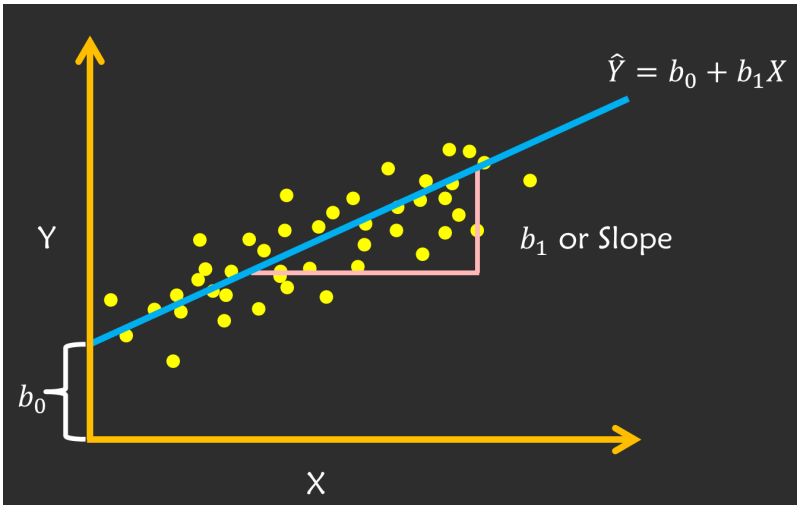
Linear regression

Linear regression aims to model the relationship between a dependent variable (target) and an independent variable (feature) by fitting a linear equation to observed data. For a single variable, this relationship can be represented as:

In the context of linear regression, the equation y=mx+c is used to describe a straight-line relationship between two variables:

- y: The dependent variable or the value we want to predict.
- x: The independent variable or the input feature.
- m: The slope of the line, representing how much y changes for a unit change in x.
- c: The y-intercept, representing the value of y when x is zero.

In linear regression, we aim to find the best-fitting line through the data points. This means we need to determine the values of m and c that minimize the difference between the actual data points and the values predicted by our line.



1. **Data Points:** Imagine we have a set of data points plotted on a graph, with x values on the horizontal axis and y values on the vertical axis.

2. **Finding the Line:** The goal of linear regression is to find the line y=mx+c that best fits these data points. This line should minimize the differences (errors) between the actual y values and the predicted y values from the line.

3. **Slope (m):** The slope m tells us how steep the line is. If m is positive, the line slopes upwards, meaning as x increases, y also increases. If m is negative, the line slopes downwards, meaning as x increases, y decreases.

4. **Intercept (c):** The intercept c is the point where the line crosses the y-axis. It represents the value of y when x is zero.

Example

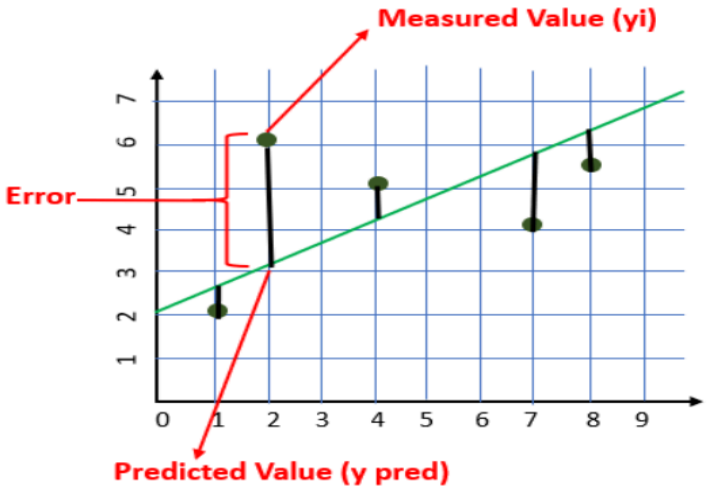
We have data on the number of hours studied (x) and the exam scores (y) of students. We want to predict the exam score based on the number of hours studied.

If our linear regression gives us the line y=5x+30

- ➢ m (slope) is 5, meaning for every additional hour studied, the predicted exam score increases by 5 points.
- ➢ c (intercept) is 30, meaning if a student studies 0 hours, the predicted exam score is 30.

Single Variable Cost Function

In linear regression with a single variable, the cost function measures how well the model fits the data. It quantifies the error between predicted values and actual outcomes.



Definition

The cost function J(θ0,θ1) is defined as:

$$L(m,b) = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

where $\hat{y}_i = mx_i + b$ are the predicted values.

- n : Number of training examples.
- $h\theta(x^{(i)})=\theta 0+\theta 1 x^{(i)}$: Predicted value for the i-th example.
- $y^{(i)}$: Actual output for the i-th example.

- The term $1/2n$ is used for scaling the error.

Purpose

The cost function helps in finding the optimal parameters θ0 and θ1 by minimizing the error between predicted and actual values.

x	y
1	2
2	2.8
3	3.6
4	4.4

Calculation

$$J(\theta_0, \theta_1) = \frac{1}{2 \cdot 4} \left((0-2)^2 + (0-2.8)^2 + (0-3.6)^2 + (0-4.4)^2 \right)$$

$$= \frac{1}{8}(4 + 7.84 + 12.96 + 19.36) = \frac{44.16}{8} = 5.52$$

Gradient Descent for Linear Regression

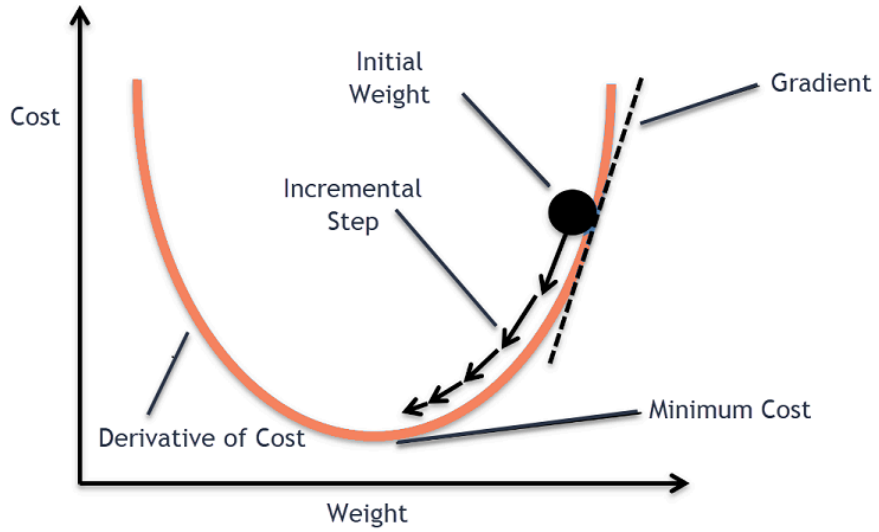
Gradient Descent is an algorithm designed to find the best-fit line for a given training dataset efficiently. If we plot the parameters m and c against the Mean Squared Error (MSE), the graph will have a bowl-like shape. For certain values of m and c, the MSE will be at its lowest. This combination of m and c defines our best-fit line.

The algorithm begins with initial values for m and c (typically m=0 and c=0). We calculate the MSE at this starting point. For example, the MSE at m=0 and c=0 might be 100. Then, we adjust the values of m and c by a small amount (known as the learning rate).

This adjustment usually results in a decrease in MSE. We continue this process until the MSE is minimized, ideally reaching 0 (which would imply 100% accuracy).

Gradient Descent Curve

The gradient descent curve visually represents how the cost function J(θ) changes as the parameters θ are updated during the optimization process.



Cost Function Surface:

- The cost function J(θ) for linear regression is typically a convex function (a bowl-shaped curve) when plotted against the parameters θ0 and θ1.
- In two dimensions, where we only have θ0 and θ1, this can be visualized as a 3D surface or a contour plot.
- The center of the contours represents the minimum cost (global minimum), where the optimal parameters θ0 and θ1 are located.

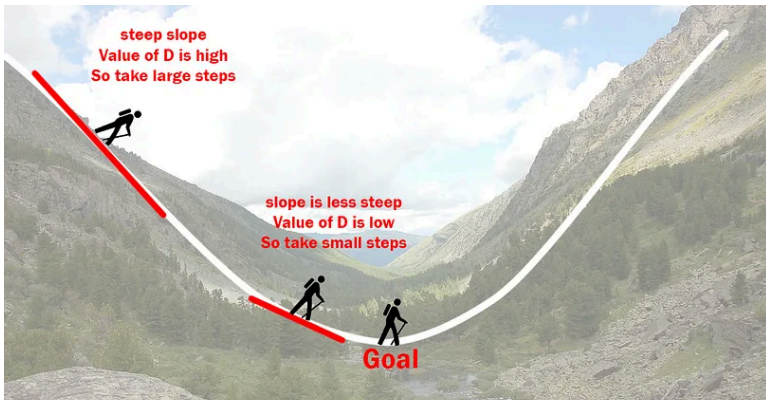
Gradient Descent Path:

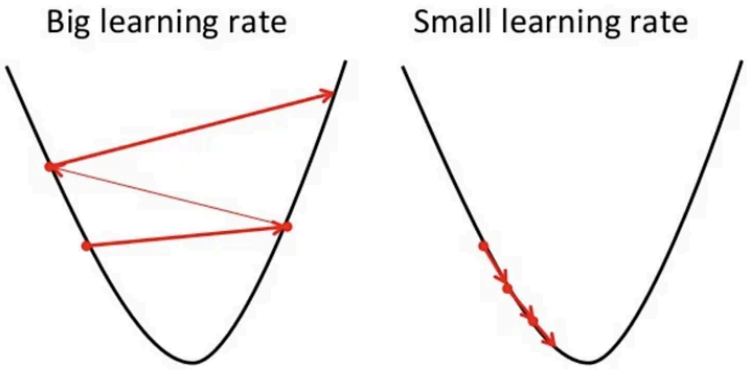
- The gradient descent algorithm starts with initial parameter values (usually chosen randomly or set to zero).
- At each iteration, the parameters are updated to move in the direction that reduces the cost function the most rapidly.
- As the parameters are updated, the gradient descent algorithm takes steps towards the minimum of the cost function. These steps can be visualized as a path on the contour plot, showing how the parameters evolve over time.
- The gradient descent path will be a series of arrows (or steps) pointing towards the center of the contours, which represent the global minimum of the cost function.

Learning Rate (α):

- The learning rate determines the size of the steps taken towards the minimum of the cost function.

- If the learning rate is too high, the algorithm might overshoot the minimum and fail to converge.
- If the learning rate is too low, the algorithm will take too long to converge.





Gradient Descent Algorithm

The goal is to minimize the difference between the predicted values from the model and the actual values y in the dataset. This is typically done by minimizing the Mean Squared Error (MSE) or the Mean Absolute Error (MAE) between and y.

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Gradient descent updates the parameters θ0 and θ1 to minimize the cost function. The update rule for gradient descent is:

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

where:

- α is the learning rate
- $\frac{\partial}{\partial \theta_j} J(\theta)$ is the partial derivative of the cost function with respect to θ_j

Algorithm

Initialize Parameters: Start with initial guesses for θ0 and θ1 (commonly 0).

Choose Learning Rate α : The learning rate determines the step size in each iteration.

Iterative Update: Repeat until convergence:

- Calculate the predictions hθ(x(i)) for all training examples.
- Compute the cost J(θ0,θ1).
- Update the parameters

Iterative Optimization: Perform iterations of gradient descent until convergence. In each iteration, compute gradients and update m and b accordingly.

Convergence Criteria: The algorithm stops when the change in the cost function is below a predefined threshold, or when the gradient is close to zero, indicating a minimum has been reached.

Derivative of Cost function

But first let's start by understanding how to find the derivative (gradient) of this cost/loss function to know which way to move.

$$m := m - \alpha \frac{\partial L}{\partial m} \qquad \frac{\partial L}{\partial m} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_i$$

$$b := b - \alpha \frac{\partial L}{\partial b} \qquad \frac{\partial L}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)$$

where α is the learning rate.

Derivatives of the Cost Function

To minimize $J(\theta)$, we need to find the partial derivatives of $J(\theta)$ with respect to each parameter θ_0 and θ_1.

Partial Derivative with Respect to θ_0

Let's start with θ_0.

1. **Cost Function**:
   $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right)^2$

2. **Derivative of the Squared Term**:
   We apply the chain rule to differentiate the squared term:
   $\frac{\partial}{\partial \theta_0} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right)^2 = 2 \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right) \cdot \frac{\partial}{\partial \theta_0} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right)$

3. **Derivative of the Inner Term**:
   $\frac{\partial}{\partial \theta_0} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right) = 1$

4. **Combining the Results**:
   $\frac{\partial}{\partial \theta_0} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right)^2 = 2 \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right) \cdot 1 = 2 \left(h_\theta(x^{(i)}) - y^{(i)}\right)$

5. **Sum Over All Training Examples**:
   $\frac{\partial}{\partial \theta_0} J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} 2 \left(h_\theta(x^{(i)}) - y^{(i)}\right)$

6. **Simplifying**:
   $\frac{\partial}{\partial \theta_0} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$

Partial Derivative with Respect to θ_1

Now, let's differentiate with respect to θ_1.

1. **Cost Function**:
   $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right)^2$

2. **Derivative of the Squared Term**:
   We apply the chain rule to differentiate the squared term:
   $\frac{\partial}{\partial \theta_1} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right)^2 = 2 \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right) \cdot \frac{\partial}{\partial \theta_1} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right)$

3. **Derivative of the Inner Term**:
   $\frac{\partial}{\partial \theta_1} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right) = x^{(i)}$

4. **Combining the Results**:
   $\frac{\partial}{\partial \theta_1} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)} \right)^2 = 2 \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$

5. **Sum Over All Training Examples**:
   $\frac{\partial}{\partial \theta_1} J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} 2 \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$

6. **Simplifying**:
   $\frac{\partial}{\partial \theta_1} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$

Gradient Descent Update Rules

Using the partial derivatives, the update rules for gradient descent are:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

Question

Dataset

Given data points:

$(x_1, y_1) = (3, 8)$
$(x_2, y_2) = (7, 7)$
$(x_3, y_3) = (2, 5)$
$(x_4, y_4) = (4, 1)$

Initialization

Let's initialize the parameters m (slope) and b (intercept):

$m = 0, \quad b = 0$

Gradient Descent Steps

Step 1: Compute Predictions

Using the current values of m and b, compute the predicted values $\hat{y}$:

$\hat{y}_1 = m \cdot x_1 + b = 0 \cdot 3 + 0 = 0$
$\hat{y}_2 = m \cdot x_2 + b = 0 \cdot 7 + 0 = 0$
$\hat{y}_3 = m \cdot x_3 + b = 0 \cdot 2 + 0 = 0$
$\hat{y}_4 = m \cdot x_4 + b = 0 \cdot 4 + 0 = 0$

Step 2: Compute the Loss Function

Calculate the Mean Squared Error (MSE) loss:

$L(m, b) = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$
$L(0, 0) = \frac{1}{2 \cdot 4} \left[(0 - 8)^2 + (0 - 7)^2 + (0 - 5)^2 + (0 - 1)^2 \right]$
$L(0, 0) = \frac{1}{8} \left[64 + 49 + 25 + 1 \right]$
$L(0, 0) = \frac{1}{8} \cdot 139 = 17.375$

Step 3: Compute Gradients

Compute the gradient of the loss function L with respect to m and b:

$\frac{\partial L}{\partial m} = \frac{1}{4} \sum_{i=1}^{4} (\hat{y}_i - y_i) x_i$
$\frac{\partial L}{\partial b} = \frac{1}{4} \sum_{i=1}^{4} (\hat{y}_i - y_i)$

For our current predictions:
$\frac{\partial L}{\partial m} = \frac{1}{4}[(0-8)\cdot 3 + (0-7)\cdot 7 + (0-5)\cdot 2 + (0-1)\cdot 4]$
$\frac{\partial L}{\partial m} = \frac{1}{4}[-24 - 49 - 10 - 4]$
$\frac{\partial L}{\partial m} = \frac{1}{4}\cdot(-87) = -21.75$

$\frac{\partial L}{\partial b} = \frac{1}{4}[(0-8) + (0-7) + (0-5) + (0-1)]$
$\frac{\partial L}{\partial b} = \frac{1}{4}[-8 - 7 - 5 - 1]$
$\frac{\partial L}{\partial b} = \frac{1}{4}\cdot(-21) = -5.25$

Step 4: Update Parameters

Update m and b using the gradients and a learning rate α:
$\alpha = 0.01$ (chosen for this example)

$$m := m - \alpha \cdot \frac{\partial L}{\partial m} = 0 - 0.01 \cdot (-21.75) = 0.2175$$

$$b := b - \alpha \cdot \frac{\partial L}{\partial b} = 0 - 0.01 \cdot (-5.25) = 0.0525$$

Step 5: Repeat

Repeat steps 1-4 until convergence criteria are met (e.g., a certain number of iterations or a sufficiently small change in loss).

Iteration 2 (Example)

Let's compute the steps for one more iteration for illustration:

- **Compute Predictions**: Using $m = 0.2175$ and $b = 0.0525$:
  $\hat{y}_1 = 0.2175 \cdot 3 + 0.0525 = 0.6525 + 0.0525 = 0.705$
  $\hat{y}_2 = 0.2175 \cdot 7 + 0.0525 = 1.5225 + 0.0525 = 1.575$
  $\hat{y}_3 = 0.2175 \cdot 2 + 0.0525 = 0.435 + 0.0525 = 0.4875$
  $\hat{y}_4 = 0.2175 \cdot 4 + 0.0525 = 0.87 + 0.0525 = 0.9225$
- **Compute Loss**: Calculate $L(m, b)$ with the updated predictions.
- **Compute Gradients**: Compute $\frac{\partial L}{\partial m}$ and $\frac{\partial L}{\partial b}$.
- **Update Parameters**: Update m and b using the gradients and the learning rate.

Logistic Regression

Logistic regression is a statistical method used for binary classification tasks in data mining. In binary classification, the goal is to predict the categorical class labels of new instances into one of two classes.

Logistic regression models the relationship between a binary dependent variable (the target class) and one or more independent variables (features). Logistic regression is used for binary classification where we use a sigmoid function that takes input as independent variables and produces a probability value between 0 and 1.

Binary Classification: Logistic regression is primarily used for binary classification tasks where the outcome is a categorical variable with two possible outcomes, often labelled as 0 and 1.

Decision Boundary: The decision boundary is the threshold at which the predicted probability is 0.5. This boundary separates the input space into two regions, each corresponding to one of the classes.

Logistic Function – Sigmoid Function

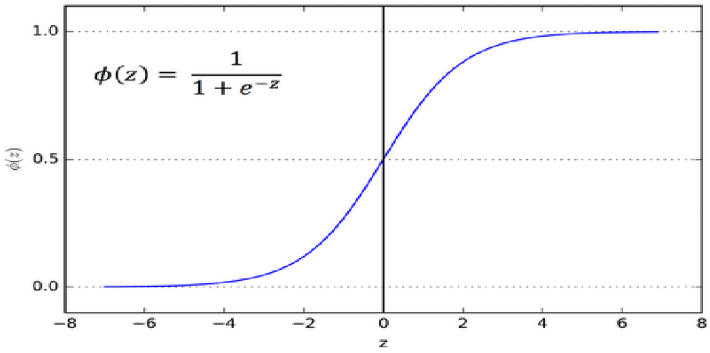
The core of logistic regression is the sigmoid function, also known as the logistic function, which maps any real-valued number into the range between 0 and 1. The sigmoid function σ(z) is defined as:

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

where z is the linear combination of input features.

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1. The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the “S” form.
- The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

For example, we have two classes Class 0 and Class 1 if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 otherwise it belongs to Class 0. The logistic function is defined as:



z is the linear combination of the feature values and model parameters.

How does Logistic Regression work?

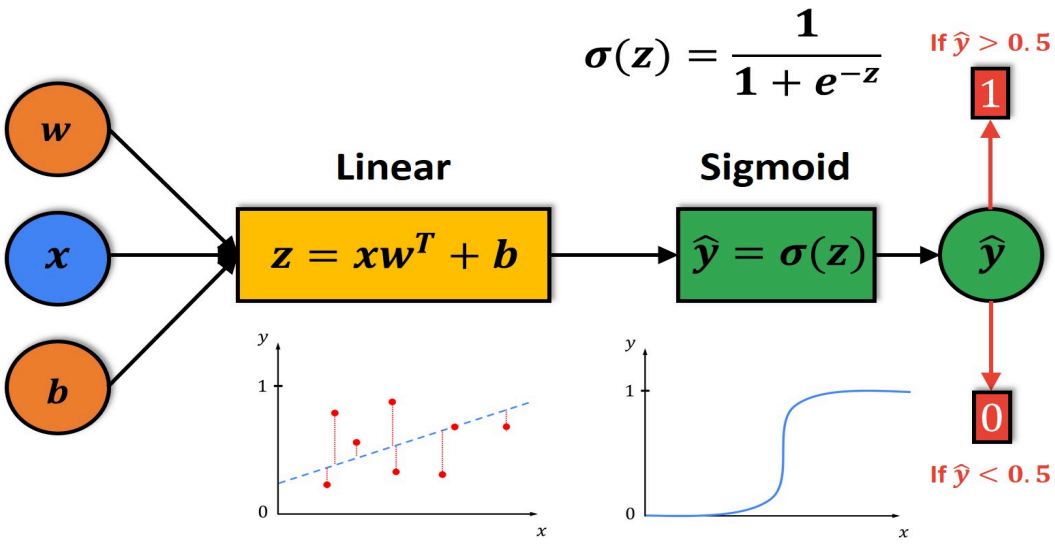
1. **Prepare the data:** The data should be in a format where each row represents a single observation and each column represents a different variable. The target variable (the variable you want to predict) should be binary (yes/no, true/false, 0/1).

2. **Train the model:** During the training phase, the model learns the optimal parameters (weights) that minimize the error between the predicted probabilities and the actual class labels in the training data. This is typically done using optimization algorithms like gradient descent.

   Once trained, logistic regression uses the learned parameters to make predictions on new data. It applies a decision boundary (often at P(y=1)=0.5) to classify instances into one of the two classes based on their predicted probabilities.

3. **Evaluate the model:** The model is evaluated on the held-out test data to assess its performance on unseen data.The performance of the logistic regression model is evaluated using metrics such as accuracy, precision, recall.

4. **Use the model to make predictions:** After the model has been trained and assessed, it can be used to forecast outcomes on new data.



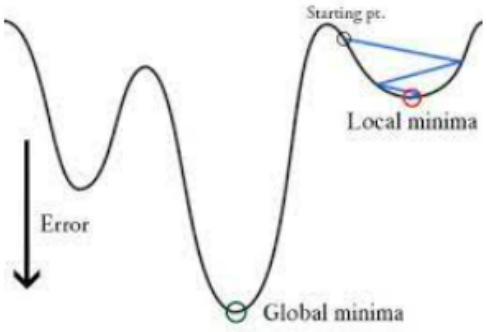
Cost Function of logistic regression

In linear regression, we use the Mean squared error which is the difference between predicted and actual value and this is derived from the maximum likelihood estimator. The graph of the cost function in linear regression is like this:

Linear Regression
Cost Function

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

In logistic regression Yi is a nonlinear function (Ŷ=1/1+ e-z). If we use this in the above MSE equation then it will give a non-convex graph with many local minima as shown.



The problem here is that this cost function will give results with local minima, which is a big problem because then we'll miss out on our global minima and our error will increase.

In order to solve this problem, we derive a different cost function for logistic regression called log loss which is also derived from the maximum likelihood estimation method.

Logistic Regression Cost Function

- **Name**: Log Loss (also called Cross-Entropy Loss).
- **Formula**:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h_\theta(x_i)) + (1 - y_i)\log(1 - h_\theta(x_i))]$$

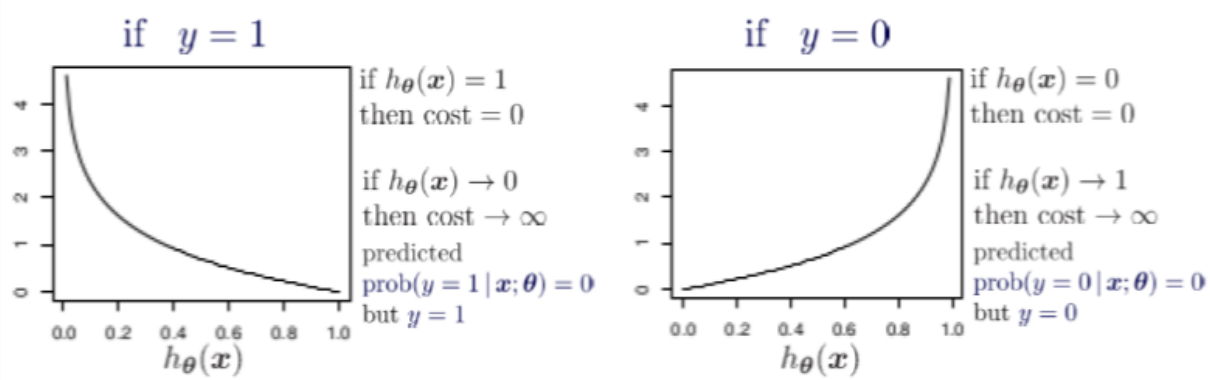
  - m: Number of training examples.
  - yi: Actual label of the i-th training example (0 or 1).
  - hθ(xi): Predicted probability of the i-th training example being 1 (using the logistic function).

Log Loss:

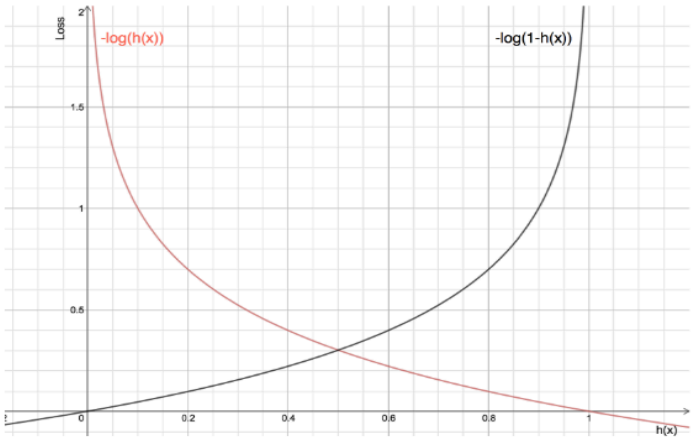
When the actual label y is 1, the cost is high if the predicted probability hθ(x) is low.

When the actual label y is 0, the cost is high if the predicted probability hθ(x) is high.

This ensures the model is penalized more for confident wrong predictions.



If we combine both the graphs, we will get a convex graph with only 1 local minimum and now it'll be easy to use gradient descent here:



The red line here represents the 1 class (y=1), the right term of cost function will vanish. Now if the predicted probability is close to 1 then our loss will be less and when probability approaches 0, our loss function reaches infinity.

The black line represents 0 class (y=0), the left term will vanish in our cost function and if the predicted probability is close to 0 then our loss function will be less but if our probability approaches 1 then our loss function reaches infinity.

Simplified Breakdown

- If $y = 1$ (positive class):

$$\text{Cost} = -\log(h_\theta(x))$$

  - High cost if $h_\theta(x)$ (predicted probability) is low.

- If $y = 0$ (negative class):

$$\text{Cost} = -\log(1 - h_\theta(x))$$

  - High cost if $h_\theta(x)$ (predicted probability) is high.

It also ensures that as the probability of the correct answer is maximized, the probability of the incorrect answer is minimized. Lower the value of this cost function higher will be the accuracy.

Example

- **Actual Label $y = 1$**:
  - If the model predicts $h_\theta(x) = 0.9$ (high probability of being 1), the cost is low.
  - If the model predicts $h_\theta(x) = 0.1$ (low probability of being 1), the cost is high.
- **Actual Label $y = 0$**:
  - If the model predicts $h_\theta(x) = 0.1$ (low probability of being 1), the cost is low.
  - If the model predicts $h_\theta(x) = 0.9$ (high probability of being 1), the cost is high.