curve, the gradient descent algorithm adds some fraction of the gradient's magnitude to the starting point
The gradient descent then repeats this process, edging ever closer to the minimum.

Loss function – Gradient Descent

- When performing gradient descent, we generalize the above process to tune all the model parameters simultaneously.
- Ex: to find the optimal values of both B1 and B0 (bias), we calculate the gradients with respect to both B1 and B0
- Next, we modify the values of B1 and B0 based on their respective gradients.
- Then we repeat these steps until we reach minimum loss.

Gradient of a function

The **gradient** of a function, denoted as follows, is the vector of partial derivatives with respect to all of the independent variables:

$$\nabla f$$

For instance, if:

$$f(x,y) = e^{2y}\sin(x)$$

then:

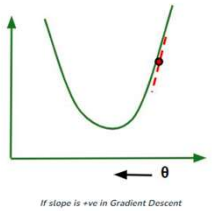
$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y)\right) = (e^{2y}\cos(x), 2e^{2y}\sin(x))$$

Gradient of a function

∇f	Points in the direction of greatest increase of the function.
$-\nabla f$	Points in the direction of greatest decrease of the function.

Loss function – Gradient Descent

- In the Gradient Descent algorithm, one can infer two points :
- If slope is +ve : θj = θj – (+ve value). Hence the value of θj decreases.

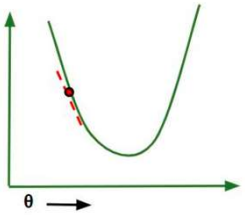


If slope is +ve in Gradient Descent

X axis is Θ = weight = w = B1. Y axis is cost

Loss function – Gradient Descent

- If slope is -ve : θj = θj – (-ve value). Hence the value of θj increases.

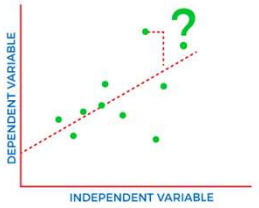


If slope is -ve in Gradient Descent

X axis is Θ = weight = w = B1. Y axis is cost

Cost function for logistic regression

Cost function for linear regression



Cost function plays a crucial role in understanding how well our model estimates the relationship between the input and output parameters.

Cost function for linear regression

- Calculates the difference between the actual values and predicted values and measures how wrong is our model in the prediction.
- Error= Predicted output - Actual Output
- By minimizing the value of the cost function, we can get the optimal solution.
- Regression cost function: MSE (Mean Square Error) , MAE (Mean Absolute Error)
- Binary classification cost function: Cross entropy Loss

Logistic regression vs linear regression

- How do we find the best values of coefficients for our model?
- In linear regression, we minimized the empirical risk which was defined as the average squared error loss, also known as the mean squared error or MSE
- In logistic regression, instead of using a squared loss and trying to minimize the empirical risk,
- we maximize the likelihood of our training set according to the model.
- likelihood function defines how likely the observation (an example) is according to our model

Logistic regression

- Because of the exp function used in the model, in practice, it's more convenient to maximize the log-likelihood instead of likelihood.
- There is no closed form solution to the above optimization problem.
- A typical numerical optimization procedure used in such cases is gradient descent

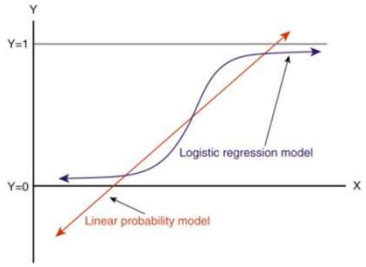
Logistic regression Example

- Take a case study of a clothing company that manufactures jackets and socks.
- They want to have a model that can predict whether the customer will buy a jacket (class 1) or a pair of socks (class 0) from their historical behavioral pattern so that they can give specific offers according to the customer's needs.
- How do we build a predictive model?
- Logistic regression is used when dependent variable (y) is binary
- Linear regression is used when dependent variable is continuous.

Logistic Regression Example

- If we use Linear Regression in our classification problem, we will have values greater than 1 and less than 0, which do not make much sense in our classification problem.
- It will make a model interpretation a challenge.
- That is where `Logistic Regression` comes in.
- Using Sigmoid function, we will get a line that remains between 0 and 1.
- Advantage of this function is all the continuous values we will get will be between 0 and 1 which we can use as a probability for making predictions.

Logistic Regression Example

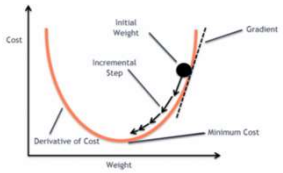


Logistic Regression Example

- In Linear Regression, we use `Mean Squared Error` for cost function

$$\frac{\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2}{n}$$

- When this error function is plotted with respect to weight parameters of the Linear Regression Model, it forms a convex curve
- So, it is possible to apply Gradient Descent Optimization Algorithm to minimize the error by finding global minima and adjust weights



Why not `Mean Squared Error' as a cost function in Logistic Regression

- In Logistic Regression Ŷi is a nonlinear function(Ŷ=1/1+ e^(-z)), if we put this in the above MSE equation it will give a non-convex function as shown

1. When we try to optimize values using gradient descent it will create complications to find global minima.
2. Another reason is in classification problems, we have target values like 0/1, So (Ŷ-Y)2 will always be in between 0-1 which can make it very difficult to keep track of the errors and it is difficult to store high precision floating numbers.

- The cost function used in Logistic Regression is Log Loss.

Why not `Mean Squared Error' as a cost function in Logistic Regression

- We can not use the same cost function MSE for linear regression.
- The hypothesis of logistic regression tends to limit the cost function between 0 and 1.
- So linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression
- Because our prediction function is non-linear (due to sigmoid transform).
- Squaring this prediction as we do in MSE results in a non-convex function with many local minimums.
- If our cost function has many local minimums, gradient descent may not find the optimal global minimum.
- Instead of Mean Squared Error, we use a cost function called Cross-Entropy, also known as Log Loss.

Log Loss

- What is Log Loss?
- Log loss, also known as logarithmic loss or cross-entropy loss, is a common evaluation metric for binary classification models.
- It measures the performance of a model by quantifying the difference between predicted probabilities and actual values.
- Log-loss is indicative of how close the prediction probability is to the corresponding actual/true value (0 or 1 in case of binary classification)
- Log Loss is a standard way of interpreting the cross-entropy that comes from the field of information theory.
- The logistic loss or cross-entropy loss (or cross entropy) is used in classification problems.
- Binary Cross-Entropy = (Sum of Cross-Entropy for N data)/N

Log Loss - Interpretation

- Assume that we have a labeled example (xi, yi) in our training data.
- Assume also that we have found (guessed) some specific values B0, B1 of our parameters.
- If we now apply our model f(z) to xi we will get some value $0 < p < 1$ as output.
- If yi is the positive class, the likelihood of yi being the positive class, according to our model, is given by p.
- Similarly, if yi is the negative class, the likelihood of it being the negative class is given by 1 - p.

Log Loss

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

Yi represents the actual class
log(p(yi)) is the probability of that class
p(yi) is the probability of 1.
1-p(yi) is the probability of 0

Log Loss - Interpretation

- Cross-entropy loss can be divided into two separate cost functions: one for y=1 and one for y=0
- Note: Detailed Reference: Neural networks and Deep learning by Michael Neilson

Visualization of cost function (binary cross entropy function) for logistic regression

Visualization of binary cross entropy function for logistic regression

$$L = -\frac{1}{m}\sum_{i=1}^{m}(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i))$$

m or n — number of training samples
i — ith training sample in a data set
y(i) — Actual value for the ith training sample
y_hat(i) — Predicted value for the ith training sample

Graph of -log(1-X) : y=0 case

$$Cost(y^{(i)}, \hat{y}^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)} * \log(\hat{y}^{(i)}) - (1 - y^{(i)}) * \log(1 - \hat{y}^{(i)})$$

Step — 1:
The value of cost function when Yi=0 (only the second term)

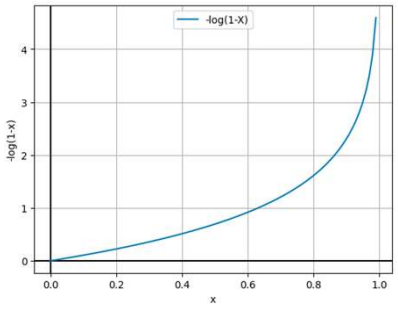
$$Cost(y^{(i)}, \hat{y}^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} -\log(1 - \hat{y}^{(i)})$$

Step — 2:
Assume we have only one training example. It means that n=1. So, the value of the cost function when Y=0

$$Cost = -\log(1 - \hat{y})$$

Graph of -log(1-X) : y=0 case

```
#Import required libraries:
import matplotlib.pyplot as plt
import numpy as np
#Define range of x values:
x = np.linspace(0,1,100)
y = -np.log(1-x)
#Plotting a vertical line at x=0:
plt.axvline(x=0, color="black")
plt.axhline(y=0, color="black")
#Plotting the values on graph:
plt.plot(x,y,label="-log(1-X)")
plt.xlabel("x")
plt.ylabel("-log(1-x)")
plt.legend()
plt.grid()
plt.show()
```



Graph of -log(X) : y=1 case

$$Cost\left(y^{(i)}, \hat{y}^{(i)}\right) = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)} * \log\left(\hat{y}^{(i)}\right) - (1 - y^{(i)}) * \log\left(1 - \hat{y}^{(i)}\right)$$

Step — 3:
The value of cost function when Yi=1 (only the first term)

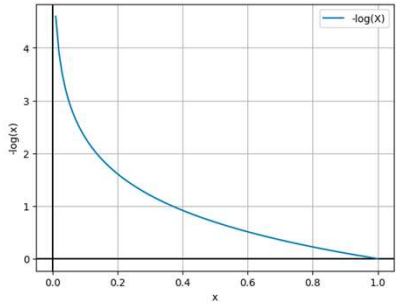
$$Cost\left(y^{(i)}, \hat{y}^{(i)}\right) = \frac{1}{m}\sum_{i=1}^{m} -\log\left(\hat{y}^{(i)}\right)$$

Step — 4:
Assume we have only one training example. It means that n=1. So, the value of the cost function when Y=1

$$Cost = -\log(\hat{y})$$

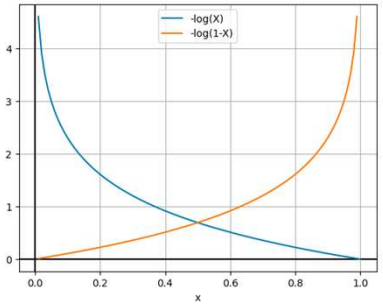
Graph of -log(X) : y=1 case

```
#Import required libraries:
import matplotlib.pyplot as plt
import numpy as np
#Define range of x values:
x = np.linspace(0,1,100)
y = -np.log(x)
#Plotting a vertical line at x=0:
plt.axvline(x=0, color="black")
plt.axhline(y=0, color="black")
#Plotting the values on graph:
plt.plot(x,y,label="-log(X)")
plt.xlabel("x")
plt.ylabel("-log(x)")
plt.legend()
plt.grid()
plt.show()
```



Combined Graph

```
#Import required libraries:
import matplotlib.pyplot as plt
import numpy as np
#Define range of x values:
x = np.linspace(0,1,100)
y = -np.log(x)
z = -np.log(1-x)
#Plotting a vertical line at x=0:
plt.axvline(x=0, color="black")
plt.axhline(y=0, color="black")
#Plotting the values on graph:
plt.plot(x,y,label="-log(X)")
plt.plot(x,z,label="-log(1-X)")
plt.xlabel("x")
plt.legend()
plt.grid()
plt.show()
```



Cost function graphs - Interpretation

- The Red line represents 0 class
- when the predicted probability (x-axis) is close to 0, the loss is less
- when the predicted probability is close to 1, loss approaches infinity.
- The Blue line represents 1 class
- when the predicted probability (x-axis) is close to 1, the loss is less
- when the predicted probability is close to 0, loss approaches infinity.

Cost function graphs - Interpretation

$$Cost\left(y^{(i)}, \hat{y}^{(i)}\right) = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)} * \log\left(\hat{y}^{(i)}\right) - (1 - y^{(i)}) * \log\left(1 - \hat{y}^{(i)}\right)$$

- When the actual class is 1:
- Second term = -(1-1).log(1-p(yi) = 0
- First term i.e. -yi.log(p(yi)) = -log(p(yi))
- When the actual class is 0:
- First term = -0.log(p(yi)) = 0
- Second term i.e -(1-yi).log(1-p(yi))= -log(1-p(yi))

Derivation of the update equation for logistic regression using stochastic gradient descent

Logistic Regression Model:

- The logistic regression model predicts the probability that a given input point belongs to a certain class.
- The probability is modeled using the sigmoid function:

  sigmoid(z)=1/(1+e^(-z))
- $z=b0+b1\cdot x1+b2\cdot x2$ is the linear combination of the
  - parameters (b0 b1, b2) and
  - input features (x1, x2).

Cost function

- **Cost Function**:
- The cost function for logistic regression is the negative log-likelihood (cross-entropy) loss
- For a single training example (*x*1,*x*2,*y*), the cost can be defined as:

$$\text{Cost}(b_0, b_1, b_2) = -y \cdot \log(\text{sigmoid}(z)) - (1 - y) \cdot \log(1 - \text{sigmoid}(z))$$

Explanation of the terms

1. Prediction: In logistic regression, the prediction for a given input (*x*1,*x*2) is the probability that the input belongs to a certain class (usually the positive class). It is computed using the sigmoid function:

$$\text{prediction} = \frac{1}{1+e^{-(b_0+b_1 \cdot x_1+b_2 \cdot x_2)}}$$

- Here, *b*0, *b*1, and *b*2 - are the model parameters (coefficients)
- *x*1, *x*2 - are the input features.

2. True Label (*y*): The true label *y* represents the actual class label associated with the input. It's either 0 (negative class) or 1 (positive class) in the context of binary classification.

Explanation of the terms used

- Error term:(y-prediction): This part represents the difference between the true label and the predicted probability.
  - It indicates how far off the prediction is from the actual label.
- prediction*(1−prediction): This part incorporates the sigmoid function's derivative.
  - It ensures that the gradient is scaled based on the confidence of the prediction.
  - When the prediction is close to 0 or 1, this term will be small, leading to smaller updates.
  - When the prediction is around 0.5, this term will be relatively large, leading to larger updates.
- Summarizing above two terms:
- 3. (y-prediction)*prediction * (1−prediction): incorporates the error or discrepancy between the true label and the predicted probability, while also considering the confidence of the prediction through the sigmoid function's derivative (considers influence of the sigmoid function to ensure that the gradient is scaled appropriately)

Derivation of the update equations

consider a single training example with input features *x*1 and *x*2 and a true label *y*.

- The cost function for logistic regression is given by:
- Cost(*b*0,*b*1,*b*2)= - *y*·log(prediction)−(1−*y*)·log(1−prediction)

  = -y. log(sigmoid(z))-(1-y)*log(1-sigmoid(z))
- Taking the partial derivative of the cost function with respect to *b*0, *b*1, and *b*2, we get
- Note: sigmoid(z)= prediction, so we get (prediction–y) as derivative of cost function

$$\frac{\partial \text{Cost}}{\partial b_0} = \text{sigmoid}(z) - y$$
$$\frac{\partial \text{Cost}}{\partial b_1} = (\text{sigmoid}(z) - y) \cdot x_1$$
$$\frac{\partial \text{Cost}}{\partial b_2} = (\text{sigmoid}(z) - y) \cdot x_2$$

Derivation of the update equations

- These are the gradients that guide the parameter updates in the direction that reduces the cost function and improves the model's predictions.
- The learning rate α scales the size of these updates during each iteration of stochastic gradient descent.
- update equations for stochastic gradient descent involve subtracting the gradient term from the current parameter values, in order to move in the direction of minimizing the cost function

$$b_0 = b_0 - \alpha \cdot (\text{prediction} - y) \cdot \text{prediction} \cdot (1 - \text{prediction})$$
$$b_1 = b_1 - \alpha \cdot (\text{prediction} - y) \cdot \text{prediction} \cdot (1 - \text{prediction}) \cdot x_1$$
$$b_2 = b_2 - \alpha \cdot (\text{prediction} - y) \cdot \text{prediction} \cdot (1 - \text{prediction}) \cdot x_2$$

B0 = B0 + alpha * (y - prediction) * prediction * (1 - prediction)* 1
B1 = B1 + alpha * (y - prediction) * prediction * (1 - prediction)* x1
B2 = B2 + alpha * (y - prediction) * prediction * (1 - prediction)* x 2

Note: prediction*(1−prediction): This part incorporates the sigmoid function's derivative.
(prediction–y) : Error term (which is also derivative of cost function)

Derivation of the update equations – alternate form

Stochastic Gradient Descent Update:

- In stochastic gradient descent (SGD), we update the parameters using the negative gradient of the cost function

$$b_0 = b_0 - \alpha \cdot \frac{\partial \text{Cost}}{\partial b_0}$$
$$b_1 = b_1 - \alpha \cdot \frac{\partial \text{Cost}}{\partial b_1}$$
$$b_2 = b_2 - \alpha \cdot \frac{\partial \text{Cost}}{\partial b_2}$$

$$\frac{\partial \text{Cost}}{\partial b_0} = \text{sigmoid}(z) - y$$
$$\frac{\partial \text{Cost}}{\partial b_1} = (\text{sigmoid}(z) - y) \cdot x_1$$
$$\frac{\partial \text{Cost}}{\partial b_2} = (\text{sigmoid}(z) - y) \cdot x_2$$

- Substituting the gradient expressions

$$b_0 = b_0 - \alpha \cdot (\text{sigmoid}(z) - y)$$
$$b_1 = b_1 - \alpha \cdot (\text{sigmoid}(z) - y) \cdot x_1$$
$$b_2 = b_2 - \alpha \cdot (\text{sigmoid}(z) - y) \cdot x_2$$

Note: In the alternate form of update equation, the term prediction * (1-prediction) which is sigmoid function's derivative is not made use. Only sigmoid(z)-y = prediction – y which is derivative of cost function is made use.