UNIT-2

Machine Learning

Dr. P. Chanthini

What is Linear Regression?

- Linear Regression is a key data science tool for predicting continuous outcomes
- Linear regression predicts the relationship between two variables by assuming they have a straight-line connection.
- It finds the best line that minimizes the differences between predicted and actual values.

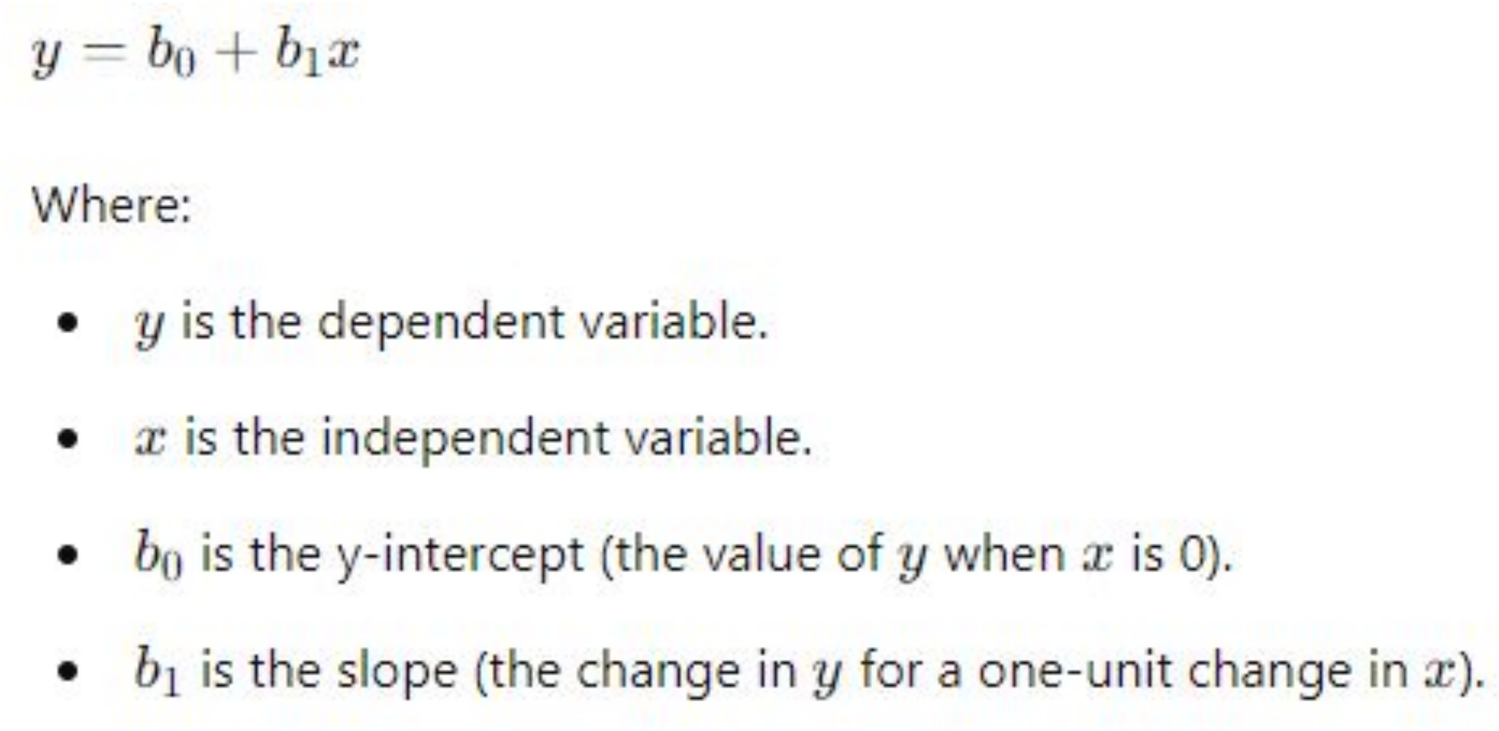
Types of Linear Regression

Simple Linear Regression involves one independent variable, while **Multiple Linear Regression** involves two or more independent variables.

Simple Linear Regression

- In a simple linear regression, there is one independent variable and one dependent variable.
- The model estimates the slope and intercept of the l**ine of best fit**, which represents the relationship between the variables.
- The slope represents the change in the dependent variable for each unit change in the independent variable, while the **intercept** represents the predicted value of the dependent variable when the independent variable is zero.

Equation of a Line: The relationship in simple linear regression (with one independent variable) is described by the equation:

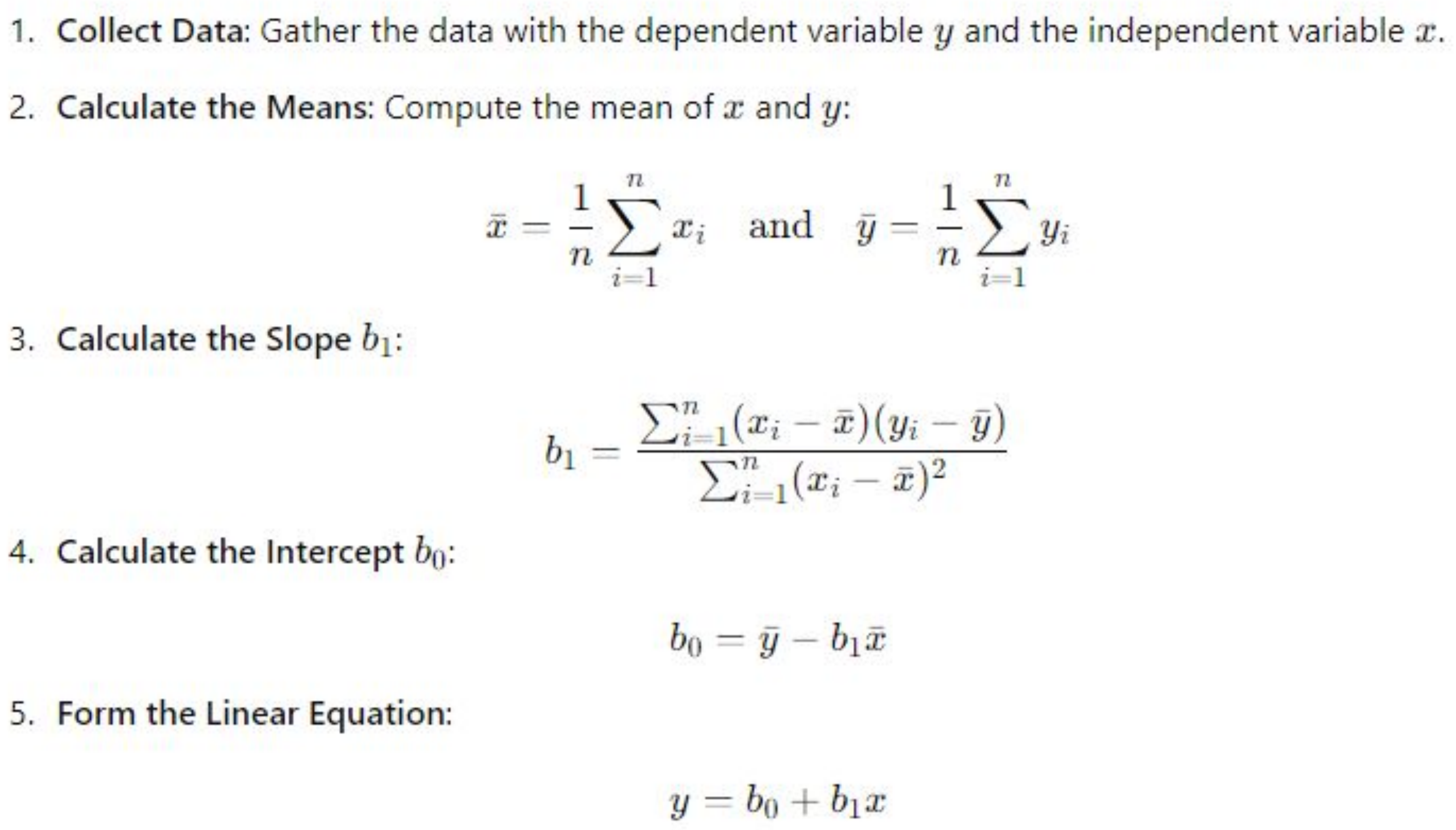


$$y = b_0 + b_1 x$$

Where:

- y is the dependent variable.
- x is the independent variable.
- b_0 is the y-intercept (the value of y when x is 0).
- b_1 is the slope (the change in y for a one-unit change in x).

Steps to Perform Linear Regression



1. **Collect Data:** Gather the data with the dependent variable y and the independent variable x.
2. **Calculate the Means:** Compute the mean of x and y:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

3. **Calculate the Slope b_1:**

$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

4. **Calculate the Intercept b_0:**

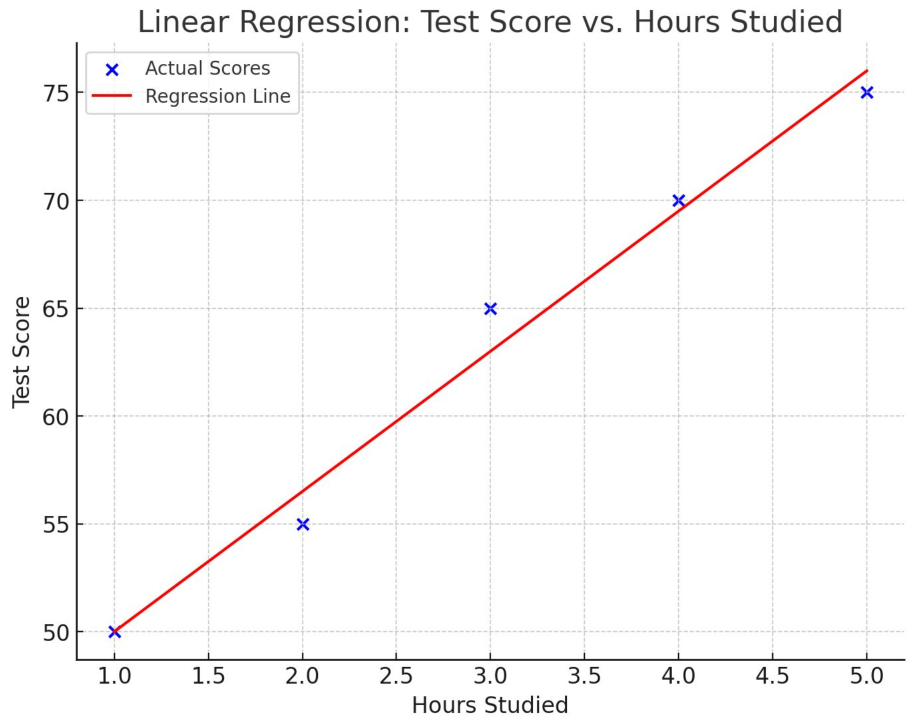
$$b_0 = \bar{y} - b_1\bar{x}$$

5. **Form the Linear Equation:**

$$y = b_0 + b_1 x$$

Dataset

Hours Studied (x)	Test Score (y)
1	50
2	55
3	65
4	70
5	75



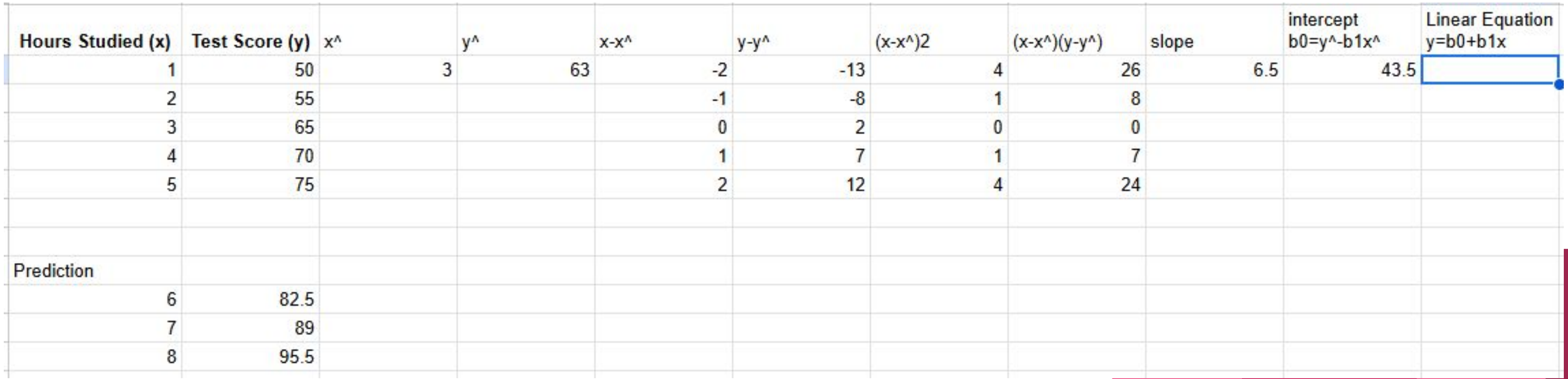
https://docs.google.com/spreadsheets/d/1yyMW4g9jkmzHGOMKuGnFoF7BsL1qsQc0HPR2oBoq9zY/edit?usp=sharing

MEAN

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

SLOPE

$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$



Hours Studied (x)	Test Score (y)	x^	y^	x-x^	y-y^	(x-x^)2	(x-x^)(y-y^)	slope	intercept b0=y^-b1x^	Linear Equation y=b0+b1x
1	50	3	63	-2	-13	4	26	6.5	43.5	
2	55			-1	-8	1	8			
3	65			0	2	0	0			
4	70			1	7	1	7			
5	75			2	12	4	24			
Prediction										
6	82.5									
7	89									
8	95.5									

Residual plots — Before evaluation of a model

We know that linear regression tries to fit a line that produces the smallest difference between predicted and actual values, where these differences are unbiased as well. This difference or error is also known as **residual. (***Unbiased means there is no systematic pattern of distribution of the predicted values)*

Residual = actual value — predicted value

$$e = y - \hat{y}$$

One of the assumptions of a linear regression model is that the errors must be normally distributed. This means, make sure your residuals are distributed around zero for the entire range of predicted values. Thus, if the residuals are evenly scattered, then your model may perform well.

What is the Best Fit Line?

In simple terms, the best-fit line is a line that best fits the given scatter plot. Mathematically, you obtain the best-fit line by minimizing the Residual Sum of Squares (RSS).

Evaluating the Model (Cost Function for Linear Regression)

- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R-squared (R^2)

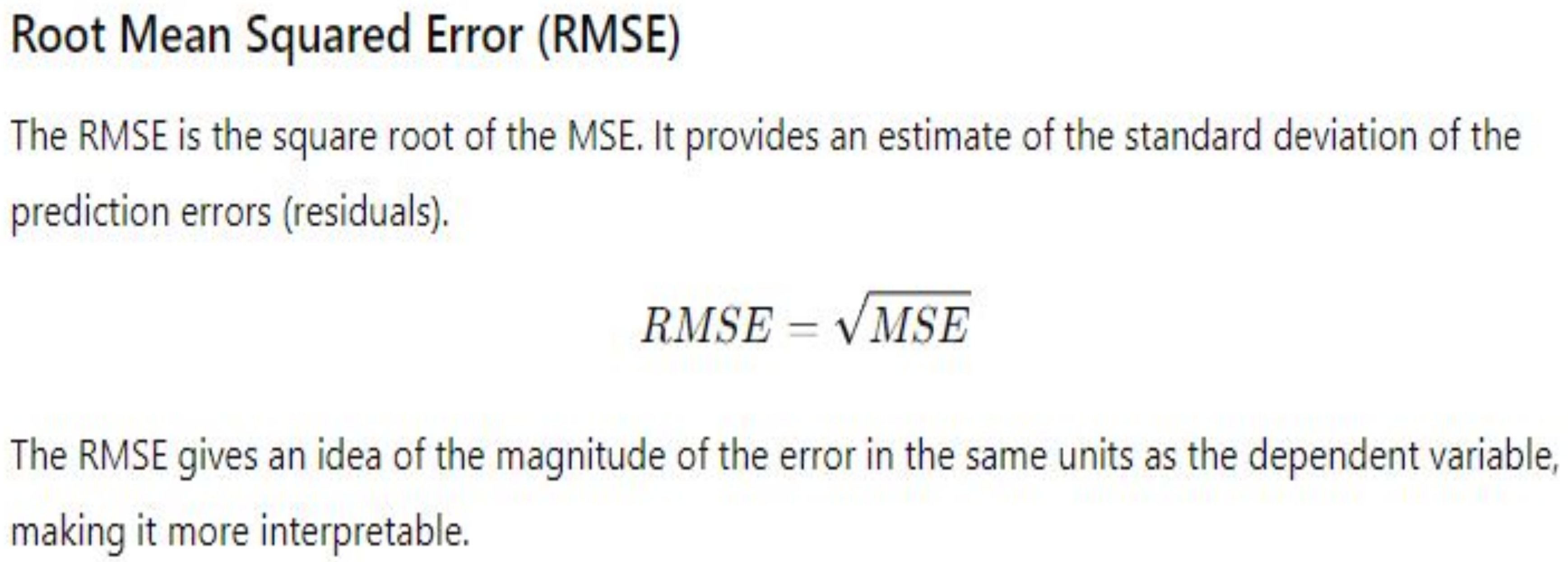
Mean Squared Error (MSE)

The MSE is the average of the squared differences between the actual values and the predicted values. It measures the average of the squares of the errors.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where:

- y_i are the actual values.
- $\hat{y}_i$ are the predicted values.
- n is the number of observations.



Root Mean Squared Error (RMSE)

The RMSE is the square root of the MSE. It provides an estimate of the standard deviation of the prediction errors (residuals).

$$RMSE = \sqrt{MSE}$$

The RMSE gives an idea of the magnitude of the error in the same units as the dependent variable, making it more interpretable.

R-squared (R^2)

R-squared is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. It provides an indication of how well the independent variable(s) predict the dependent variable.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Where:

- y_i are the actual values.
- $\hat{y}_i$ are the predicted values.
- $\bar{y}$ is the mean of the actual values.
- n is the number of observations.

Let's predict the scores using our model:

$$\hat{y}_1 = 43.5 + 6.5 \times 1 = 50$$

$$\hat{y}_2 = 43.5 + 6.5 \times 2 = 56$$

$$\hat{y}_3 = 43.5 + 6.5 \times 3 = 63$$

$$\hat{y}_4 = 43.5 + 6.5 \times 4 = 69.5$$

$$\hat{y}_5 = 43.5 + 6.5 \times 5 = 76$$

Errors and Squared Errors

$$(50 - 50)^2 = 0$$

$$(55 - 56)^2 = 1$$

$$(65 - 63)^2 = 4$$

$$(70 - 69.5)^2 = 0.25$$

$$(75 - 76)^2 = 1$$

Sum of Squared Errors

$$0 + 1 + 4 + 0.25 + 1 = 6.25$$

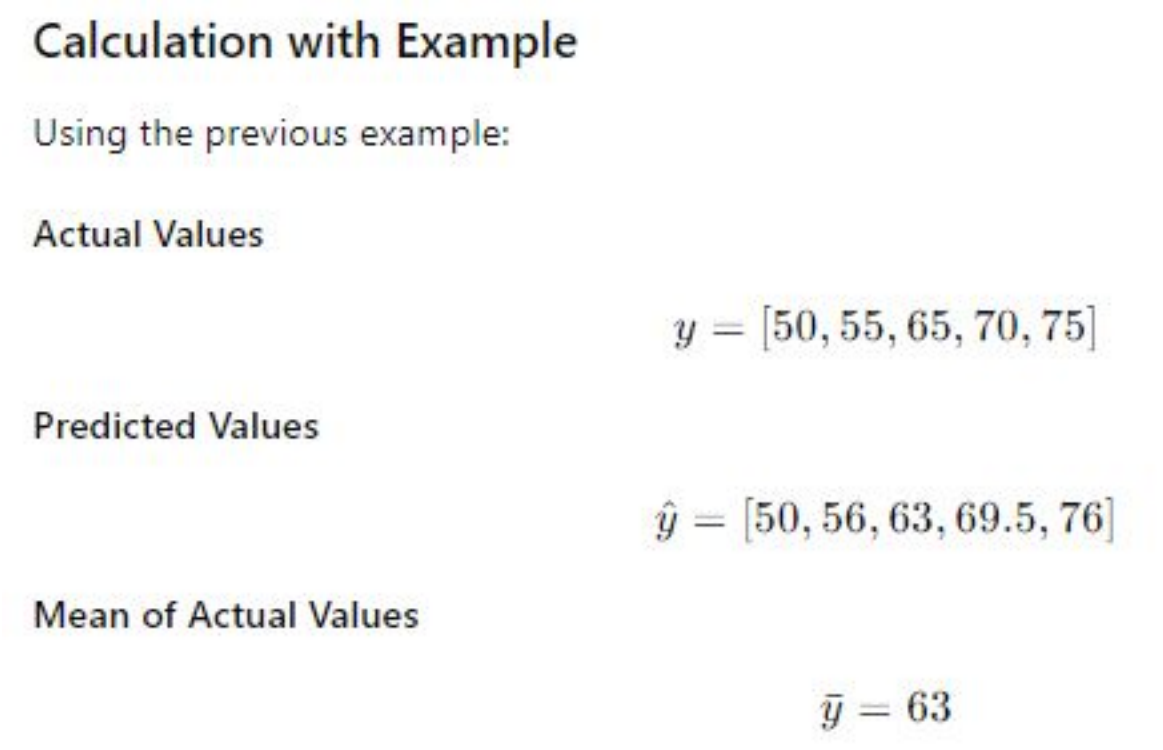
MSE

$$MSE = \frac{6.25}{5} = 1.25$$

RMSE

$$RMSE = \sqrt{1.25} \approx 1.12$$

R-squared (R^2)



Calculation with Example

Using the previous example:

Actual Values

$$y = [50, 55, 65, 70, 75]$$

Predicted Values

$$\hat{y} = [50, 56, 63, 69.5, 76]$$

Mean of Actual Values

$$\bar{y} = 63$$

Sum of Squares of Residuals (SSR)

$$SSR = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = (50 - 50)^2 + (55 - 56)^2 + (65 - 63)^2 + (70 - 69.5)^2 + (75 - 76$$

$$= 0 + 1 + 4 + 0.25 + 1 = 6.25$$

Total Sum of Squares (TSS)

$$TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2 = (50 - 63)^2 + (55 - 63)^2 + (65 - 63)^2 + (70 - 63)^2 + (75 - 63)^2$$

$$= (-13)^2 + (-8)^2 + (2)^2 + (7)^2 + (12)^2 = 169 + 64 + 4 + 49 + 144 = 430$$

R-squared (R^2)

$$R^2 = 1 - \frac{SSR}{TSS} = 1 - \frac{6.25}{430} \approx 1 - 0.0145 \approx 0.9855$$

The R-squared value of approximately 0.9855 indicates that approximately 98.55% of the variance in test scores is explained by the number of hours studied. This suggests a very good fit of the model to the data.

Implementation in Python

```
import numpy as np
from sklearn.linear model import LinearRegression
import matplotlib.pyplot as plt

# Your data
hours_studied = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
# Reshape to 2D array
test_scores = np.array([50, 55, 65, 70, 75])

# Create a linear regression model
model = LinearRegression()

# Fit the model to the data
model.fit(hours_studied, test_scores)

# Make predictions
predicted_scores = model.predict(hours_studied)
```

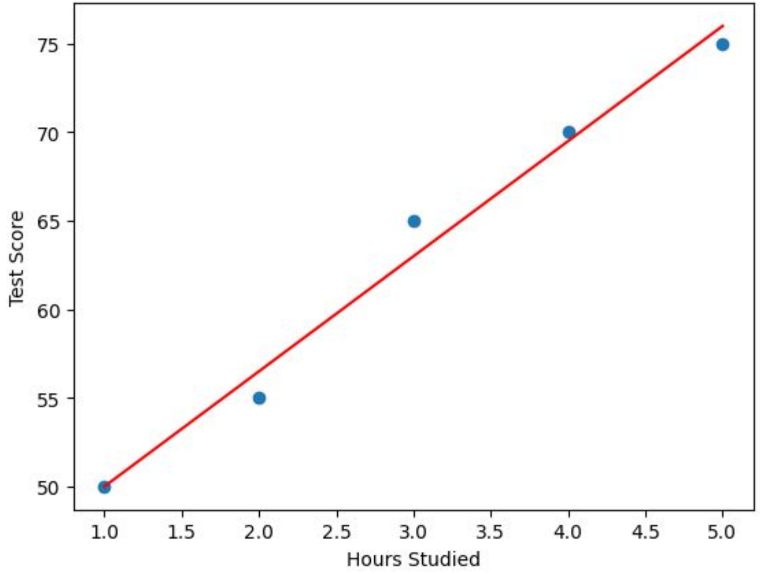
```python
# Print coefficients
print("Coefficient:", model.coef )
print("Intercept:", model.intercept_)

# Plot the data and the regression line
plt.scatter(hours_studied, test_scores)
plt.plot(hours_studied, predicted_scores,
color='red')
plt.xlabel('Hours Studied')
plt.ylabel('Test Score')
plt.show()
```

https://colab.research.google.com/drive/1hcrChy5_KIjn2JstJ4QPLTvn51gCVbSx?usp=drive_link

Output:

```
Coefficient: [6.5]
Intercept: 43.5
```



```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
# Make predictions
y_pred = model.predict(hours_studied)
y_true = test_scores  # Actual values

# Calculate evaluation metrics
mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_true, y_pred)
r2 = r2_score(y_true, y_pred)

print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("Mean Absolute Error:", mae)
print("R-squared:", r2)
```

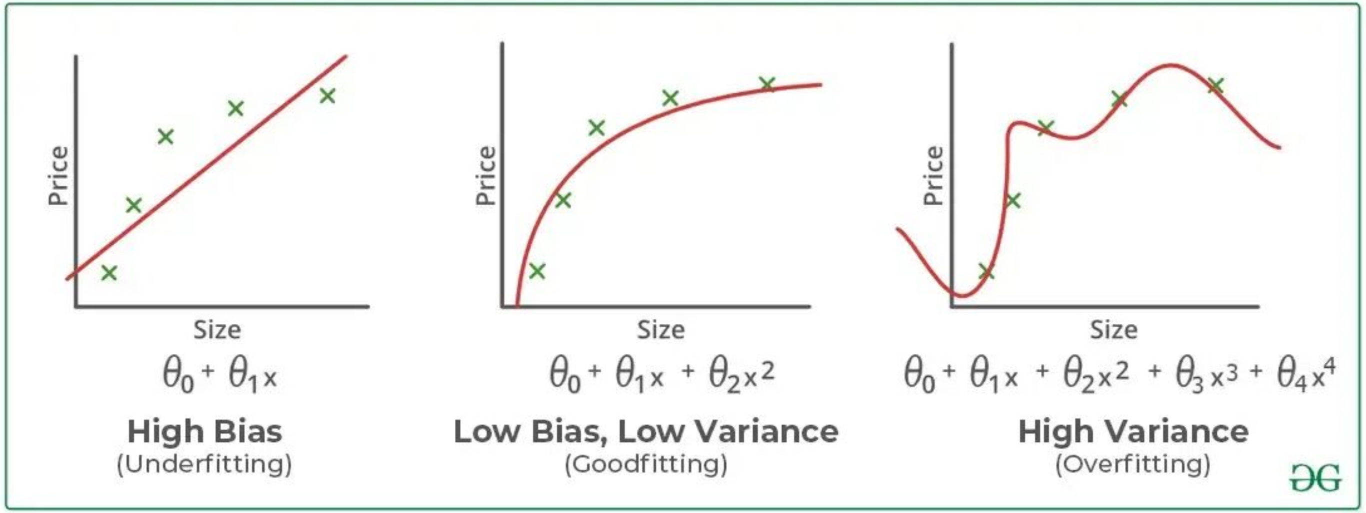
Output:

```
Mean Squared Error: 1.5
Root Mean Squared Error:
1.224744871391589
Mean Absolute Error: 1.0
R-squared: 0.9825581395348837
```

Underfitting and Overfitting in Regression Models

- ❖ Underfitting occurs when a model is too simple to capture the underlying pattern of the data. It performs poorly on both the training and test datasets because it cannot represent the complexity of the data.

- ❖ Overfitting occurs when a model is too complex and captures the noise in the training data along with the underlying pattern. It performs well on the training data but poorly on new, unseen test data.



Bias and Variance in Machine Learning

Bias: Bias refers to the error due to overly simplistic assumptions in the learning algorithm.

- ❖ It is the error due to the model's inability to represent the true relationship between input and output accurately.
- ❖ When a model has poor performance both on the training and testing data means high bias because of the simple model, indicating underfitting.

Characteristics:

- High bias can result in underfitting, where the model performs poorly on both the training and test datasets.

For example, a linear model trying to fit a complex, non-linear dataset will have high bias.

Variance: Variance, on the other hand, is the error due to the model's sensitivity to fluctuations in the training data.

- ❖ High variance occurs when a model learns the training data's noise and random fluctuations rather than the underlying pattern.

Characteristics:

- High variance can result in **overfitting**, where the model performs well on the training dataset but poorly on the test dataset.
- For instance, a highly complex model (like a deep neural network with many layers) might fit the training data very closely but fail to generalize to new data.

Bias-Variance Tradeoff

- The relationship between bias and variance is often depicted as a tradeoff:
  - Reducing bias typically increases variance, and vice versa.
  - The goal in machine learning is to find a balance between bias and variance to minimize the overall error on unseen data, often measured by cross-validation techniques.

Techniques to Address Underfitting and Overfitting

1. Regularization:
   - Ridge Regression (L2 Regularization): Adds a penalty for larger coefficients to prevent overfitting.
   - Lasso Regression (L1 Regularization): Can set some coefficients to zero, effectively performing feature selection and preventing overfitting.
   - Elastic Net Regression: Combines L1 and L2 regularization to balance their benefits.
2. Cross-Validation:
   - Split the data into training and validation sets multiple times to ensure the model performs well on unseen data.
3. Model Complexity:
   - Choose a model with the right complexity for the data. Use simpler models for simpler patterns and more complex models for more complex patterns.
4. Feature Selection:
   - Select only relevant features to reduce model complexity and prevent overfitting.
5. Data Augmentation:
   - Increase the size of the training dataset to provide the model with more examples to learn from.

Regularization

- Ridge and Lasso regularization are techniques used to prevent overfitting in linear regression by adding a penalty to the model for having large coefficients.
- This helps the model generalize better to new data. Let's go through each one step-by-step, using the given example of predicting test scores based on hours studied and hours slept.

Ridge Regression (L2 Regularization)

Ridge regression adds a penalty equal to the sum of the squared values of the coefficients (excluding the intercept). The goal is to minimize the following objective function:

$$\text{Objective Function} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

$$\text{Objective Function} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Where:

- y_i are the actual values.
- $\hat{y}_i$ are the predicted values.
- β_j are the coefficients.
- λ is the regularization parameter (a non-negative value).

Lasso Regression (L1 Regularization)

Lasso regression adds a penalty equal to the sum of the absolute values of the coefficients (excluding the intercept). The goal is to minimize the following objective function:

$$\text{Objective Function} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Regularised Regression- Auto Selection of parameters Evaluation of Best Model representation

- To perform regularized regression with automatic selection of parameters and evaluate the best model representation, you can use cross-validation techniques.

- One common approach is to use GridSearchCV from `sklearn` to find the best hyperparameters for Ridge and Lasso regression.

https://colab.research.google.com/drive/1hcrChy5_KIjn2JstJ4QPLTvn51gCVbSx?usp=sharing

Steps to follow:

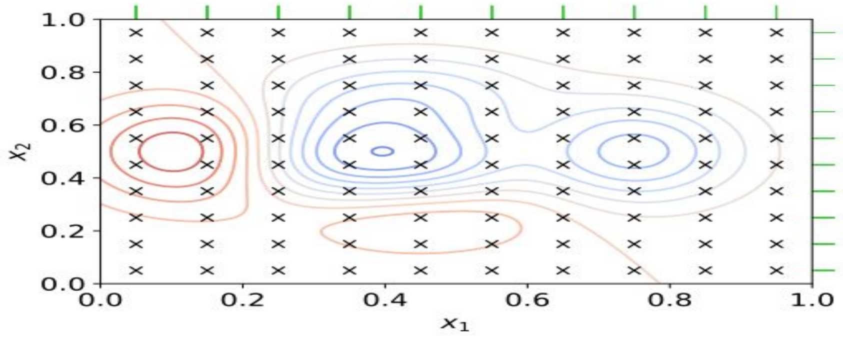
1. **Prepare the Data:**
   - Define the predictor variables (X) and target variable (y).
2. **Set up GridSearchCV for Hyperparameter Tuning:**
   - Define a range of hyperparameters (e.g., alpha values) to search for the best model.
3. **Evaluate Models:**
   - Use cross-validation to evaluate different models and select the best hyperparameters.
4. **Fit and Compare Models:**
   - Fit the models with the best hyperparameters and compare their performance.
5. **Visualize Results:**
   - Plot the results to visualize the best model fit.

Hyperparameters for a model can be chosen using several techniques such as Random Search, Grid Search, Manual Search, Bayesian Optimizations, etc.

Set up GridSearchCV for Hyperparameter Tuning

- ➔ **Hyperparameters are the parameters that are set before training a machine learning model.**
- ➔ **These parameters can have a significant impact on the performance of a model and, therefore, need to be carefully chosen.**
- ➔ **The process of selecting the best hyperparameters is called hyperparameter tuning.**

GridSearchCV is a scikit-learn function that performs hyperparameter tuning by training and evaluating a machine learning model using different combinations of hyperparameters.



Some popular cross-validation techniques:

Holdout:

Simplest method, splitting data into training and testing sets (typically 80/20 or 50/50).

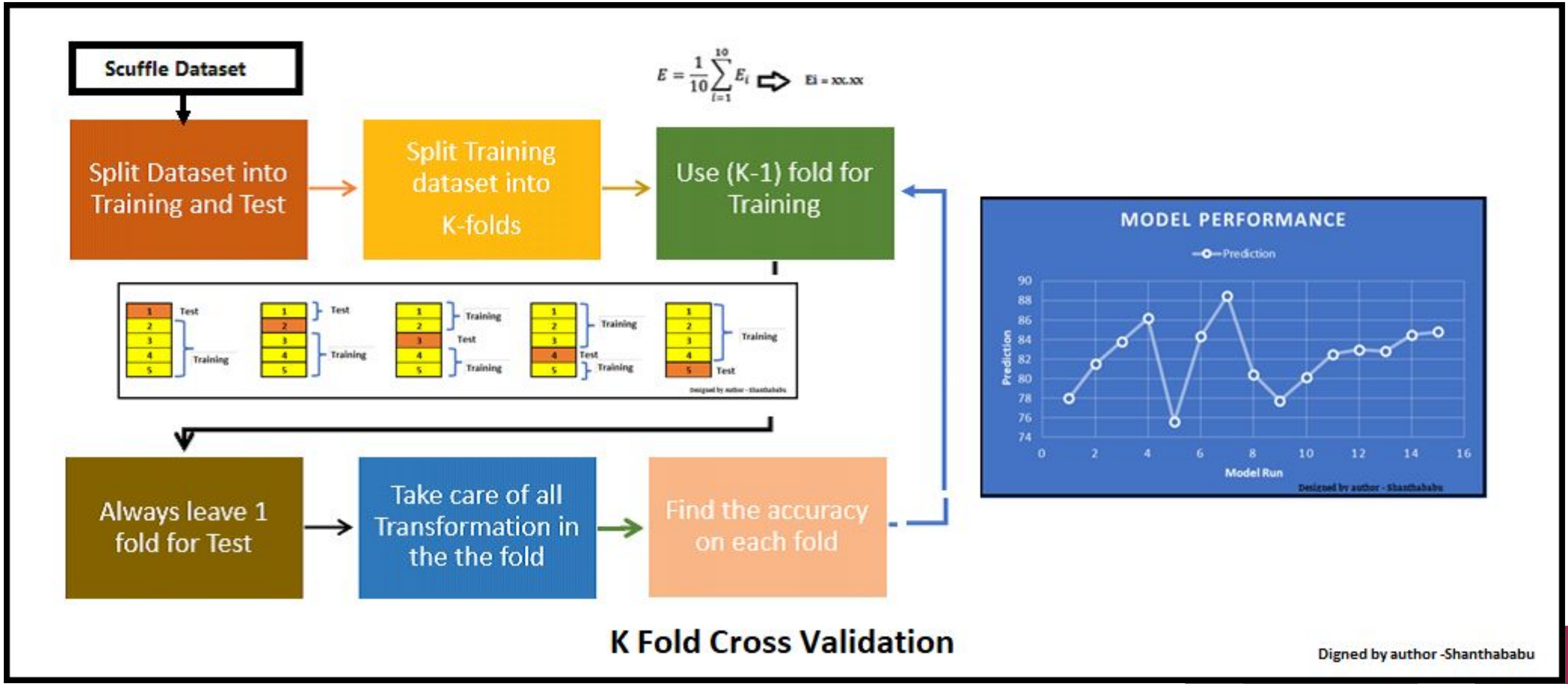
K-fold:

Splits data into k folds, using one for validation each time and training on the remaining k-1 folds.

LOOCV (Leave One Out Cross Validation) - leaves only one data-point of the available dataset and then iterates for each data-point.

Nested cross-validation:

Also known as double cross-validation, it involves an inner loop for hyperparameter tuning and an outer loop for model evaluation, reducing bias.


Scuffle Dataset
Split Dataset into Training and Test
Split Training dataset into K-folds
Use (K-1) fold for Training
$E = \frac{1}{10}\sum_{i=1}^{10} E_i$
Ei = xx.xx
Test
Training
Designed by author - Shanthababu
MODEL PERFORMANCE
Prediction
Model Run
Designed by author - Shanthababu
Always leave 1 fold for Test
Take care of all Transformation in the the fold
Find the accuracy on each fold
K Fold Cross Validation
Digned by author -Shanthababu

https://colab.research.google.com/drive/1hcrChy5_KIjn2JstJ4QPLTvn51gCVbSx3[illegible]rive_link

What is the Classification Algorithm?

- The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data.
- In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups.
- Such as, **Yes or No, 0 or 1, Spam or Not Spam, cat or dog,** etc. Classes can be called as targets/labels or categories.

Step 1: Understanding the Problem

Classification involves predicting a categorical label (class) for given input data. Examples include classifying emails as spam or not spam, recognizing digits in images, or predicting whether a patient has a disease based on medical tests.

Step 2: Collecting and Preparing the Data

1. **Collect Data**: Gather data relevant to the problem. The data should include features (inputs) and labels (outputs).
2. **Explore and Clean Data**: Inspect the data for any inconsistencies, missing values, or outliers. Clean the data to ensure it's suitable for training the model.
3. **Feature Selection/Extraction**: Select relevant features or create new features that might help improve the model's performance.

Step 3: Splitting the Data

Split the dataset into two parts:

- **Training Set**: Used to train the model.
- **Test Set**: Used to evaluate the model's performance on unseen data.

A common split is 80% for training and 20% for testing.

Step 4: Choosing a Classification Algorithm

Select an appropriate classification algorithm based on the problem, data size, and nature of the data. Common algorithms include:

- k-Nearest Neighbors (k-NN)
- Decision Trees
- Random Forests
- Logistic Regression
- Support Vector Machines (SVM)
- Neural Networks

Step 5: Training the Model

1. **Initialize the Model**: Set up the chosen classification algorithm.
2. **Train the Model**: Fit the model to the training data. The model learns the relationship between features and labels during this step.

Step 6: Making Predictions

Use the trained model to predict labels for new, unseen data. This step involves inputting the features into the model and obtaining the predicted class labels.

Step 7: Evaluating the Model

1. **Accuracy:** The proportion of correctly predicted instances out of the total instances.
2. **Confusion Matrix:** A table showing the counts of true positives, true negatives, false positives, and false negatives.
3. **Precision, Recall, and F1-Score:** Metrics that provide insights into the performance, especially for imbalanced datasets.
4. **ROC Curve and AUC:** Graphical representation of the true positive rate vs. false positive rate and the area under the curve.

Step 8: Fine-Tuning the Model

1. **Hyperparameter Tuning**: Adjust the model's hyperparameters to improve performance. Techniques include Grid Search, Random Search, and Bayesian Optimization.
2. **Cross-Validation**: Use cross-validation to ensure the model's performance is robust and not due to a particular train-test split.

Step 9: Deploying the Model

Once the model is trained and evaluated, it can be deployed to make predictions on new data in real-world applications.

Understanding Evaluation Metrics in Detail

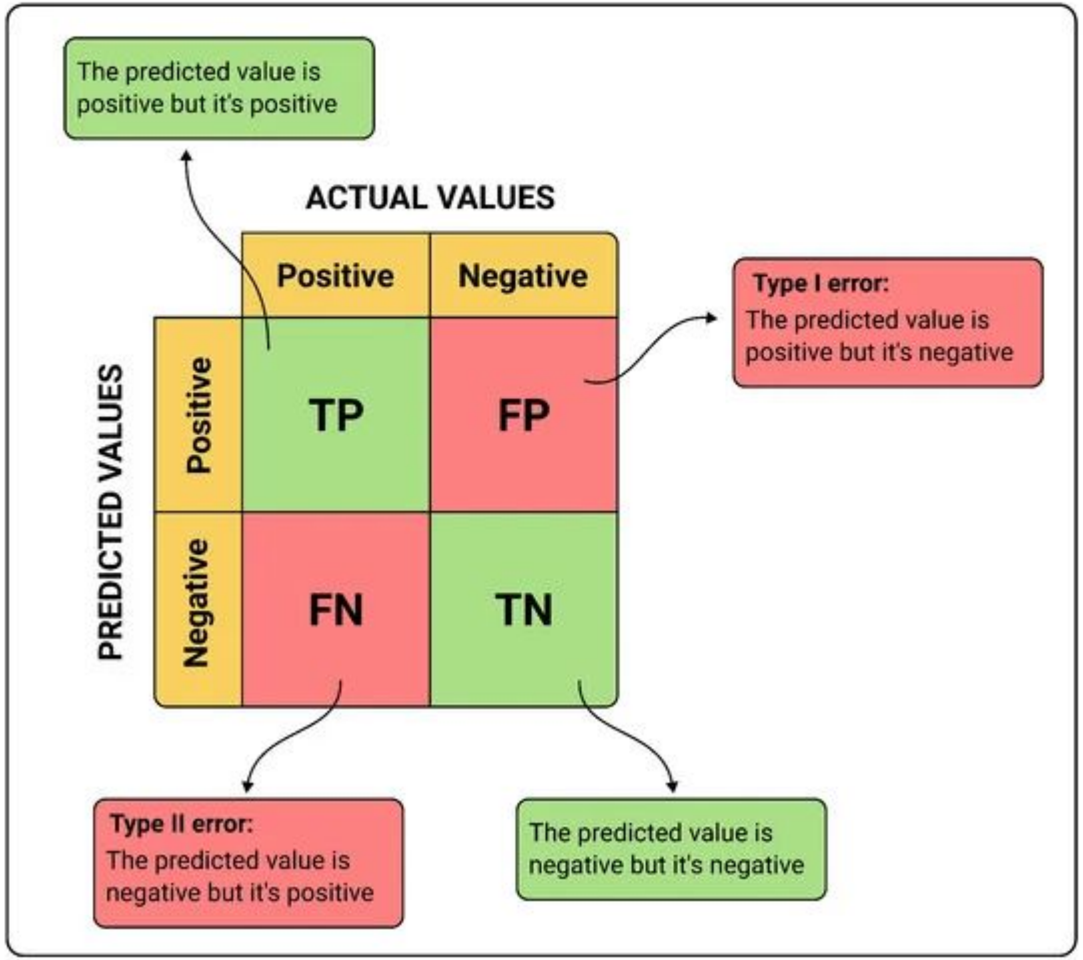
Evaluating the performance of a binary classification model is crucial to understand how well it generalizes to unseen data.

1. Confusion Matrix

A **confusion matrix** is a table that summarizes the performance of a classification model by showing the counts of true positives, true negatives, false positives, and false negatives.

Structure:

- **True Positives (TP):** The number of instances correctly predicted as the positive class (e.g., correctly identifying an Iris-Setosa).
- **True Negatives (TN):** The number of instances correctly predicted as the negative class (e.g., correctly identifying a flower as not Iris-Setosa).
- **False Positives (FP):** The number of instances incorrectly predicted as the positive class (e.g., identifying a flower as Iris-Setosa when it is not).
- **False Negatives (FN):** The number of instances incorrectly predicted as the negative class (e.g., identifying a flower as not Iris-Setosa when it is).


The predicted value is positive but it's positive
ACTUAL VALUES
Positive
Negative
PREDICTED VALUES
Positive
Negative
TP
FP
FN
TN
Type I error:
The predicted value is positive but it's negative
Type II error:
The predicted value is negative but it's positive
The predicted value is negative but it's negative

Example Scenario:

Imagine you're building a model to predict whether a patient has a particular disease (`1` for having the disease, `0` for not having the disease) based on some medical data. After testing the model, you compare its predictions with the actual outcomes for 10 patients.

Actual vs. Predicted Labels:

Let's say the actual and predicted outcomes are as follows:

Patient	Actual Label	Predicted Label
1	1	1
2	0	0
3	1	0
4	0	1
5	1	1
6	0	0
7	1	1
8	0	0
9	0	0
10	1	1

Confusion Matrix:

Using the above data, the confusion matrix is constructed as follows:

	Predicted: 0	Predicted: 1
Actual: 0	**4 (TN)**	**1 (FP)**
Actual: 1	**1 (FN)**	**4 (TP)**

2. Interpreting the Confusion Matrix:

2.1)Accuracy

Accuracy measures the proportion of correctly predicted instances (both positive and negative) out of the total instances.

Formula:

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{All predictions}}$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

For the previous example

Accuracy= (4+4)/(4+4+1+1)
= 8/10
= 0.8 (80%)

2.2) Precision

Precision measures the proportion of correctly predicted positive instances out of all instances predicted as positive. It’s particularly important **when the cost of false positives is high**.

Formula:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

For the previous example

Accuracy= 4/4+1
= 4/5
= 0.8 (80%)

2.3) Recall (Sensitivity or True Positive Rate)

Recall measures the proportion of correctly predicted positive instances out of all actual positive instances. It's crucial **when the cost of false negatives is high**.

Formula:

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

$$\text{Recall} = \frac{TP}{TP+FN}$$

For the previous example

Accuracy= 4/4+1

= 4/5

= 0.8 (80%)

2.4) F1 Score

- Harmonic mean of precision and recall.
- $\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{0.8 \times 0.8}{0.8 + 0.8} = 0.8 \text{ or } 80\%$

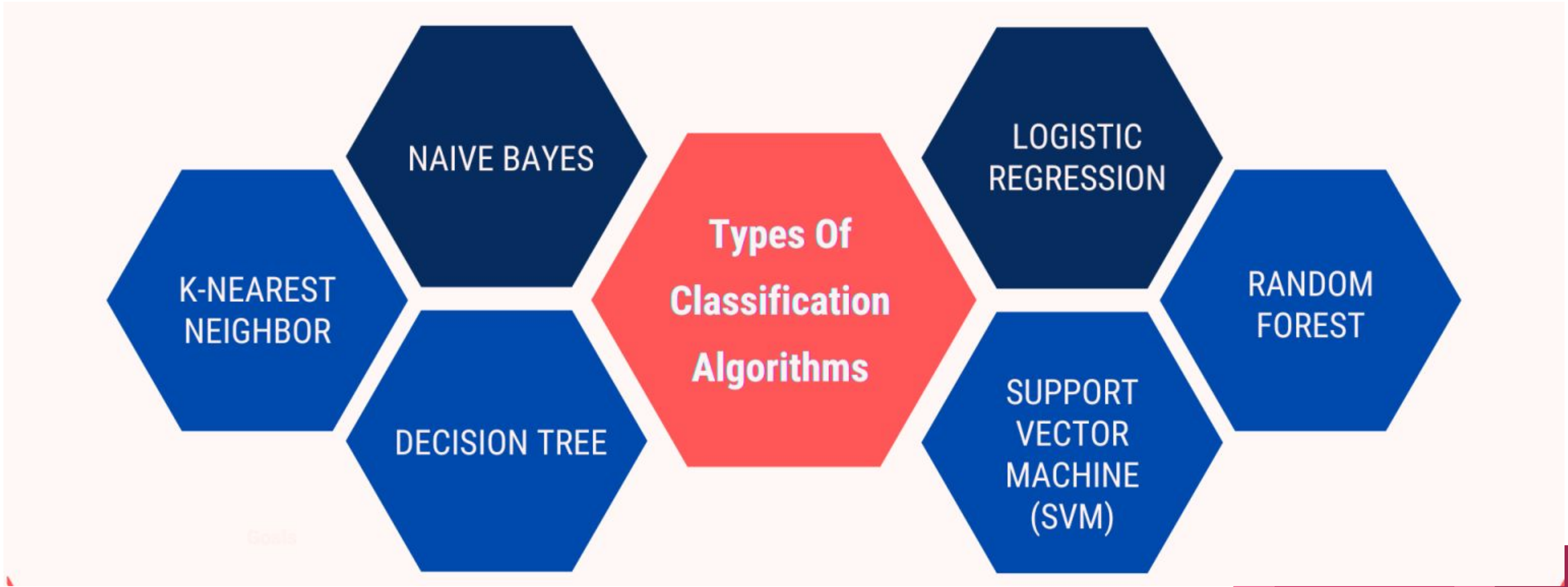
Note: What is Harmonic Mean?

The harmonic mean (HM) is a type of average, best for rates and ratios (like speed or density), calculated as the reciprocal of the arithmetic mean of the reciprocals of the data points, giving more weight to smaller values

Arithmetic Mean = $(a_1 + a_2 + a_3 + \ldots.. + a_n) / n$

Harmonic Mean = $n / [(1/a_1)+(1/a_2)+(1/a_3)+\ldots+(1/a_n)]$

Types of Classification Algorithms



Classification Predictive Modeling

There are mainly 4 different types of classification tasks that you might encounter in your day to day challenges. Generally, the different types of predictive models in machine learning are as follows :

- Binary classification
- Multi-Label Classification
- Multi-Class Classification
- Imbalanced Classification

Binary classification

A binary classification refers to those tasks which can give either of any two class labels as the output. Generally, one is considered as the normal state and the other is considered to be the abnormal state. The following examples will help you to understand them better.

- **Email Spam detection:** Normal State – Not Spam, Abnormal State – Spam
- **Conversion prediction:** Normal State – Not churned, Abnormal State – Churn

The most popular algorithms which are used for binary classification are :

- K-Nearest Neighbours
- Logistic Regression
- Support Vector Machine
- Decision Trees
- Naive Bayes

Some algorithms are specifically designed for binary classification and do not natively support more than two classes; examples include **Logistic Regression** and **Support Vector Machines**.

Multi-Class Classification

These types of classification problems have no fixed two labels but can have any number of labels. Some popular examples of multi-class classification are :

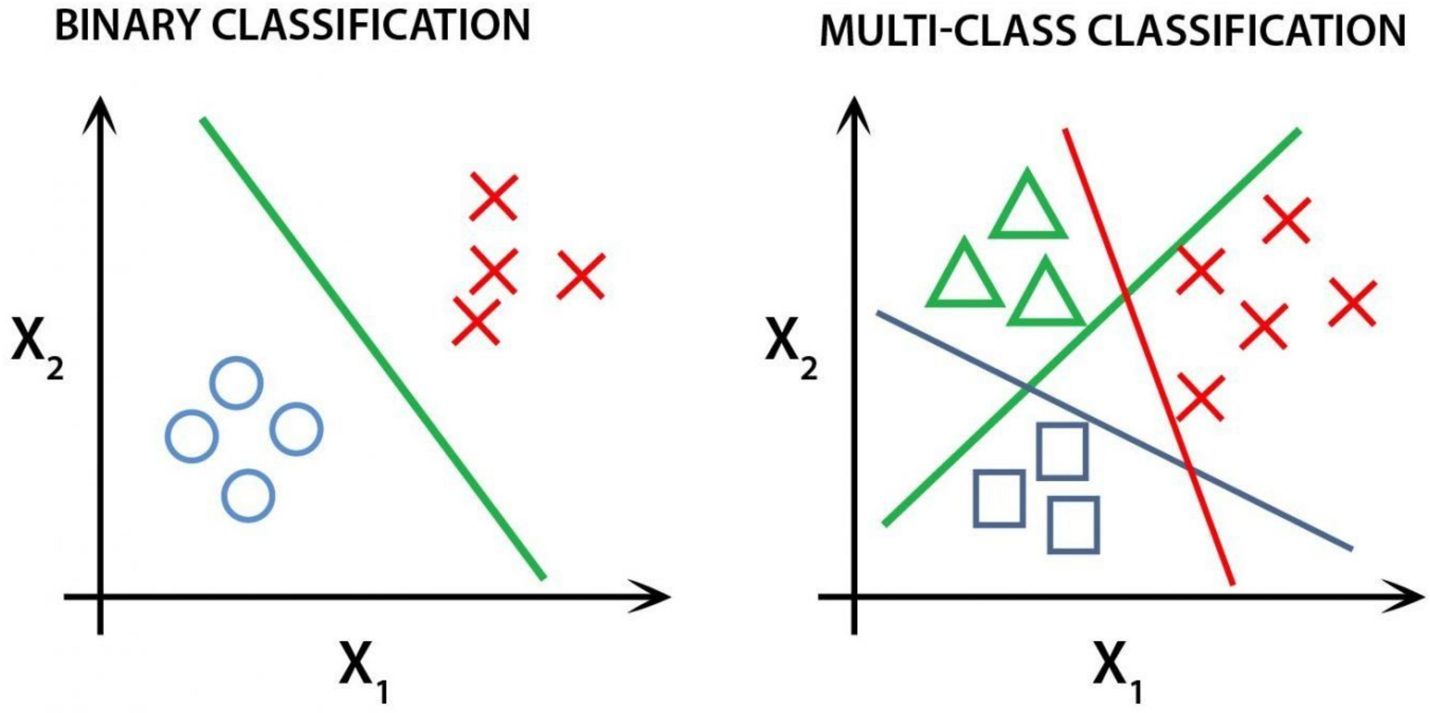
- Plant Species Classification
- Face Classification
- Optical Character recognition

Example: Classifying images of animals into categories like "cat," "dog," "rabbit," and "horse." Each image can belong to only one of these classes.

The most common algorithms which are used for Multi-Class Classification are :

- K-Nearest Neighbours
- Naive Bayes
- Decision trees
- Gradient Boosting
- Random Forest

ANALYTIXLABS



Multi-Label Classification:

Definition: In multi-label classification, each data point can belong to multiple classes simultaneously. This is different from binary and multi-class classification, where each instance is associated with only one label.

Examples:

- Image Tagging: An image might be tagged with multiple labels such as "beach," "sunset," and "vacation."
- Document Classification: A news article might be categorized under multiple topics like "politics," "economics," and "health."
- In a music recommendation system, a song might be labeled as "rock," "classic," and "instrumental" all at the same time. The model's task is to assign all appropriate labels to each song.

The common algorithms used here are :

- Multi-label Random Forests
- Multi-label Decision trees
- Multi-label Gradient Boosting

Imbalanced Classification:

Definition: Imbalanced classification refers to situations where the classes in the dataset are not represented equally.

One class might be significantly more frequent than others, which can lead to challenges in training effective models.

Example: In fraud detection, fraudulent transactions (positive class) are typically much less common than non-fraudulent ones (negative class).

Logistic Regression is a statistical method for binary classification. Despite its name, it's a linear model for classification, not regression. It predicts the probability that a given input point belongs to a particular class.

Step 1: Input Features: Collect the features (input variables) that will be used for predicting the binary outcome.

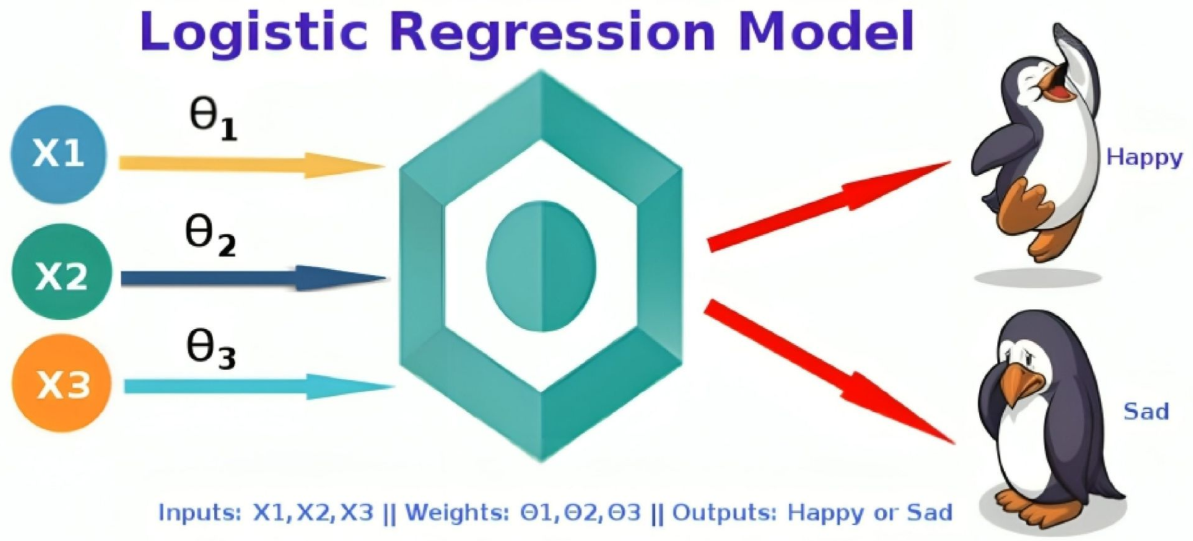
Step 2: Apply Logistic Function: Use the logistic function (sigmoid) to model the probability that a given input belongs to a certain class.

$$P(y=1) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n)}}$$

Step 3: Cost Function: Use the log-loss function to measure the difference between the predicted probabilities and the actual class labels.

Step 4: Optimization: Use techniques like gradient descent to adjust the coefficients (β0, β1, etc.) to minimize the cost function.

Step 5: Prediction: Predict the class label based on the probability; usually, a threshold of 0.5 is used (probability > 0.5 → Class 1, otherwise Class 0).


Logistic Regression Model
X1
X2
X3
θ1
θ2
θ3
Happy
Sad
Inputs: X1,X2,X3 || Weights: Θ1,Θ2,Θ3 || Outputs: Happy or Sad

Mathematical Foundation

- Logistic regression models the probability of the default class (usually denoted as 1) using a logistic function.
- The logistic function (also called the sigmoid function) is defined as:

$$P(y=1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n)}}$$

Here:

- y is the binary target variable.
- $x_1, x_2, \ldots, x_n$ are the input features.
- β_0 is the intercept (bias term).
- $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients (weights) for each feature.

Understanding the Sigmoid Function

- The sigmoid function maps any real-valued number into a value between 0 and 1, which is interpreted as a probability.
- If the output of the sigmoid function is greater than 0.5, the prediction is class 1 (positive class); otherwise, it’s class 0 (negative class).

In machine learning, the sigmoid function is a mathematical function that maps real numbers to a value between 0 and 1. It's also known as a logistic function. The sigmoid function is used as an activation function in neural networks to add non-linearity to a model.

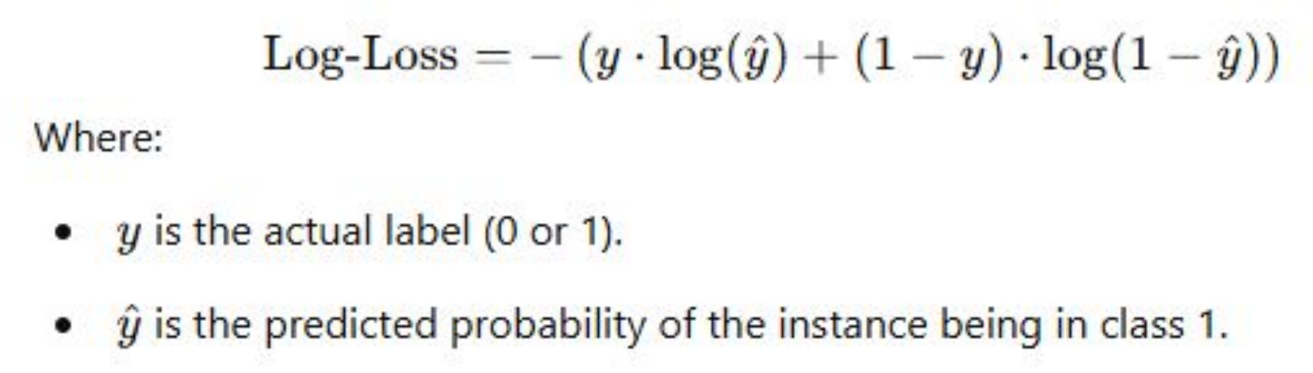
Cost Function

- Logistic regression uses a cost function called **Log-Loss** (or binary cross-entropy), which is defined as:

To Understand **Log-Loss** (or binary cross-entropy)

Log-loss measures the performance of a classification model whose output is a probability value between 0 and 1. It quantifies the difference between the predicted probabilities and the actual class labels (which are either 0 or 1).

Formula: For binary classification, the log-loss for a single instance is calculated as:



$$\text{Log-Loss} = -\left(y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})\right)$$

Where:

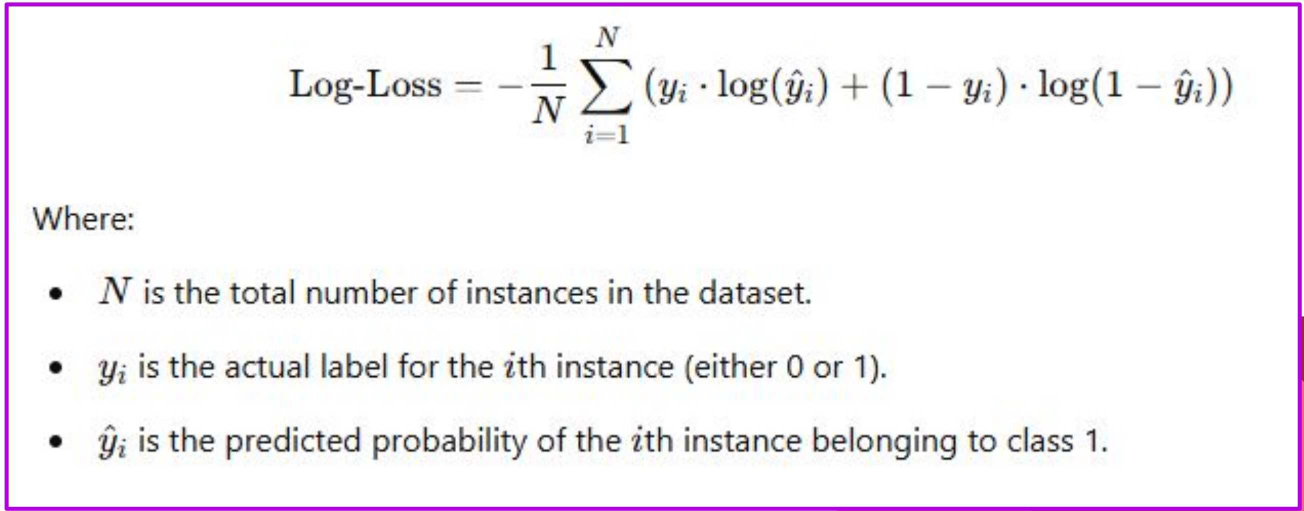
- y is the actual label (0 or 1).
- $\hat{y}$ is the predicted probability of the instance being in class 1.

The overall log-loss for a dataset is the average log-loss over all instances.

Interpretation:

- **Perfect Prediction**: If the model predicts exactly 0 for class 0 and exactly 1 for class 1, the log-loss will be 0, which is ideal.
- **Incorrect Prediction**: If the predicted probability is far from the actual label, the log-loss increases significantly.
- **Probabilistic Output**: Log-loss is sensitive to how confident the model is about its predictions. A prediction of 0.99 for class 1 that turns out to be wrong incurs a much higher log-loss than a prediction of 0.51.

For Multiple Instances

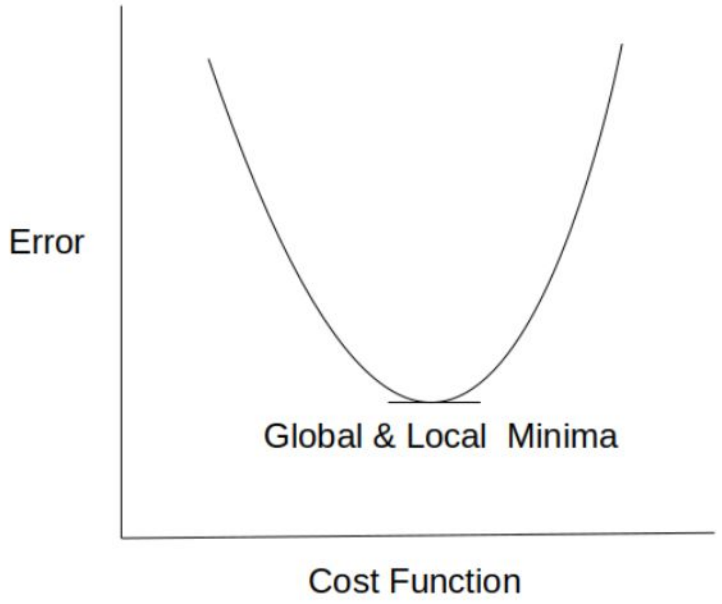


$$\text{Log-Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\right)$$

Where:

- N is the total number of instances in the dataset.
- y_i is the actual label for the ith instance (either 0 or 1).
- $\hat{y}_i$ is the predicted probability of the ith instance belonging to class 1.

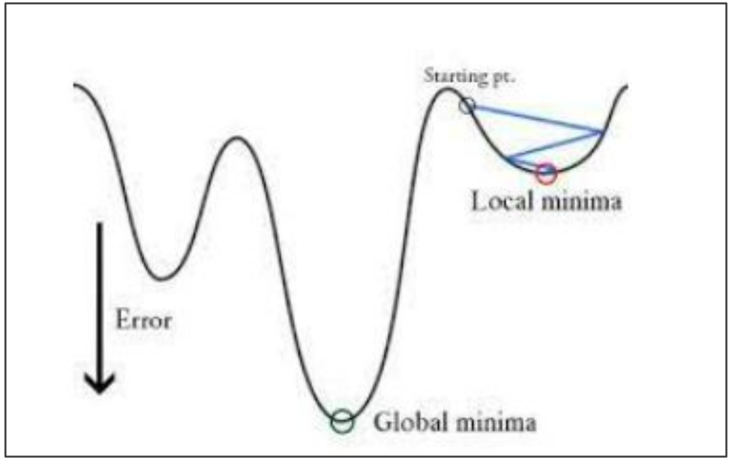
In linear regression, we use the Mean squared error which was the difference between y_predicted and y_actual and this is derived from the maximum likelihood estimator. The graph of the cost function in linear regression is like this:



Linear Regression
Cost Function

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

In logistic regression Yi is a non-linear function ($\hat{Y} = \frac{1}{1+e^{-x}}$). If we use this in the above MSE equation then it will give a non-convex graph with many local minima as snown.



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.

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^{N} -(y_i * \log(\hat{Y}_i) + (1-y_i) * \log(1-\hat{Y}_i))$$

To Understand - local minima, global minima, local maxima, and global maxima

Global Minimum:

- **Definition:** The global minimum of a function is the lowest point over the entire domain of the function. It is the point where the function value is the smallest compared to all other points in the domain.

Example: In the function $f(x) = x^2$, the global minimum is at $x = 0$, where $f(x) = 0$.

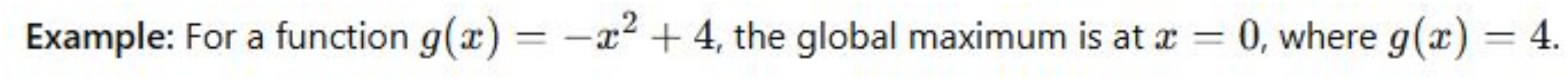
Local Minimum:

- **Definition:** A local minimum is a point where the function value is lower than at any nearby points. However, it may not be the lowest point over the entire domain.

Example: In a more complex function like $f(x) = x^4 - 3x^2 + 2$, there could be multiple local minima, but only one global minimum.

Global Maximum:

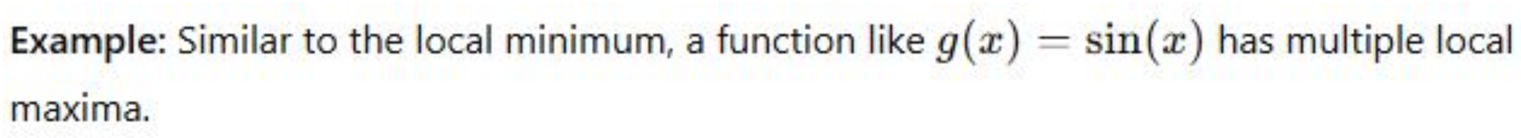
- **Definition:** The global maximum is the highest point over the entire domain of the function. It is where the function value is the greatest compared to all other points in the domain.



Example: For a function $g(x) = -x^2 + 4$, the global maximum is at $x = 0$, where $g(x) = 4$.

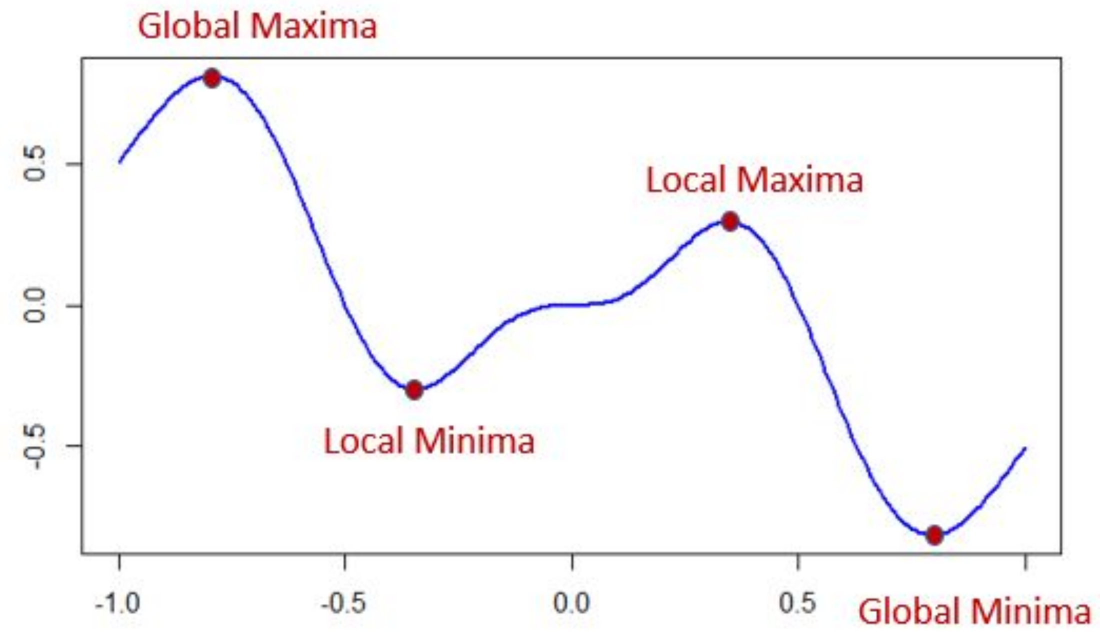
Local Maximum:

- **Definition:** A local maximum is a point where the function value is higher than at any nearby points. However, it may not be the highest point over the entire domain.



Example: Similar to the local minimum, a function like $g(x) = \sin(x)$ has multiple local maxima.

Visual Representation:



Optimization

- **Gradient Descent** is commonly used to minimize the cost function. This involves iteratively adjusting the parameters in the opposite direction of the gradient of the cost function with respect to the parameters.
- The update rule for each parameter is

$$\beta_j := \beta_j - \alpha \frac{\partial J(\beta)}{\partial \beta_j}$$

where α is the learning rate.

Prediction

- After training, for a new input x, the model computes:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_1+\ldots+\beta_n x_n)}}$$

- If this probability is greater than 0.5, the predicted class is 1; otherwise, it's 0.

To Understand - Gradient Descent (Unit-3)

- Gradient Descent is a fundamental optimization algorithm used to minimize the loss function in various machine learning and deep learning models.

- The core idea is to iteratively adjust model parameters in the direction that reduces the loss, which represents how well the model performs.

Support Vector Machines (SVM)

- Support Vector Machines (SVM) is a powerful supervised machine learning algorithm used for classification and regression tasks.
- SVM can be used for both regression and classification tasks, but generally, they work best in classification problems.
- Its primary goal is to find a hyperplane in an N-dimensional space (N is the number of features) that distinctly classifies the data points.

1. Introduction to SVM

- **Hyperplane:** A decision boundary that separates different classes in the data. In a 2D space, this is a line; in a 3D space, it's a plane, and in higher dimensions, it's a hyperplane.
- **Support Vectors:** These are the data points that are closest to the hyperplane. The position of these points is crucial in defining the hyperplane.
- **Margin:** The distance between the hyperplane and the nearest support vectors. SVM aims to maximize this margin. A larger margin is associated with better generalization ability. In SVM large margin is considered a good margin. There are two types of margins hard margin and soft margin. I will talk more about these two in the later section.

Objective:

- SVM finds the optimal hyperplane that maximizes the margin between two classes (for binary classification).

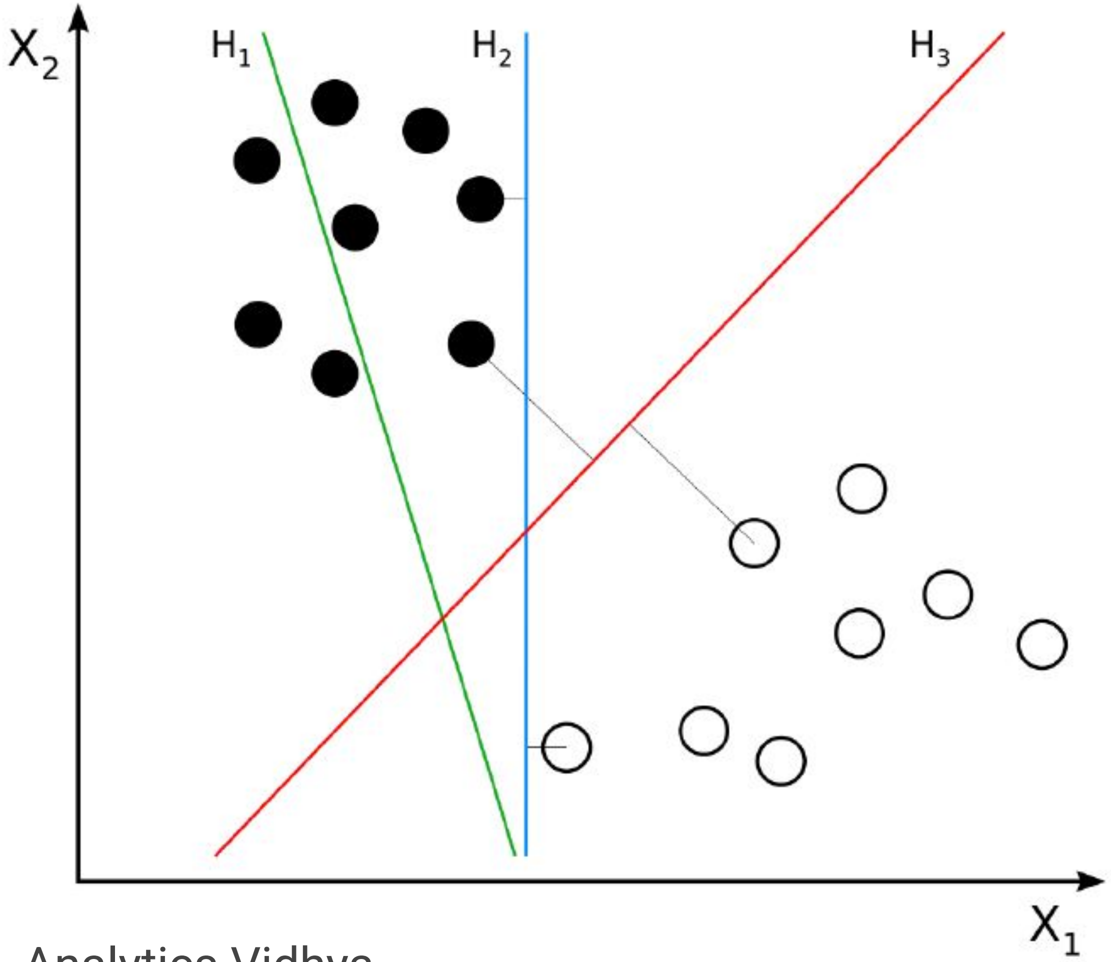
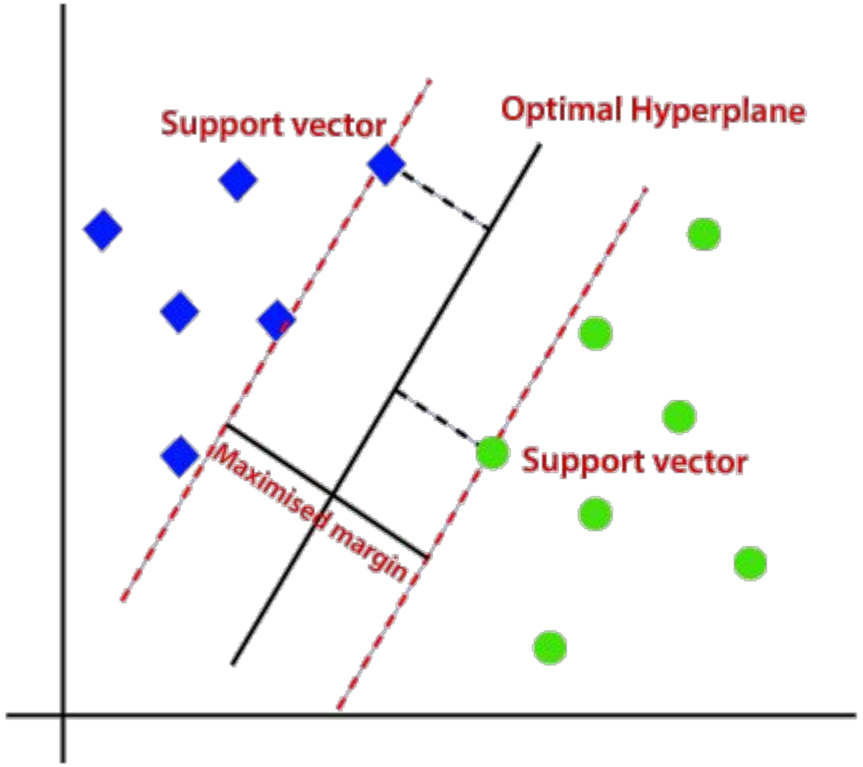
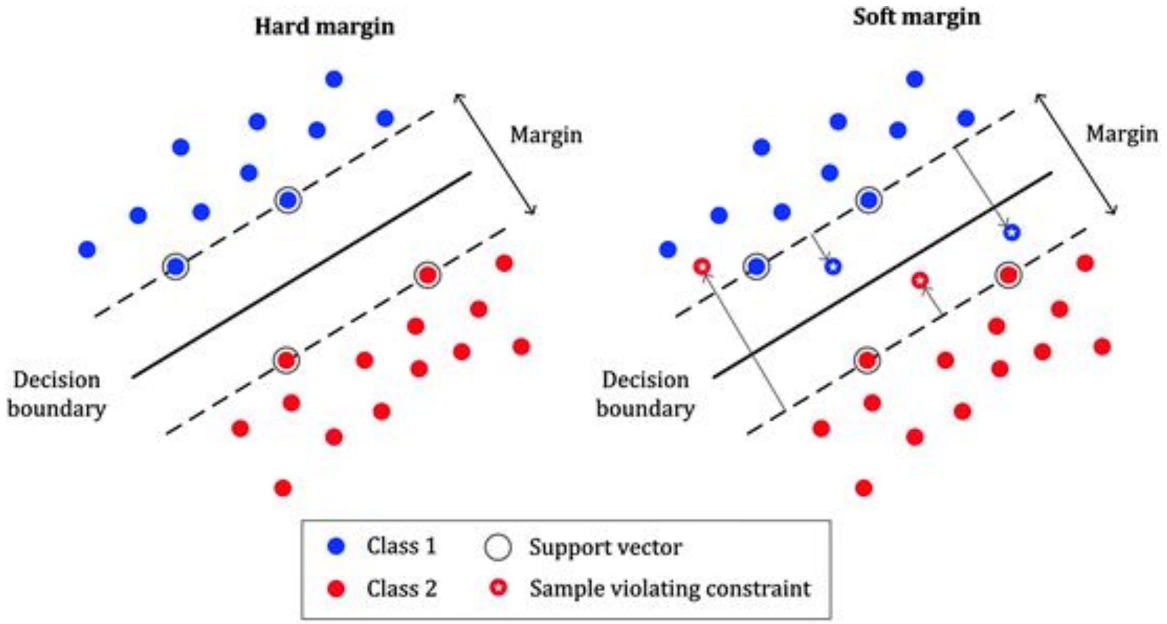


Image Source- Analytics Vidhya


Support vector
Optimal Hyperplane
Support vector
Maximised margin

Hard Margin – If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible.

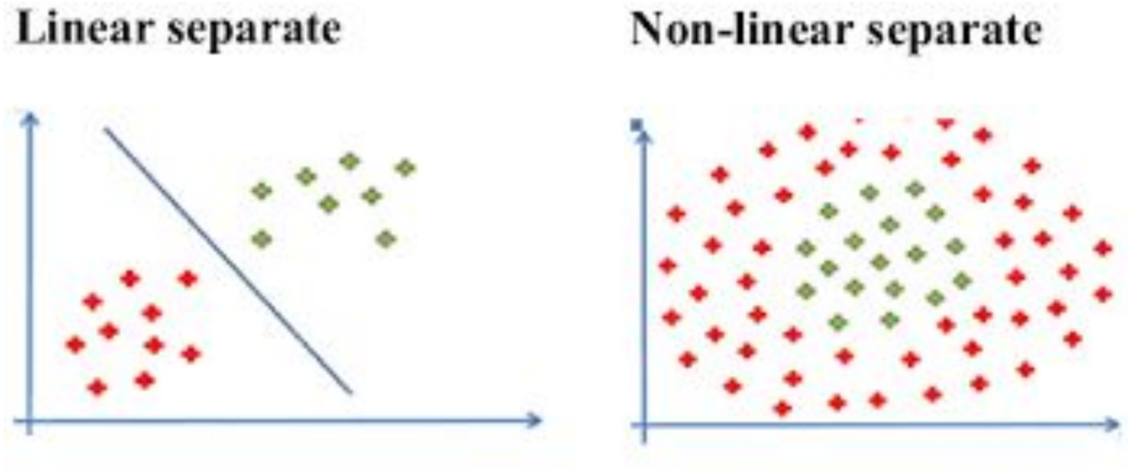


Soft Margin

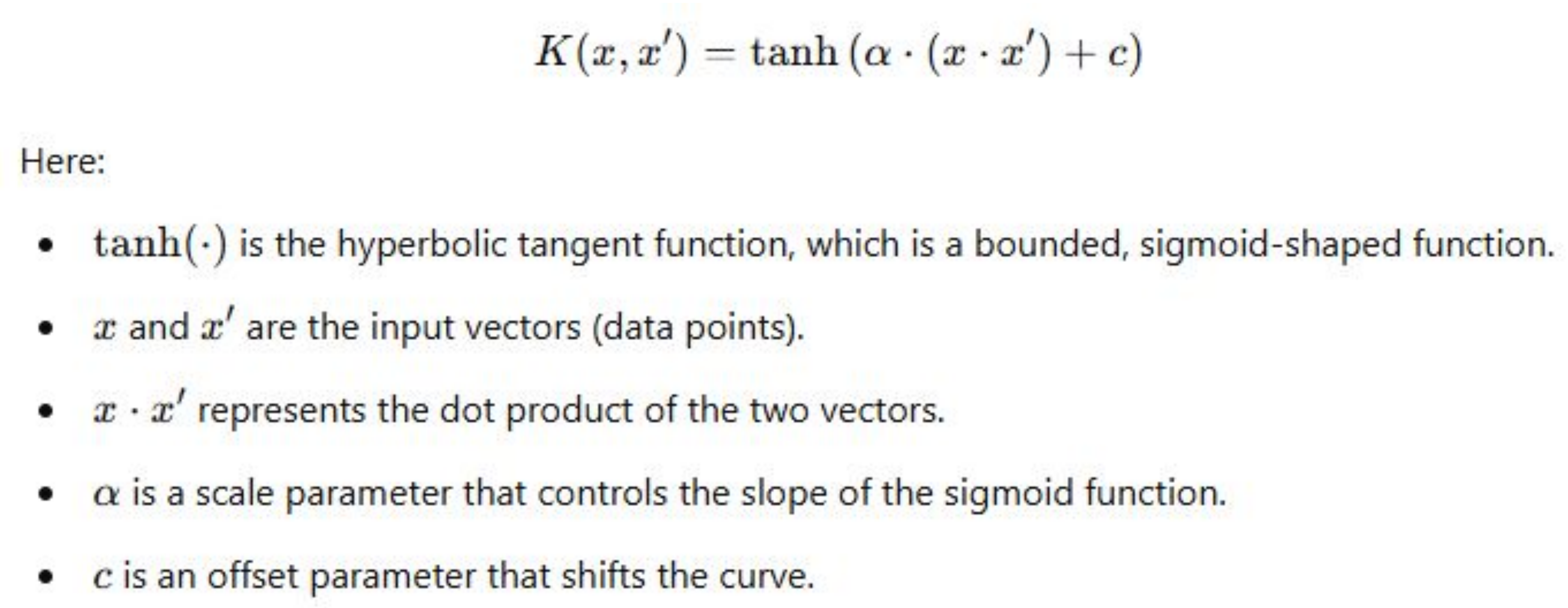
- As most of the real-world data are not fully linearly separable, we will allow some margin violation to occur which is called soft margin classification.
- It is better to have a large margin, even though some constraints are violated.
- Margin violation means choosing a hyperplane, which can allow some data points to stay on either the incorrect side of the hyperplane and between the margin and correct side of the hyperplane.

Types of SVM:

- Linear SVM: Used when the data is linearly separable.
- Non-Linear SVM: Used when the data is not linearly separable. It uses kernel functions to project data into higher dimensions where it becomes linearly separable.



The sigmoid kernel, also known as the hyperbolic tangent kernel

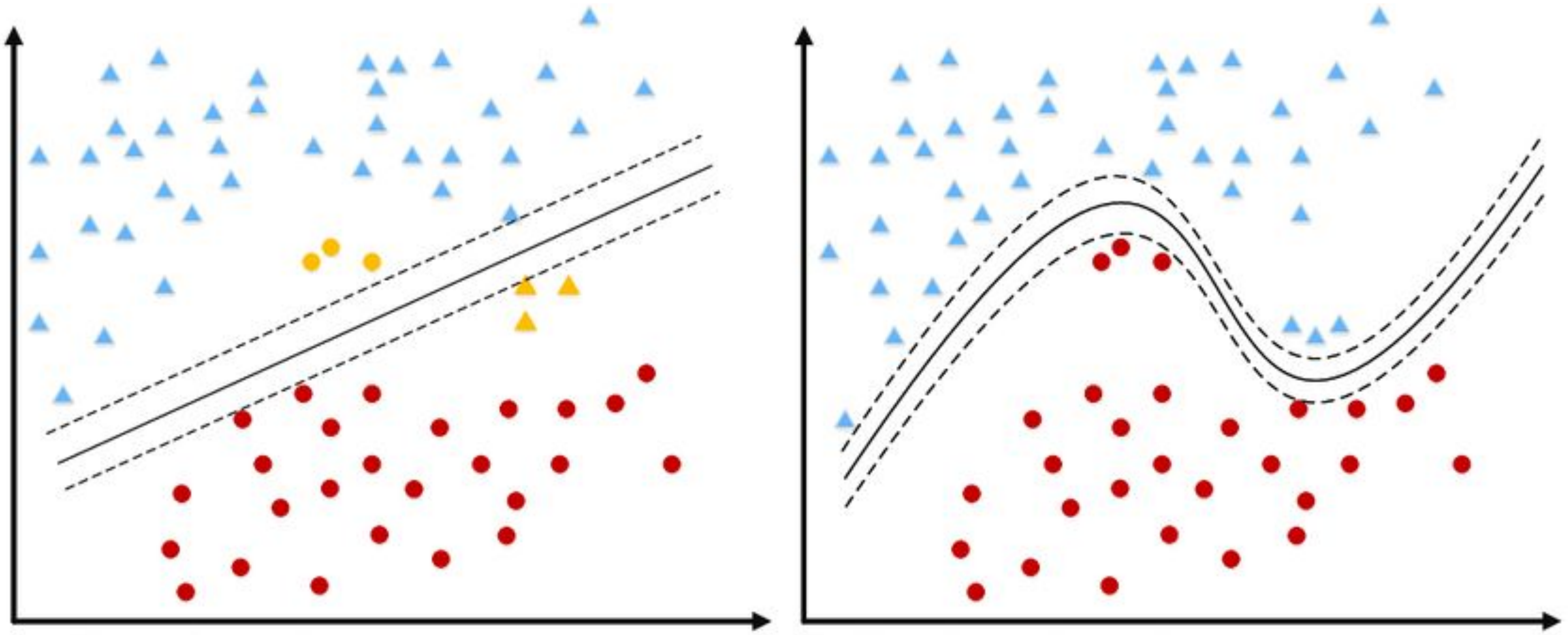


$$K(x, x') = \tanh\left(\alpha \cdot (x \cdot x') + c\right)$$

Here:

- $\tanh(\cdot)$ is the hyperbolic tangent function, which is a bounded, sigmoid-shaped function.
- x and x' are the input vectors (data points).
- $x \cdot x'$ represents the dot product of the two vectors.
- α is a scale parameter that controls the slope of the sigmoid function.
- c is an offset parameter that shifts the curve.

https://colab.research.google.com/drive/1MNSmf7KytQ8Qj5l9RBxxcMhvVUiyM3GZ?usp=drive_link


A) Linear Separation B) Non-linear Separation

Kernel Trick:

SVM has a technique called the kernel trick. These are functions that take low dimensional input space and transform it into a higher-dimensional space i.e. it converts not separable problem to separable problem.

This is done using a kernel function, which computes the dot product of the data points in this higher-dimensional space without actually having to compute the coordinates of the points in that space.

There are some famous and most frequently used Non-linear kernels in SVM are,

- 1. Linear Kernel
- 2. Polynomial SVM Kernel
- 3. Gaussian Radial Basis Function (RBF)
- 4. Sigmoid Kernel

Understanding the Kernel Trick

- Imagine you have a dataset that is not linearly separable in 2D space. A linear SVM would struggle to find a decision boundary that separates the classes.
- The kernel trick enables us to implicitly transform this data into a higher-dimensional space where a linear decision boundary can be found.

Linear Kernel: No transformation is applied. The kernel is simply the dot product of two vectors.

$$K(x, x') = x \cdot x'$$

Polynomial Kernel: Transforms the data into a higher degree polynomial space.

$$K(x, x') = (x \cdot x' + c)^d$$

d **is the degree of the polynomial.**
c **is a constant that can be adjusted to control the influence of higher-order terms.**

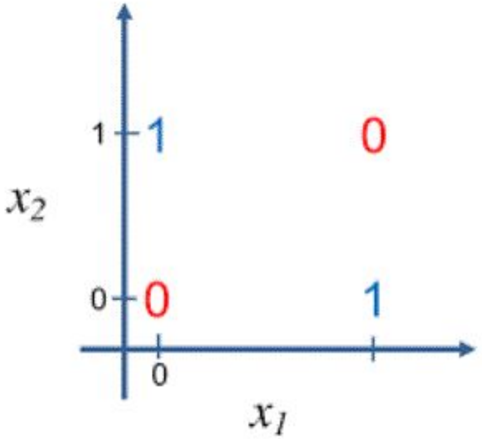
Radial Basis Function (RBF) Kernel: Maps data into an infinite-dimensional space.

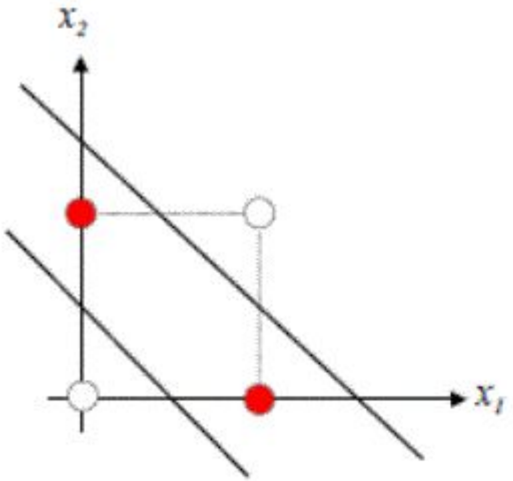
$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

∥ x−x′ ∥ is the Euclidean distance between two data points x and x′.
σ\sigma controls how quickly the kernel function value decreases as the distance between the points increases.

Example: XOR Problem with Polynomial Kernel
Consider a simple XOR problem:

x1	x2	Output
0	0	0
0	1	1
1	0	1
1	1	0



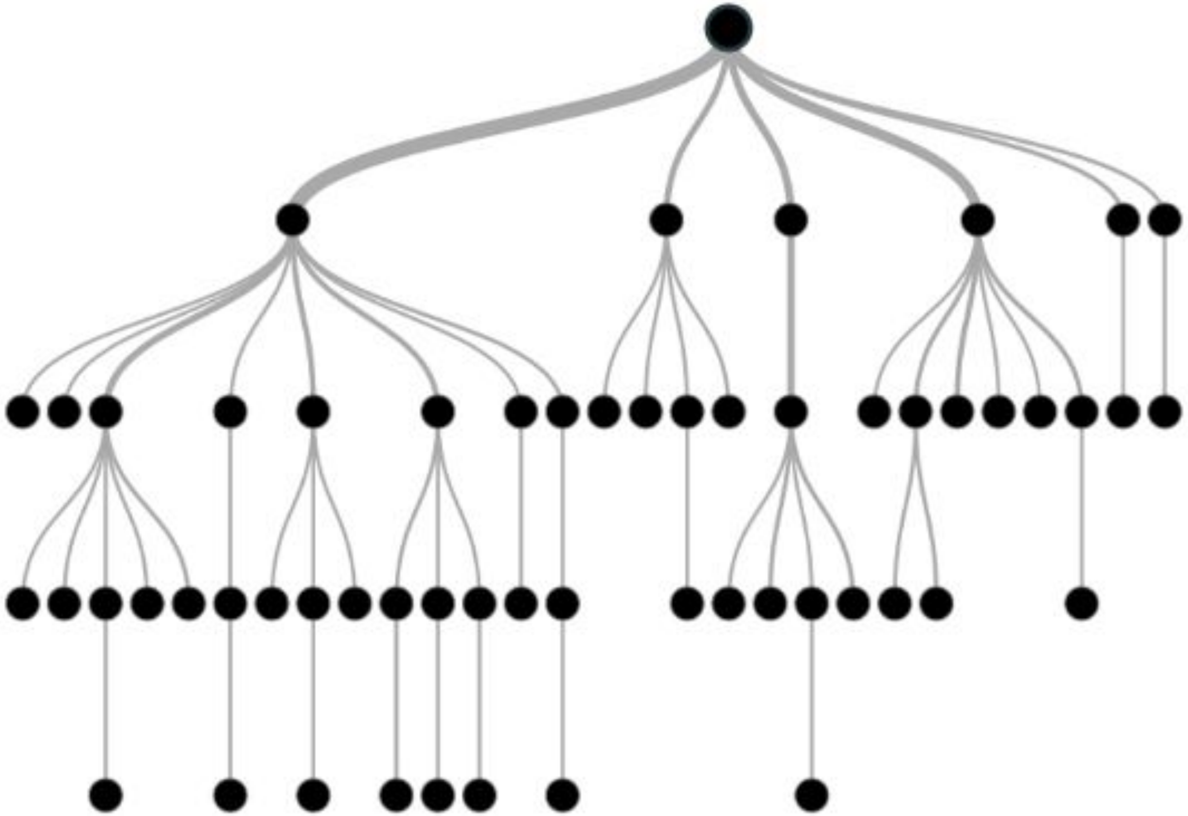


https://colab.research.google.com/drive/1MNSmf7KytQ8Qj5l9RBxxcMhvVUiyM3GZ?usp=drive_link

What is a Decision Tree?

A decision tree, which has a hierarchical structure made up of root, branches, internal, and leaf nodes, is a non-parametric supervised learning approach used for classification and regression applications.

- It is a tool that has applications spanning several different areas. Decision trees can be used for classification as well as regression problems.
- The name itself suggests that it uses a flowchart like a tree structure to show the predictions that result from a series of feature-based splits.
- It starts with a root node and ends with a decision made by leaves.



Types of Decision Tree

ID3 : This algorithm measures how mixed up the data is at a node using something called entropy. It then chooses the feature that helps to clarify the data the most.

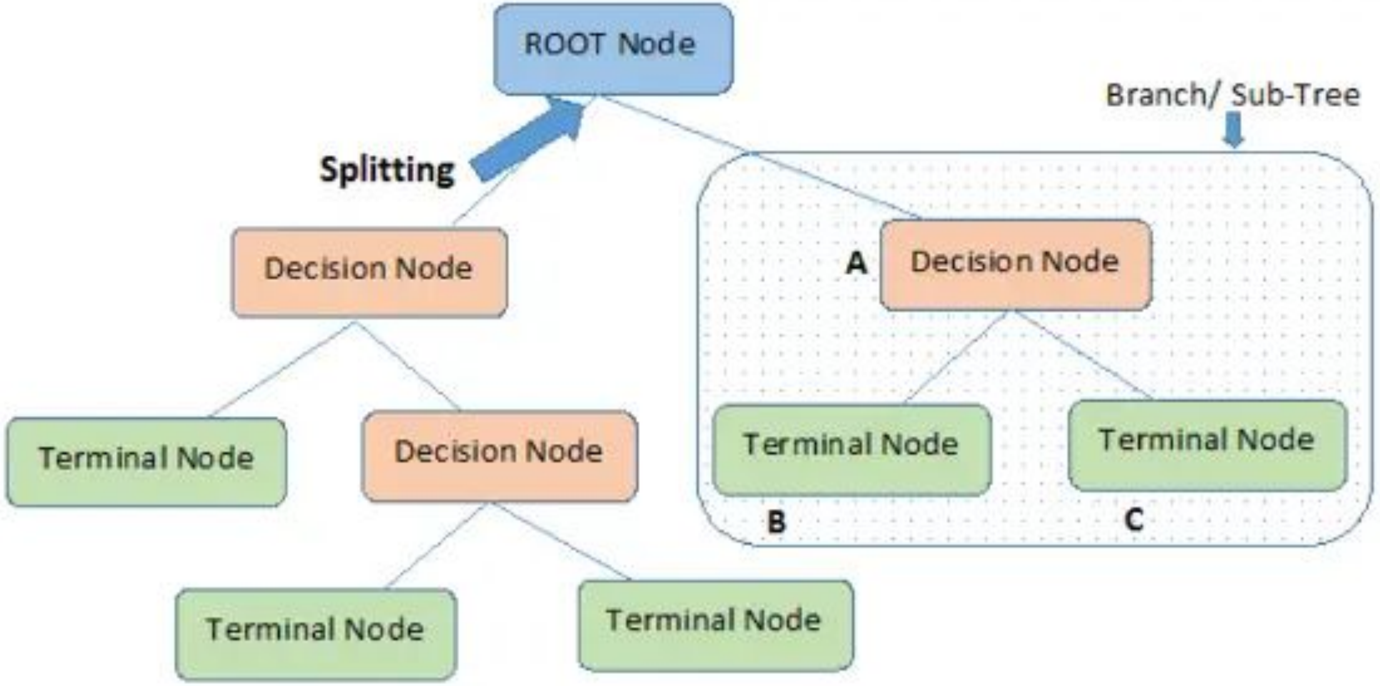
C4.5 : This is an improved version of ID3 that can handle missing data and continuous attributes.

CART : This algorithm uses a different measure called Gini impurity to decide how to split the data. It can be used for both classification (sorting data into categories) and regression (predicting continuous values) tasks.

Decision Tree Terminologies

- ***Root Node:*** The initial node at the beginning of a decision tree, where the entire population or dataset starts dividing based on various features or conditions.
- ***Decision Nodes:*** Nodes resulting from the splitting of root nodes are known as decision nodes. These nodes represent intermediate decisions or conditions within the tree.
- ***Leaf Nodes:*** Nodes where further splitting is not possible, often indicating the final classification or outcome. Leaf nodes are also referred to as terminal nodes.
- ***Parent and Child Node:*** In a decision tree, a node that is divided into sub-nodes is known as a parent node, and the sub-nodes emerging from it are referred to as child nodes. The parent node represents a decision or condition, while the child nodes represent the potential outcomes or further decisions based on that condition.

- ***Sub-Tree:*** Similar to a subsection of a graph being called a sub-graph, a subsection of a decision tree is referred to as a sub-tree. It represents a specific portion of the decision tree
- ***Pruning*****:** The process of removing or cutting down specific nodes in a decision tree to prevent overfitting and simplify the model.
- ***Branch / Sub-Tree:*** A subsection of the entire decision tree is referred to as a branch or sub-tree. It represents a specific path of decisions and outcomes within the tree.


ROOT Node
Splitting
Branch/ Sub-Tree
Decision Node
A
Decision Node
Terminal Node
Decision Node
Terminal Node
Terminal Node
B
C
Terminal Node
Terminal Node

$\text{Info}(D) = -\sum_{i=1}^{m} pi \log_2 pi$ $\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$

Where:

- Info(D) is the average amount of information needed to identify the class label of a tuple in D.
- InfoA(D) is the expected information required to classify a tuple from D based on the partitioning by A.

The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N().

How decision tree algorithms work?

Decision Tree algorithm works in simpler steps

- **Starting at the Root:** The algorithm begins at the top, called the “root node,” representing the entire dataset.
- **Asking the Best Questions:** It looks for the most important feature or question that splits the data into the most distinct groups. This is like asking a question at a fork in the tree.
- **Branching Out:** Based on the answer to that question, it divides the data into smaller subsets, creating new branches. Each branch represents a possible route through the tree.
- **Repeating the Process:** The algorithm continues asking questions and splitting the data at each branch until it reaches the final “leaf nodes,” representing the predicted outcomes or classifications.

Attribute Selection Measures

If the dataset consists of **N** attributes then deciding which attribute to place at the root or at different levels of the tree as internal nodes is a complicated step. By just randomly selecting any node to be the root can't solve the issue. If we follow a random approach, it may give us bad results with low accuracy.

For solving this attribute selection problem, researchers worked and devised some solutions. They suggested using some *criteria* like :

Entropy,

Information gain,

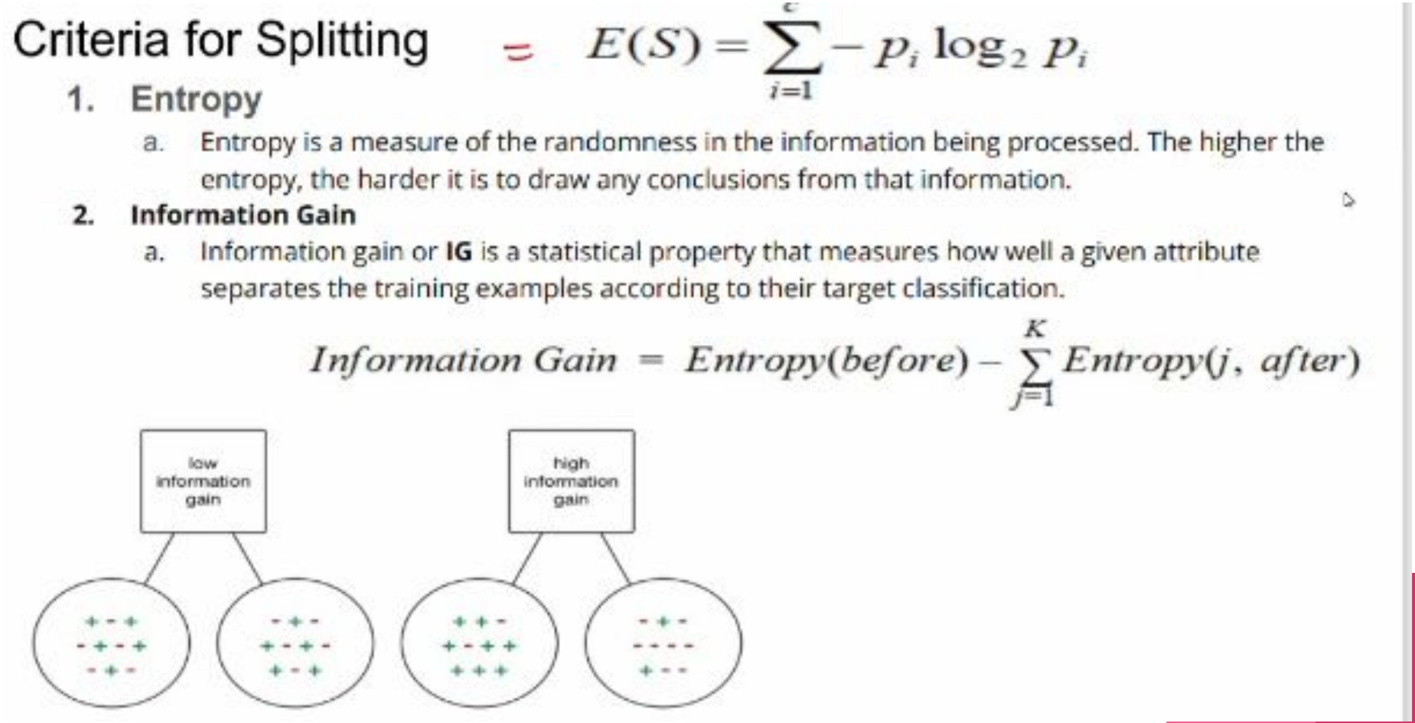
Gini index,

Gain Ratio,

Reduction in Variance

Entropy

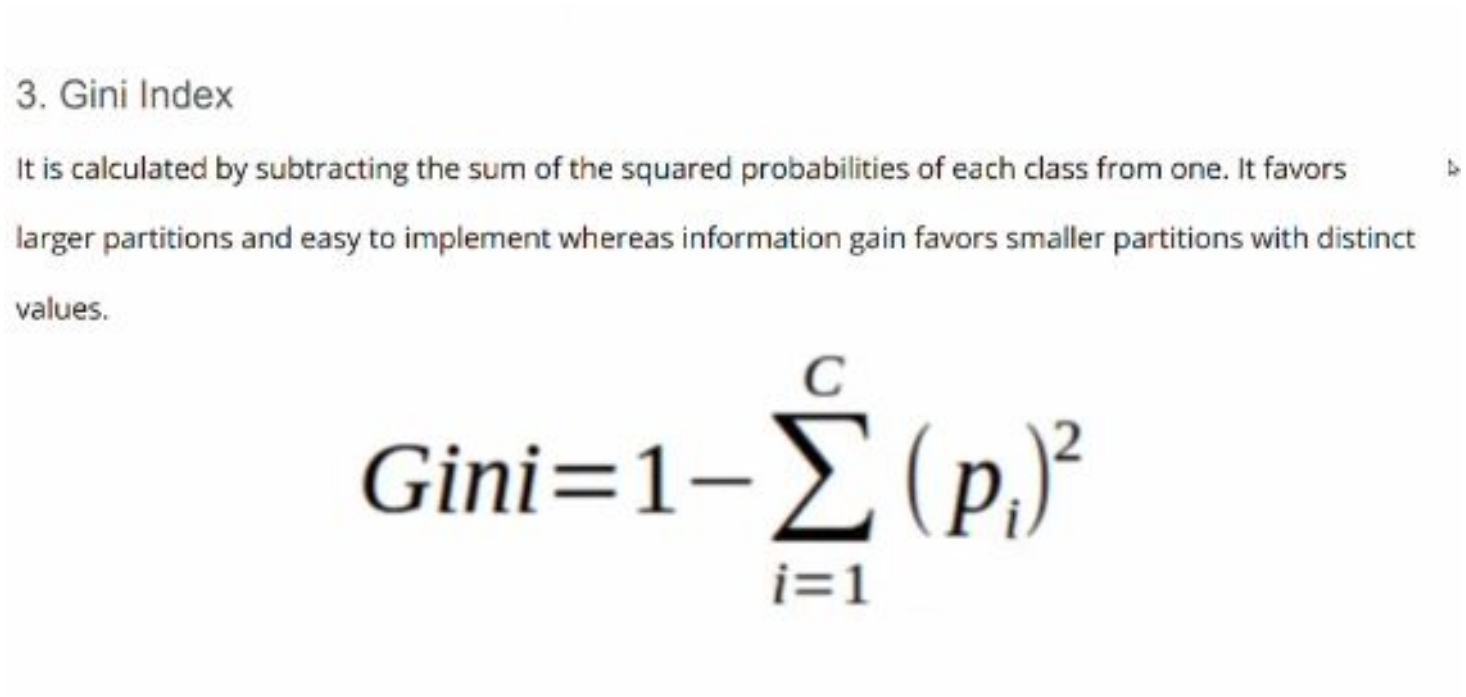
Entropy is nothing but the uncertainty in our dataset or measure of disorder.



Criteria for Splitting = $E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$

1. Entropy
   a. Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information.
2. **Information Gain**
   a. Information gain or **IG** is a statistical property that measures how well a given attribute separates the training examples according to their target classification.

$$Information\ Gain = Entropy(before) - \sum_{j=1}^{K} Entropy(j,\ after)$$

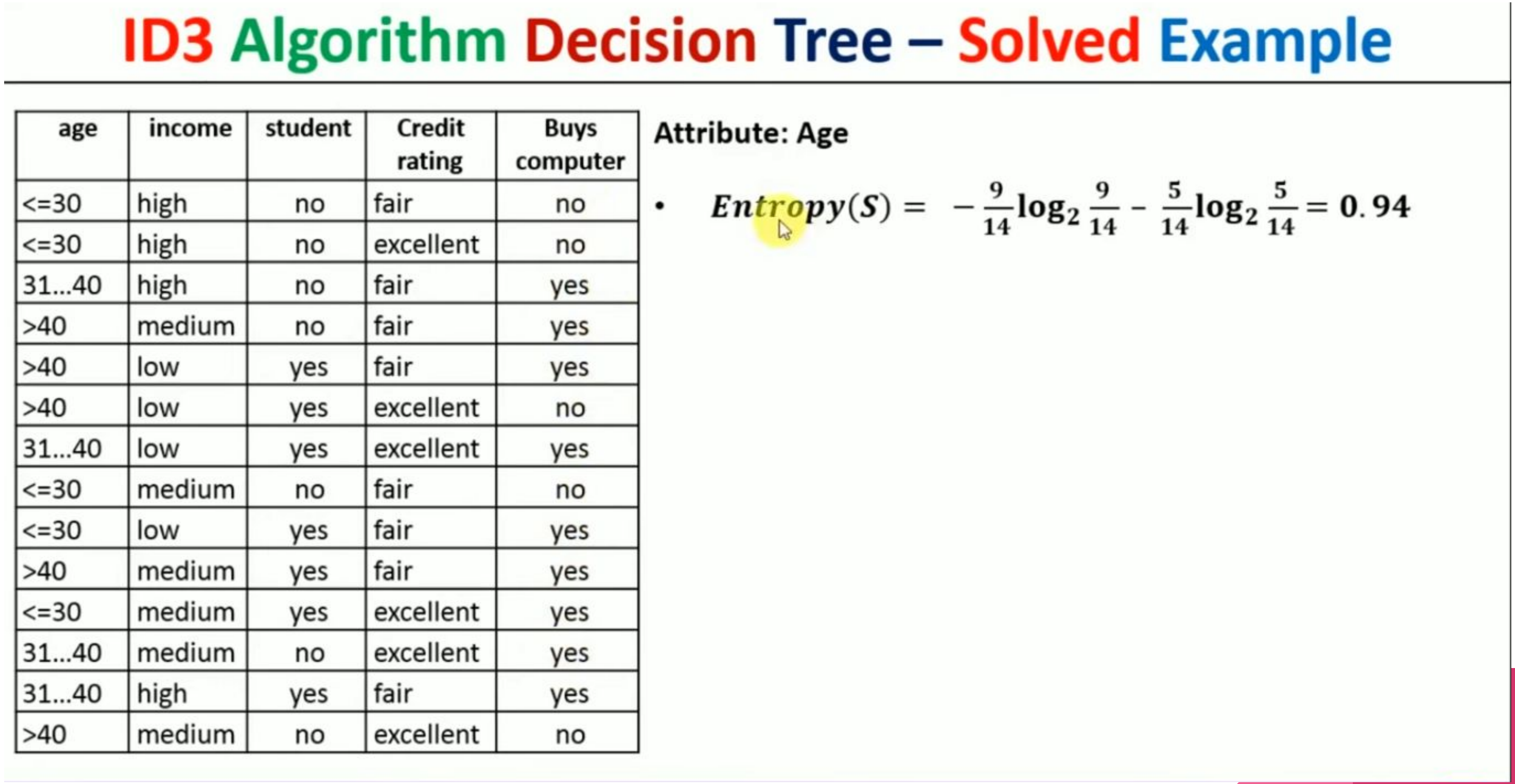


3. Gini Index

It is calculated by subtracting the sum of the squared probabilities of each class from one. It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Let's understand decision trees with the help of an example:



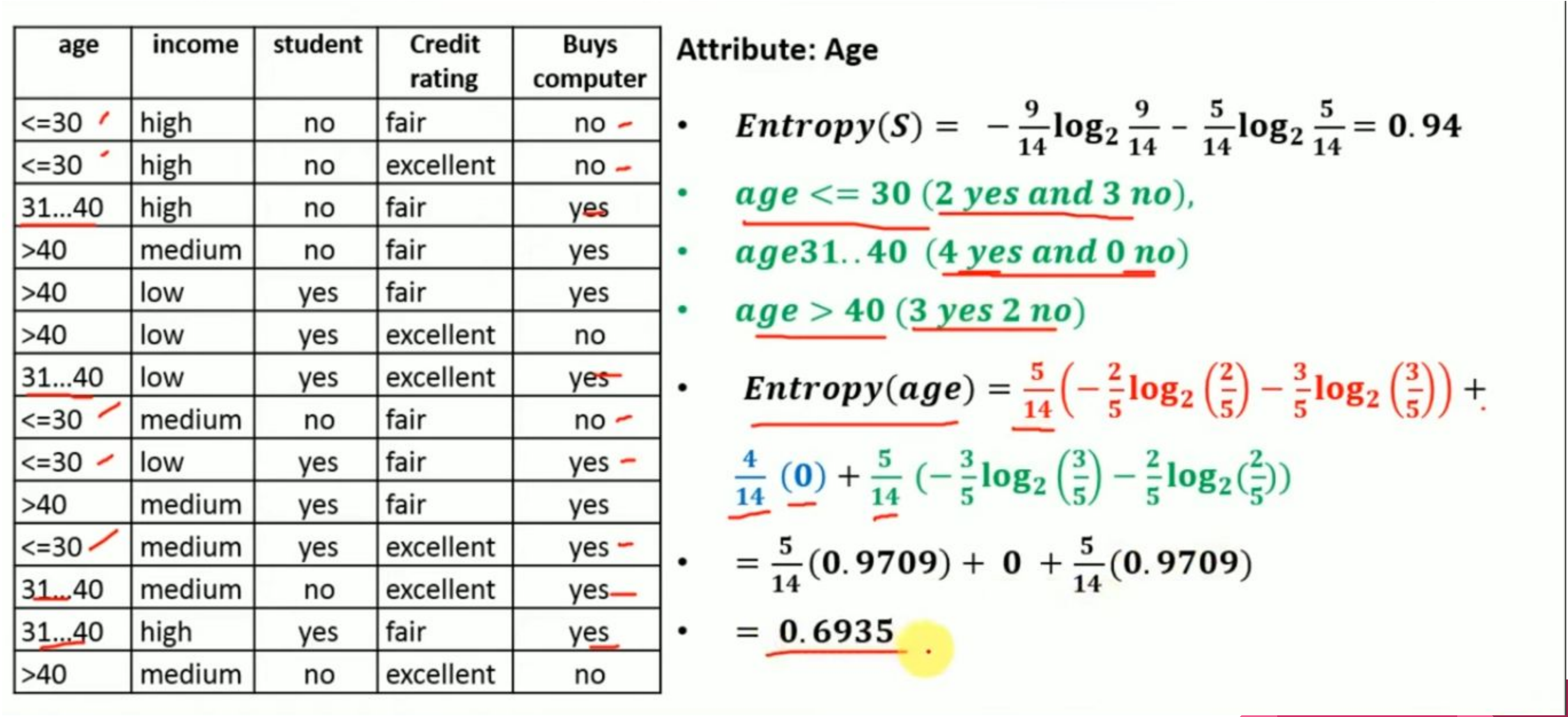
ID3 Algorithm Decision Tree – Solved Example

age	income	student	Credit rating	Buys computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Attribute: Age

- $Entropy(S) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.94$

Source from VTUPulse A Computer Science Portal



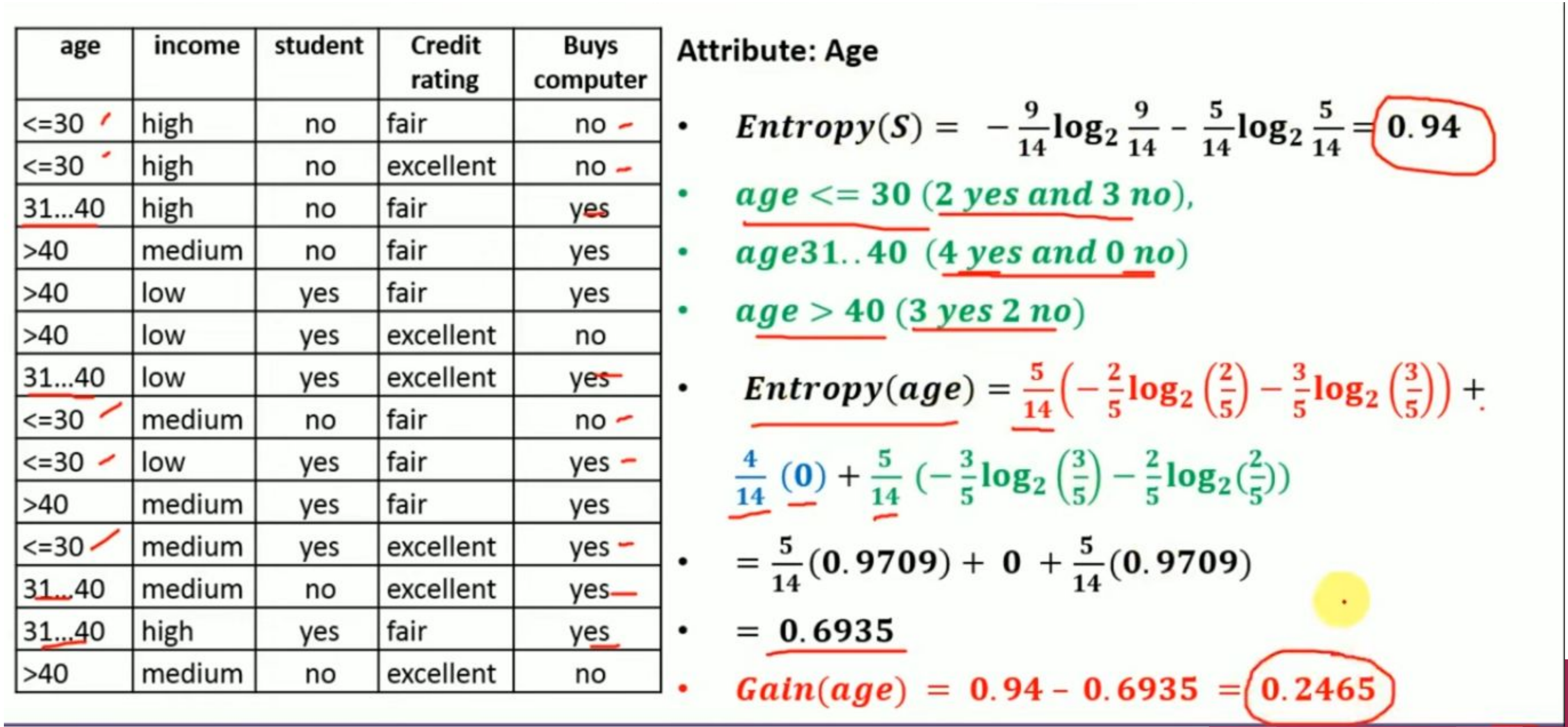
age	income	student	Credit rating	Buys computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Attribute: Age

- $Entropy(S) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.94$
- $age <= 30\ (2\ yes\ and\ 3\ no),$
- $age31..40\ \ (4\ yes\ and\ 0\ no)$
- $age > 40\ (3\ yes\ 2\ no)$
- $Entropy(age) = \frac{5}{14}\left(-\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right) +$

$\frac{4}{14}(0) + \frac{5}{14}\left(-\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right)\right)$

- $= \frac{5}{14}(0.9709) + 0 + \frac{5}{14}(0.9709)$
- $= 0.6935$

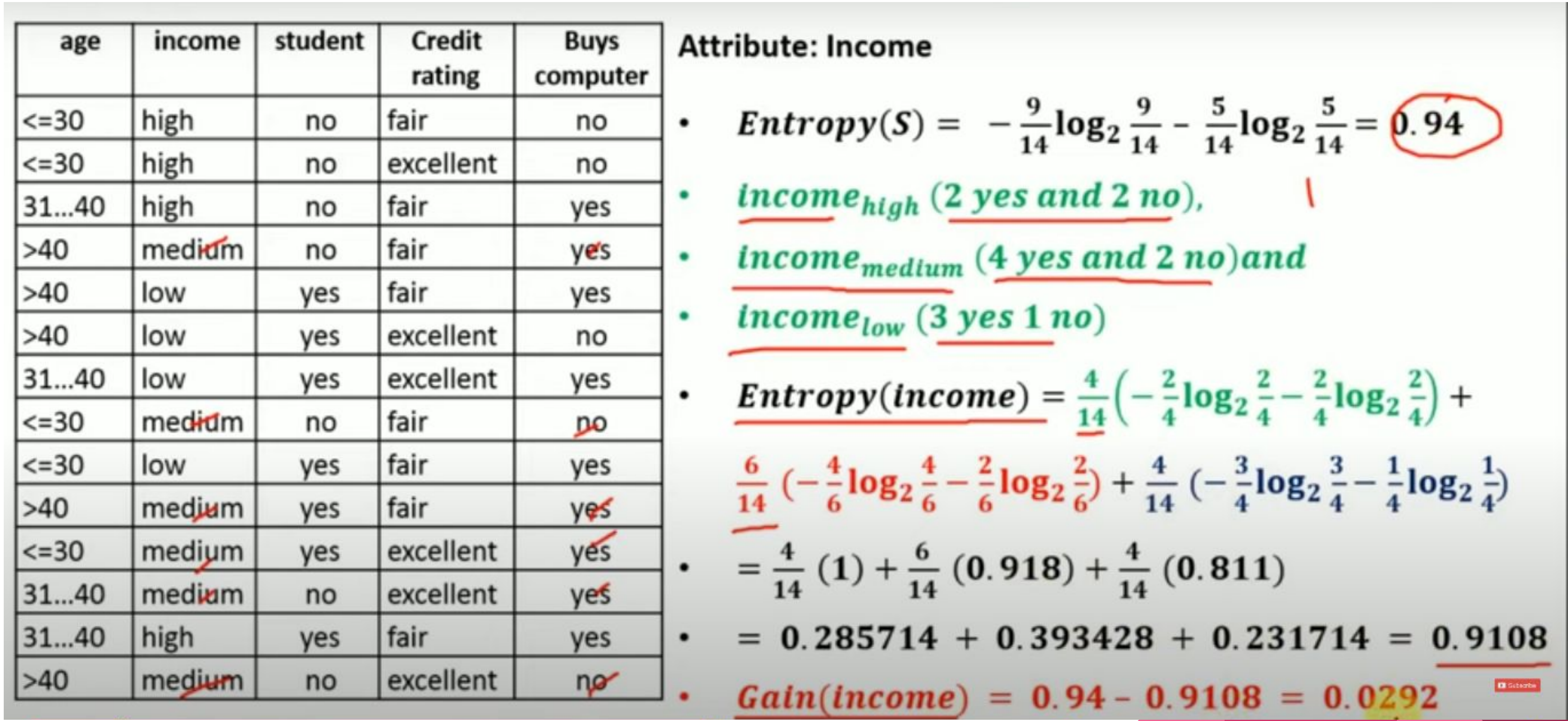


age	income	student	Credit rating	Buys computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Attribute: Age

- $Entropy(S) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = \boxed{0.94}$
- $\underline{age <= 30}\ (\underline{2\ yes\ and\ 3\ no}),$
- $age31..40\ (\underline{4\ yes\ and\ 0\ no})$
- $\underline{age > 40}\ (\underline{3\ yes\ 2\ no})$
- $\underline{Entropy(age)} = \underline{\frac{5}{14}}\left(-\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right) +$

  $\underline{\frac{4}{14}}\ \underline{(0)} + \underline{\frac{5}{14}}\left(-\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right)\right)$
- $= \frac{5}{14}(0.9709) + 0 + \frac{5}{14}(0.9709)$
- $= \underline{0.6935}$
- $Gain(age) = 0.94 - 0.6935 = \boxed{0.2465}$



age	income	student	Credit rating	Buys computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Attribute: Income

- $Entropy(S) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.94$
- $\underline{income_{high}}$ $\underline{(2\ yes\ and\ 2\ no)},$
- $\underline{income_{medium}}$ $(\underline{4\ yes\ and\ 2\ no})and$
- $\underline{income_{low}}$ $(\underline{3\ yes\ 1\ no})$
- $\underline{Entropy(income)} = \frac{4}{14}\left(-\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4}\right) +$
  $\frac{6}{14}\left(-\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6}\right) + \frac{4}{14}\left(-\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4}\right)$
- $= \frac{4}{14}(1) + \frac{6}{14}(0.918) + \frac{4}{14}(0.811)$
- $= 0.285714 + 0.393428 + 0.231714 = \underline{0.9108}$
- $\underline{Gain(income)} = 0.94 - 0.9108 = 0.0292$

age	income	student	Credit rating	Buys computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

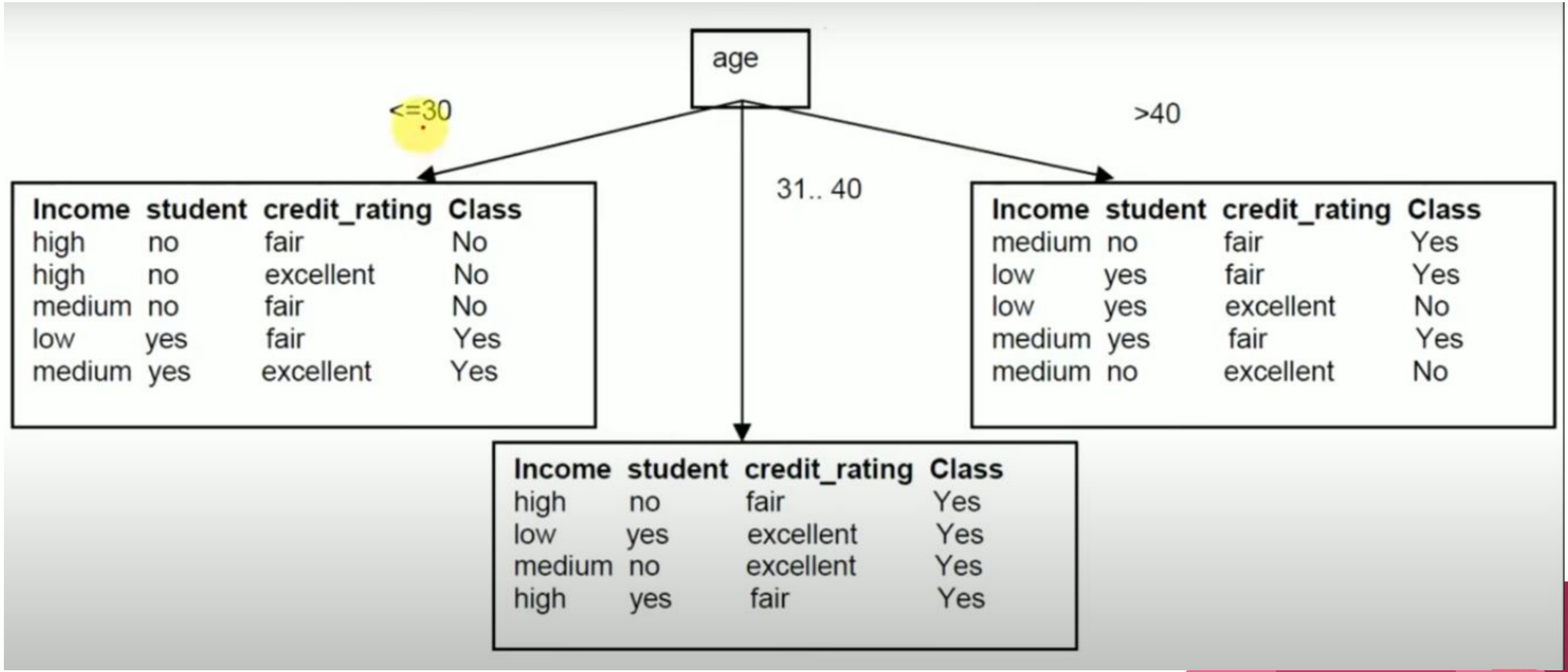
Information Gain:

$Age = 0.2465$

$Income = 0.0292$

$Student = 0.1516$

$Credit\_Rating = 0.048$



Income	student	credit_rating	Class
high	no	fair	No
high	no	excellent	No
medium	no	fair	No
low	yes	fair	Yes
medium	yes	excellent	Yes

Income	student	credit_rating	Class
high	no	fair	Yes
low	yes	excellent	Yes
medium	no	excellent	Yes
high	yes	fair	Yes

Income	student	credit_rating	Class
medium	no	fair	Yes
low	yes	fair	Yes
low	yes	excellent	No
medium	yes	fair	Yes
medium	no	excellent	No

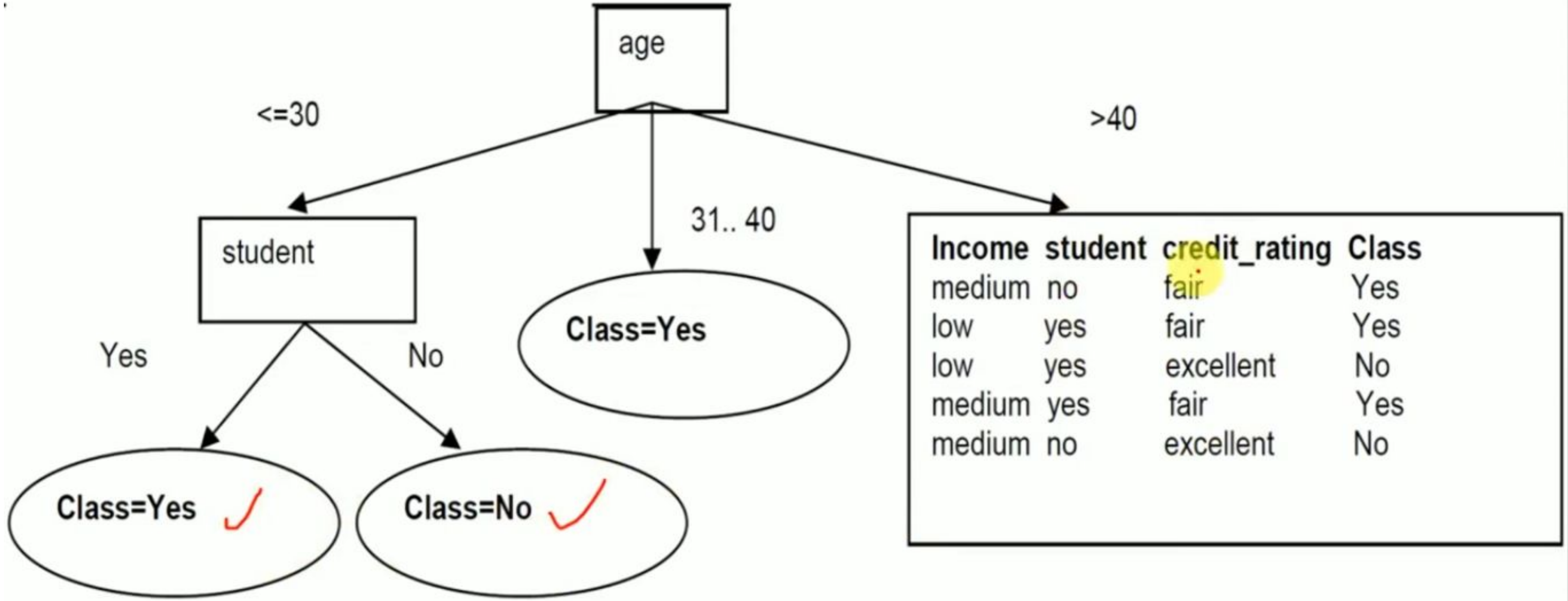
Income	student	credit_rating	Class
high	no	fair	No
high	no	excellent	No
medium	no	fair	No
low	yes	fair	Yes
medium	yes	excellent	Yes

Information Gain:

$$Income = 0.57$$

$$Student = 0.97$$

$$Credit\_Rating = 0.0192$$



Income	student	credit_rating	Class
medium	no	fair	Yes
low	yes	fair	Yes
low	yes	excellent	No
medium	yes	fair	Yes
medium	no	excellent	No

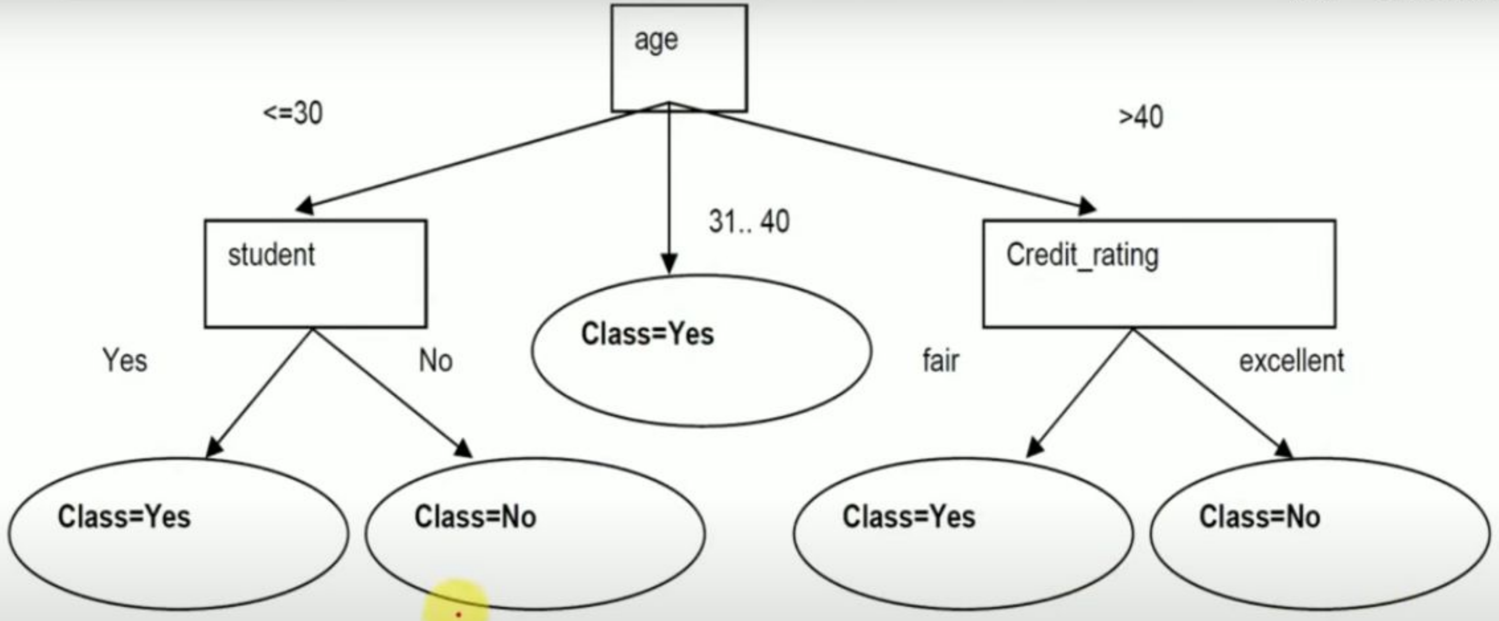
Income	student	credit_rating	Class
medium	no	fair	Yes
low	yes	fair	Yes
low	yes	excellent	No
medium	yes	fair	Yes
medium	no	excellent	No

Information Gain:

$$Income = 0.02$$

$$Student = 0.02$$

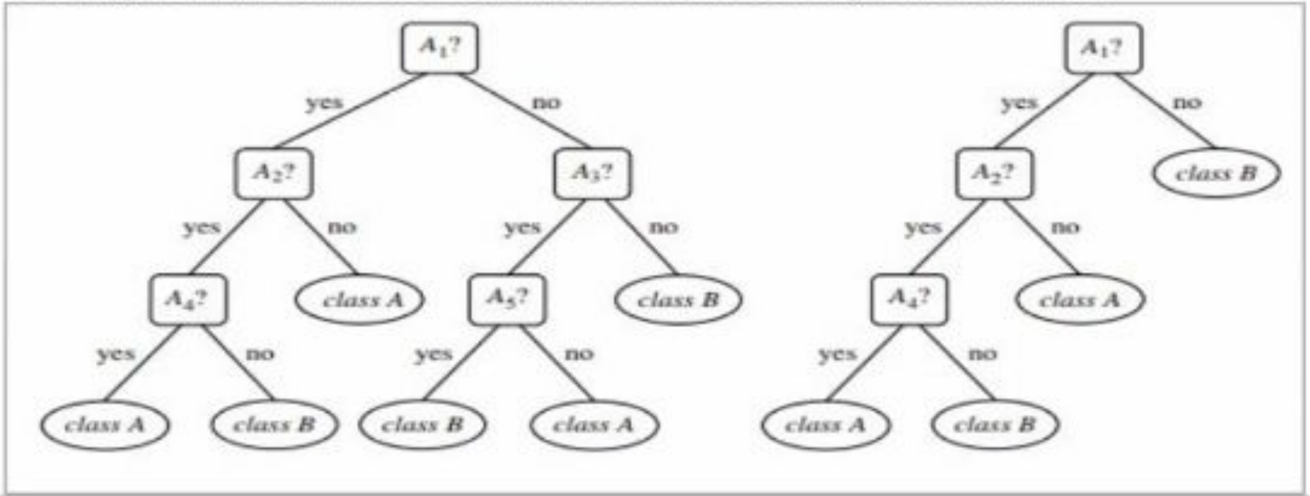
$$Credit\_Rating = 0.97$$



$New\ example: age <= 30, income = medium, student = yes, credit - rating = fair$

Decision Tree Pruning

- The splitting process results in fully grown trees until the stopping criteria are reached. But, the fully grown tree is likely to overfit the data, leading to poor accuracy on unseen data.
- In **pruning**, you trim off the branches of the tree, i.e., remove the decision nodes starting from the leaf node such that the overall accuracy is not disturbed.



https://colab.research.google.com/drive/1Cs9yuBShNGnDJq09L-L8XxYJM1USoDeh?usp=sharing

1. Pre-Pruning (Early Stopping)

Pre-pruning stops the growth of the tree early by imposing certain conditions. If the condition is met during the tree's construction, the node becomes a leaf node, and the tree is not allowed to grow further.

Common Pre-Pruning Techniques:

- **Minimum Samples per Node:** Stop splitting a node if the number of samples at the node is below a threshold.
- **Maximum Depth:** Set a limit on the depth of the tree.
- **Minimum Gain:** Only split a node if the information gain from the split exceeds a certain threshold.

Advantages:

- Reduces the complexity of the tree.
- Decreases computation time.
- Prevents overfitting to some extent.

Disadvantages:

- May lead to underfitting if the tree stops growing too early.

2. Post-Pruning (Cost Complexity Pruning)

Post-pruning involves first generating the entire tree and then trimming it back. Nodes are removed if the tree's performance improves or remains the same after removing them.

Steps in Post-Pruning:

1. **Grow the Full Tree:** Allow the decision tree to grow to its full depth.
2. **Prune the Tree:** Prune the tree by removing nodes that add little predictive power, based on a cost function that balances tree complexity and accuracy.

Common Post-Pruning Techniques:

- **Reduced Error Pruning:** Remove nodes that do not lead to a decrease in accuracy on a validation set.
- **Cost Complexity Pruning (CCP):** Calculate a cost-complexity metric for each subtree and prune the one with the lowest increase in this metric. CCP is implemented in algorithms like CART (Classification and Regression Trees).

Advantages:

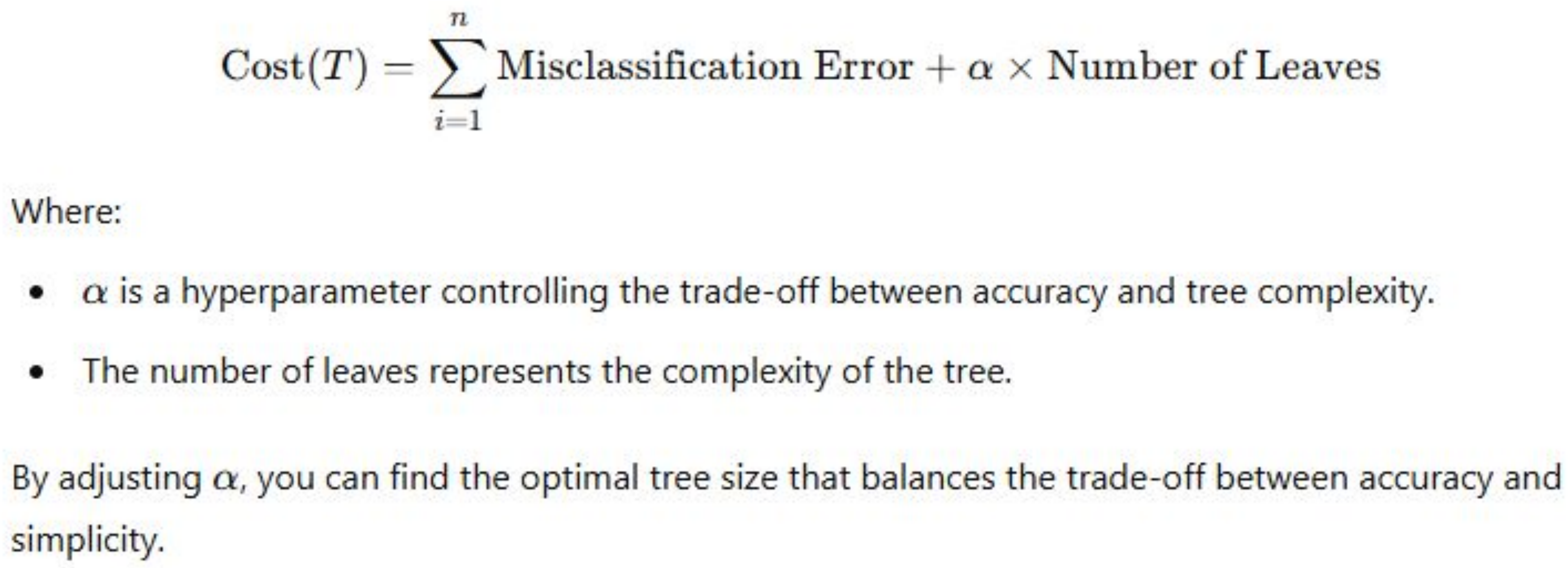
- Helps in reducing overfitting by removing unnecessary branches.
- Results in a simpler, more interpretable model.

Disadvantages:

- Requires the tree to be fully grown, which can be computationally expensive.

3. Minimal Cost-Complexity Pruning (CCP) in CART

In CCP, the complexity of the tree is measured by a parameter α\alphaα, which penalizes the complexity (size) of the tree. The goal is to find the subtree that minimizes the total cost, defined as:



$$\text{Cost}(T) = \sum_{i=1}^{n} \text{Misclassification Error} + \alpha \times \text{Number of Leaves}$$

Where:

- α is a hyperparameter controlling the trade-off between accuracy and tree complexity.
- The number of leaves represents the complexity of the tree.

By adjusting α, you can find the optimal tree size that balances the trade-off between accuracy and simplicity.

4. Pruning with Cross-Validation

Cross-validation can be combined with pruning to choose the optimal size of the tree. Here, the tree is pruned to different extents, and the performance of each pruned tree is evaluated using cross-validation. The tree with the best cross-validation score is selected.

Advantages:

- Provides a good balance between bias and variance.
- Ensures the pruned tree generalizes well to unseen data.

Disadvantages:

- Computationally expensive due to repeated training and evaluation.

Random Forest

- **Random Forest** is a popular and powerful ensemble learning technique used for both classification and regression tasks.

- It builds upon the decision tree algorithm by combining the predictions of multiple decision trees to produce a more accurate and stable prediction.

1. Basic Concepts:

- **Ensemble Learning:** This is the process of combining multiple models (in this case, decision trees) to improve overall performance.
- **Decision Trees:** Random Forest is based on decision trees. A decision tree is a model that splits the data into branches based on feature values, making decisions at each node to eventually classify or predict the outcome.

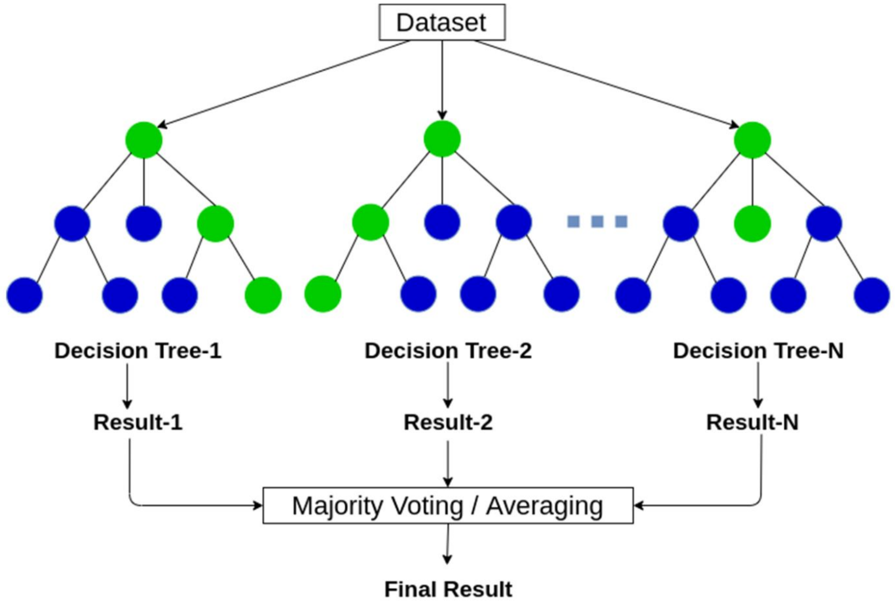
2. How Random Forest Works:

- **Bootstrap Aggregating (Bagging):**
  - Random Forest creates multiple decision trees, each trained on a different subset of the training data. These subsets are created by randomly sampling the data with replacement (bootstrap sampling).
  - This means that each tree in the forest is trained on slightly different data, leading to diversity among the trees.
- **Random Feature Selection:**
  - At each split in the tree, a random subset of features is chosen, and the best split is selected from this subset. This process of feature randomness helps in making the trees less correlated with each other, further increasing the diversity.
- **Prediction:**
  - For classification tasks, each tree in the forest makes a prediction (vote), and the class that receives the most votes across all trees is chosen as the final prediction.
  - For regression tasks, the predictions from all trees are averaged to produce the final output.

3. Why Random Forest is Effective:

- **Reduces Overfitting:** Individual decision trees are prone to overfitting, especially when they grow deep and capture noise in the data. By averaging the results of multiple trees, Random Forest reduces the likelihood of overfitting and improves generalization.
- **Handles High Dimensional Data:** Random Forest can handle large numbers of features without significant performance degradation. The random selection of features at each split ensures that not all trees will make decisions based on the same set of features.
- **Robust to Noise:** Since Random Forest builds multiple trees on different subsets of the data, it is less sensitive to noise and outliers in the data.
- **Feature Importance:** Random Forest provides an estimate of feature importance, helping to understand which features are most influential in the prediction process.

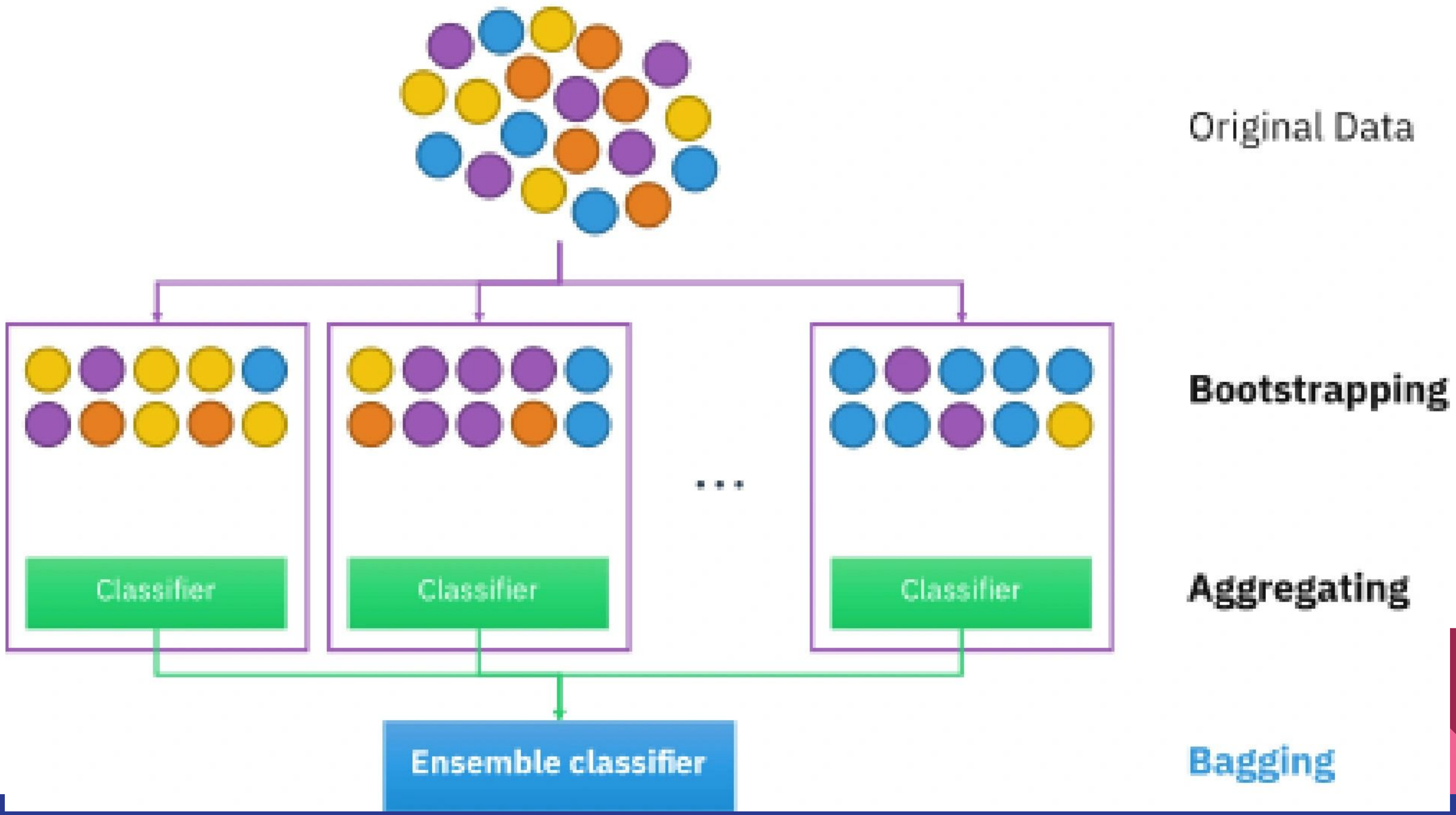
Random Forest



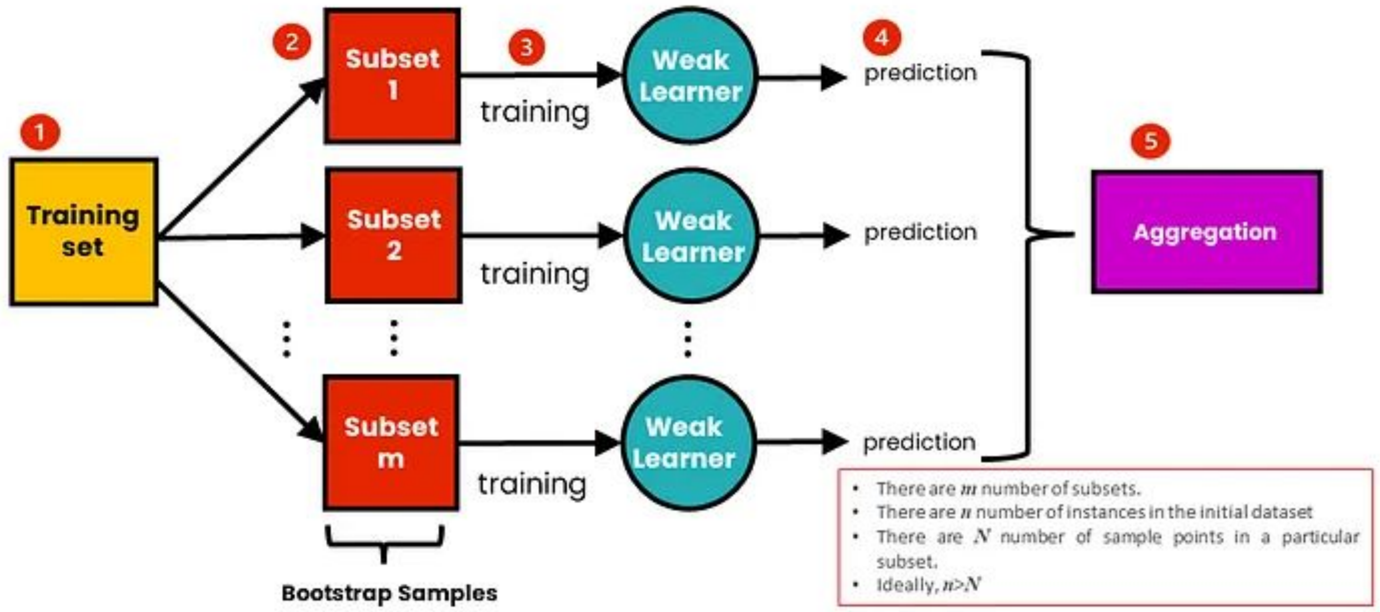
What is Bagging? **(Ensemble methods)**

Bagging (Bootstrap Aggregating) is an ensemble learning technique designed to improve the accuracy and stability of machine learning algorithms. It involves the following steps:

1. **Data Sampling:** Creating multiple subsets of the training dataset using bootstrap sampling (random sampling with replacement).
2. **Model Training:** Training a separate model on each subset of the data.
3. **Aggregation:** Combining the predictions from all individual models (averaged for regression or majority voting for classification) to produce the final output.


Original Data
Bootstrapping
Aggregating
Bagging
Classifier
Classifier
Classifier
. . .
Ensemble classifier

The Process of Bagging (Bootstrap Aggregation)



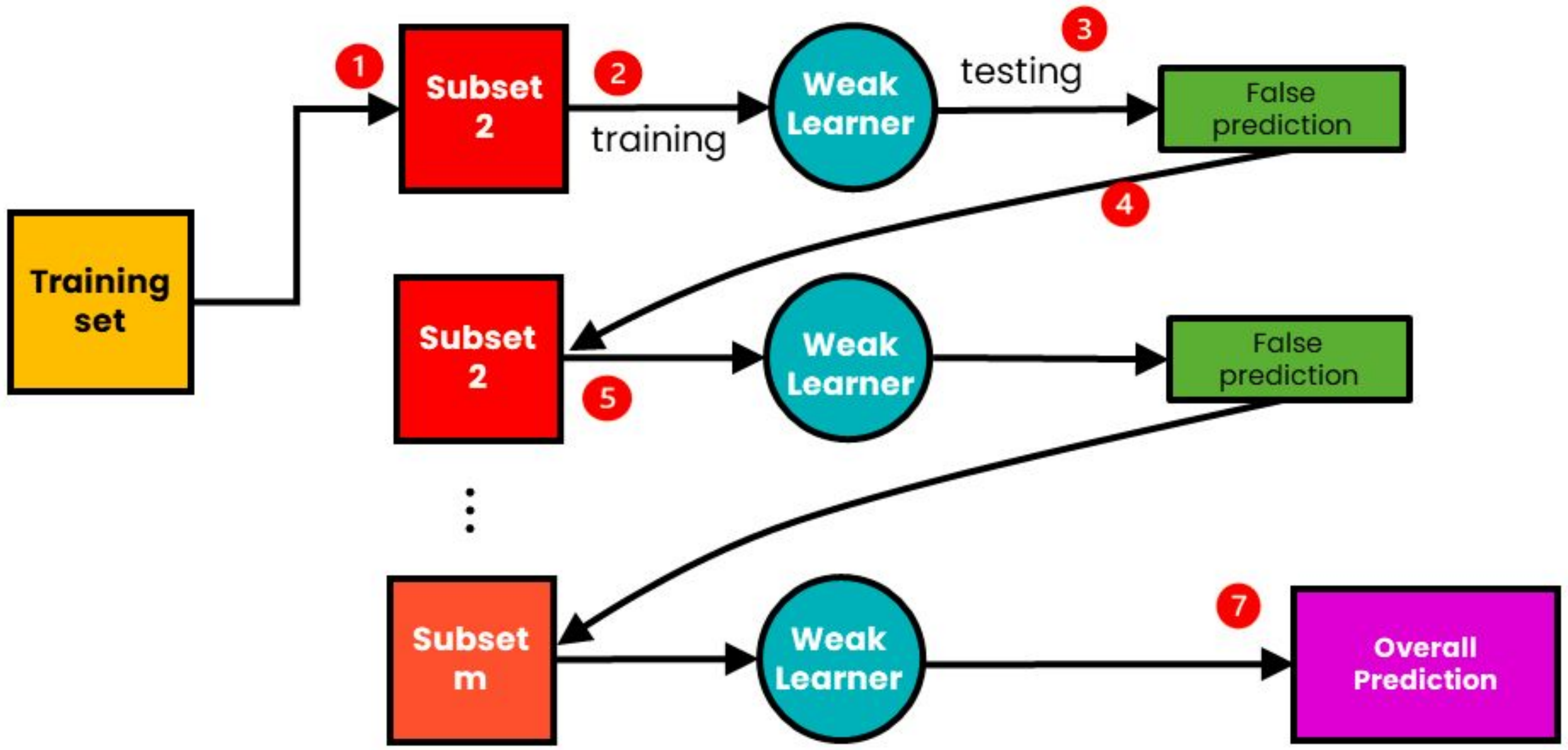
- There are m number of subsets.
- There are n number of instances in the initial dataset
- There are N number of sample points in a particular subset.
- Ideally, $n>N$

What is Boosting? **(Ensemble methods)**

Boosting is another ensemble learning technique that focuses on creating a strong model by combining several weak models. It involves the following steps:

1. **Sequential Training:** Training models sequentially, each one trying to correct the errors made by the previous models.
2. **Weight Adjustment:** Each instance in the training set is weighted. Initially, all instances have equal weights. After each model is trained, the weights of misclassified instances are increased so that the next model focuses more on difficult cases.
3. **Model Combination:** Combining the predictions from all models to produce the final output, typically by weighted voting or weighted averaging.

The Process of Boosting



Difference Between Decision Tree and Random Forest

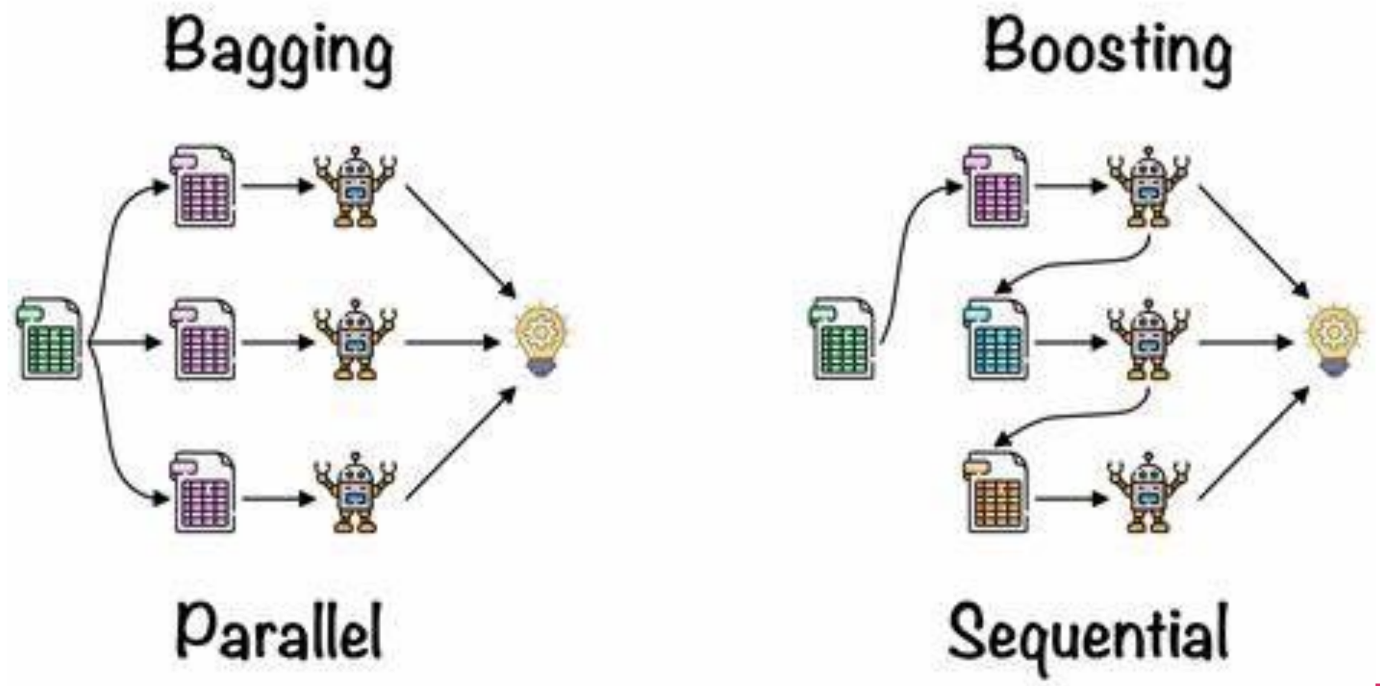
Random forest is a collection of decision trees; still, there are a lot of differences in their behavior.

Decision trees	Random Forest
1. Decision trees normally suffer from the problem of overfitting if it's allowed to grow without any control.	1. Random forests are created from subsets of data, and the final output is based on average or majority ranking; hence the problem of overfitting is taken care of.
2. A single decision tree is faster in computation.	2. It is comparatively slower.
3. When a data set with features is taken as input by a decision tree, it will formulate some rules to make predictions.	3. Random forest randomly selects observations, builds a decision tree, and takes the average result. It doesn't use any set of formulas.

There are various sorts of boosting algorithms that can be employed in machine learning. Here are a few of the most well-known:

1. ***AdaBoost (Adaptive Boosting):*** AdaBoost is one of the most extensively used boosting algorithms. It gives weights to each data point in the training set based on the accuracy of prior models, and then trains a new model using the updated weights. AdaBoost is very useful for classification tasks.
2. ***Gradient Boosting:*** Gradient Boosting works by fitting new models to the residual errors of prior models. It minimizes the loss function using gradient descent and may be applied to both regression and classification problems.

1. ***Stochastic Gradient Boosting:*** Similar to Gradient Boosting, Stochastic Gradient Boosting fits each new model with random subsets of the training data and random subsets of the features. This helps to avoid overfitting and may result in improved performance.
2. ***LPBoost (Linear Programming Boosting):*** LPBoost is a boosting algorithm that minimizes the exponential loss function using linear programming. It is capable of handling a wide range of loss functions and may be applied to both regression and classification issues.
3. ***TotalBoost (Total Boosting):*** TotalBoost is an AdaBoost and LPBoost boosting method. It works by minimizing a mixture of exponential and linear programming losses, and it can increase accuracy for certain types of problems.


Bagging
Boosting
Parallel
Sequential

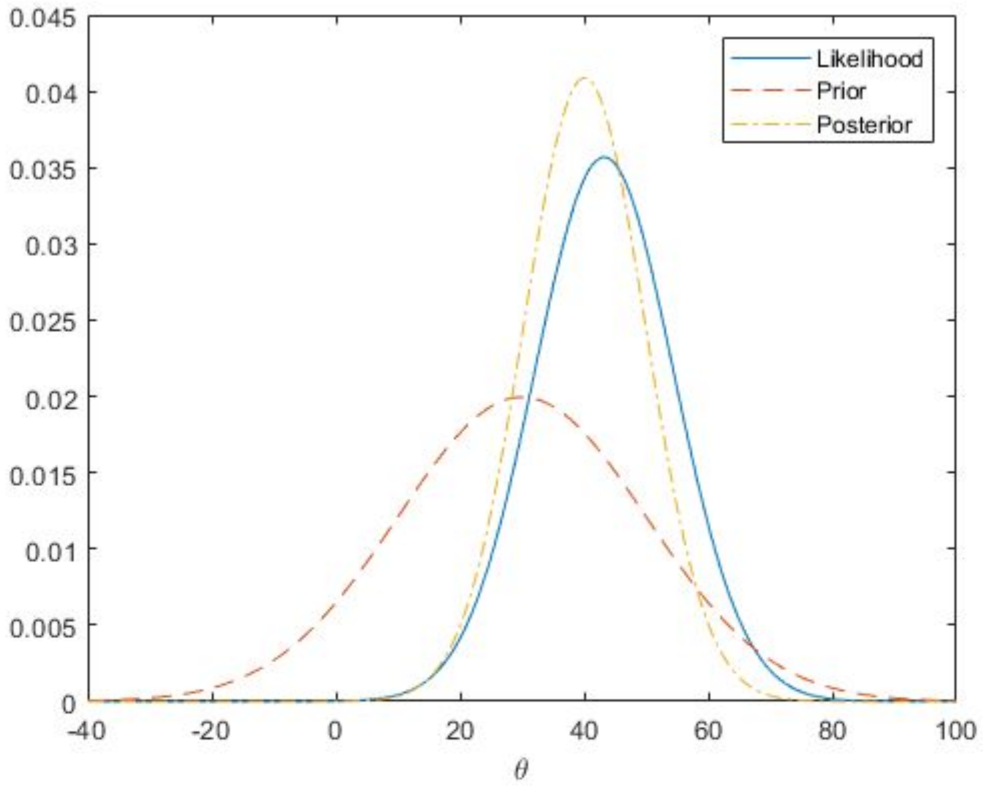
Bayesian Regression

Bayesian regression is a statistical method where the linear regression model is updated based on prior distributions for the model parameters. Instead of finding a single best estimate for the regression coefficients, Bayesian regression provides a distribution of possible coefficients, allowing for uncertainty quantification.

1. **Prior Distribution**: Before observing the data, you define prior beliefs about the model parameters. For example, you might believe that the coefficients are centered around zero with some variance.
2. **Likelihood Function**: This represents the probability of observing the data given the model parameters. In linear regression, the likelihood is usually based on a Gaussian distribution.
3. **Posterior Distribution**: After observing the data, the prior beliefs are updated to form the posterior distribution using Bayes' theorem:

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}$$

4. This posterior distribution represents the updated belief about the model parameters after considering the observed data.
5. **Prediction**: Predictions are made by averaging over the posterior distribution, often leading to more accurate and robust predictions, especially when dealing with limited or noisy data.


0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
-40
-20
0
20
40
60
80
100
θ
Likelihood
Prior
Posterior