Chapter 2

Supervised Learning

2. Supervised Learning [10 hours]

2.1 Types of Supervised Learning: Regression and Classification

2.2 Regression

2.2.1 Linear Regression

Simple and Multiple Regression
Polynomial Regression

2.2.2 Regularization Techniques

Ridge Regression
Lasso Regression
Bias-variance Tradeoff

2.2.3 Support Vector Regression

2.3 Classification

2.3.1 Logistic Regression

Binary Classification
Multi-class classification

2.3.2 K-Nearest Neighbors (KNN)

2.3.3 Support Vector Machine (SVM)

Hyperplane ad Support Vectors
Kernels and its Types: Linear, Polynomial, Radial Basis Function (RBF)
SVM for Linear and Non-linear classification

2.3.4 Decision Trees

Construction and pruning of decision trees
Ensemble Methods: Bagging, Random forests

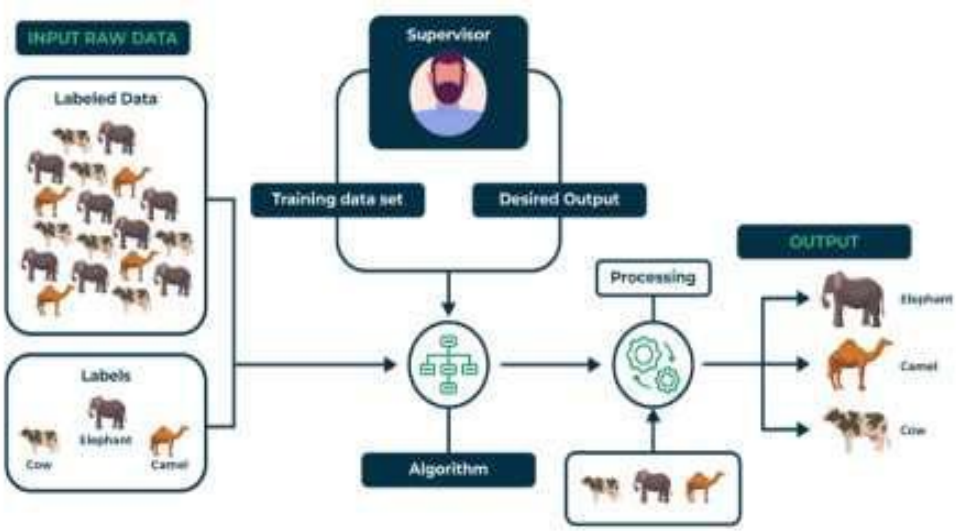
2. Supervised Learning [10 hours]

In supervised learning, the model is trained with labeled data where each input is paired with a corresponding output.

Supervised Machine Learning is a type of machine learning where the algorithm is trained on a **labeled dataset** — meaning each training example is paired with a correct output. The goal is for the model to **learn the mapping from inputs to outputs**, so it can accurately predict outcomes on new, unseen data.

Key Characteristics of Supervised Learning:

- **Uses labeled data** (input-output pairs).
- Learns a function that maps inputs to outputs.
- Requires a **training phase** where the model learns from known data.
- Performance is evaluated using metrics like accuracy, precision, recall, etc.



How Supervised Machine Learning Works?

Where supervised learning algorithm consists of input features and corresponding output labels. The process works through:

- **Training Data:** The model is provided with a training dataset that includes input data (features) and corresponding output data (labels or target variables).
- **Learning Process:** The algorithm processes the training data, learning the relationships between the input features and the output labels. This is achieved by adjusting the model's parameters to minimize the difference between its predictions and the actual labels.

- **Training Phase** involves feeding the algorithm labeled data, where each data point is paired with its correct output. The algorithm learns to identify patterns and relationships between the input and output data.
- **Testing Phase** involves feeding the algorithm new, unseen data and evaluating its ability to predict the correct output based on the learned patterns.

2.1 Types of Supervised Learning: Regression and Classification

Supervised learning is broadly categorized into **two main types** based on the nature of the output variable: **Regression and Classification.**

- Regression is used when the **output variable is continuous**, meaning it can take any numeric value within a range.
- Classification is used when the **output variable is categorical**, i.e., it belongs to one of several predefined classes or categories.

2.2 Regression

Regression in machine learning refers to a **supervised learning** technique where the goal is to predict a continuous numerical value based on one or more independent features. It finds relationships between variables so that predictions can be made.

We have two types of variables present in regression:

- **Dependent Variable (Target)**: The variable we are trying to predict e.g house price.
- **Independent Variables (Features)**: The input variables that influence the prediction e.g locality, number of rooms.

2.3.1 Linear Regression

a) Simple Linear Regression

Linear regression is one of the simplest and most widely used statistical models. This assumes that there is a linear relationship between the independent and dependent variables. This means that the change in the dependent variable is proportional to the change in the independent variables.

For example predicting the price of a house based on its size.

Simple Linear Regression models the relationship between a single independent variable (X) and a dependent variable (Y). It tries to find the best-fitting line for the form:

$$Y = mX + c$$

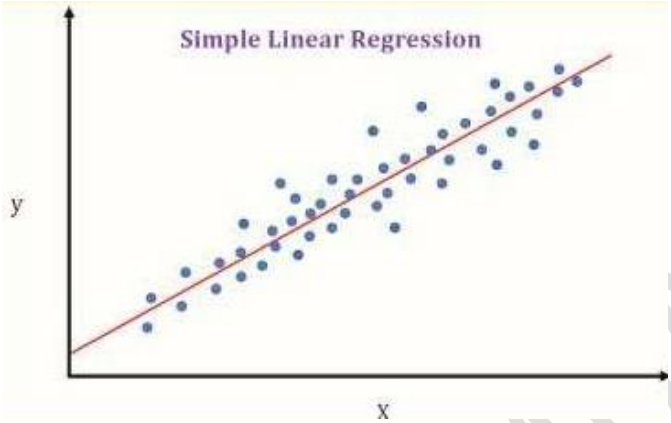
Where:

Y= Dependent variable (target/output)

X= Independent variable (input feature)

m= Slope of the line (coefficient)

c = Intercept



In the above Diagram X (Independent variable) is plotted against Y (Dependent variable).

- Blue points represent the data.
- The red line in the above graph is referred to as the best-fit straight line.
- Based on the given data points, we try to plot a line that models the points the best. The best fit line is the line that **minimizes the difference** between the **actual values** and the **predicted values**.
-

How to Find the Best Fit Line?

The **Least Squares Method** is a mathematical technique used to **find the best-fitting line** (or curve) through a set of data points by **minimizing the sum of the squares of the vertical distances (errors)** between the observed values and the predicted values on the line.

In the context of **simple linear regression**, it is used to find the **best fit line** y=mx+c.

To find the values of **slope** m and **intercept** c that minimize this error, the least squares method uses these formulas:

$$m = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2} \qquad c = \frac{\sum y - m\sum x}{n}$$

Where:

- n: number of data points
- $\sum xy$: sum of the product of co rresponding x and y values
- $\sum x$, $\sum y$, $\sum x^2$: sums over the dataset

Example:

Given a dataset with height and weight, find the regression line that best fits to predict the weight of a person having a height of 172 cm.

X (Height)	Y (Weight)
150	50
160	56
170	62
180	68

Solution:

Here, n=4

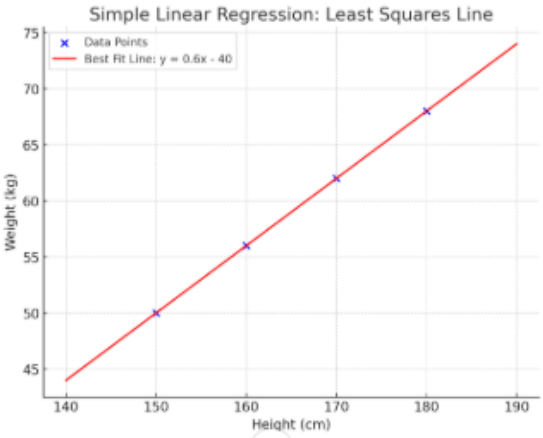
X (Height)	Y (Weight)	X * Y	X^2
150	50	7500	22500
160	56	8960	25600
170	62	10540	28900
180	68	12240	32400
$\sum x = 660$	$\sum y = 236$	$\sum XY = 39240$	$\sum X^2 = 109400$

$$m = \frac{n\sum xy - \sum x \sum y}{n\sum X - (\sum x)^2} = \frac{(4 * 39240 - 660 * 236)}{4 * 109400 - (660)^2} = 4 * 109400 - (660)^2 = 0.6$$

$$c = \frac{\sum y - m\sum x}{n} = \frac{236 - 0.6 * 660}{4} = -40$$

Final Simple Linear Regression Model is: **y = 0.6x - 40**

This can be fit in a graph as below:



Now, if you want to predict the weight whose height is 172 then, Weight= 0.6*172-40 = 63.2

Assignment:

The following data shows the sales (in million dollars) of a company. Estimate the sales in the year 2020 using the regression line.

x	2015	2016	2017	2018	2019
y	12	19	29	37	45

b) Multiple Linear Regression

Multiple Regression is an extension of **simple linear regression**, where instead of using just **one independent variable**, we use **two or more independent variables** to predict the value of a dependent variable. Here, multiple features contribute to the prediction in the multiple linear regression.

When there are multiple independent variables (X_1, X_2, ..., Xn), the equation extends to:

$$Y=m_0 + m_1X_1 + m_2X_2 + ... + m_nX_n$$

Where:

- y: dependent variable (what we are trying to predict)
- $x_1,x_2,\ldots,x_n$: independent variables (features)
- m_0: intercept (constant term)
- $m_1,m_2,\ldots,m_n$: coefficients (slopes showing the effect of each x_i on y)

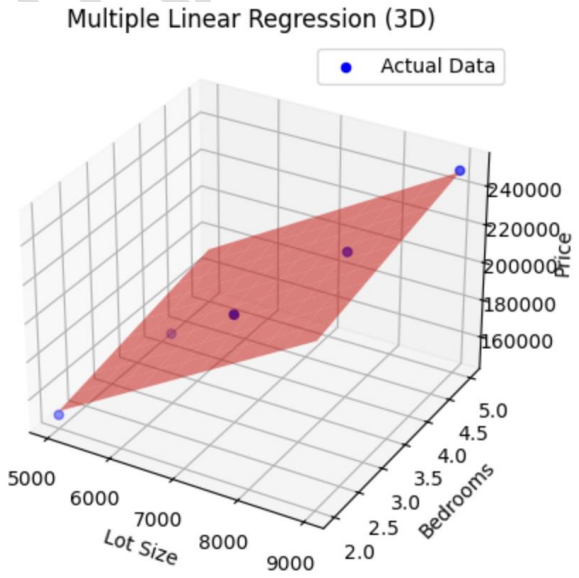


Figure: Multiple linear regression plane

Example:

Given a dataset with study hours and sleep hours for 5 students. Use linear regression to predict the exam score for a new student that studies 6 hours and sleeps 4 hours.

Student	Hours Studied (x1)	Hours Slept (x2)	Exam Score (y)
1	2	9	60
2	1	6	50
3	3	7	65
4	4	5	70
5	5	5	75

Solution:

Here:

The multiple regression of two variables x1 and x2 is given by:

$y = f(X_1, X_2)$

$y = m_0 + m_1X_1 + m_2 X_2$

Here the matrices for X and Y are given as :

$$X = \begin{bmatrix} 1 & 2 & 9 \\ 1 & 1 & 6 \\ 1 & 3 & 7 \\ 1 & 4 & 5 \\ 1 & 5 & 5 \end{bmatrix}, \quad y = \begin{bmatrix} 60 \\ 50 \\ 65 \\ 70 \\ 75 \end{bmatrix}$$

Now slope matrix is given by: $(X^T X)^{-1} X^T y$

Here,

$$X^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 3 & 4 & 5 \\ 9 & 6 & 7 & 5 & 5 \end{bmatrix}$$

$$X^TX= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 3 & 4 & 5 \\ 9 & 6 & 7 & 5 & 5 \end{bmatrix} \begin{bmatrix} 1 & 2 & 9 \\ 1 & 1 & 6 \\ 1 & 3 & 7 \\ 1 & 4 & 5 \\ 1 & 5 & 5 \end{bmatrix} = \begin{bmatrix} 5 & 15 & 32 \\ 15 & 55 & 90 \\ 32 & 90 & 216 \end{bmatrix}$$

$$(X^TX)^{-1} \approx \begin{bmatrix} 1.476 & -0.706 & -0.118 \\ -0.706 & 0.412 & -0.010 \\ -0.118 & -0.010 & 0.059 \end{bmatrix}$$

$$X^Ty = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 3 & 4 & 5 \\ 9 & 6 & 7 & 5 & 5 \end{bmatrix} \begin{bmatrix} 60 \\ 50 \\ 65 \\ 70 \\ 75 \end{bmatrix} = \begin{bmatrix} 320 \\ 1020 \\ 2020 \end{bmatrix}$$

Now:

$$(X^TX)^{-1}X^Ty = \begin{bmatrix} 37.37 \\ 6.63 \\ 1.05 \end{bmatrix}$$

Thus, m $= \begin{bmatrix} 37.37 \\ 6.63 \\ 1.05 \end{bmatrix}$

Hence the final multiple regression line is:

$y = 37.37 + 6.63\ X_1 + 1.05\ X_2$

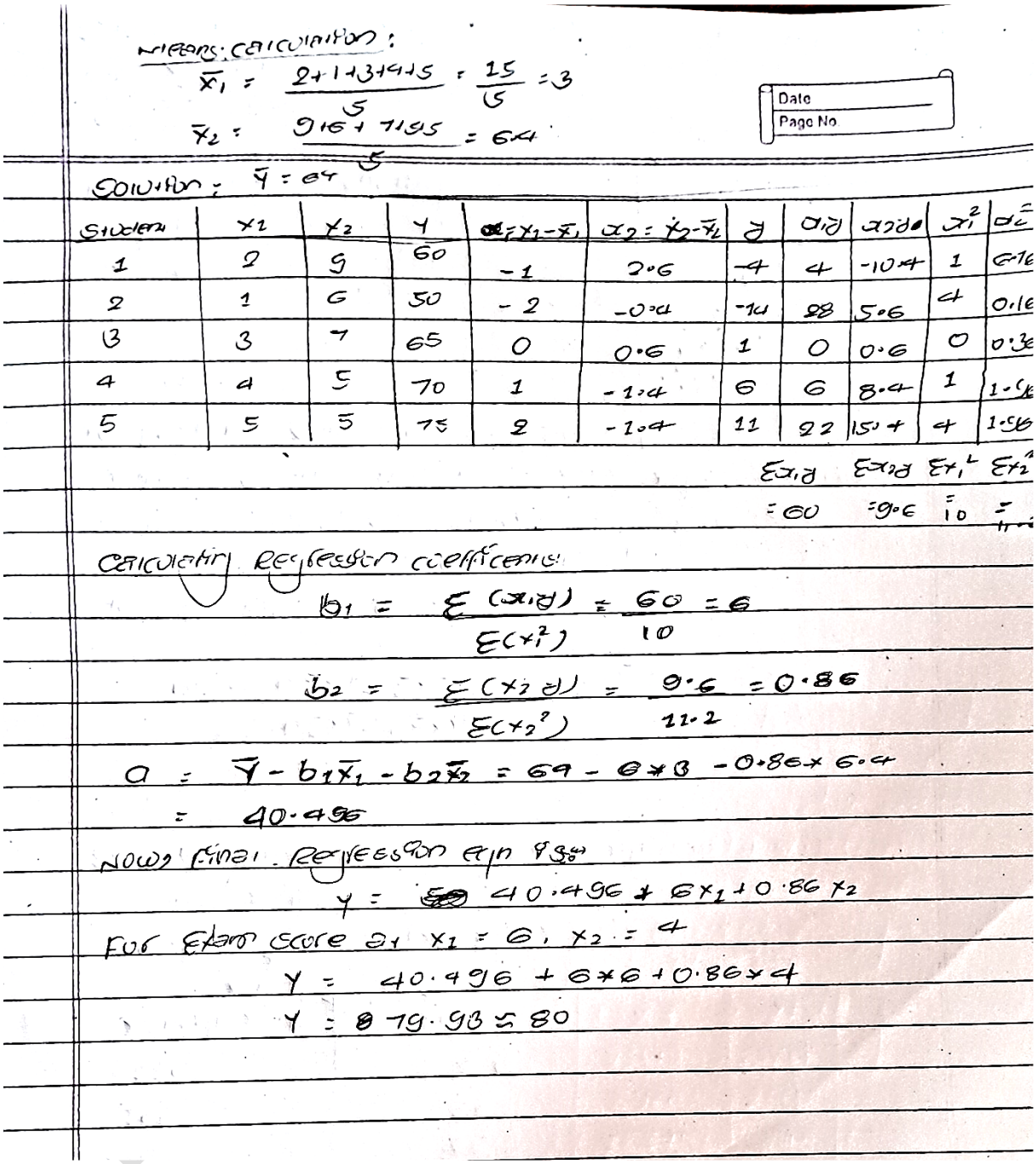
Finally, the score for student who studies 6 hours and sleeps 4 hours can be predicted as:

y = 37.37 + 6.63·x1 + 1.05·x2

= 37.37 + 6.63 * 6 + 1.05 *4

= 81.35 score

Alternative way



Means calculation:

$\bar{x}_1 = \frac{2+1+3+4+5}{5} = \frac{15}{5} = 3$

$\bar{x}_2 = \frac{9+6+7+5+5}{5} = 6.4$

Solution: $\bar{y} = 64$

Student	x_1	x_2	y	$x_1 - \bar{x}_1$	$x_2 - \bar{x}_2$	y	xy	$x_1 y$	x_1^2	x_2^2
1	2	9	60	-1	2.6	-4	4	-10.4	1	6.76
2	1	6	50	-2	-0.4	-14	28	5.6	4	0.16
3	3	7	65	0	0.6	1	0	0.6	0	0.36
4	4	5	70	1	-1.4	6	6	8.4	1	1.96
5	5	5	75	2	-1.4	11	22	15.4	4	1.96

Σxy $\Sigma x_1 y$ Σx_1^2 Σx_2^2

$= 60$ $= 9.6$ $= 10$ $= 11.2$

Calculating Regression coefficients:

$$b_1 = \frac{\Sigma(x_1 y)}{\Sigma(x_1^2)} = \frac{60}{10} = 6$$

$$b_2 = \frac{\Sigma(x_2 y)}{\Sigma(x_2^2)} = \frac{9.6}{11.2} = 0.86$$

$$a = \bar{y} - b_1\bar{x}_1 - b_2\bar{x}_2 = 64 - 6*3 - 0.86 \times 6.4$$

$$= 40.496$$

Now, final Regression eqn is

$$y = 40.496 + 6x_1 + 0.86x_2$$

For Exam score of $x_1 = 6$, $x_2 = 4$

$$y = 40.496 + 6*6 + 0.86*4$$

$$y = 79.93 \simeq 80$$

c) Polynomial Regression

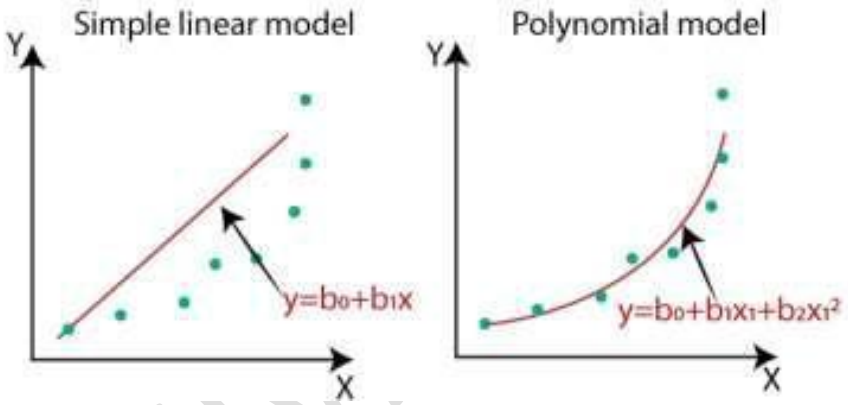
Polynomial Regression is an extension of Linear Regression where the relationship between input variables and output is modeled as an **nth-degree polynomial**:

$$Y = a_0 + a_1X + a_2X^2 + a_3X^3 + ... + a_nX^n$$

Where:

- x, x^2, x^3 … are features (powers of x).
- a_0, a_1,...,a_n are coefficients.

This is useful when data shows **non-linear trends** that a straight line cannot fit well. In Polynomial Regression, the relationship between the input variables (X) and output (y) is non-linear (curved), But we model it using higher-degree polynomials. It still fits a line, but a curved line (like a parabola, cubic curve, etc.).



2.3.2 Model Evaluation for Regression

Sometimes when we train a machine-learning model, it may capture all the underlying noise and may not train well. Some other times, the machine-learning model may be trained well, but may not predict well on new data.

Model evaluation is a process that uses some metrics which help us to analyze the performance of the model. It is a crucial step in assessing how well a machine learning model performs. It helps in:

- Comparing different models.
- Detecting overfitting/underfitting.
- Fine-tuning hyperparameters.
- Ensuring generalization to unseen data.

Regression Metrics:

Regression Metrics helps to evaluate how well the model predicts compared to actual values.

Sometimes it is also known as loss function. It is used when the output is a continuous value.

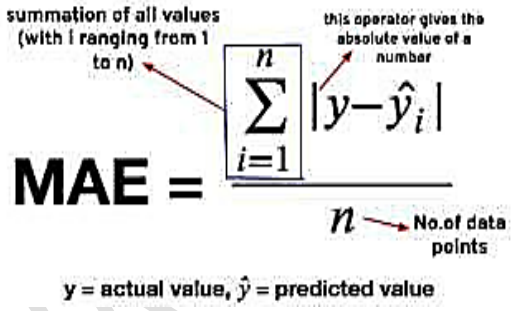
i.e. These metrics are used to evaluate the performance of regression models.

Some of the evaluation metrics are:

i) Mean Absolute Error (MAE)
ii) Mean Squared Error (MSE)
iii) Root Mean Squared Error (RMSE)
iv) R-Squared (R^2)

i) Mean Absolute Error (MAE)

- It's defined as the average of the absolute difference between actual and predicted values.



- **Interpretation:**
  - How much the predictions deviate from the true values, on average.
  - Lower MAE means better predictions.

ii) Mean Squared Error (MSE)

- The average of the squared differences between the predicted values and actual values.

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

- **Interpretation:**
  - Measures how far predictions are from the actual values.
  - Lower MSE indicates better model performance.

iii) Root Mean Squared Error (RMSE)

- The **square root** of the MSE.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

- **Interpretation:**
  - Brings the error back to the same units as the output (y).
  - Easy to interpret like MAE.

iv) R-Squared (R^2)

- It measures the proportion of variance in the dependent variable that is predictable from the independent variables.

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

- **Interpretation:**
  - R^2=1 Perfect fit.
  - R^2=0: Model performs as badly as the mean.
  - R^2<0: Worse than the mean.

Example:

Compute the MAE, MSE, RMSE and R2 a model that performed below prediction as in the report.

Observation	Actual Value (y)	Predicted Value (ŷ)
1	3	2.5
2	5	5.1
3	7	6.8
4	9	9.3

Actual values: y = [3,5,7,9]

Predicted values: ŷ = [2.5,5.1,6.8,9.3]

$$\text{MAE} = \frac{1}{n}\sum|y_i - \hat{y}_i|$$

$$= \frac{|3-2.5|+|5-5.1|+|7-6.8|+|9-9.3}{4}$$

$$= \frac{0.5+0.1+0.2+0.3}{4}$$

$$= 0.257$$

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

$$= \frac{1}{4} \text{ x } (3\text{-}2.5)^2 + (5\text{-}5.1)^2 + (7\text{-}6.8)^2 + (9\text{-}9.3)^2$$

$$= 0.25+0.01+0.04+0.09$$

$$=0.39$$

$$\text{RMSE} = \sqrt{MSE}$$

$= \sqrt{0.0975}$

$= 0.312$

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

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

Where:

- SSE = Sum of Squared Errors (already calculated = 0.39)
- SST = Total Sum of Squares = Sum of $(y_i - \bar{y})^2$
- $\bar{y}$ = mean of actual y values

$$\bar{y} = \frac{3+5+7+9}{4} = 6$$

We have:

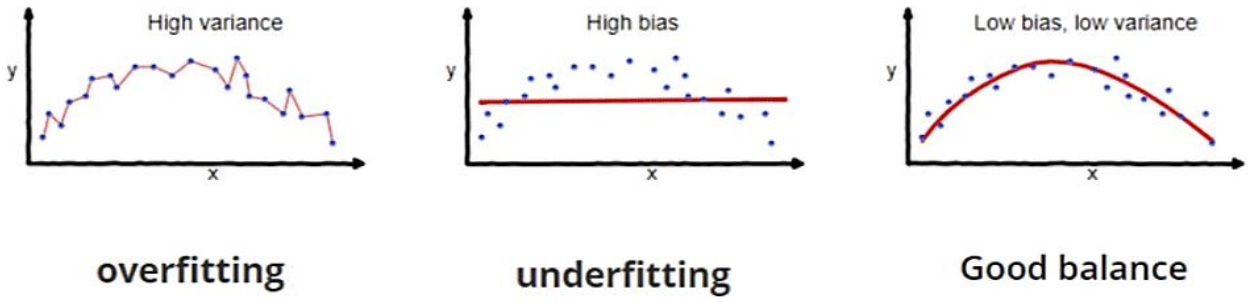
i	**y_i**	**$\hat{y}$**	**$(y_i - \hat{y})^2$**
1	3	2.5	0.25
2	5	5.1	0.01
3	7	6.8	0.04
4	9	9.3	0.09

i	**yi**	**$y_i - \bar{y}$**	**$(y_i - \bar{y})^2$**
1	3	-3	9
2	5	-1	1
3	7	1	1
4	9	3	9

Hence $R^2 = 1 - \frac{0.39}{20} = 0.0805$

2.3.3 Regularization Techniques

Concept of Overfitting and Underfitting

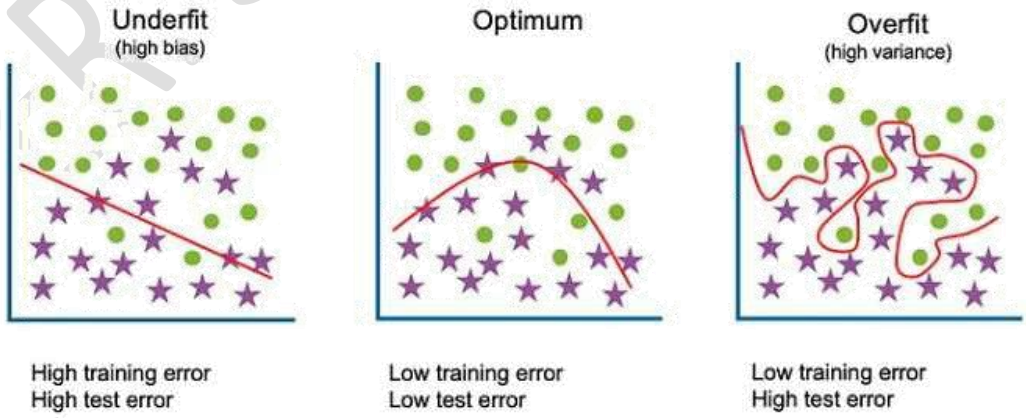


The image illustrates three scenarios in model performance:

- **Overfitting** – The model is too complex, capturing noise and outliers leading to poor generalization.
- **Underfitting** – The model is too simple failing to capture the underlying data patterns.
- **Optimal Fit** – A balanced model that generalizes well achieving low bias and low variance and it can be achieved by using regularization techniques.

Underfitting and overfitting are two common challenges faced in machine learning.

- **Underfitting** happens when a model is not good enough to understand all the details in the data. It's like the model is too simple and misses important stuff. This leads to poorperformance on both the training and test sets.
- **Overfitting** on the other hand, occurs when a model is too complex and memorizes the training data too well. This leads to good performance on the training set but poor performance on the test set.



Regularization:

Regularization techniques are essential tools in **machine learning**—especially in **regression problems**—to prevent **overfitting**, improve **generalization**, and handle **multicollinearity** among features. Without regularization, a model may memorize training data (overfit). Regularization forces the model to stay simpler by shrinking the weights (coefficients).

Why regularization?

Regularization is **crucial in regression models** to improve their **generalization** and **robustness**, especially when the model is at risk of **overfitting**

i) **Prevent Overfitting:**
It controls model complexity by penalizing large coefficients, helping the model generalize better to unseen data.

ii) **Handle Multicollinearity:**
Regularization (like Ridge) stabilizes coefficients when predictor variables are highly correlated.

iii) **Feature Selection:**
Lasso (L1) can shrink some coefficients to zero, removing irrelevant features. This simplifies the model and improves interpretability.

iv) **Improve Generalization:**
Balances bias and variance for better prediction on test data.

How Regularization Works?

Original loss function (example for Linear Regression):

Loss = Mean Squared Error (MSE)

Regularized loss function:

Loss = MSE + Penalty Term

The **penalty term** depends on the regularization technique.

Types of Regularization:

a) Ridge Regression (L2 Regularization)

- **Ridge Regression** is a regularized version of linear regression that adds an **L2 penalty** to the loss function to prevent overfitting and handle multicollinearity (when independent variables are highly correlated).
- **Analogy:**
  Imagine you're packing a **backpack for hiking**.
  - If you pack **without any restrictions**, you might fill your bag with:
    Snacks: 3kg, Jacket: 4kg , Books: 2kg , Toy: 2kg , Mirror: 1kg
    Total = **12kg** → Too heavy and hard to carry (overfit model).
  - Now, you add a **penalty for excess weight**, so you **shrink the less important items**:
    Snacks: 2.5kg,Jacket: 3.5kg, Books: 1kg, Toy: 0.5kg , Mirror: 0.5kg
    Total = **8kg** → Still includes everything, but in **controlled amounts**.

- When to Use?
  - You suspect **overfitting** due to high model complexity.
  - Your data has **multicollinearity** (correlated predictors).
  - You want to keep all features but reduce their impact.
- The Ridge cost function modifies the standard least squares as:

$$\text{Loss} = \text{MSE} + \lambda \sum_{j=1}^{p} \beta_j^2$$

where,

λ = Regularization strength (hyperparameter) [λ=0 to ∞]

βj = Model coefficients [For simple LR, βj=1 coefficient, and for Multiple LR, βj = n number of coefficients]

b) Lasso Regression

- **Lasso Regression** is a type of linear regression that uses **L1 regularization** to shrink some coefficients to exactly **zero**, effectively performing **feature selection**.
- For example, if we're predicting house prices based on features like location, square footage and number of bedrooms.
  - Lasso Regression can identify most important features.
  - It might determine that location and square footage are the key factors influencing price while others has less impact.
  - By making coefficient for the bedroom feature to zero it simplifies the model and improves its accuracy.

- It adds the absolute value of the sum of coefficients as a penalty term to the loss function.

$$\text{Loss} = \text{MSE} + \lambda \sum_{j=1}^{p} |\beta_j|$$

Where:
$\lambda\sum|\beta_j|$: **L1 penalty** (sum of absolute values of coefficients)
λ: regularization strength (tuned via cross-validation)

c) Bias-variance Tradeoff

Bias (Underfitting Problem)

- Bias is calculated as the difference between average prediction and actual value.
- Bias is the error introduced because the model is too simple and does not capture the true patterns in the data.
- In machine learning, bias (systematic error) occurs when a model makes incorrect assumptions about data.
- A model with high bias does not match well training data as well as test data. It leads to high errors in training and test data.
- **High Bias:** The model makes strong assumptions (e.g., assuming data is linear when it's actually complex).
- **Low Bias:** The model is flexible enough to learn the true trends.
- Example:
    - **Model:** Predicting house prices using just the size of the house (ignoring location, age, etc.).
    - **Problem:** The model is oversimplified and misses key factors, leading to high errors even on training data.
    - **Effect:** Underfitting → Poor performance on both training and test data

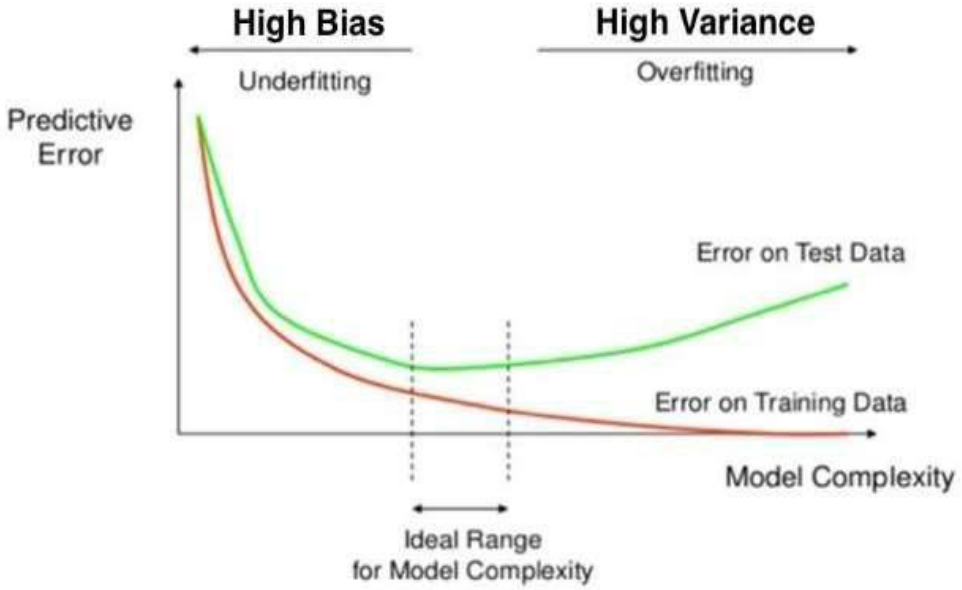
Variance (Overfitting Problem)

- Variance is the error introduced because the model is too complex and learns noise instead of the real pattern.
- **High Variance**: The model memorizes training data but fails on new data.
- **Low Variance:** The model generalizes well to unseen data.
- Example:
    - **Model:** A polynomial regression with a very high degree that fits every tiny fluctuation in training data.

- **Problem:** The model performs well on training data but fails on test data because it learned noise.
- **Effect:** Overfitting → Good on training data, bad on test data.

Bias-variance tradeoff

- The bias-variance tradeoff is finding a balance between the error introduced by bias and the error introduced by variance.
- With increased model complexity, the bias will decrease, but the variance will increase. However, when we decrease the model complexity, the bias will increase, and the variance will decrease.
- So, we need a balance between bias and variance so total prediction error is minimized.
- A machine-learning model will not perform well on new, unseen data if it has a high bias or variance in training.
- A good model should not have either high bias or variance.
- We cannot reduce both bias and variance at the same time. When bias reduces, variance will increase.
- So, we need to find an optimal bias and variance such that the prediction error is minimized.
- How to find the optimal point for Bias- Variance?
  - Regularization
  - Cross validation (Unit 5)
  - Grid search or random search (Unit 5)



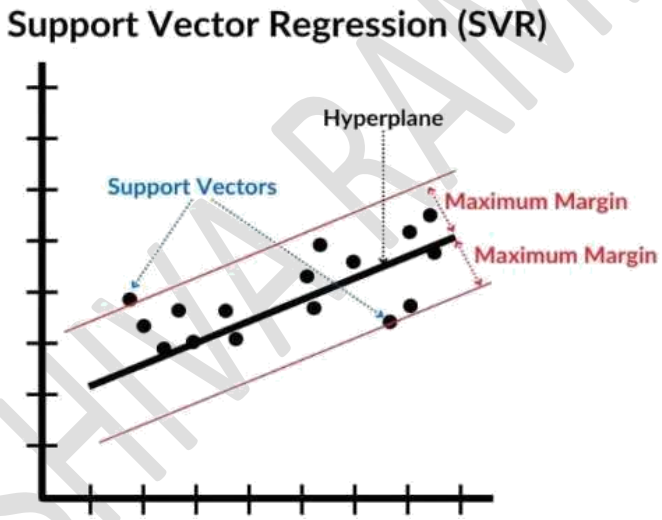
Bias Variance Tradeoff: Models with low complexity have high Bias and low Variance, and models with high complexity have low Bias and high Variance (Source: Al-Behadili et al. Rule pruning techniques in the ant-miner classification algorithm and its variants: A review)

The figure shows the relationship between model complexity and error.

- As the complexity of the model increases, the bias decreases and the variance increases.
- At the same time, total error of the model first decreases and then increases.
- The optimal model complexity is the point at which the total error is minimum.

2.3.4 Support Vector Regression

- Support Vector Regression (SVR) is a machine learning technique for regression tasks.
- It extends the principles of Support Vector Machines (SVM) from classification to regression.
- In SVR, the goal is to predict continuous target variables rather than discrete classes.
- SVR works by finding a hyperplane that best fits the training data while also maintaining a maximum margin, where the margin is defined as the distance between the hyperplane and the support vectors.



Note: [Will be covered in Support Vector Machine which is the classification task]

2.3 Classification

Classification is a supervised learning technique where the model predicts discrete class labels (categories) instead of continuous values.

Example

Problem	Input	Output (Class)
Email spam detection	Email text	"Spam" or "Not Spam/Ham"
Tumor classification	Medical scan data	"Benign" or "Malignant"
Handwritten digit recognition	Image of digit	0 through 9 (10 classes)
Sentiment analysis	Movie review	"Positive", "Negative", etc.

2.3.1 Logistic Regression

- Logistic regression is a supervised machine learning algorithm used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not.
- It is used when the **output (dependent variable)** is **categorical**, typically **binary** (e.g., 0 or 1, Yes or No, Spam or Not Spam).
- Example:
    - Suppose we want to **predict whether a student passes** based on their study hours.
        - Input: x = hours studied
        - Output: y = 1 (pass) or y = 0 (fail)

How It Works?

Step 1: *Linear Combination*

First, it calculates a weighted sum of input features (just like linear regression):

$$z = b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n$$

where:

b0: Bias term

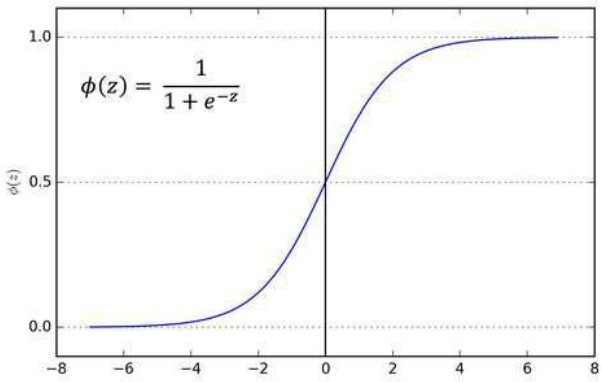
b1,…,bn: Coefficients (learned from data)

x1,…,xn: Input features (e.g., age, salary)

Step 2 : *Sigmoid Function* (Logistic Function)

The output z is passed through a sigmoid function to squash it into [0, 1]. To convert this into a probability between 0 and 1, it uses:

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



Step 3: Decision Rule

- If P(y=1) ≥ 0.5 (default threshold value can be adjusted according to our task), predict class 1.
- If P(y=1) < 0.5, predict class 0.

Example: Spam Detection

Email Text (Features)	Is Spam? (Y)
"Win money now!"	1 (Yes)
"Meeting tomorrow"	0 (No)

Model Prediction:

For a new email "Free lottery!",

The model's output might be: P(Spam)=0.9 >= 0.5 ⇒ Predict "Spam"

a) Binary Classification

- A classification task where the output can be **one of two classes** (i.e., two possible outcomes).
- In binary classification, the goal is to sort the data into **two distinct categories**.
- Usually output is encoded as **0 or 1**

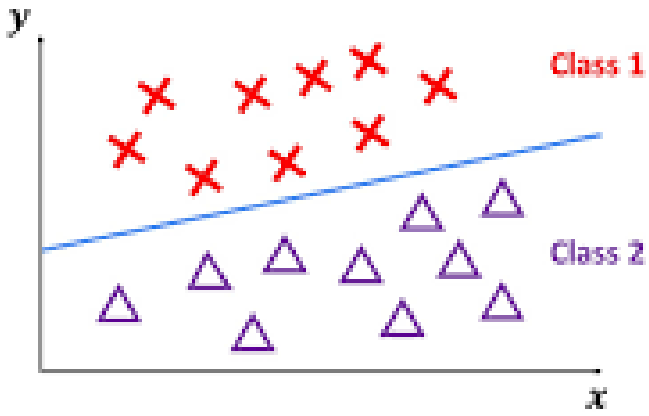


Figure: Binary Classification

- Logistic regression (with sigmoid) is commonly used
- some of the most common algorithms used by binary classification are;
    - **Logistic Regression**
    - **k-Nearest Neighbors**
    - **Decision Trees**
    - **Support Vector Machine**
    - **Naive Bayes**
- Examples:
    - Spam vs. Not Spam
    - Pass vs. Fail
    - Disease vs. No Disease
    - Click vs. No Click

b) Multi-class classification

A classification task where the output can be **one of three or more classes**.

Output class is one of **n categories**, e.g., {0, 1, 2, ..., n−1}

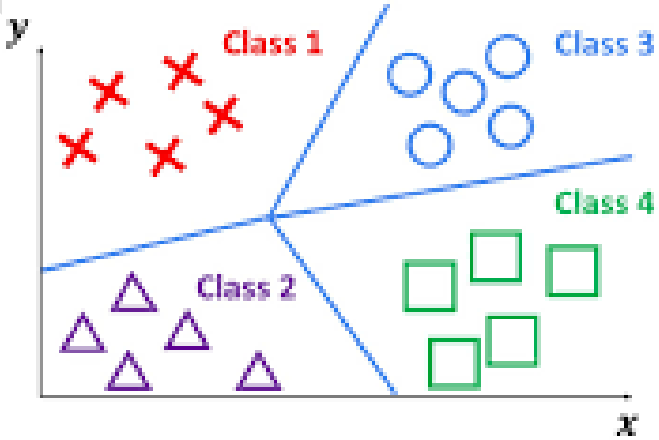


Figure: Multiclass Classification

Examples:

- Classifying emails into **Primary**, **Social**, or **Promotions**
- Identifying animal type: **Dog**, **Cat**, **Horse**
- Predicting exam grade: **A**, **B**, **C**, **D**, **F**

Common approaches:

i) **One-vs-Rest (OvR)**:

- Train one binary classifier per class vs. all others.
- Pick the class with the highest score.
- Python code snippet:

```
# One-vs-Rest (default in sklearn for multi-class)
model = LogisticRegression(multi_class='ovr', solver='liblinear')
model.fit(X_train, y_train)
```

ii) **Softmax Regression**:

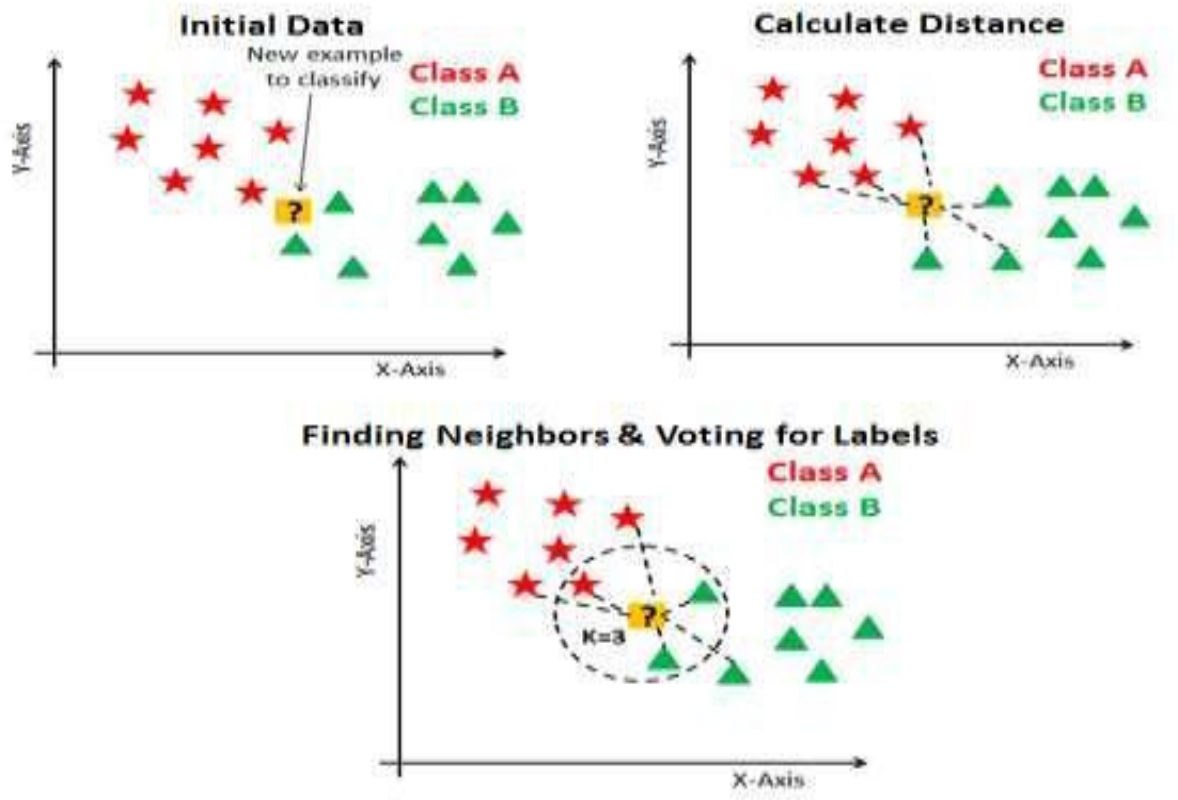
- Also known as Multinomial Logistic Regression
- Extension of logistic regression using the **softmax function** to output class probabilities that sum to 1.
- Directly handles multiple classes
- Uses softmax function to output probability for each class. Generalizes sigmoid to K classes.
- Python Implementation:

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
# Load dataset (10 classes: digits 0- 9)
digits = load_digits()
X, y = digits.data, digits.target
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train Softmax model (multi_class='multinomial')
```

```
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
model.fit(X_train, y_train)
# Predict
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test) # Probabilities for all classes
# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

2.3.2 K-Nearest Neighbors (KNN)

- K-Nearest Neighbors (KNN) is a simple, instance-based machine learning algorithm used for classification (and regression).
- It classifies a data point by checking the majority class of its "k" closest neighbors in the training data.
- KNN is a non-parametric and lazy learning algorithm.
    - Non-parametric means there is no assumption for underlying data distribution
    - The lazy algorithm means it does not need any training phase for model generation. All training data used in the testing phase.
- This makes training faster and the testing phase slower and costlier.
- K- NN algorithm is based on the principle that, **"the similar things or objects exist closer to each other."**



Algorithm:

- **Step-1:** Select the hyper-parameter K (Eg., 3,5,7, …)
    - **Small k** → Sensitive to noise (overfitting).
    - **Large k** → Smoother decision boundaries (may underfit).
    - So, one method to choose k is use cross-validation (Will be discussed in unit
- **Step-2:** Calculate the distance of the new data (test instance) with all the training data using Euclidean distance, Manhattan distance, Hamming distance, etc.

- Euclidean distance is given by:

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- **Step 3:** Sort distances in ascending order. And Select K nearest neighbors.
- **Step-4:** Assign the new data points to that *category (i.e.. Target class)* for which the number of the neighbor is maximum (i.e. majority voting)

Numerical Example 1:

Consider the following dataset. Given a new sample with X1 = 3, X2 = 7, classify it using KNN (k=3). What class will it belong to?

X1	X2	Y (Classification)
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

Solution:

Step 1: Compute Euclidean Distance from the new sample to each training sample

$$\text{Distance} = \sqrt{(X1_{new} - X1_i)^2 + (X2_{new} - X2_i)^2}$$

Let the new sample be **(3, 7)**.

Step 2: Tabulate distances

Training Sample	X_1	X_2	Class	Distance to (3,7)
A	7	7	Bad	$\sqrt{[(3-7)^2 + (7-7)^2]} = \sqrt{16} =$ **4.00**
B	7	4	Bad	$\sqrt{[(3-7)^2 + (7-4)^2]} = \sqrt{25} =$ **5.00**
C	3	4	Good	$\sqrt{[(3-3)^2 + (7-4)^2]} = \sqrt{9} =$ **3.00**
D	1	4	Good	$\sqrt{[(3-1)^2 + (7-4)^2]} = \sqrt{13} \approx$ **3.61**

Step 3: Pick 3 nearest Neighbors

Nearest Neighbor	Distance	Class
C (3, 4)	3.00	Good
D (1, 4)	3.61	Good
A (7, 7)	4.00	Bad

Step 4: Classify based on Majority Vote

- **Good**: 2 votes
- **Bad**: 1 vote

Final Classification:

The new sample (3, 7) will be classified as: Good

Numerical Example 2:

Assume the following training set with two classes, Food and Beverage. Apply KNN with K=3 to classify the new document "turkey soda".

Food : "turkey stuffing"
Food : "buffalo wings"
Beverage : "cream soda"
Beverage : "orange soda"

Solution:

Here, the vocabulary is: **Buffalo, Cream, Orange, Soda, Stuffing, Turkey, Wings**

Applying one-hot encoding and calculating the Euclidean distance of each point with the test data, we get:

	Buffalo	Cream	Orange	Soda	Stuffing	Turkey	Wings	Category	Euclidian Distance	Rank with 3NN
D1: "Turkey Stuffing"	0	0	0	0	1	1	0	Food	$\sqrt{(0-1)^2+(1-0)^2} = \sqrt{2} = 1.414$	1
D2: "Buffalo Wings"	1	0	0	0	0	0	1	Food	$\sqrt{(1-0)^2+(0-1)^2+(0-1)^2+(1-0)^2} = \sqrt{4} = 2$	
D3: "Cream Soda"	0	1	0	1	0	0	0	Beverage	$\sqrt{(1-0)^2+(0-1)^2} = \sqrt{2} = 1.414$	2
D4: " Orange Soda"	0	0	1	1	0	0	0	Beverage	$\sqrt{(1-0)^2+(0-1)^2} = \sqrt{2} = 1.414$	3
Q: "Turkey Soda"	0	0	0	1	0	1	0			

Hence, the classification for "Turkey Soda" by majority voting is : **Beverage**

Assignment:

1. Apply K Nearest Neighbor classifier to predict the diabetic patient with the given features (BMI, Age) of Training dataset. The target label is Sugar. The test example is: BMI= 43.6 and Age= 40, Sugar= ? . Assume K=3

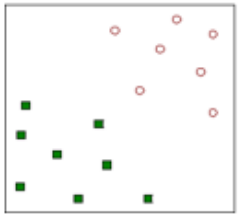
BMI	Age	Sugar
33.6	50	1
26.6	30	0
23.4	40	0
43.1	67	0
35.3	23	1
35.9	67	1
36.7	45	1
25.7	46	0
23.3	29	0
31	56	1

2. "Restaurant A" sells burger with optional flavors: Pepper, Ginger and Chilly. Below small dataset shows the Like and Unlike by customer. Use KNN classifier (k=3) to classify (Like of Unlike) for a burger with below flavor: Pepper: Yes, Ginger: No, Chilly: Yes.

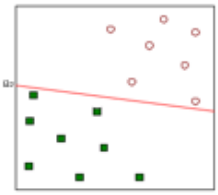
Pepper	Ginger	Chilly	Remarks
Yes	Yes	Yes	Unlike
Yes	No	No	Like
No	Yes	Yes	Unlike
No	Yes	No	Like
Yes	No	No	Like

2.3.3 Support Vector Machine (SVM)

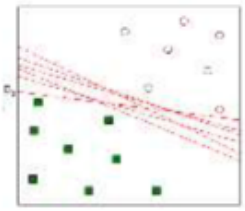
Problem: Find the line/plane that best separates the data.



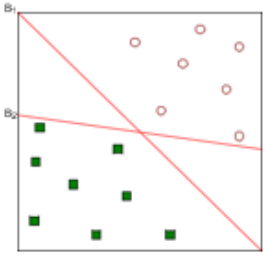
Find the decision boundary (line/hyperplane) that will separate the data.



one possible solutions

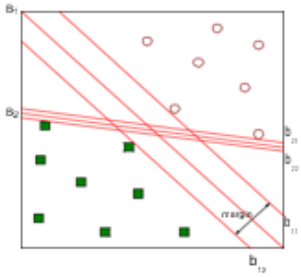


Other possible solutions



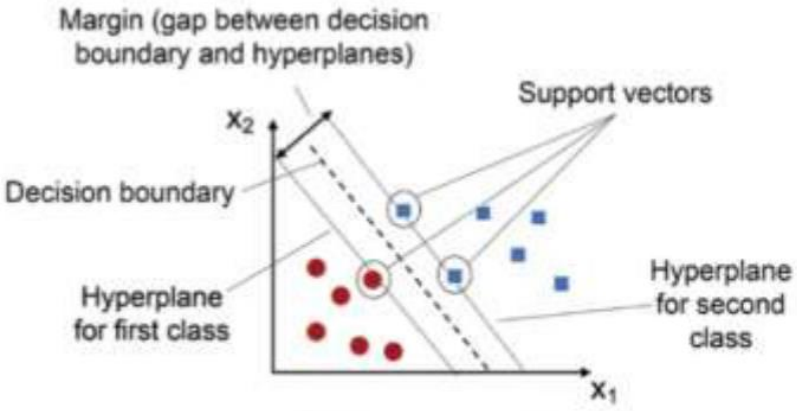
Which one is better? B1 or B2?

How do you define better?



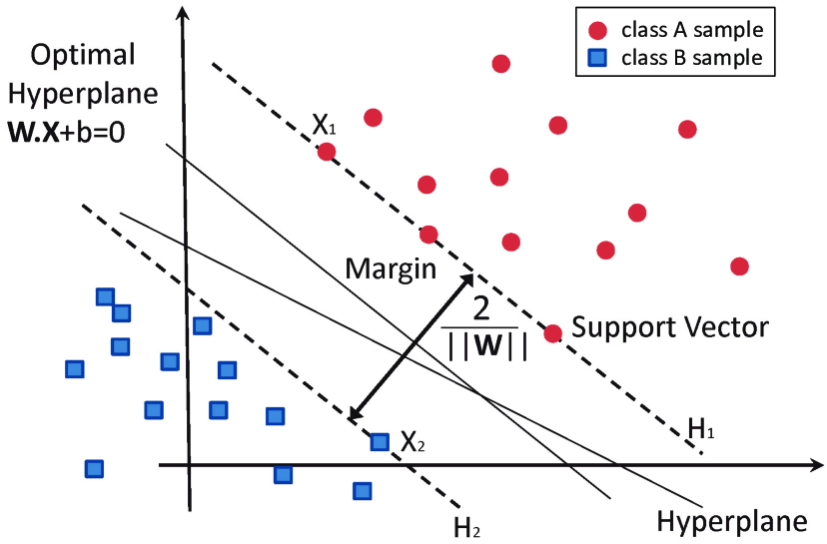
Find hyperplane maximizes the margin => B1 is better than B2

- Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression tasks.
- It tries to find the best line (in 2D) or hyperplane (in higher dimensions) that separates the data points of different classes with the maximum margin.
- The **margin** is the distance between the decision boundary and the closest data points from each class.
- These closest points are called **support vectors.**



- **Hyperplane:** A decision boundary separating different classes in feature space, represented by the equation wx + b = 0 in linear classification.
- **Support Vectors:** The closest data points to the hyperplane, crucial for determining the hyperplane and margin in SVM.
- **Margin:** Margin is the total distance between the two support vector hyperplanes (H1 and H2), on either side of the main hyperplane. SVM aims to maximize this margin for better classification performance.
- **Hard Margin:** A maximum-margin hyperplane that perfectly separates the data without misclassifications.
- **Soft Margin:** Allows some misclassifications by introducing slack variables, balancing margin maximization and misclassification penalties when data is not perfectly separable.
- **Kernel:** A function that maps data to a higher-dimensional space, enabling SVM to handle non-linearly separable data.
-

a) Hyperplane



i) Main Optimal Hyperplane (Decision Boundary):

This is the line (or plane) that separates the two classes.

$$w^T x + b = 0$$

Where:

- w = weight vector (normal to the hyperplane)

- x = input feature vector
- b = bias (offset from origin)

ii) Margin Boundaries (H1 and H2):

SVM constructs **two parallel hyperplanes** at equal distance from the main decision boundary. These pass through the **support vectors**.

H1 (for Class +1):

$$w^T x + b = +1$$

H2 (for Class –1):

$$w^T x + b = -1$$

The region between **H1 and H2** is called the **margin**.

iii) Margin Width:

The margin is the **total distance between the two support vector lines (H1 and H2).**

Margin equals two divided by the norm of **w**.

The distance between H1 and H2 is:

$$\text{Margin} = \frac{2}{\|w\|}$$

Where: **‖w‖** is the magnitude (or length) of the vector **w**.

‖w‖ represent the **Euclidean norm** (or L2 norm) of the vector **w**.

SVM tries to **maximize this margin** by minimizing $\|w\|^2$ under the constraint:

$$y_i(w^T x_i + b) \geq 1 \quad \text{for all } i$$

Where $y_i \in \{-1, +1\}$ are the class labels.

Decision in SVM:

$$\hat{y} = \begin{cases} 1 & : w^T x + b \geq 0 \\ 0 & : w^T x + b < 0 \end{cases}$$

Where:

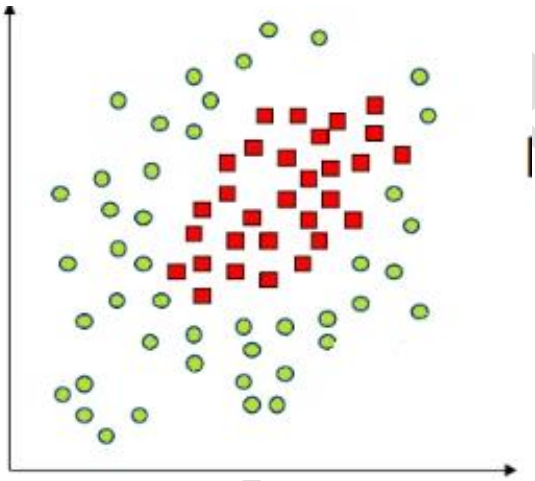
- y^: The predicted output (either **1** or **0**).
- $w^T x$: The **dot product** of the weight vector www and input vector xxx.
- b: The **bias** term.
- $w^T x + b$: The **decision function** (a linear equation representing a hyperplane).

If $w^T x + b \geq 0$ then, the point lies on or above the decision boundary. Hence, the model predicts the data point to be in class 1.

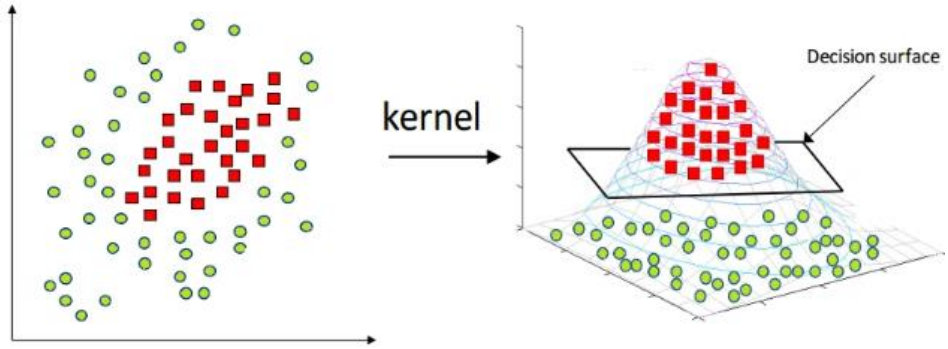
If $w^T x + b < 0$ then, the point lies below the decision boundary. Hence, the model predicts the data point to be in class 0.

b) Kernels and its Types: Linear, Polynomial, Radial Basis Function (RBF)

- Kernels in Support Vector Machines (SVM) are functions that transform input data into a higher-dimensional space to make it easier to classify data that is not linearly separable.
- A kernel function computes the similarity between two data points without explicitly transforming them to high-dimensional space (this trick is called the kernel trick).
- The most interesting feature of SVM is that it can even work with a non-linear dataset and for this, we use "Kernel Trick" which makes it easier to classifies the points.
- *They are used to solve a non-linear problem by using a linear classifier.*
- Suppose we have a dataset like this:



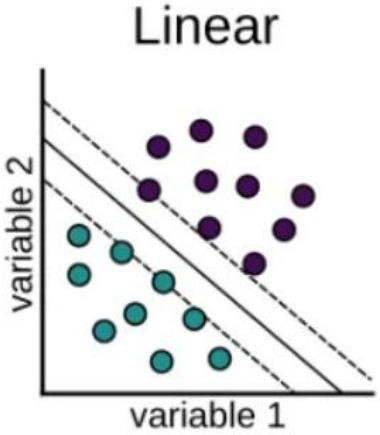
- Here we see we cannot draw a single line or say hyperplane which can classify the points correctly.
- So what we do is try converting this lower dimension space to a higher dimension space using some quadratic functions which will allow us to find a decision boundary that clearly divides the data points.
- These functions which help us do this are called Kernels and which kernel to use is purely determined by hyperparameter tuning.



Types of SVM Kernels:

i) Linear Kernel

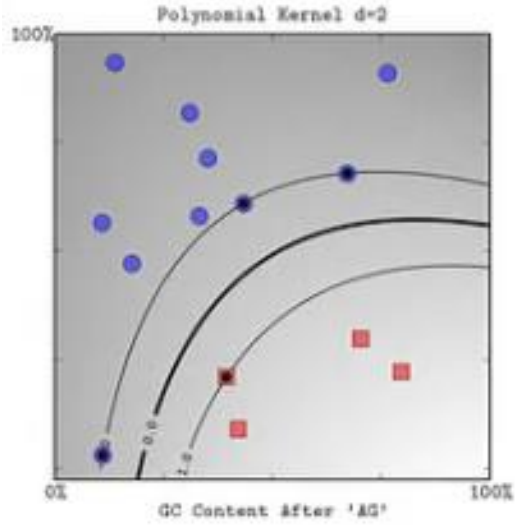
- A **linear kernel** is the simplest form of kernel used in SVM.
- It is suitable when the data is linearly separable meaning that a straight line (or hyperplane in higher dimensions) can effectively separate the classes.



- Let us say that we have two vectors with name x and y, then the linear kernel is defined by the dot product of these two vectors:
  $K(x,y) = x.y$
- It is used for text classification problems such as spam detection

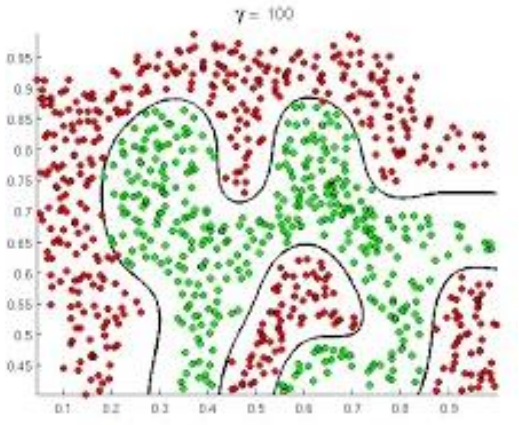
ii) Polynomial Kernel

- The **polynomial kernel** allows SVM to model more complex relationships by introducing polynomial terms.
- It is useful when the data is not linearly separable but still follows a pattern.
- It is used in Complex problems like image recognition where relationships between features can be non-linear.
- Let us say that we have two vectors with name x and y, then the polynomial kernel is defined by the dot product of these two vectors:
  $K(x,y) = (x.y + c)^d$
  where c is a constant and d is the polynomial degree.



iii) Radial Basis Function (RBF)

- The **RBF kernel** is the most widely used kernel in SVM.
- It maps the data into an infinite-dimensional space making it highly effective for complex classification problems.
- We use RBF kernel When the decision boundary is highly non-linear and we have no prior knowledge about the data's structure is available.



- The formula of RBF kernel is:

$$K(x, y) = \exp\left(-\gamma \, ||x - y||^2\right)$$

where

K(x, y) is the similarity score between points x and y

γ is a hyperparameter that controls the "spread" of the kernel.

$||x - y||^2$ is the squared Euclidean distance between x and y.

c) SVM for Linear and Non-linear classification

SVM can handle both linearly separable and non-linearly separable data using different kernel

Linear SVM:

- **Linear SVM** is a type of Support Vector Machine used for **linearly separable data**, where a **straight line (in 2D)** or **hyperplane (in higher dimensions)** can separate the classes.
- When data can be separated by a straight line (or hyperplane) then Linear Classification is used. In this case no kernel transformation is needed.
- Characteristics:
    - Uses a **linear decision boundary**.
    - Tries to find the **maximum-margin hyperplane** that best separates the two classes.
    - Fast and effective when data is linearly separable or approximately so.

Non-Linear SVM:

- **Non-linear SVM** is used when the data is **not linearly separable** — i.e., a straight line cannot separate the classes.
- When data cannot be separated by a straight line then Non-Linear Classification is used.
- In this case kernel trick is used. Common non-linear kernels are Polynomial kernel, RBF (Radial Basis Function / Gaussian) kernel.
- **Characteristics:**
    - Uses the **kernel trick** to transform input data into a **higher-dimensional space**.
    - Finds a **linear decision boundary in that higher-dimensional space**, which corresponds to a **non-linear boundary** in the original space.
    - Can handle complex patterns and shapes in the data.
- **Common Kernels:**
    - **Polynomial Kernel**
    - **Radial Basis Function (RBF) Kernel**
    - **Sigmoid Kernel**

2.3.4 Decision Trees

- A decision tree is a tree-structured model used for classification and regression tasks.
- It splits the dataset into subsets based on feature values, using decision rules at each node.
- It consists of:
    - Root Node: Represents the entire dataset.
    - Internal Nodes: Represent tests on features.
    - Leaf Nodes: Represent class labels or output values.

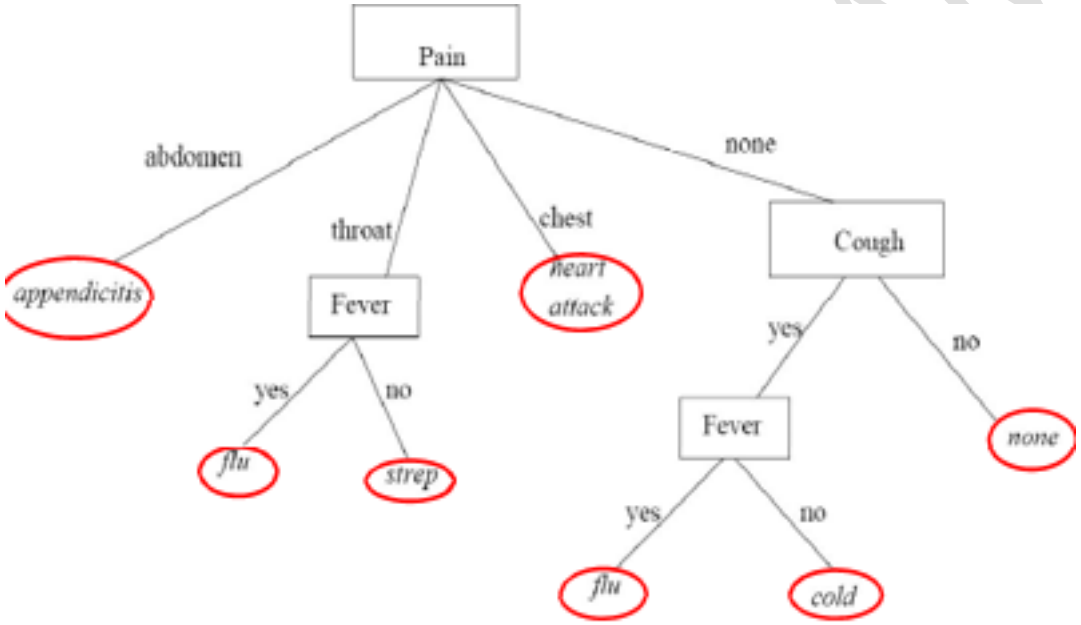


Figure: Decision tree

How does the Decision Tree algorithm Work?

- In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree.
- This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.
- For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree.

a) Decision Tree Construction

Building a **Decision Tree** involves recursively splitting the dataset based on feature values to create a structure that predicts the output labels with high accuracy. The goal is to **partition data into subsets** that are as **pure** as possible (i.e., containing mostly one class).

Steps in Decision Tree Construction

1. **Select the Best Attribute for Splitting**

   At each node, choose the **attribute** that best splits the dataset into subsets that are **homogeneous** (i.e., mostly one class). There are three popular techniques for ASM, which are:

   - **Information Gain** (used in ID3)
   - Gain Ratio (used in C4.5)
   - Gini Index (used in CART)

2. Split the Dataset Based on the Selected Attribute

   Create branches for each value (or range) of the selected attribute.

   - For categorical attributes: one branch per value.
   - For numerical attributes: binary split using a threshold (e.g., Age < 30?).

3. Repeat for Each Branch

   For each subset of the data:

   - If it is **pure** (contains only one class): make it a **leaf node**.
   - If not pure: repeat the **splitting process**.

4. Stopping Criteria

   Stop when:

   - All samples belong to the same class.
   - No more features to split on.
   - Maximum tree **depth** is reached.
   - **Minimum number of samples** in a node is reached.
   - **Information gain** is too small (no further improvement).

Decision Tree construction using Information Gain:

- Information Gain (IG) is a metric used in decision trees to measure how much a split at a node reduces the uncertainty (or entropy) in the dataset.
- Information gain is calculated as:

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

Where:
T= Target class
X= particular attribute

- Entropy is a measure of uncertainty, randomness, or impurity in a set of data.
  - If the data is all of one type (like all "Yes" or all "No"), entropy is low because there's no uncertainty.
  - If the data is a mix of types (like half "Yes" and half "No"), entropy is high because it's harder to predict the outcome.
  - Entropy is calculate as:

$$Entropy\,(D) \equiv \sum_{i=1} - p_i \log_2(p_i)$$

- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first.

Numerical:

Compute the Entropy of Target class (PlayGolf) and Entropy of (PlayGolf, Outlook) and also the Information Gain of (PlayGolf, Outtlook) from below dataset:

Predictors				Target
Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Solution:

i) Entropy of PlayGolf is given by:

$$Entropy\,(D) \equiv \sum_{i=1} - p_i \log_2(p_i)$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

ii) Entropy of (PlayGolf, Outlook) is given by:

$$E(T,X) = \sum_{c \in X} P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)
= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971
= 0.693

iii) Information Gain of (PlayGolf, Outlook) is given by:

$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)
= 0.940 – 0.693 = 0.247

Numerical 2:

Construct a Decision Tree using Information Gain for below dataset;

Day	Temp	Humidity	Wind	Play
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D3	Cool	High	Weak	Yes
D4	Cool	High	Strong	Yes
D5	Hot	Normal	Weak	Yes
D6	Cool	Normal	Weak	Yes
D7	Cool	Normal	Strong	No

Solution:

Step 1:

Computation of entropy:

$$\text{Entropy (Play)} = \text{Entropy}(4,3)$$
$$= -\left(\frac{4}{7}\log_2\frac{4}{7}\right) - \left(\frac{3}{7}\log_2\frac{3}{7}\right)$$
$$= -(-0.461) - (-0.523)$$
$$= 0.985$$

Now,

$$\text{Entropy (Play, Temp)} = P(\text{Hot}) * E(1,2) + P(\text{Cool}) * E(3,1)$$
$$= \frac{3}{7} * \left[-\left(\frac{1}{3}\log_2\frac{1}{3}\right) - \left(\frac{2}{3}\log_2\frac{2}{3}\right)\right] + \frac{4}{7}\left[-\left(\frac{3}{4}\log_2\frac{3}{4}\right) - \left(\frac{1}{4}\log_2\frac{1}{4}\right)\right]$$
$$= 0.3935 + 0.4636$$
$$= 0.8570$$

$$\text{Entropy (Play, Humidity)} = P(\text{High}) * E(2,2) + P(\text{Normal}) * E(2,1)$$
$$= \frac{4}{7} * \left[-\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4}\right] + \frac{3}{7}\left[-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}\right]$$
$$= 0.5714 + 0.3935$$
$$= 0.9649$$

$$\text{Entropy (Play, Wind)} = P(\text{Strong}) * E(1,2) + P(\text{Weak}) * E(3,1)$$
$$= \frac{3}{7} * \left[-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}\right] + \frac{4}{7} * \left[-\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4}\right]$$
$$= 0.3935 + 0.4636$$
$$= 0.8571$$

Computation of information gain:

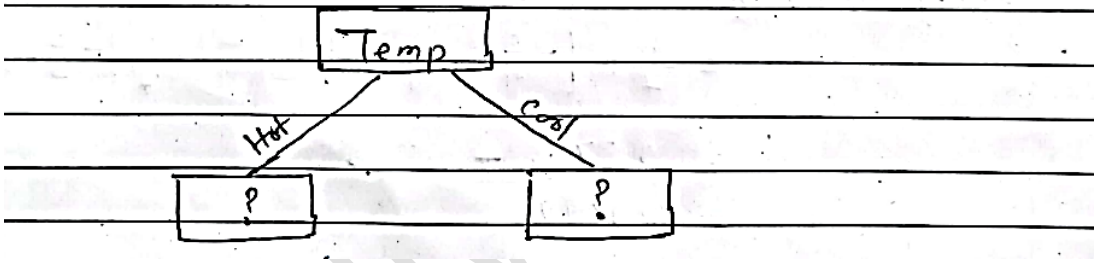
Gain (Play, Temp) = Entropy (Play) − Entropy (Play, Temp)
= 0.985 − 0.8570
= 0.128

Gain (Play, Humidity) = Entropy (Play) − Entropy (Play, Humidity)
= 0.985 − 0.9649
= 0.0201

Gain (Play, Wind) = Entropy (Play) − Entropy (Play, Wind)
= 0.985 − 0.8571
= 0.1279

Construction of partial Decision Tree with root node:

Since, the Gain (Play, Temp) has the highest information gain, we place this attribute (Temp) at the root node.



Step 2:

Now again we continue to compute the entropy of each attribute (*Humidity*, *wind*) with reference to the data value of *Temp* attribute. The data values are *Hot* and *Cool*.

The new table with reference to *Hot* is:

Day	Humidity	Wind	Play
D1	High	Weak	No
D2	High	Strong	No
D5	Normal	Weak	Yes

Now we compute Entropy and Information gain as in Step 1.

$$\text{Entropy}(\text{Play}_{hot}) = \text{Entropy}(1,2)$$
$$= -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}$$
$$= 0.9183$$

$$\text{Entropy}(\text{Play}_{hot}, \text{humidity}) = P(\text{high}) * E(0,2) + P(\text{Normal}) * E(1,0)$$
$$= \frac{2}{3}\left[-\frac{0}{2}\log_2\frac{0}{2} - \frac{2}{2}\log_2\frac{2}{2}\right] + \frac{1}{3}\left[-\frac{1}{1}\log_2\frac{1}{1} - \frac{0}{1}\log_2\frac{0}{1}\right]$$
$$= 0 + 0$$
$$= 0$$

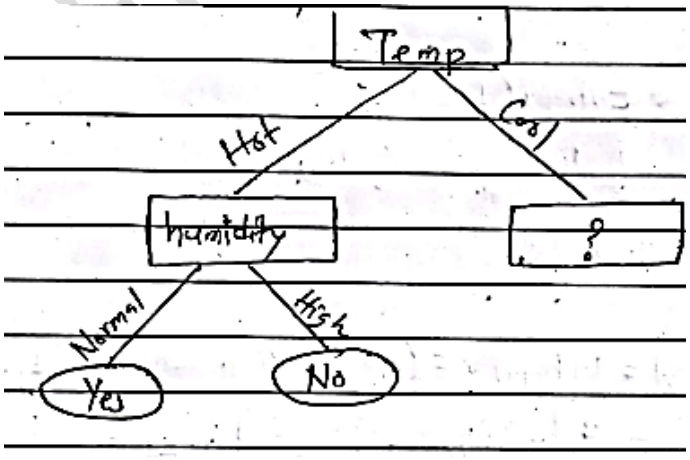
$$\text{Entropy}(\text{Play}_{hot}, \text{Wind}) = P(\text{Strong}) * E(0,1) + P(\text{Weak}) * E(1,1)$$
$$= \frac{1}{3}\left[-\frac{0}{1}\log_2\frac{0}{1} - \frac{1}{1}\log_2\frac{1}{1}\right] + \frac{2}{3}\left[-\frac{1}{1}\log_2\frac{1}{1} - \frac{1}{1}\log_2\frac{1}{1}\right]$$
$$= 0 + 0$$
$$= 0$$

Now,

$$\text{Gain}(\text{Play}_{hot}, \text{humidity}) = \text{Entropy}(\text{Play}_{hot}) - \text{Entropy}(\text{Play}_{hot}, \text{humidity})$$
$$= 0.9183 - 0$$
$$= 0.9183$$

$$\text{Gain}(\text{Play}_{hot}, \text{wind}) = \text{Entropy}(\text{Play}_{hot}) - \text{Entropy}(\text{Play}_{hot}, \text{Wind})$$
$$= 0.9183 - 0$$
$$= 0.9183$$

Since the information gain of humidity is **high** we take it as the parent node. Our partial decision tree is:

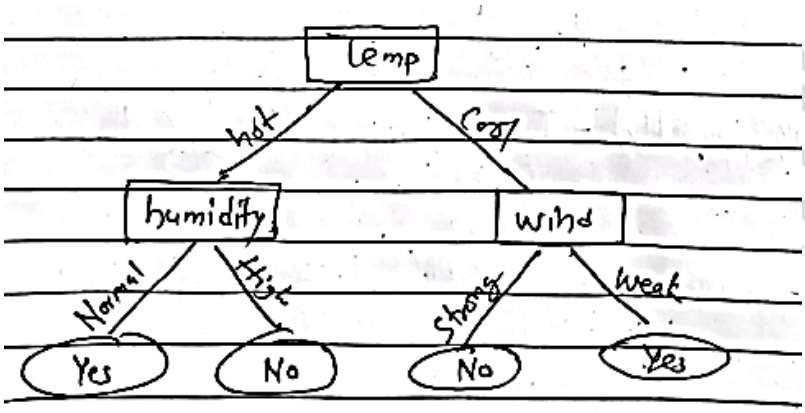


Step 3:

Similar to step 2, we need to compute the entropy and information gain for ($Play_{Cool}$) with reference to *wind* and *humidity* attribute with below table

Day	Humidity	Wind	Play
D3	High	Weak	Yes
D4	High	Strong	Yes
D6	Normal	Weak	Yes
D7	Normal	Strong	No

But, since the remaining is *wind* attribute only. So, we can directly draw the final decision tree as below:



Assignment:

Construct a Decision Tree using Information Gain for below dataset:

ID	StudyHours	Attendance	Result
1	High	Good	Pass
2	Low	Poor	Fail
3	Medium	Good	Pass
4	Low	Good	Fail
5	High	Poor	Pass

Decision Tree construction using Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART (Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

Numerical Example 3:

Construct a decision tree using Gini Index for the below playGolf dataset.

Day	Temp	Humidity	Wind	Play
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D3	Cool	High	Weak	Yes
D4	Cool	High	Strong	Yes
D5	Hot	Normal	Weak	Yes
D6	Cool	Normal	Weak	Yes
D7	Cool	Normal	Strong	No

Solution:

Step 1: Gini Index of the Entire Dataset

Class counts:

- **Yes** = 4
- **No** = 3

$$Gini(S) = 1 - \left(\frac{4}{7}\right)^2 - \left(\frac{3}{7}\right)^2 = 1 - 0.327 - 0.184 = 0.489$$

Step 2: Gini for each attribute

Attribute: **Temp**

Temp	Count	Yes	No	Gini
Hot	3	1	2	$1-(1/3)^2-(2/3)^2 = 0.444$
Cool	4	3	1	$1-(3/4)^2-(1/4)^2 = 0.375$

$$Gini_{Temp} = \frac{3}{7} \cdot 0.444 + \frac{4}{7} \cdot 0.375 = 0.190 + 0.214 = 0.404$$

Attribute: **Humidity**

Humidity	Count	Yes	No	Gini
High	4	2	2	$1 - 0.5^2 - 0.5^2 = 0.5$
Normal	3	2	1	$1 - (2/3)^2 - (1/3)^2 = 0.444$

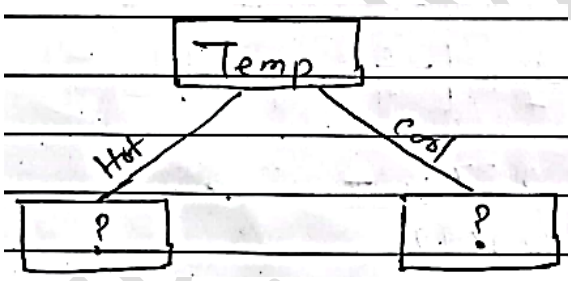
$$Gini_{Humidity} = \frac{4}{7} \cdot 0.5 + \frac{3}{7} \cdot 0.444 = 0.286 + 0.190 = 0.476$$

Attribute: **Wind**

Wind	Count	Yes	No	Gini
Weak	4	3	1	$1 - (3/4)^2 - (1/4)^2 = 0.375$
Strong	3	1	2	$1 - (1/3)^2 - (2/3)^2 = 0.444$

$$Gini_{Wind} = \frac{4}{7} \cdot 0.375 + \frac{3}{7} \cdot 0.444 = 0.214 + 0.190 = 0.404$$

Here, we have 0.404 as the lowest Gini Index, so we take Temp as the root node for our decision tree.



Step 2:

Now again we continue to build subtrees based on the values Hot and Cool of attribute Temp.

The new table with reference to *Hot* is:

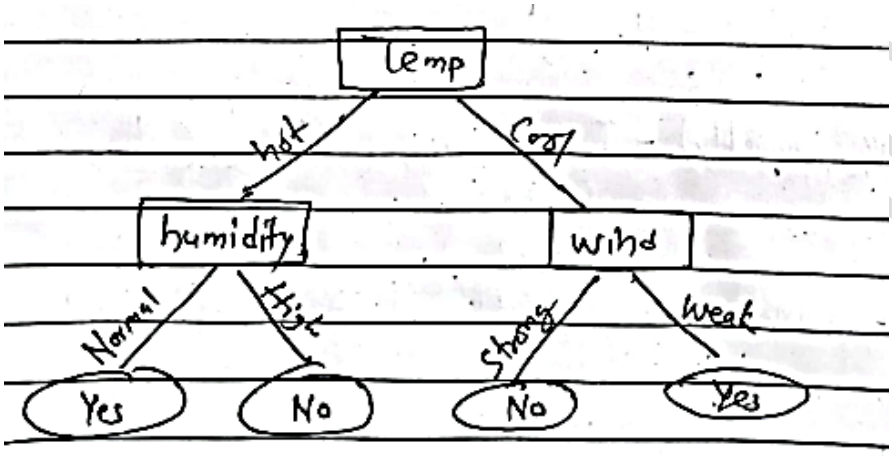
Day	Humidity	Wind	Play
D1	High	Weak	No
D2	High	Strong	No
D5	Normal	Weak	Yes

The new table with reference to *Cool* is:

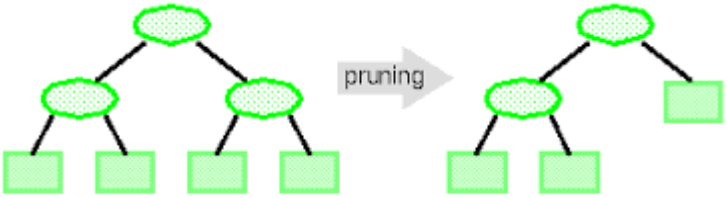
Day	Humidity	Wind	Play
D3	High	Weak	Yes
D4	High	Strong	Yes
D6	Normal	Weak	Yes
D7	Normal	Strong	No

We proceed further for each of these new tables to compute Gini Index for attributes Humidity and Wind within each branch.

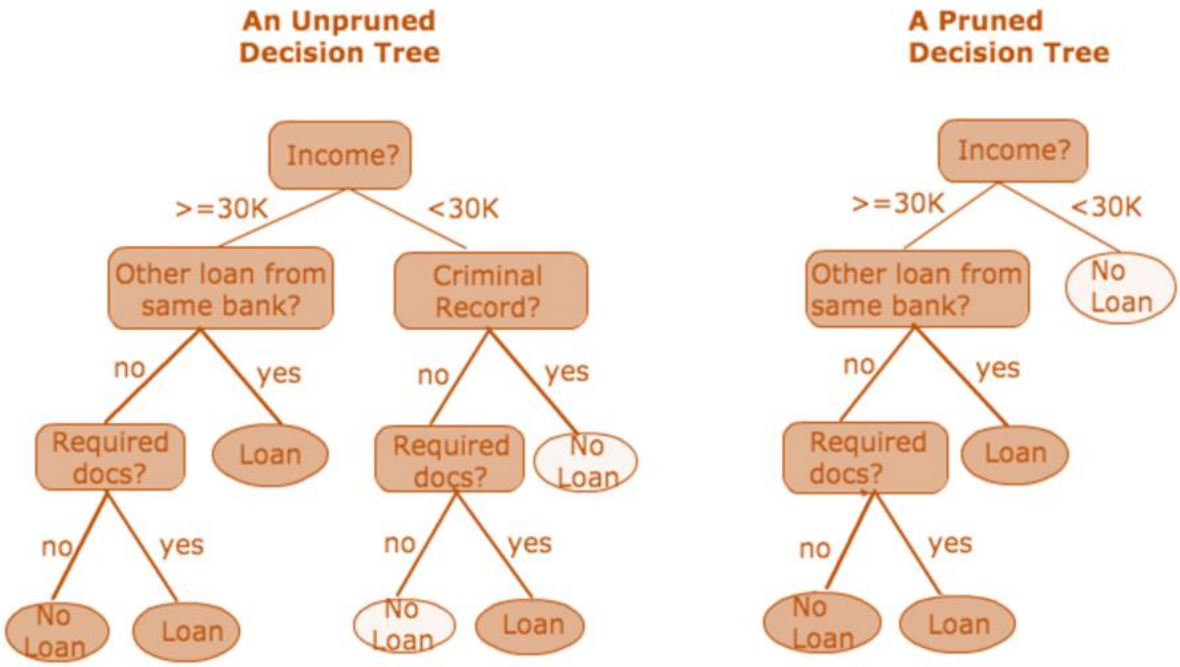
Continuing splitting based on the one with the lowest Gini index in each subset, we obtain the decision tree as:



b) Pruning of decision trees



Pruning in a decision tree is a technique used to reduce the size of the tree by removing sections that provide little to no power in classifying instances. It helps to prevent **overfitting**, which happens when the tree models the training data too closely, including noise, and thus performs poorly on unseen data.



Why prune?

- To **improve the generalization** of the model on new data.
- To make the tree **simpler and more interpretable**.
- To reduce the **complexity and size** of the tree.

Types of Pruning:

i) Pre-Pruning (Early Stopping)

- **Pre-pruning**, also known as **early stopping**, is a technique in decision tree learning where the tree construction is **halted early**, before it perfectly classifies the training data.

- Stop growing the tree before it becomes overly complex.
- While building the decision tree, pre-pruning **stops further splitting** of a node if a certain condition is met. Common stopping criteria include:
    - **Maximum depth** of the tree is reached.
    - **Minimum number of samples** in a node (e.g., don't split if fewer than 5 samples).
    - **Minimum information gain** or impurity reduction from a split.
    - **Validation performance** does not improve after a split.

ii) Post-Pruning (Cost Complexity Pruning)

- **Post-pruning**, also known as **"pruning after tree construction"**, is a method used in decision tree learning to **simplify** a fully grown tree **after** it has been built.
- It involves **removing branches** that do not contribute significantly to the model's predictive accuracy — especially those that were likely fitted to noise or specific quirks in the training data.

- Let the tree grow fully, then prune back branches that don't improve generalization.
- How post-pruning works:
    1) **Grow the full tree** using the training data.
    2) **Evaluate subtrees** (starting from the leaves upward).
    3) **Replace subtrees with leaf nodes** if doing so improves or does not significantly reduce accuracy on a **validation set** or based on a pruning metric (like error rate, cost-complexity, or statistical significance)

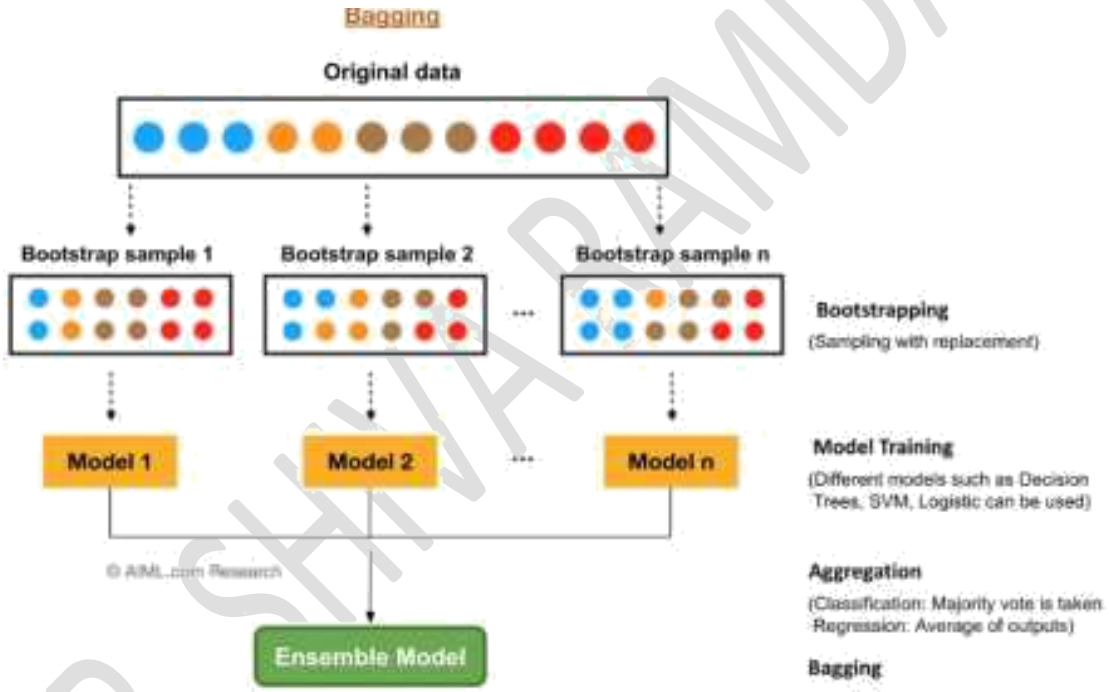
c) Ensemble Methods: Bagging, Random forests

Ensemble refers to a technique that combines multiple individual models to produce a more powerful predictive model**.** This is achieved by training multiple decision trees on different subsets of the data or with different random states, and then aggregating their predictions. Main goal of the Ensemble methods are to improve performance and reduce overfitting.

Two powerful ensemble techniques for are Bagging and Random Forests.

Bagging (Bootstrap Aggregating)

It create multiple subsets of the training data by randomly selecting samples with replacement. Each subset (bootstrap sample) has the same size as the original dataset. Then train a base model (e.g., decision tree) on each subset and combine predictions via averaging (regression) or majority voting (classification).



- Bootstrapping is a statistical method that involves repeatedly sampling data points from a dataset **with replacement** to create multiple datasets.
- Each of these datasets (called bootstrap samples) is used to estimate statistics or build machine learning models.
- The key feature of bootstrapping is that the same data point can appear multiple times in each bootstrap sample, while others may not appear at all.
- Bootstrapping is a key component of **Random Forest**, an ensemble learning method that combines multiple decision trees to improve prediction accuracy and robustness.
- It uses **Bagging (Bootstrap Aggregating)**, where bootstrapping creates multiple datasets to train each decision tree.

Random forest

Random Forest is an ensemble learning method that combines multiple decision trees to improve accuracy and reduce overfitting. A Random Forest builds many decision trees using:

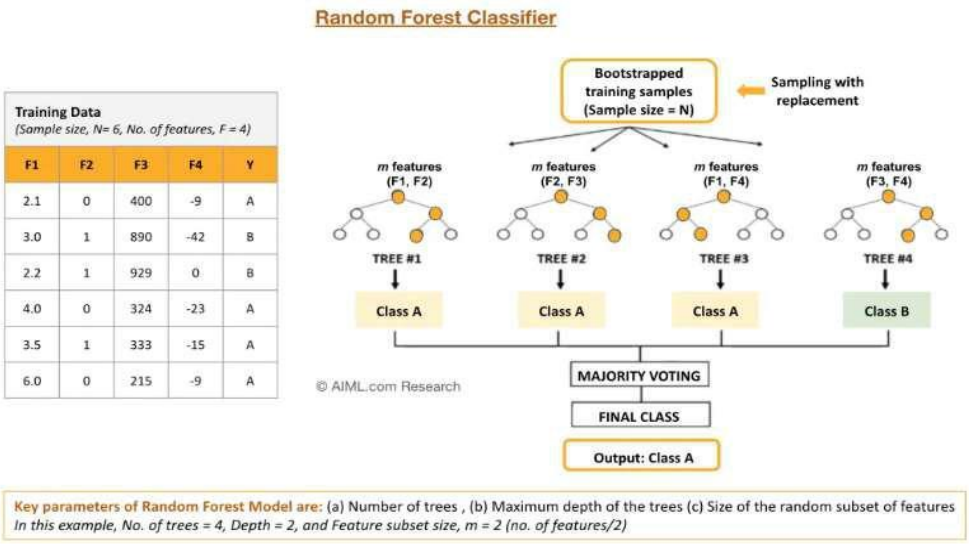
i. Randomly sample the training data with replacement (bootstrapping).

ii. Train a decision tree on this sample, but at each split, consider only a random subset of features, not all.

iii. Repeat for many trees (e.g., 100 or 500).

iv. Combine the output of all trees for the final prediction.

- Classification: majority vote
- Regression: average prediction



Random Forest Classifier (Source: AIML.com Research)