COMP30027 MACHINE LEARNING TUTORIAL

Workshop - 6

Understanding

- Parameters
- Hyperparameters
- Linear Regression
- Gradient Descent
- Error Analysis

Parameters vs Hyperparameter

Parameters

Parameters are learned from training data. They define the model's ability to make predictions and are saved as part of the trained model.

Hyperparameter

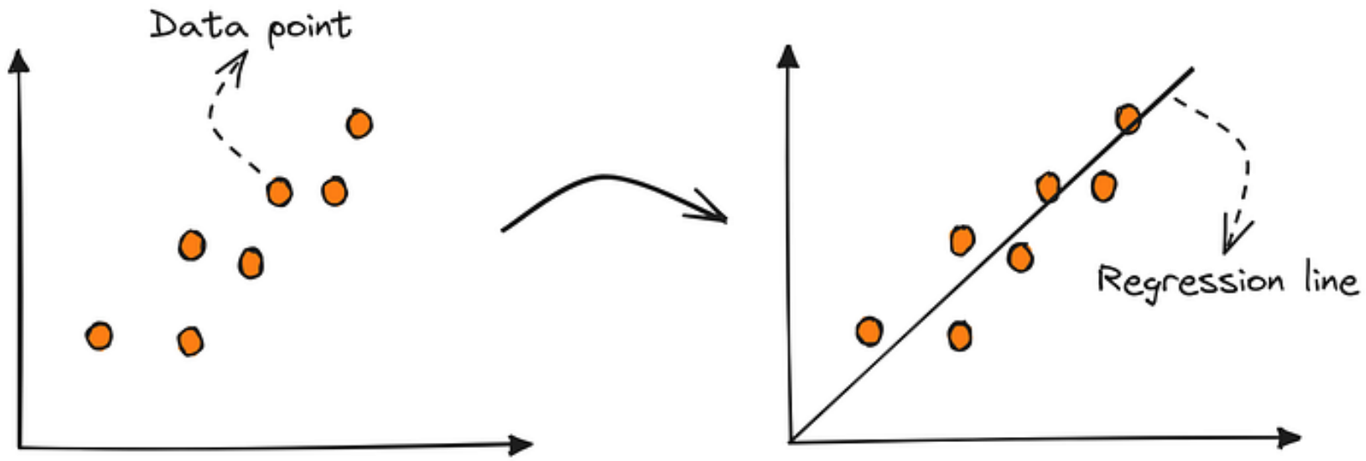
Hyperparameters are manually set before training begins and control how the model learns.

- Unlike parameters, which are learned automatically, hyperparameters represent choices made by the data scientist about how the learning should proceed.
- The same algorithm with different hyperparameter values can yield dramatically different models and prediction accuracy.

Model-Specific Parameters vs Hyperparameters

Algorithm	Parameters	Hyperparameters
Support Vector Machine	Support vectors, weights, bias/intercept	C (regularization), kernel type, gamma
K-Nearest Neighbors	None (stores training data)	k (number of neighbours), distance metric, voting scheme
Decision Trees	Split thresholds, leaf values	Max depth, min samples per leaf, split criterion
Naive Bayes	Prior probabilities, feature likelihoods	Smoothing parameter (alpha)

Linear Regression



- Linear regression captures a **linear relationship** between:
  - An **outcome variable** y (or called response variable, dependent variable, label) and
  - One or more **predictors** $x_1, \ldots, x_D$ (or called independent variable, attribute, feature)

https://pub.towardsai.net/machine-learning-the-first-step-is-to-understand-simple-linear-regression-30dde8b4739

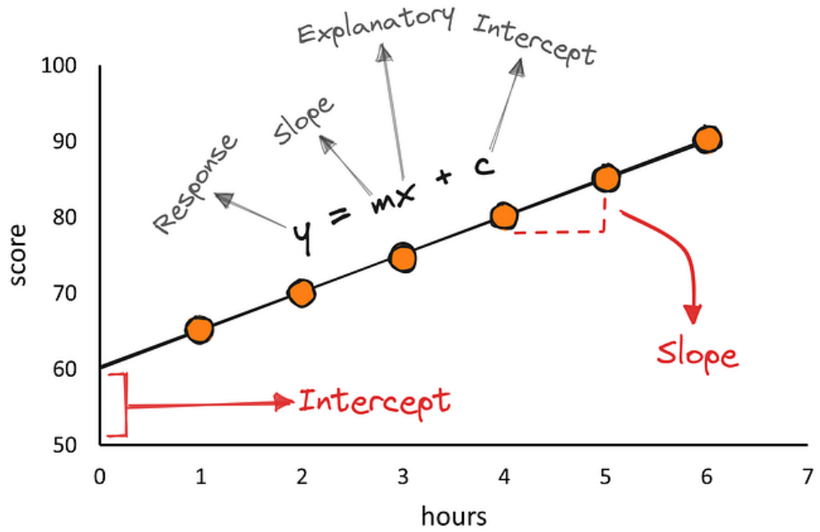
Linear Regression

Model Equation

y = mx + c, where m and c are parameters learned from data

Parameter Learning

m and c are optimized to minimize prediction error on training data



Slope (m)

Represents the slope of the line, indicating how much y changes for each unit of x

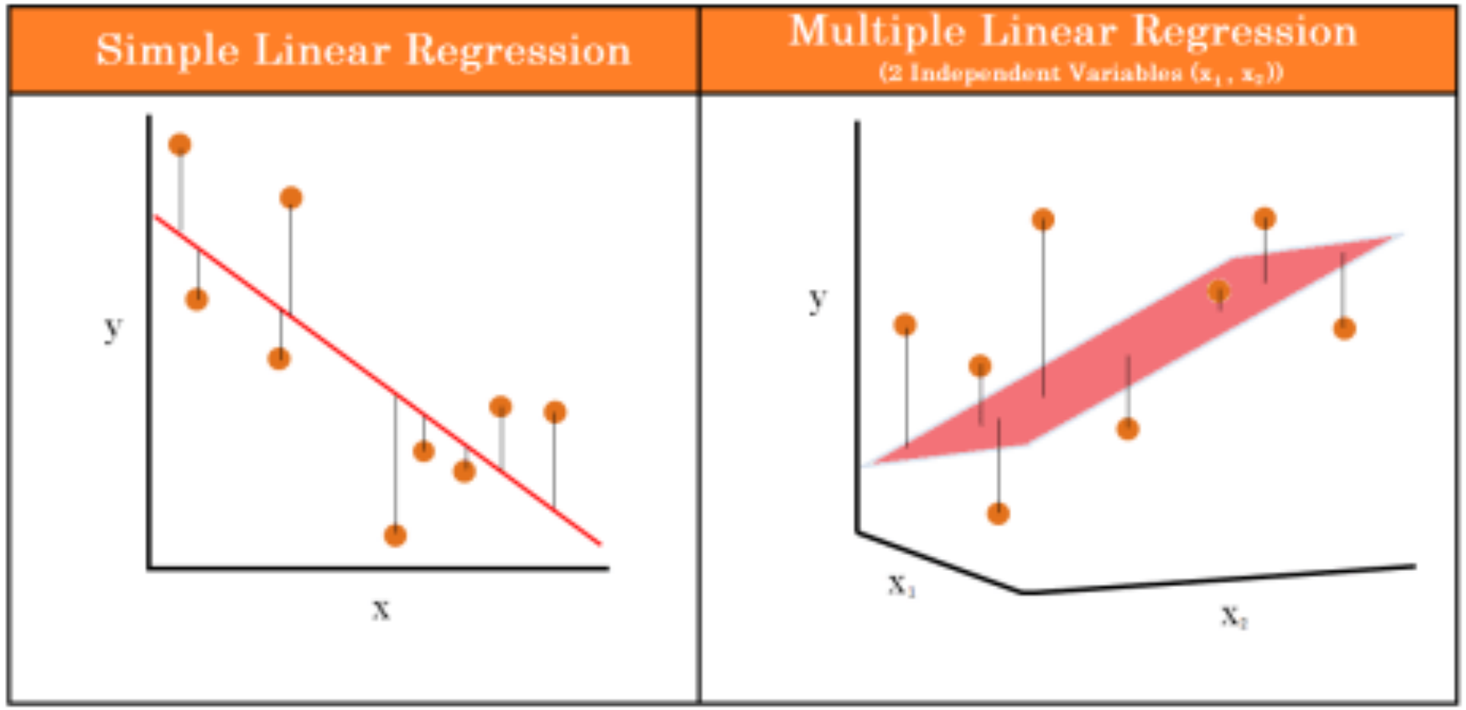
Intercept (c)

The y-intercept, showing the predicted value when x=0

https://pub.towardsai.net/machine-learning-the-first-step-is-to-understand-simple-linear-regression-30dde8b4739

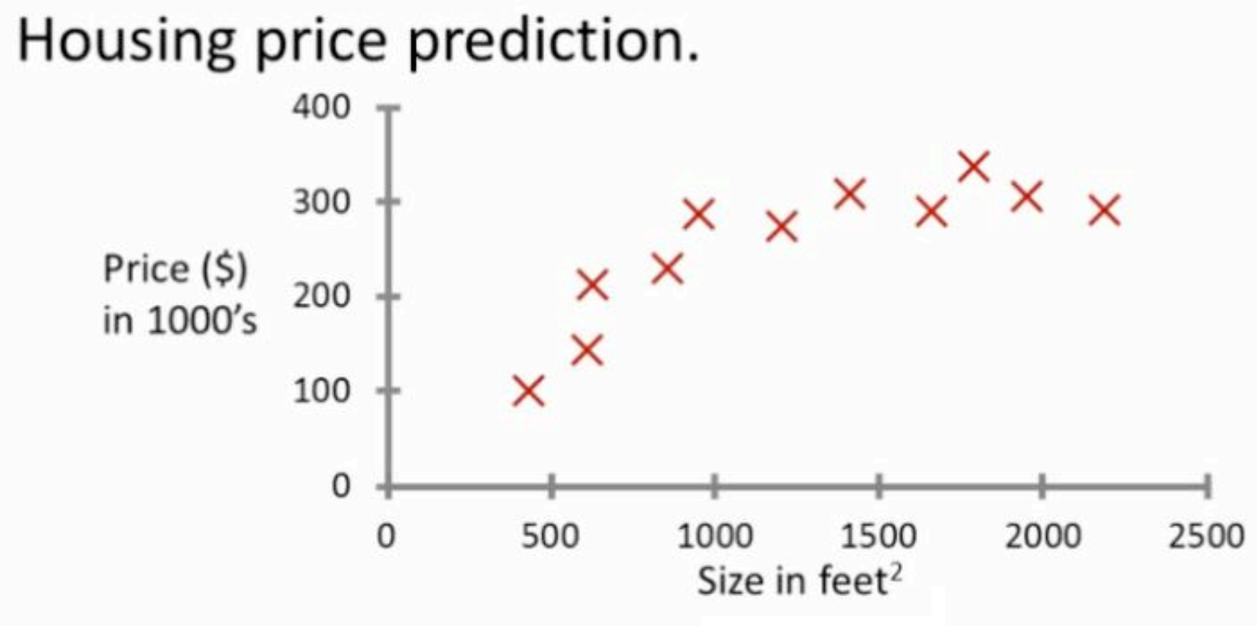
Linear Regression

$$y = f(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_D x_D = \boldsymbol{\beta} \cdot \boldsymbol{x}$$

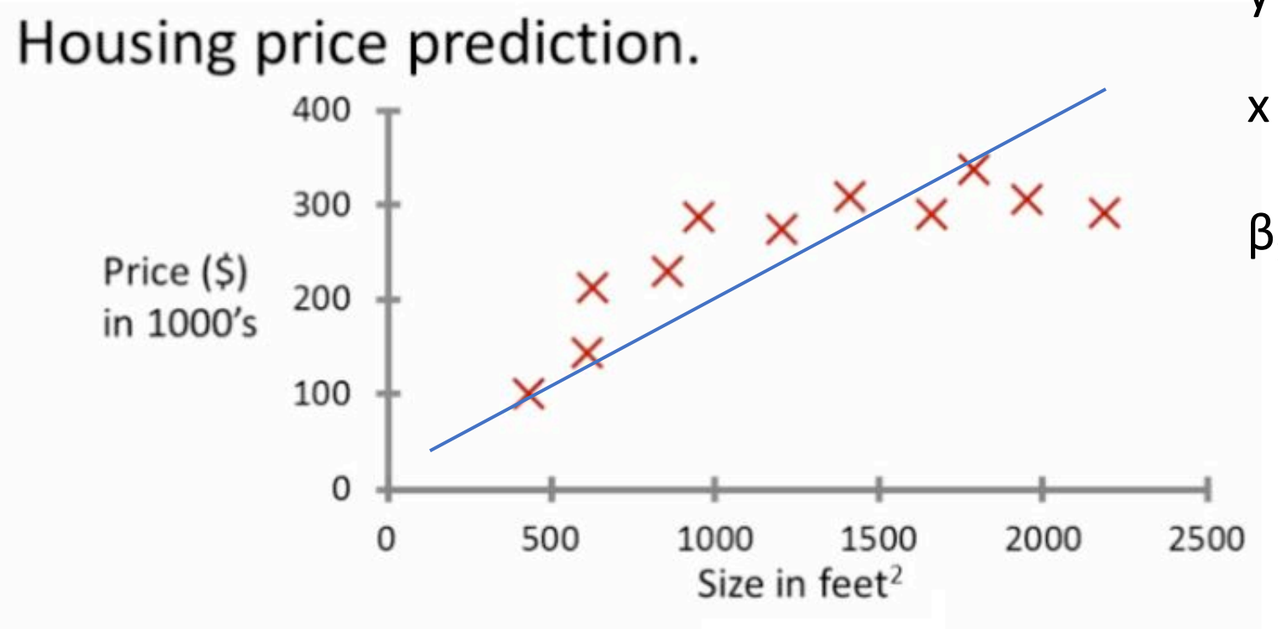


https://thaddeus-segura.com/multiple-linear/

Linear Regression Example



Linear Regression Example



This relationship can be represented as a straight line:

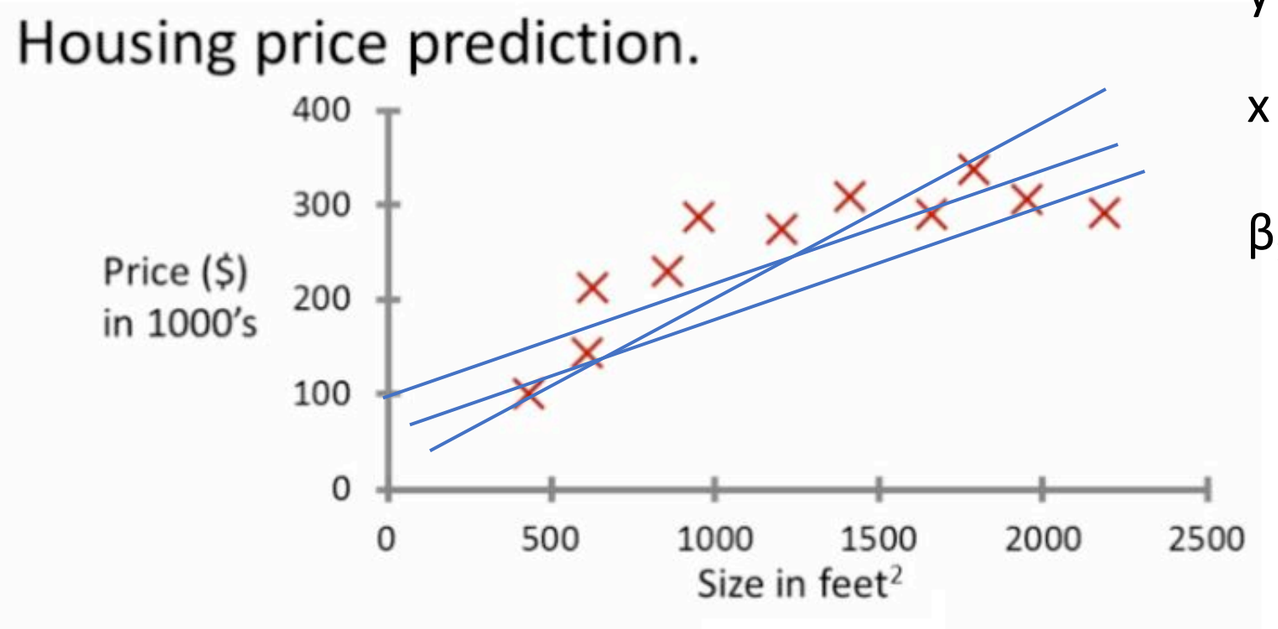
$y = \beta_1 x + \beta_0$

y (Dependent Variable) → Price of house

x (Independent Variable) → Size of house

β_1 and β_0 are model parameters

Linear Regression Example



This relationship can be represented as a straight line:

$y = \beta_1 x + \beta_0$

y (Dependent Variable) → Price of house

x (Independent Variable) → Size of house

β_1 and β_0 are model parameters

Based on different values of β_1 and β_0, we can fit have different lines to the data

We need to find β_1 and β_0 values of the straight line that best fits to the data. Solution: **Least Squares Method**

Fitting the Model- Least Squares Method

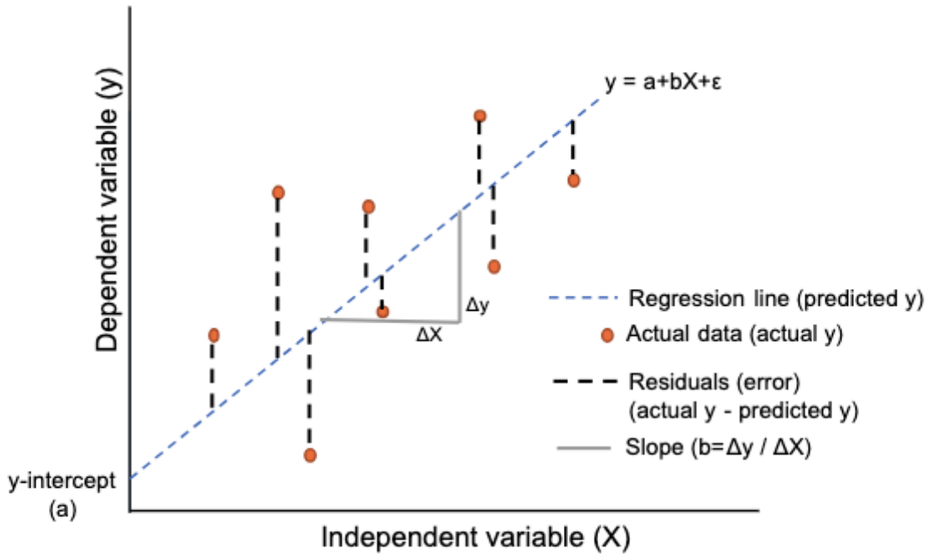
- **Least squares method**: find the line that minimises the sum of the squares of the vertical distances between predicted $\hat{y}$ and actual y.
- **Minimise** the Mean Squared Error (MSE) of N data points

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{\beta}\cdot\boldsymbol{x}_i)^2$$

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \text{MSE}$$

loss function

$$residuals = actual\ y(y_i) - predicted\ y\ (\hat{y}_i)$$



$$\hat{y} = f(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_D x_D = \boldsymbol{\beta}\cdot\boldsymbol{x}$$

We need to find β_1 and β_0 values that have the lowest MSE. Solution: **Gradient Descent**

https://www.reneshbedre.com/blog/linear-regression.html

Gradient Descent

- **Minimise** the Mean Squared Error (MSE) of N data points

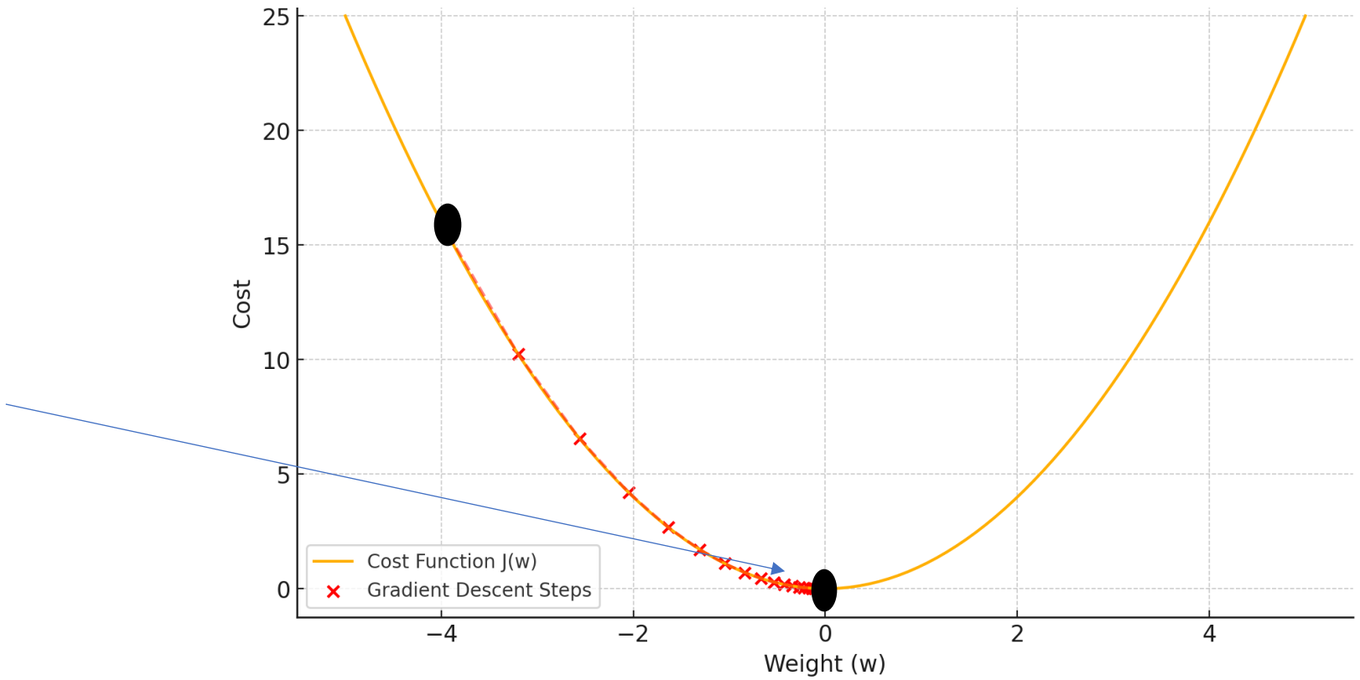
$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{\beta}\cdot\boldsymbol{x}_i)^2$$

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \mathrm{MSE}$$

loss function

Gradient Descent is an optimisation algorithm used to **find the best values** of a model's parameters (like weights in linear regression) that **minimise a loss function** — basically, the error the model makes on training data.

We want to reach here; we want to know for what value of 'w', J=f(w) will assume the minimum value



This curve J=f(w) represents the values of loss function will assume for different values of w

Gradient Descent

- **Minimise** the Mean Squared Error (MSE) of N data points

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{\beta}\cdot\boldsymbol{x}_i)^2$$

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \mathrm{MSE}$$ ← **loss function**

The **gradient** is just the **slope** of the function — it tells us **how steep** the function is and in **which direction** to move to reach a minimum.

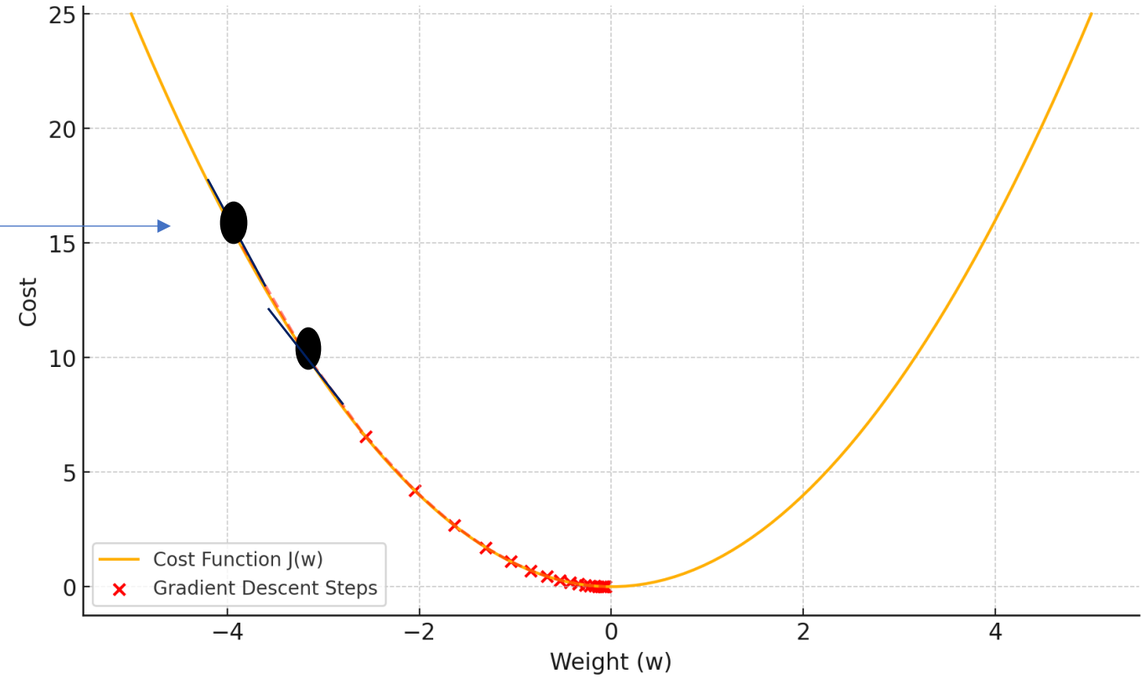
Update rule:

$$w_{\text{new}} = w_{\text{old}} - \alpha \cdot \frac{dJ}{dw}$$

gradient

learning rate (step size)

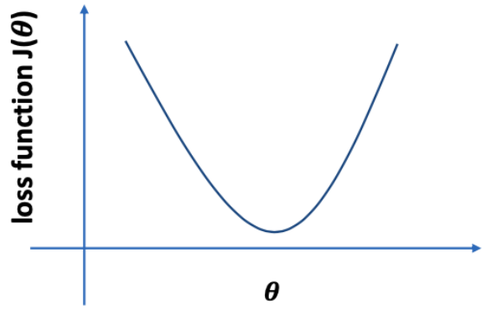
$\frac{dJ}{dw}$



This curve J=f(w) represents the values of loss function will assume for different values of w

Gradient Descent

- α is learning rate, which defines how big a step to update θ_k^{iter}
  - If α is too small, the algorithm might be slow.
  - If α is too large, you might miss the minimum.



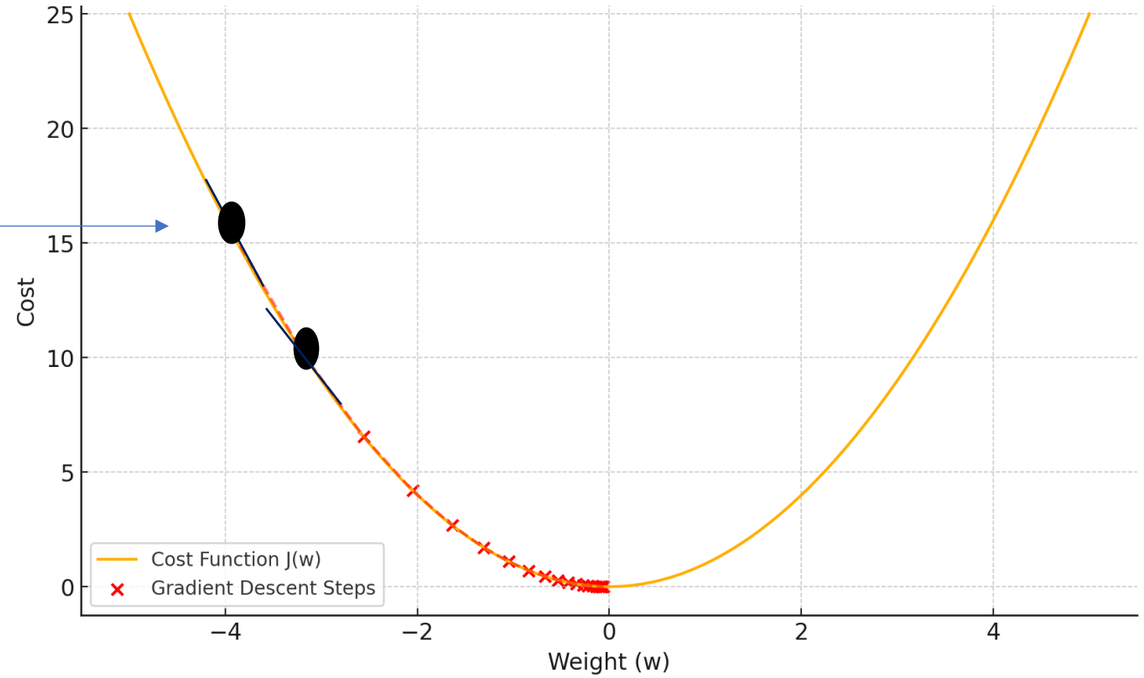
Update rule:

$$w_{\text{new}} = w_{\text{old}} - \alpha \cdot \frac{dJ}{dw}$$

gradient

learning rate (step size)

$\frac{dJ}{dw}$



This curve J=f(w) represents the values of loss function will assume for different values of w

Gradient Descent: Iterative Parameter Optimization

Initialize Parameters

Start with random or zero values for parameters β_1 and β_0

Calculate predictions

Using current β_1 and β_0 compute $\hat{y}$

Compute Gradient

Calculate partial derivatives of the error function with respect to each parameter

Update Parameters

$$w_{\text{new}} = w_{\text{old}} - \alpha \cdot \frac{dJ}{dw}$$

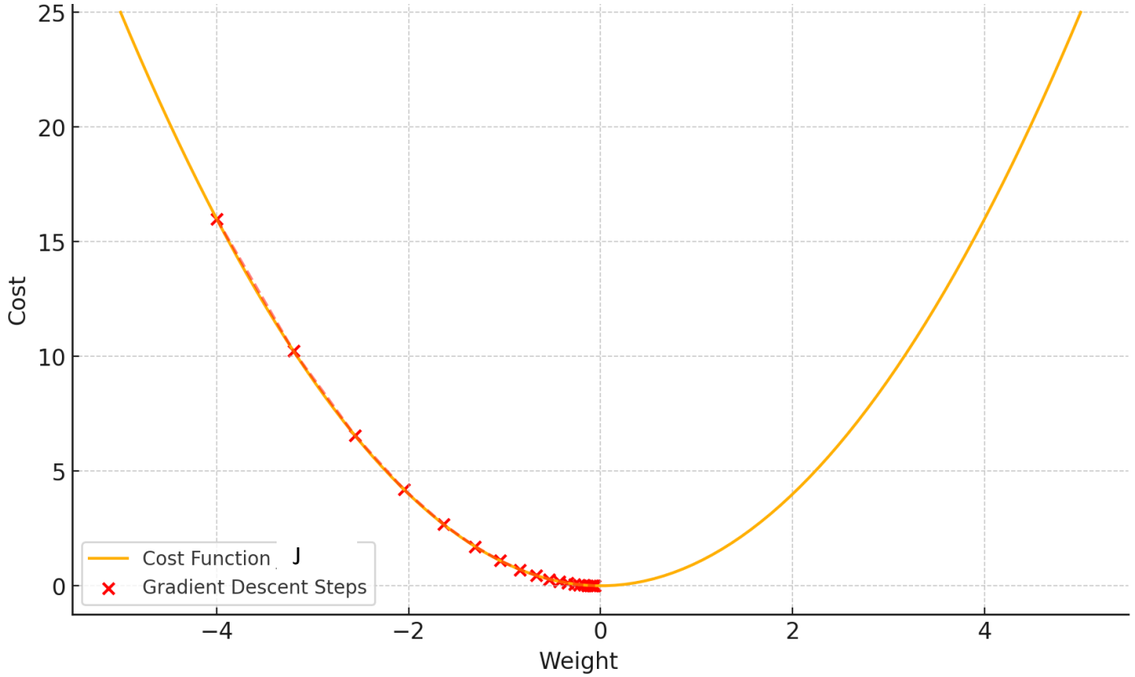
θ: parameter (like weight)
α: **learning rate** (step size)
$\frac{dJ}{d\theta}$: slope (gradient) of the error

Learning rate α controls step size and affects convergence

Iterate Until Convergence

Repeat steps 2-3 until parameters stabilize or error is minimized

Convergence criteria is a hyperparameter that must be set beforehand



Gradient Descent: For Linear Regression

$$w_{\text{new}} = w_{\text{old}} - \alpha \cdot \frac{dJ}{dw}$$

For linear regression

$$\beta_k^{iter+1} := \beta_k^{iter} - \alpha \frac{\partial\, Error(\beta_k^{iter})}{\partial \beta_k^{iter}} = \beta_k^{iter} + \frac{2\alpha}{N} \sum_{i=1}^{N} x_{ik}(y_i - \widehat{y_i^{iter}})$$

Gradient descent steps for iteration $iter$

- Calculate prediction $\widehat{y_i^{iter}}$ for each training instance
- Compare prediction with actual value y_i
- Multiply by the corresponding attribute value x_{ik}
- Update weight β_k after all the training instances are processed

Q3

Recall that the update rule for Gradient Descent with respect to Mean Squared Error (MSE) is as follows:

$$\beta_k^{t+1} = \beta_k^t + \frac{2\alpha}{N} \sum_{i=1}^{N} x_{ik}(y_i - \hat{y}_i^t)$$

Suppose we wish to fit a linear regression model to predict y from x given the following instances:

x	y
1	1
2	2
2	3

Intialise the model parameters to 0, so the initial model is $y = 0 + 0x$. Set the learning rate to $\alpha = 0.15$ and fit the model using gradient descent.

For linear regression

$$\beta_k^{iter+1} := \beta_k^{iter} - \alpha \frac{\partial\, Error(\beta_k^{iter})}{\partial \beta_k^{iter}} = \beta_k^{iter} + \frac{2\alpha}{N} \sum_{i=1}^{N} x_{ik}(y_i - \widehat{y_i^{iter}})$$

Gradient descent steps for iteration $iter$

- Calculate prediction $\widehat{y_i^{iter}}$ for each training instance
- Compare prediction with actual value y_i
- Multiply by the corresponding attribute value x_{ik}
- Update weight β_k after all the training instances are processed

We will use the **gradient descent update rule for MSE**:

$$\beta_k^{(t+1)} = \beta_k^{(t)} + \frac{2\alpha}{N} \sum_{i=1}^{N} x_{ik}(y_i - \hat{y}_i)$$

Where:

- β_0 is the bias (so $x_{i0} = 1$)
- β_1 is the weight for the feature x
- $\alpha = 0.15$ (learning rate)
- $N = 3$ (number of data points)

Step 1: Initialize parameters

$$\beta_0^{(0)} = 0, \quad \beta_1^{(0)} = 0$$

The model predictions $\hat{y}$ and ground truth labels y are:

$$\hat{y}_1 = 0 + 0(1) = 0, y_1 = 1$$

$$\hat{y}_2 = 0 + 0(2) = 0, y_2 = 2$$

$$\hat{y}_3 = 0 + 0(2) = 0, y_3 = 3$$

x	y
1	1
2	2
2	3

For linear regression

$$\beta_k^{iter+1} := \beta_k^{iter} - \alpha \frac{\partial\, Error(\beta_k^{iter})}{\partial \beta_k^{iter}} = \beta_k^{iter} + \frac{2\alpha}{N} \sum_{i=1}^{N} x_{ik}(y_i - \widehat{y_i^{iter}})$$

Gradient descent steps for iteration $iter$

- Calculate prediction $\widehat{y_i^{iter}}$ for each training instance
- Compare prediction with actual value y_i
- Multiply by the corresponding attribute value x_{ik}
- Update weight β_k after all the training instances are processed

Step 2: Compute First Update

$$\beta_0^{(1)} = 0 + \frac{2 * 0.15}{3}(1(1-0) + 1(2-0) + 1(3-0)) = 0.6$$

$$\beta_1^{(1)} = 0 + \frac{2 * 0.15}{3}(1(1-0) + 2(2-0) + 2(3-0)) = 1.1$$

New model after step 1:

$$\hat{y} = 0.6 + 1.1x$$

Step 3: Compute New Predictions

$$\hat{y}_1 = 0.6 + 1.1 \cdot 1 = 1.7, \quad y_1 = 1$$

$$\hat{y}_2 = 0.6 + 1.1 \cdot 2 = 2.8, \quad y_2 = 2$$

$$\hat{y}_3 = 0.6 + 1.1 \cdot 2 = 2.8, \quad y_3 = 3$$

Step 4: Compute Second Update

$$\beta_0^{(2)} = 0.6 + \frac{2 * 0.15}{3}(1(1 - 1.7) + 1(2 - 2.8) + 1(3 - 2.8)) = 0.47$$

$$\beta_1^{(2)} = 1.1 + \frac{2 * 0.15}{3}(1(1 - 1.7) + 2(2 - 2.8) + 2(3 - 2.8)) = 0.91$$

New model after step 4:

$$\hat{y} = 0.47 + 0.91x$$

Analytical Solution

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{m} w_j x_j = \mathbf{w} \cdot \mathbf{x}$$

where

- y is the *target variable*;
- $\mathbf{x} = [x_0, x_1, \ldots, x_m]$ is a vector of *features* (we define $x_0 = 1$ which is our bias); and
- $\mathbf{w} = [w_0, \ldots, w_m]$ are the *weights*.

We saw in lectures that for finding the optimum weights we can *minimize* the following mean squared errors:

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{n} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2$$

It's possible to solve for the optimal weights $\mathbf{w}^\star$ analytically by solving for $\nabla_{\mathbf{w}} E(\mathbf{w}) = 0$. This yields the *normal equations* for the least squares problem:

$$\mathbf{w}^* = \left[\mathbf{X}^\top \mathbf{X}\right]^{-1} \mathbf{X}^\top \mathbf{y}$$

where X is the matrix of attributes, which is also called *design matrix*. In our simple 1-feature case this is:

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \quad \text{and} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

When to Use Linear Regression (And When Not To)

When to Use

- Data shows approximately linear relationships
- Features are independent
- Limited or no strong outliers present
- You want a simple and interpretable model

When to Avoid

- Data exhibits clear nonlinear patterns
- Significant outliers distort relationships when using MSE

When linear regression assumptions are violated, alternatives like polynomial regression or entirely different algorithms may be more appropriate.

Polynomial Regression: Beyond Linear Relationships

When data exhibits clear nonlinear patterns, linear regression falls short. Polynomial regression extends the model by adding higher-degree terms.

Equation Form

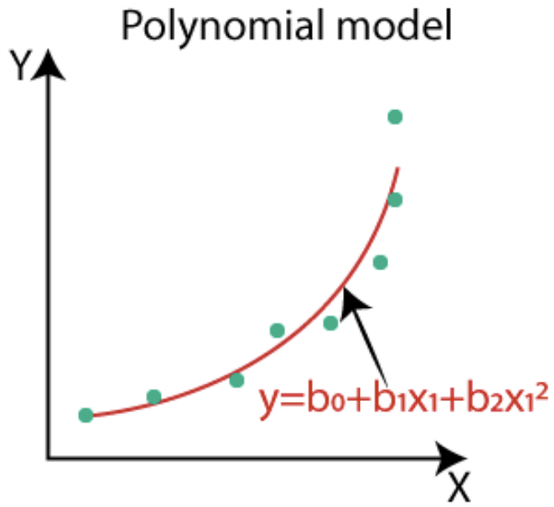
$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_n x^n + \varepsilon$

Increased Flexibility

Higher degree polynomials can fit more complex data relationships.

Overfitting Risk

Too many degrees can create a model that memorizes training data.



https://medium.com/analytics-vidhya/understanding-polynomial-regression-5ac25b970e18

Error Analysis

- Analysis of the sorts of errors that a given model makes
  - **identifying** different “classes” of error that the system makes (predicted vs. actual labels)
  - **hypothesising** as to what has caused the different errors, and testing those hypotheses against the actual data
  - **quantifying** whether (for different classes) it is a question of data quantity/sparsity, or something more fundamental than that
  - **feeding** those hypotheses **back** into feature/model engineering to see if the model can be improved

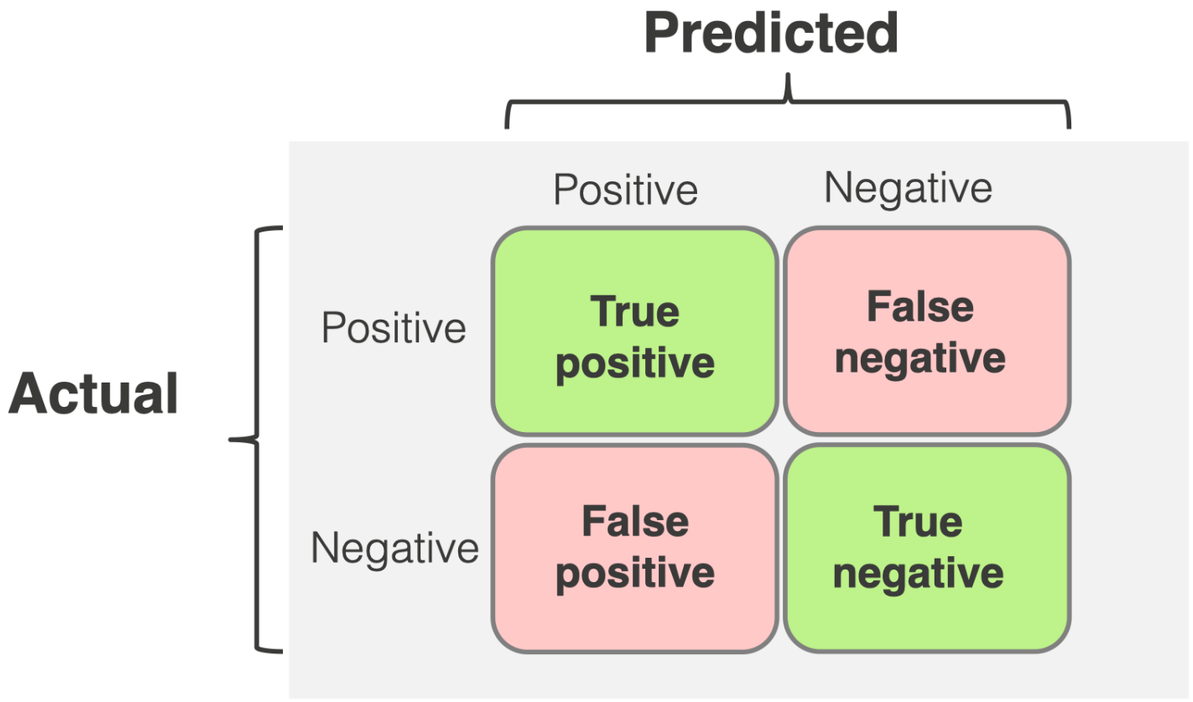
Q5

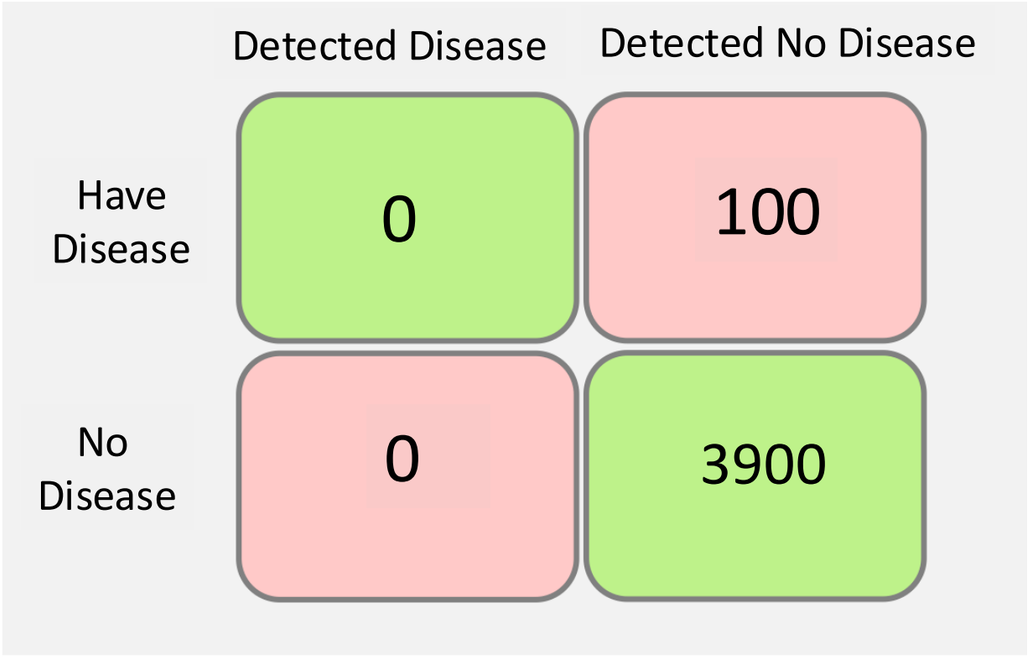
You are developing a model to detect an extremely contagious disease. Your data consists of 4000 patients, out of which 100 are known to have the disease. You achieve 96% classification accuracy.

1.Would you trust this model to identify patients with the disease, based on this accuracy result? Why or why not?

2.What type of error is most important in this task?

3.Name at least one appropriate evaluation metric that you would choose to evaluate your model.





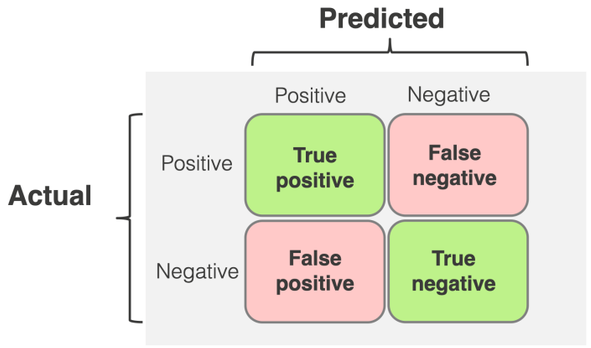
Q5 -Solution

You are developing a model to detect an extremely contagious disease. Your data consists of 4000 patients, out of which 100 are known to have the disease. You achieve 96% classification accuracy.

1.Would you trust this model to identify patients with the disease, based on this accuracy result? Why or why not? No, heavily imbalanced class

2.What type of error is most important in this task? A Type II error, or false negatives, is the most critical error because undetected cases can continue to spread the disease to others, amplifying public health risks.

3.Name at least one appropriate evaluation metric that you would choose to evaluate your model. Recall



$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$