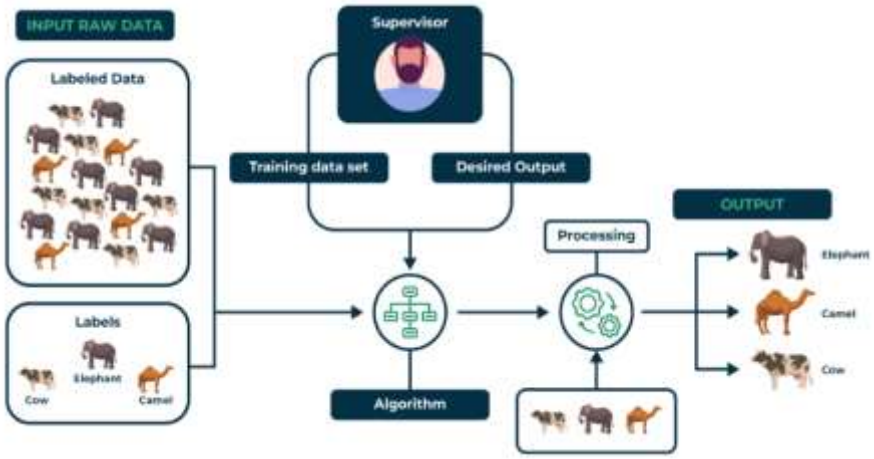
Unit 2 Supervised Learning (10 hrs.)

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



2.1. Types of Supervised Learning

2.1.1. **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

2.1.2. **Classification**: A classification problem is when the output variable is a category, such as “Yes” or “No”, “disease” or “no disease”.

2.2. Regression

Regression is a type of supervised learning that is used to predict continuous values, such as house prices, stock prices, or customer churn. Regression algorithms learn a function that maps from the input features to the output value.

2.2.1. Linear Regression

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task i.e. to predict a continuous value. Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. It assumes a **linear relationship** between the input variables (**X**) and the target variable (**Y**).

2.2.1.1 Simple Linear Regression

Simple Linear Regression models the relationship between a single independent variable (X) and a dependent variable (Y) using the equation:

$$Y=mX+c$$

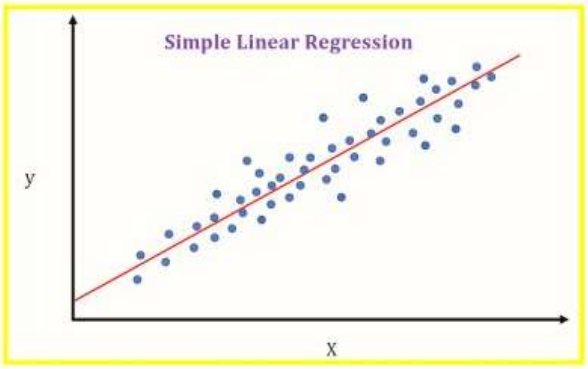
Where: Y= Dependent variable (target/output)

X= Independent variable (input feature)

m= Slope of the line (coefficient)

c = Intercept

Simple linear regression is used to find out the best relationship between a single input variable (predictor, independent variable, input feature, input parameter) & output variable (predicted, dependent variable, output feature, output parameter).



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 line can be modelled based on the linear equation shown below.

$$Y=mX+c$$

where:

Y= Dependent variable (target/output)

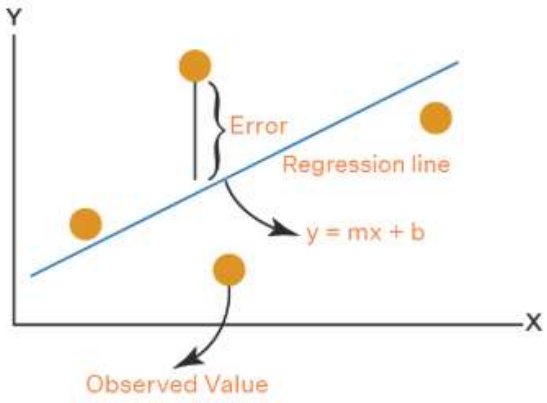
X= Independent variable (input feature)

m= Slope of the line (coefficient)

c = y Intercept or bias

The motive of the simple linear regression algorithm is to find the best values for m and c and then use to predict values. The line having that value of m and c which is used for prediction is called Best Fit Line. For above given example red line is the best fit line.

Least square method is the process of finding a regression line or best-fitted line for any data set that is described by an equation.



Least-square method is the curve that best fits a set of observations with a minimum sum of squared errors. Let us assume that the given points of data are (x1, y1), (x2, y2), (x3, y3), …, (xn, yn) in which all x's are independent variables, while all y's are dependent ones. This method is used to find a linear line of the form y = mx + b, where y and x are variables, m is the slope, and b is the y-intercept. The formula to calculate slope m and the value of b is given by:

$$m = (n\sum xy - \sum y\sum x)/[n\sum x^2 - (\sum x)^2]$$

$$b = (\sum y - m\sum x)/n$$

Here, n is the number of data points

Example:

- Predicting weight based on height. Here, **only one** predictor variable (height) is used.

Given data: **Height (X)** and **Weight (Y)**

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

Now,

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

n=4

Now Calculate,

$m = (n\sum xy - \sum x\sum y)/[n\sum x^2 - (\sum x)^2]$

m = (4*39240-660*236)/(4*109400-(660)2) =1200/2000=0.6

$b = (\sum y - m\sum x)/n$

b= (236-0.6*660)/4= - 40

Final Simple Linear Regression Model is

y= 0.6x-40 Here, x is height and y is weight.

Now, if you want to predict the weight whose height is 172 then,

Weight= 0.6*172-40=63.2

Question:

The following data shows the sales (in million dollars) of a company.

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

Estimate the sales in the year 2020 using the regression line.

Multiple Linear Regression

When there are multiple independent variables (X1,X2,...,Xn), the equation extends to:

Y=m1X1+m2X2+...+mnXn+c

Example: Predicting house price based on: Size of the house (sq. ft), Location , Number of rooms, Age of the house , etc.

Here, multiple features contribute to the prediction in the multiple linear regression.

[Note: Refer Lab class for the implementation of multiple linear regression]

2.2.1.2 Polynomial Regression

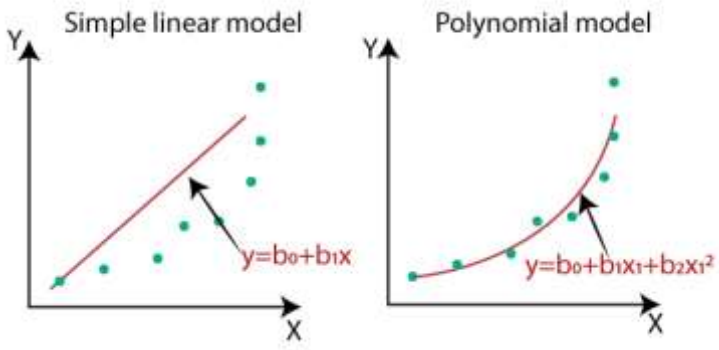
Polynomial Regression is an extension of Linear Regression where the relationship between input variables and output is modeled as an **nth-degree polynomial**:

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

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.).



5.2 Model Evaluation:

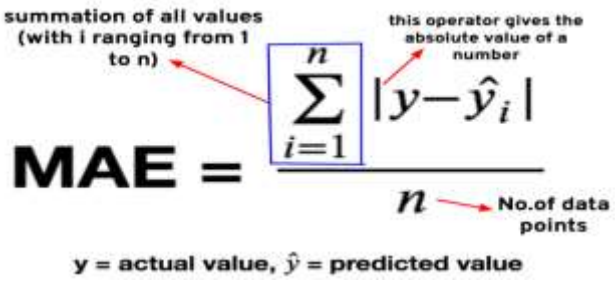
Model evaluation is a process that uses some metrics which help us to analyze the performance of the model. It is a crucial step in assessing how well a machine learning model performs. It helps in:

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

5.2.2 Regression Metrics (Unit 5)

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.

a. **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.

b. **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.

c. **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.

d. **R-Squared (Coefficient of Determination) :** It measures the proportion of variance in the dependent variable that is predictable from the independent variables.

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

$\bar{y}$ = mean of actual y values

Interpretation:

R^2=1 Perfect fit.

R^2=0: Model performs as badly as the mean.

R^2<0: Worse than the mean.

Example

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

So:

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

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

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

= |3-2.5|+|5-5.1|+|7-6.8|+|9-9.3|/4

MAE =(0.5+0.1+0.2+0.3)/4=0.257

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

First, calculate squared errors:

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

Sum of squared errors = 0.25+0.01+0.04+0.09=0.39

MSE=0.39/4=0.0975

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

RMSE=√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

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

Now,

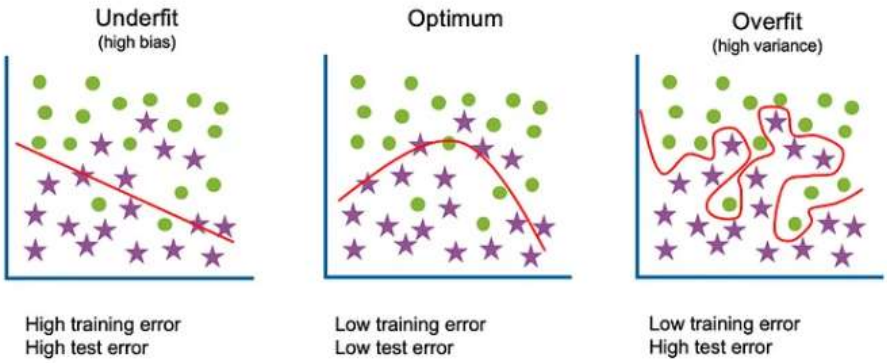
i	yi	$yi-\bar{y}$	$(yi-\bar{y})^2$
1	3	-3	9
2	5	-1	1
3	7	1	1
4	9	3	9

Sum of $(yi-\bar{y})2=9+1+1+9=20$

R^2= 1- 0.39/20 =0.9805

NOTE: 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 poor performance 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.



2.2.2 Regularization Techniques

- Regularization is a technique used to prevent overfitting in machine learning models.
- It works by adding a penalty to the loss function to control model complexity.
- A very complex model (like very large coefficients in linear regression) might fit the training data perfectly but perform poorly on unseen data.
- Without regularization, a model may memorize training data (overfit).
- Regularization forces the model to stay simpler by shrinking the weights (coefficients).
- Simple models often generalize better to new, unseen 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:

2.2.2.1 Ridge Regression (L2 Regularization)

- Adds a penalty proportional to the square of the magnitude of coefficients.
- Ridge regression is a type of linear regression used when data suffers from multicollinearity (when independent variables are highly correlated).

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

where,

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

β_j = Model coefficients [For simple LR β_j=1 coefficient, and for Multiple LR β_j=n no. of coefficients]

Why use L2 Regularization?

- Keeps model weights **small and smooth**
- Helps reduce **variance** (overfitting)
- Doesn't eliminate features, just shrinks their importance

2.2.2.2 Lasso regression (L1 Regularization)

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

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

- Can shrink some coefficients exactly to zero.
- Performs feature selection (automatically picks important features).

Let's say you're predicting house price using: Size, Number of rooms, Distance to city, Number of ceiling fans, Age of house with L1, the model might decide: "Ceiling fans don't matter" → sets its weight to zero so that it keeps only the important features.

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 doesn't 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.

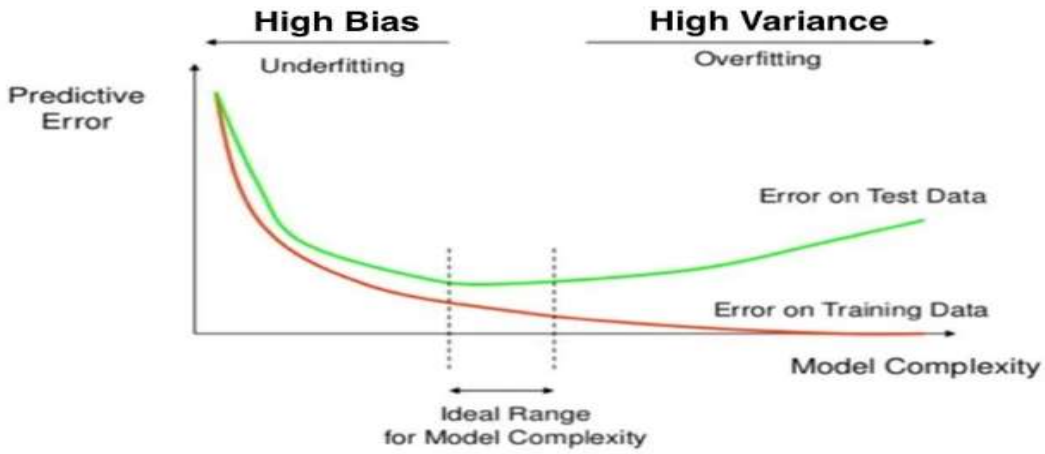
Problem: The model performs well on training data but fails on test data because it learned noise.

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 can't 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.



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)

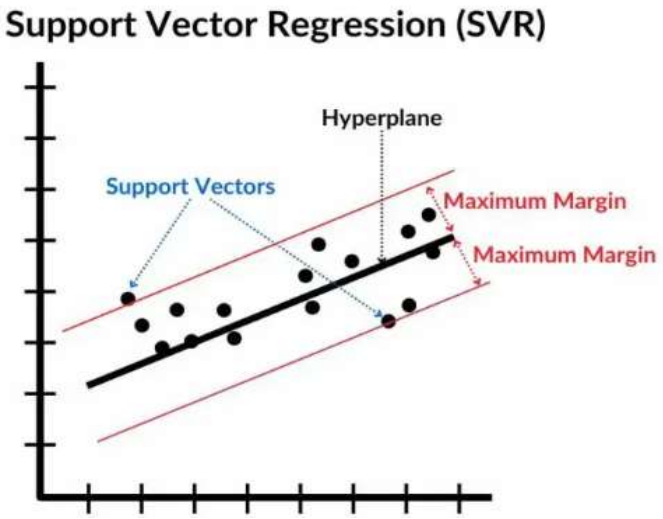
The figure shows the relationship between model complexity and error. As the complexity of the model increases, the bias decreases and the variance increases. At the same time total error of the model is first decreases and then increases. The optimal model complexity is the point at which the total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

How to find the optimal point for Bias- Variance?

- ➔ Regularization
- ➔ Cross validation (Unit 5)
- ➔ Grid search or random search (Unit 5)

2.2.3 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.



[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.

How It Works ?

Step 1: *Linear Combination*

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

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

b_0: Bias term

$b_1,\ldots,b_n$: Coefficients (learned from data)

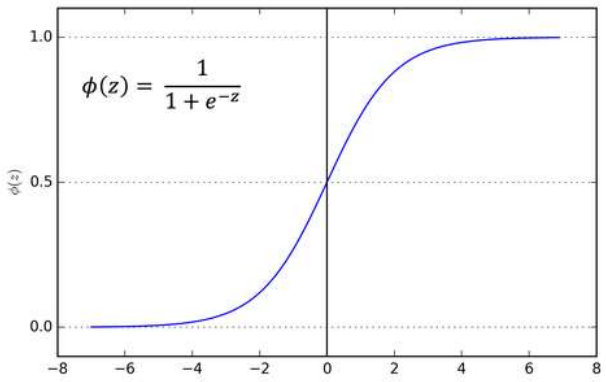
$x_1,\ldots,x_n$: Input features (e.g., age, salary)

Step 2 : *Sigmoid Function*

The output z is passed through a sigmoid function to squash it into [0, 1]:

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

- 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"

2.3.1.1 Binary Classification

Binary classification involves classifying data into two classes.

Sigmoid Function: Maps linear output $z=\beta_0+\beta_1X_1+\cdots+\beta_nX_n$ to probabilities between 0 and 1.

$$P(y=1) = \frac{1}{1+e^{-z}}, \quad P(y=0) = 1 - P(y=1)$$

Python Implementation: Binary Classification

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Load dataset (binary: malignant=0, benign=1)
```

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
X, y = data.data, data.target
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train model
model = LogisticRegression(solver='lbfgs', max_iter=1000)
model.fit(X_train, y_train)
# Predict probabilities for class 1 (benign)
y_pred = model.predict(X_test)          # Class labels (0/1)
# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
```

2.3.1.2 Multi-Class Classification

Multi-class classification handles problems where the output can belong to more than two classes**.**

Examples: Classifying digits (0–9), predicting the type of animal (cat, dog, bird)
Strategies:

a. Softmax Regression (a.k.a. Multinomial Logistic Regression):

Directly handles multiple classes.

Uses softmax function to output probability for each class.

Generalizes sigmoid to K classes.

Probability for class k:

$$P(y=k)=\frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \quad z_k=\beta_{k0}+\beta_{k1}X_1+\cdots+\beta_{kn}X_n$$

Python Implementation: Multi-Class (Softmax)

```
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))
```

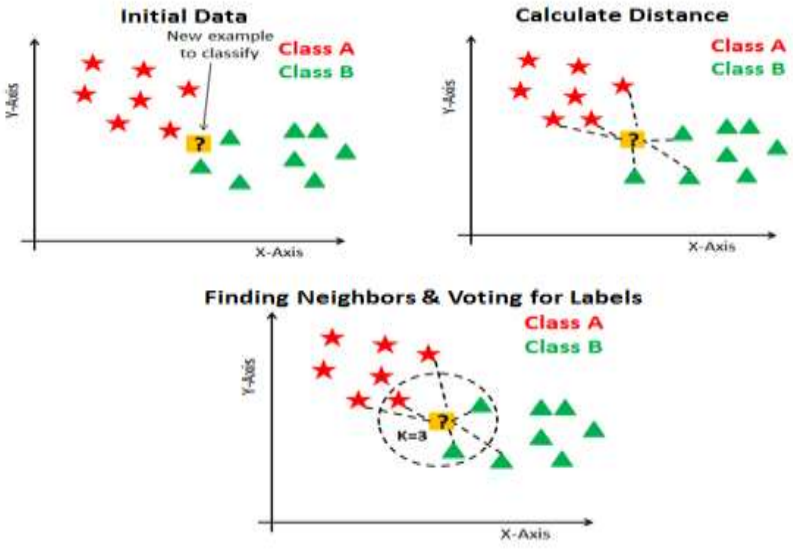
b. One-Vs-Rest (OvR):

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

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

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.

How KNN Works?

Step 1: Choose Hyperparameter "k"

k = Number of nearest neighbors to consider (e.g., k=3, 5, 10….).

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 5)

Step 2: Calculate Distance

Compute the distance between the new data point and all training points using a distance metric (use Euclidean Distance (common), Manhattan Distance, etc**)**:

Euclidean Distance

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

Step 3: Find "k" Nearest Neighbors

Select the k closest training points based on the computed distances.

Step 4: Majority Vote

Assign the class that appears most frequently among the **k** neighbors.

Example: Consider the following dataset

X_1	X_2	Y (Classification)
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

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

Calculate Euclidean Distance

Euclidean Distance Formula:

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

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

New point: (3, 7)

Distances to existing data:

1. To (7, 7) →

$$\sqrt{(3-7)^2 + (7-7)^2} = \sqrt{16+0} = 4.0 \quad \text{(Bad)}$$

2. To (7, 4) →

$$\sqrt{(3-7)^2 + (7-4)^2} = \sqrt{16+9} = \sqrt{25} = 5.0 \quad \text{(Bad)}$$

3. To (3, 4) →

$$\sqrt{(3-3)^2 + (7-4)^2} = \sqrt{0+9} = 3.0 \quad \text{(Good)}$$

4. To (1, 4) →

$$\sqrt{(3-1)^2 + (7-4)^2} = \sqrt{4+9} = \sqrt{13} \approx 3.61 \quad \text{(Good)}$$

Find 3(K=3 given) Nearest Neighbors

Sorted by distance:

1. (3, 4) → 3.0 → Good
2. (1, 4) → 3.61 → Good
3. (7, 7) → 4.0 → Bad

So, out of the 3 nearest neighbors: 2 are Good and 1 is Bad

Final Classification (Majority Vote)

Predicted Class = Good

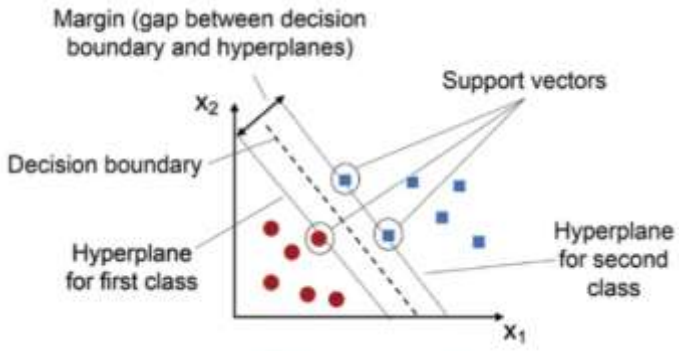
Using KNN (k=3), the new point (3,7) is classified as "Good".

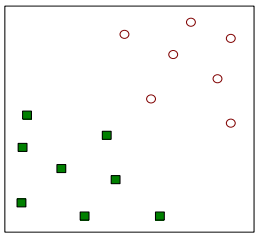
Question: For the data in the given table, find the class of the new data using KNN algorithm.

Age	Income	No. of Credit	Class
20	5000	3	Yes
21	4000	2	No
22	3000	1	Yes
23	2500	3	No
19	3500	2	Yes
24	1700	1	Yes
22	2200	2	No
25	3100	3	?

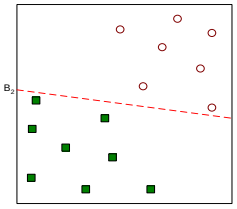
2.3.3. Support Vector Machine (SVM)

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.

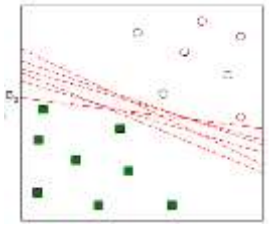




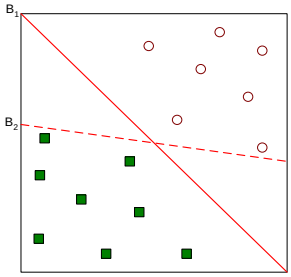
Find the decision boundary (line/hyperplane) that will separate the data.



one possible solutions

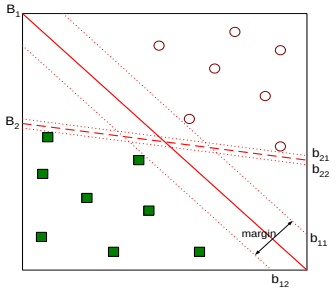


Other possible solutions



Which one is better? B1 or B2?

How do you define better?



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

Let D be a data set given as (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) where x_i is a set of training data with associated class labels y_i. Each y_i can take values, either +1 or -1.

The plane H_0 is the median in between where,

$$H_0: w.x_i + b = 0$$

Where, w is the weight vector; x is the input vector and b is the bias.

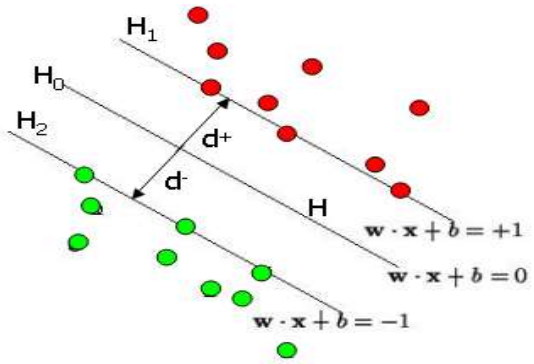
H_1 and H_2 are the planes defined by

$$H_1: w.x_i + b = +1$$

$$H_2: w.x_i + b = -1$$

We want a classifier (linear separator) with as big a margin as possible. The distance between H_1 and H_2 (the margin) is given by:

$Margin = \frac{2}{||w||}$, $|w|$ is the cardinality of w



SVM Kernels and Their Types

- 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).

Types:

1. Linear Kernel: It used when data is linearly separable - a straight line (or hyperplane) can divide the classes.

   K(x,y)=x·y (Dot product of input vectors x and y).

2. Polynomial Kernel**:** It **c**aptures non-linear relationships by adding polynomial terms.

   K(x,y)=(x.y+c)d where is c a constant and d is the polynomial degree.

3. Radial Basis Function Kernel (RBF) Kernel: 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.

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

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

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

γ (or σ) is a hyperparameter that controls the "spread" of the kernel.

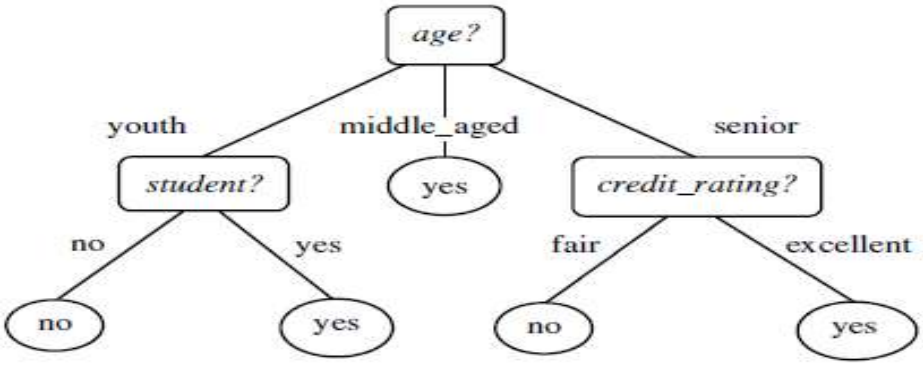
Support Vector Machine (SVM) for Linear and Non-linear Classification

SVM can handle both linearly separable and non-linearly separable data using different kernels**.**

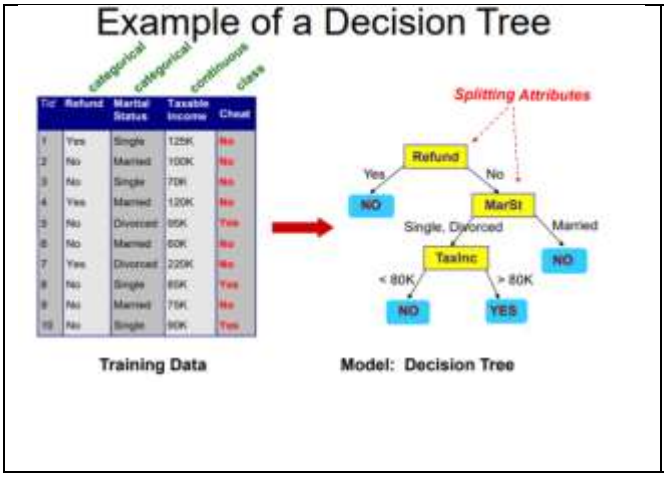
- ➔ When data can be separated by a straight line (or hyperplane) then Linear Classification is used. In this case no kernel transformation is needed.
- ➔ When data cannot be separated by a straight line then Non-Linear Classification is used. In this case kernel trick is used. Common non-linear kernels are Polynomial kernel, RBF (Radial Basis Function / Gaussian) kernel.

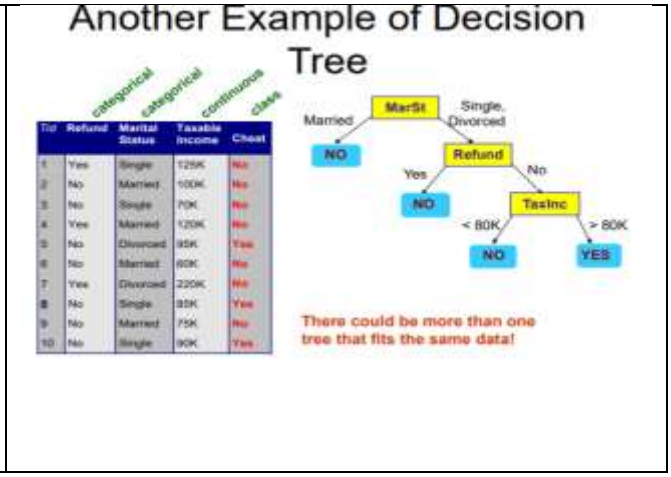
2.3.4 Decision Tree

A Decision Tree is a supervised machine learning algorithm used for classification and regression. It is a flowchart-like tree structure, where each internal node (nonleaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node.



There are many decision tree that can be construct from a given set of attributes. Finding all the tree is computationally infeasible because of the exponential size of the search space.





Attribute Selection Measures

An attribute selection measure is a heuristic for selecting the splitting criterion that "best" separates a given data partition, D, of class-labeled training tuples into individual classes. There are different popular attribute selection measures:- information gain, gini index, etc.

Information gain

ID3(Iterative Dichotomiser) uses information gain as its attribute selection measure.

The expected information needed to classify a tuple in D is given by

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

where pi is the probability that an arbitrary tuple in D belongs to class C_i and is estimated by $|C_i,D|/|D|$. A log function to the base 2 is used, because the information is encoded in bits. Info(D) is also known as the entropy of D.

Now, suppose we were to partition the tuples in D on some attribute A having v distinct values, $\{a_1, a_2, : : : , a_V\}$ as observed from the training data. If A is discrete-valued, these values correspond directly to the v outcomes of a test on A. Attribute A can be used to split D into v partitions or subsets, $\{D_1, D_2, : : : , D_v\}$, where D_j contains those tuples in D that have outcome a_j of A.

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

$$Gain(A) = Info(D) - Info_A(D).$$

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

Example: Construction of Decision tree using information gain

Class-labeled training tuples from the *AllElectronics* customer database.

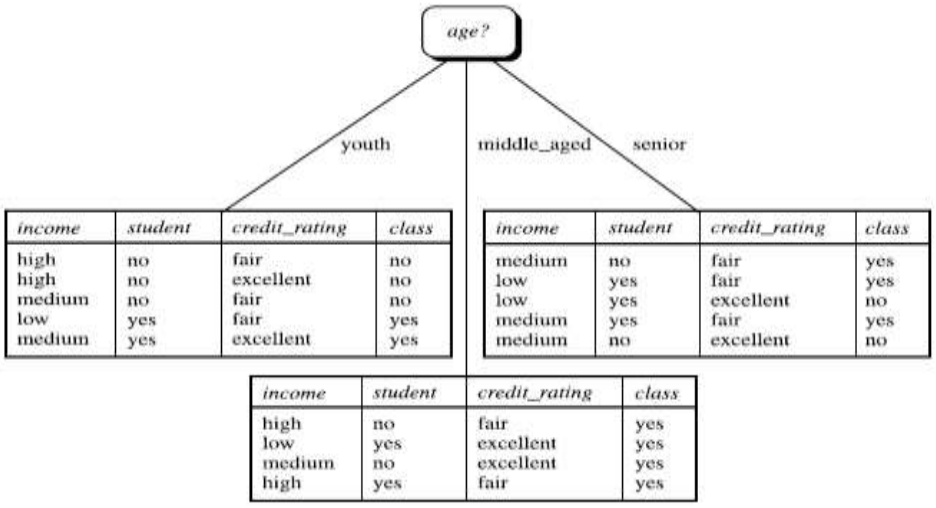
RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits.}$$

$$\begin{aligned} Info_{age}(D) &= \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) \\ &+ \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4}\right) \\ &+ \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) \\ &= 0.694 \text{ bits.} \end{aligned}$$

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits.}$$

Similarly, we can compute Gain(income) = 0.029 bits, Gain(student) = 0.151 bits, and Gain(credit rating) = 0.048 bits. Because age has the highest information gain among the attributes, it is selected as the splitting attribute.



Same process is done for the each branch until:

- All data in a node belong to the same class.
- A maximum depth is reached.
- A minimum number of samples per node is reached

Pruning of Decision Tree

Pruning is the process of trimming a decision tree to reduce its size and complexity to avoid overfitting**.**

a. Pre-Pruning (Early Stopping)
   - Stop growing the tree before it becomes overly complex.
   - Control via parameters:
     - max_depth: Maximum allowed depth.
     - min_samples_split: Minimum samples to split an internal node.
     - min_samples_leaf: Minimum samples required at a leaf.
     - max_leaf_nodes: Limit number of leaf nodes.

b. Post-Pruning (Cost Complexity Pruning)
   - Let the tree grow fully, then prune back branches that don't improve generalization.

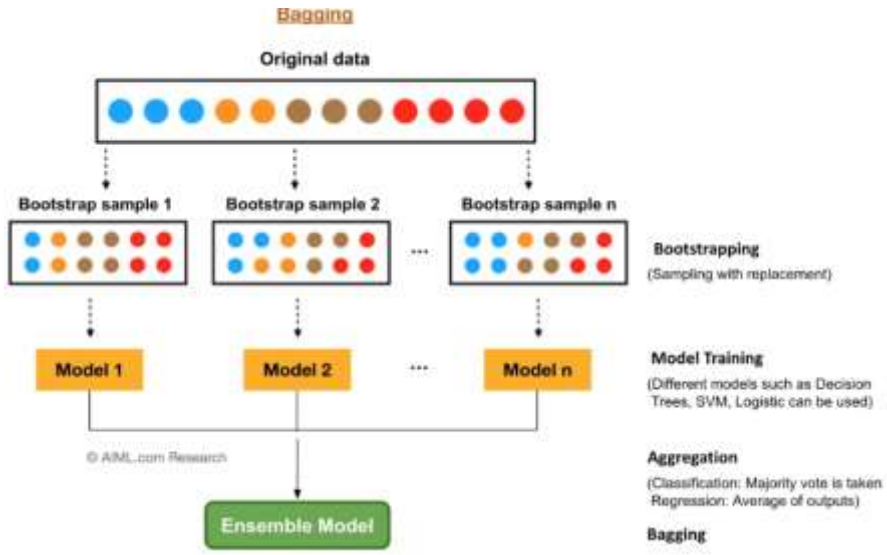
Ensemble Methods

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).



(Source: aiml.com Research)

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:

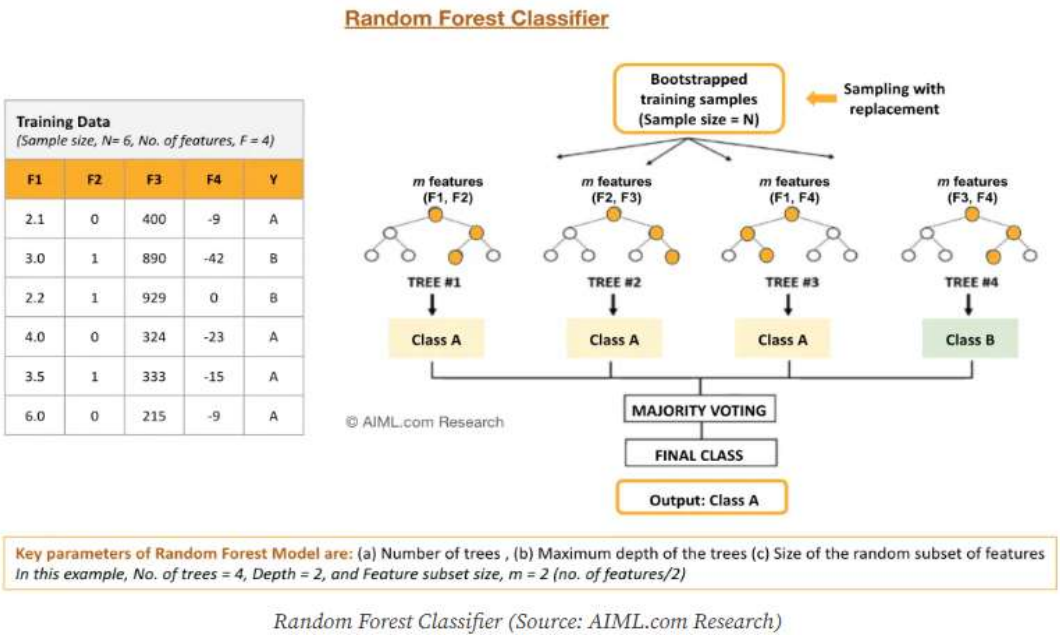
i. Randomly sample the training data with replacement (bootstrapping).

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



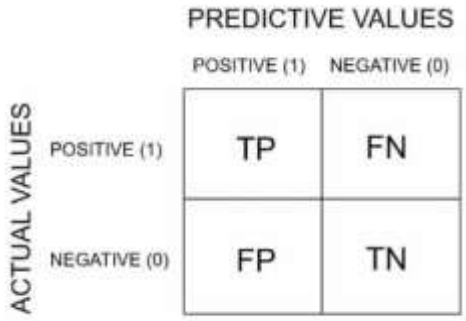
Random Forest Classifier (Source: AIML.com Research)

5.2 Model Evaluation Metrics

5.2.1. Classification Metrics (Unit 5)

Classification metrics help evaluate the performance of classification models, especially when the output is categorical (e.g., spam vs. not spam, disease vs. no disease).

1. **Confusion Matrix**: A confusion matrix is a table used in machine learning to evaluate the performance of a classification model. It summarizes the model's predictions by comparing them against the actual (true) labels. The matrix displays the number of correct and incorrect predictions, categorized as true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).
   - True Positives (TP): The model correctly predicts a positive class.
   - True Negatives (TN): The model correctly predicts a negative class.
   - False Positives (FP): The model incorrectly predicts a positive class when the actual class is negative (also known as Type I error).
   - False Negatives (FN): The model incorrectly predicts a negative class when the actual class is positive (also known as Type II error).



Confusion Matrix for the Binary Classification

2. **Accuracy:** A general measure of how well the model is doing. It is calculated as
   (TP + TN) /N Where N(total no. of data)=TP+TN+FP+FN

3. **Recall:** The proportion of actual positives correctly identified by the model. It is calculated as

   TP / (TP + FN)

   High Recall = very few false negatives

4. **Precision:** The proportion of correct positive predictions out of all positive predictions made. It is calculated as

   TP / (TP + FP)

High Precision = very few false positives

5. **F1-score:** The harmonic mean of precision and recall, providing a balanced view of the model's performance. It is calculated as

   2 × (Precision × Recall) / (Precision + Recall)

Metric	Formula	Meaning
Accuracy	(TP + TN) / (TP + TN + FP + FN)	Overall correctness
Precision	TP / (TP + FP)	How many predicted positives are correct
Recall (Sensitivity)	TP / (TP + FN)	How many actual positives are caught
F1 Score	2 × (Precision × Recall) / (Precision + Recall)	Harmonic mean of Precision and Recall

Note: Accuracy is a common metric to evaluate classification models, but it can be highly misleading when dealing with imbalanced ((i.e., one class dominates) datasets.

Example:

Dataset: 100 samples

95 negative (no disease)

5 positive (has disease)

Suppose, model predicts all "negative"

TP=0	FN=5
FP=0	TN=95

Then, Accuracy= (TP+TN)/100=95/100=95%

But, Precision = 0, Recall = 0 and F1 = 0

Therefore, model has high accuracy but not useful for catching positives cases.

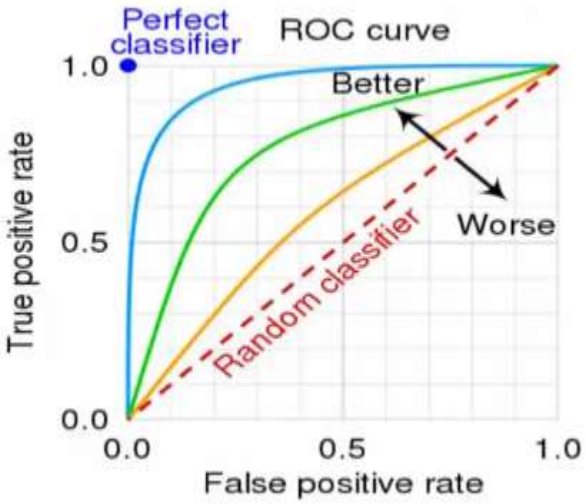
ROC Curve

The ROC (Receiver Operating Characteristic) curve is a graphical representation used to evaluate the performance of binary classification models. It plots two key metrics: True Positive Rate (TPR) and False Positive Rate (FPR).

True Positive Rate (TPR)=Recall=sensitivity=TP/TP+FN

False Positive Rate (FPR)=specificity=FP/FP+TN

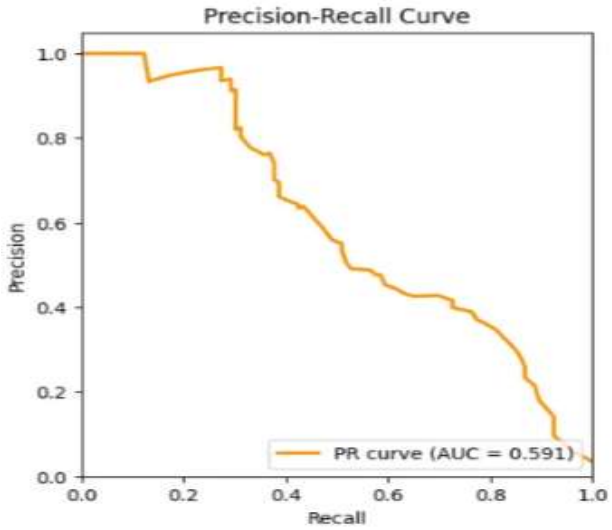
The ROC curve is generated by plotting TPR against FPR. The curve helps visualize the trade-off between sensitivity and specificity for different thresholds.



The AUC (Area Under the Curve) is a single scalar value that summarizes the performance of a binary classification model. It represents the area under the ROC curve and ranges from 0 to 1. Higher AUC value indicates better performance.

PR Curve

A PR (Precision-Recall) curve is a graphical representation used in machine learning to evaluate the performance of classification models, particularly when the classes are imbalanced. It plots precision against recall at various threshold settings, highlighting the trade-off between these two metrics. A high area under the PR curve (AUC-PR) indicates a good model, where both precision and recall are high.



****Thank You****