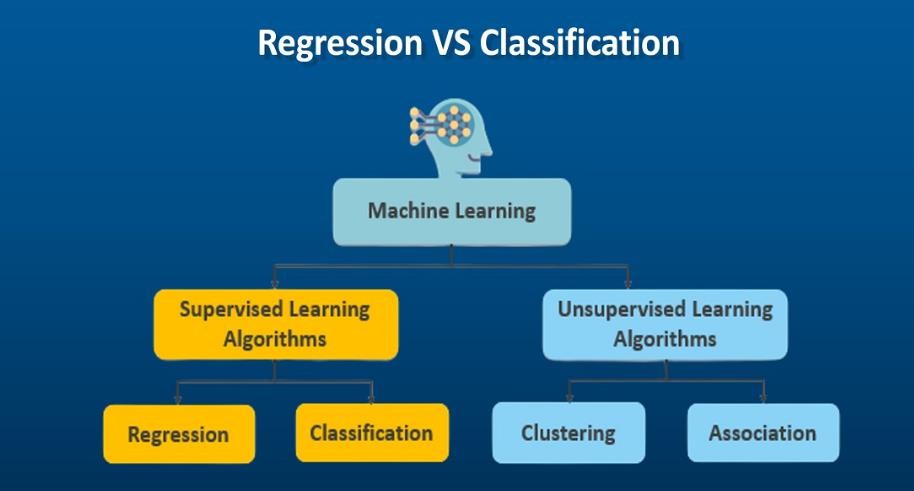
**Supervised Learning -Classification Unit-4**

# Classification problem- Probability based approach

* Definition
* Types of classification problems
* Examples of classification tasks
* Difference between classification and regression

Regression and classification are two fundamental types of supervised learning tasks in machine learning, and they have distinct purposes and characteristics.



# Introduction to Classification

Classification is a supervised learning task where the goal is to predict the class or category of given data points. It involves assigning labels to inputs based on their features.

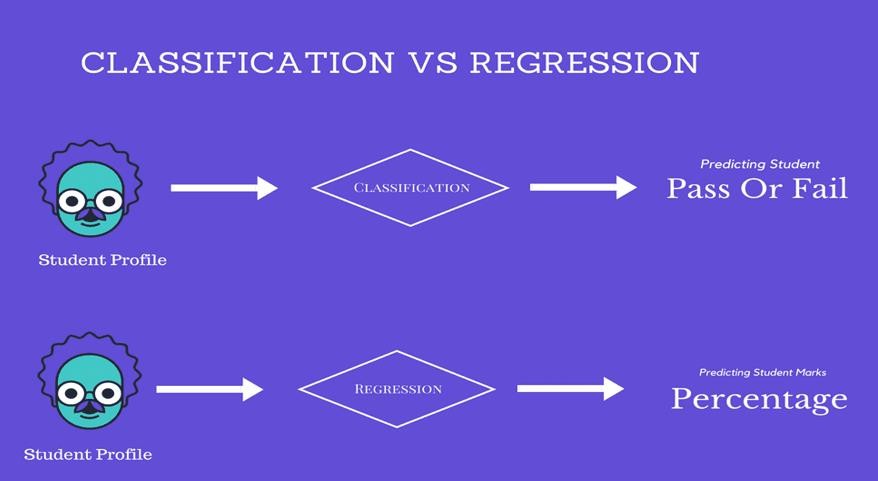
# Key Features of Classification:

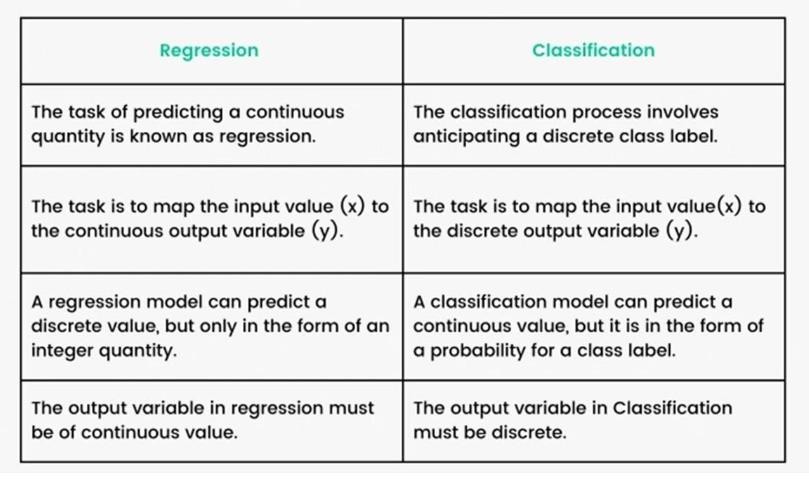
* + **Input**: Features (numerical, categorical, or mixed data).
  + **Output**: Discrete class labels (e.g., spam or not spam, 0 or 1).

# Types:

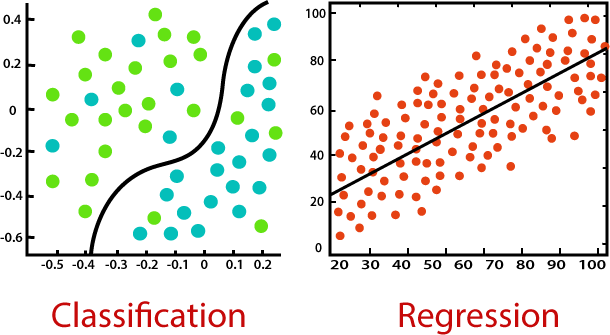
* + - **Binary Classification**: Two classes (e.g., fraud detection).
    - **Multi-class Classification**: More than two classes (e.g., digit recognition).

**Example**: Predicting whether a patient has a disease (Yes/No).





We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the ‘**Sigmoid function**’ or also known as the ‘logistic function’ instead of a linear function.



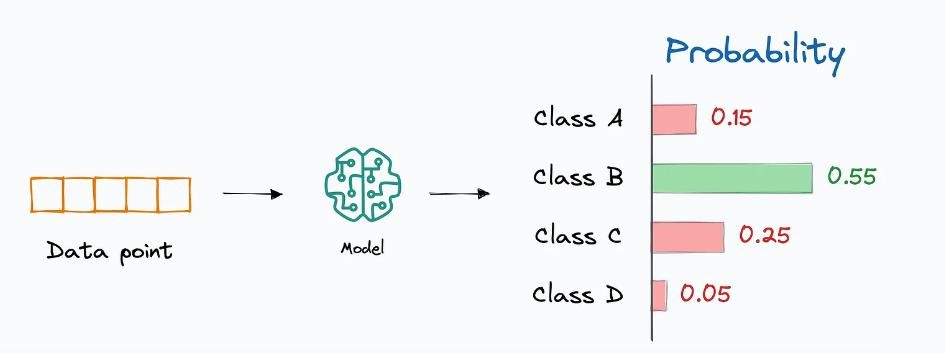
# Probability-Based Classification

In probability-based classification, the model estimates the likelihood that an input belongs to a particular class and assigns it to the most probable class.

* + **Output**: A probability score for each class (e.g., 70% spam, 30% not spam).

# Advantages:

* + - Provides confidence levels for predictions.
    - Useful for threshold-based decision-making.



# Bayesian Probability Basics

Probability plays a foundational role in classification. Bayesian methods use probabilities to make predictions.

# Key Concepts:

* + **Joint Probability** (P(A,B) Probability of both events A and B.
  + **Marginal Probability** (P(A): Probability of an event irrespective of others.
  + **Conditional Probability** (P(A∣B): Probability of A given B.

# Bayes' Theorem:

* + **Example**: In medical diagnosis, P(Disease∣Symptoms) is derived using P(Symptoms∣Disease).

# Logistic Regression as a Probability-Based Approach

Logistic regression predicts the probability of a binary outcome using the logistic function.

# Key Concepts:

* + **Sigmoid Function:**
  + The sigmoid maps any real number into the range [0, 1].
  + **Decision Rule**: Assign the class based on a threshold (commonly 0.5).

# Naive Bayes Classifier

Naive Bayes is a family of probabilistic classifiers based on Bayes' Theorem, with an assumption of independence between features. There are different variations of Naive Bayes classifiers, tailored for specific types of data:

Naive Bayes is based on Bayes' theorem with the assumption that features are independent given the class.

# Formula:

****

Where:

* + C: Class
  + X: Feature set

# Types:

1. **Gaussian Naive Bayes**: Assumes normal distribution of features.
2. **Multinomial Naive Bayes**: Used for discrete data like word counts.
3. **Bernoulli Naive Bayes**: Used for binary features.

# Evaluation Metrics for Classification Metrics:

* + **Accuracy**: Ratio of correct predictions to total predictions.
  + **Precision**: Proportion of true positives among predicted positives.
  + **Recall (Sensitivity)**: Proportion of true positives among actual positives.
  + **F1-Score**: Harmonic mean of precision and recall.

# ROC Curve:

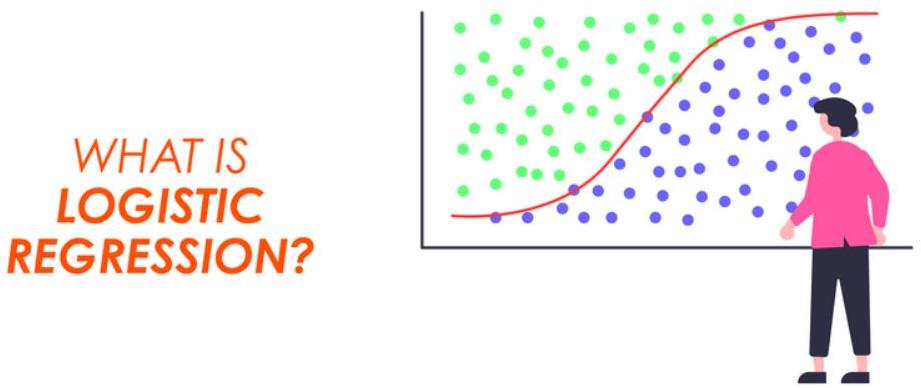
A plot of the True Positive Rate (TPR) vs. False Positive Rate (FPR), showing the tradeoff between sensitivity and specificity.

**AUC** measures the overall performance.

**Supervised Learning -Classification**

# Logistic Regression- log-odd, sigmoid transformation

* Introduction to Logistic Regression
* The Problem with Linear Regression for Classification
* Log-Odds in Logistic Regression
* Sigmoid Function and Transformation
* Relationship Between Log-Odds and Probability
* Logistic Regression Model
* Decision Boundary in Logistic Regression
* Applications of Logistic Regression



# Introduction to Logistic Regression

Logistic regression is a supervised learning algorithm used for binary classification. It predicts the probability of a data point belonging to one of two classes.

* + Input: Features (x1,x2,…,xn).
  + Output: A probability score (P) that an input belongs to a class.

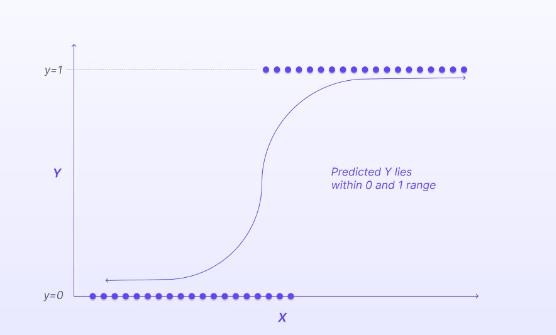
Example: Predict whether an email is spam (1) or not spam (0).

# The Problem with Linear Regression for Classification

Linear regression is unsuitable for classification because:

* + It predicts continuous values, which are unbounded.
  + Outputs cannot be interpreted as probabilities.

Solution: Logistic regression uses a sigmoid function to constrain predictions to the range [0, 1], making them interpretable as probabilities.



# Log-Odds in Logistic Regression

Logistic regression models the log-odds of an event. Odds

The odds represent the ratio of the probability of success (P) to the probability of failure (1−P):



Log-Odds

The log-odds, also known as the logit, are the natural logarithm of the odds:



In logistic regression, the log-odds are modeled as a linear combination of the input features:



Here:

* β0: Intercept term.
* β1,β2,…,βn: Coefficients for the features.

# Sigmoid Function and Transformation

The sigmoid function maps the log-odds to a probability between 0 and 1. Sigmoid Function Formula:



* + z: Log-odds (linear combination of features).
  + h(x): Predicted probability of the positive class.

Properties of the Sigmoid Function

* + h(x)→1 as z→+∞
  + h(x)→0 as z→−∞
  + h(x)=0.5 when z=0.

# 5. Relationship Between Log-Odds and Probability

* + The probability (P) can be expressed in terms of the log-odds:



* + Similarly, the log-odds can be obtained from the probability:

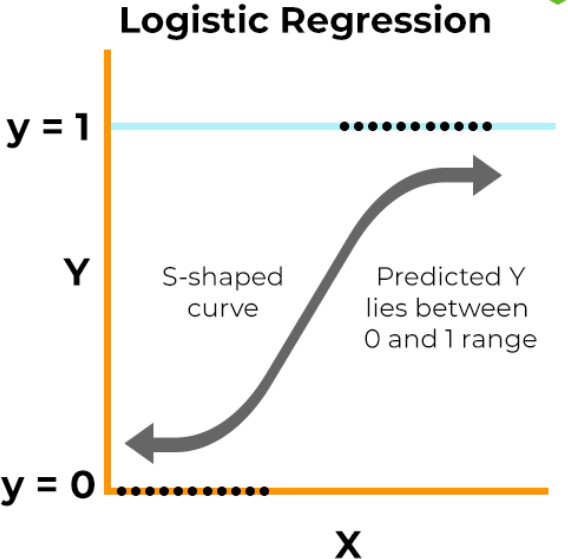


# Logistic Regression Model

Logistic regression predicts the class based on a probability threshold:

* + If P≥0.5, predict class 1.
  + If P<0.5, predict class 0. Steps in Logistic Regression:

1. Compute the log-odds 
2. Apply the sigmoid function to obtain the probability (P).
3. Classify based on the threshold.



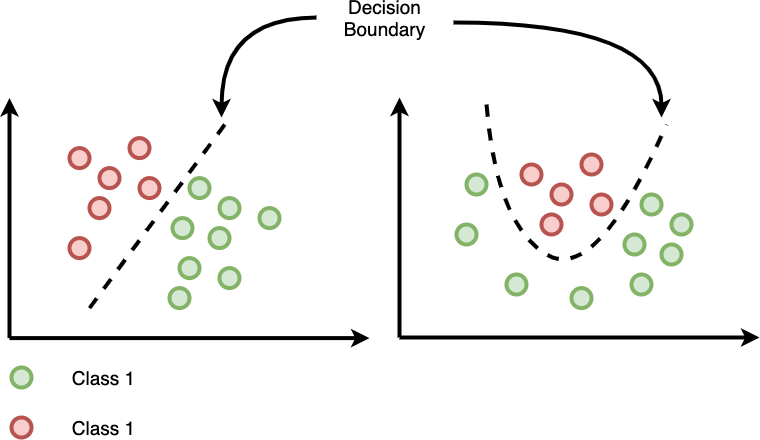
# Decision Boundary in Logistic Regression

The decision boundary is the point where the predicted probability is 0.5:



For binary features, this is a straight line in 2D space.

For more complex data, the boundary can be nonlinear.



# Applications of Logistic Regression

* + Medical Diagnosis: Predicting whether a patient has a disease.
  + Credit Scoring: Determining the likelihood of loan default.
  + Spam Detection: Classifying emails as spam or not spam.

# Example: Medical Diagnosis: Predicting Disease Presence

* Purpose: Logistic regression can help predict whether a patient has a particular disease based on medical data.
* Process:
  + Medical features like age, blood pressure, cholesterol levels, and test results are used as input variables.
  + Logistic regression predicts the probability of disease presence (e.g., heart disease, diabetes).
* Outcome:
  + The output is a probability score between 0 and 1.
  + Threshold-based classification determines the outcome (e.g., probability > 0.5 indicates disease presence).

**Supervised Learning -Classification**

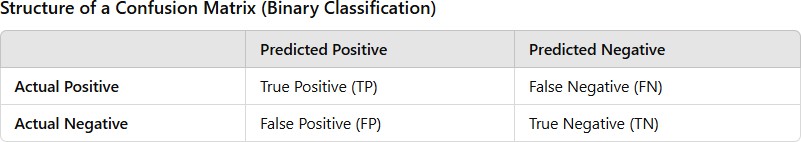
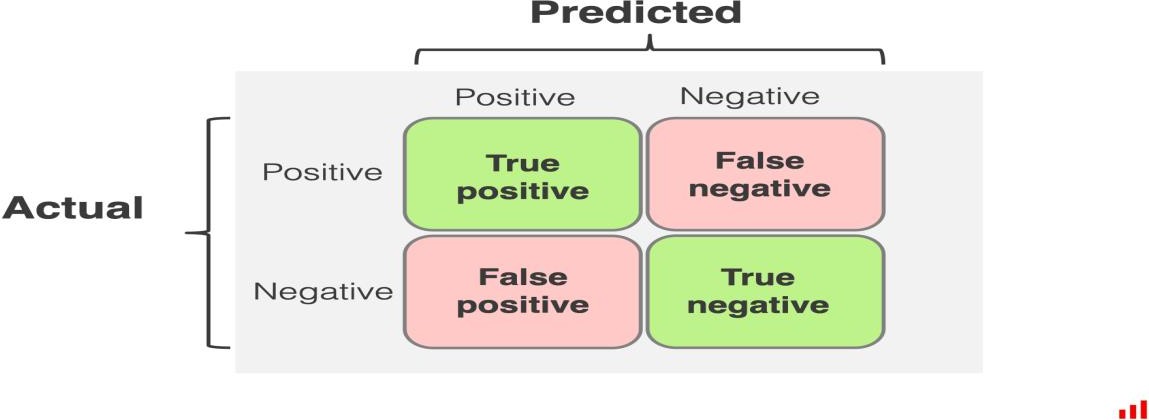
**Classification Metrics**

1. Confusion Matrix
2. Accuracy
3. Error Rate
4. Precision
5. Recall
6. ROC curve
7. F1 score

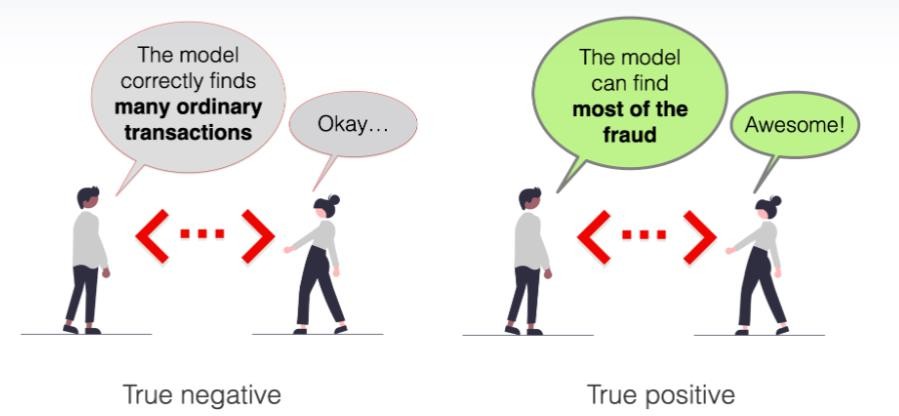
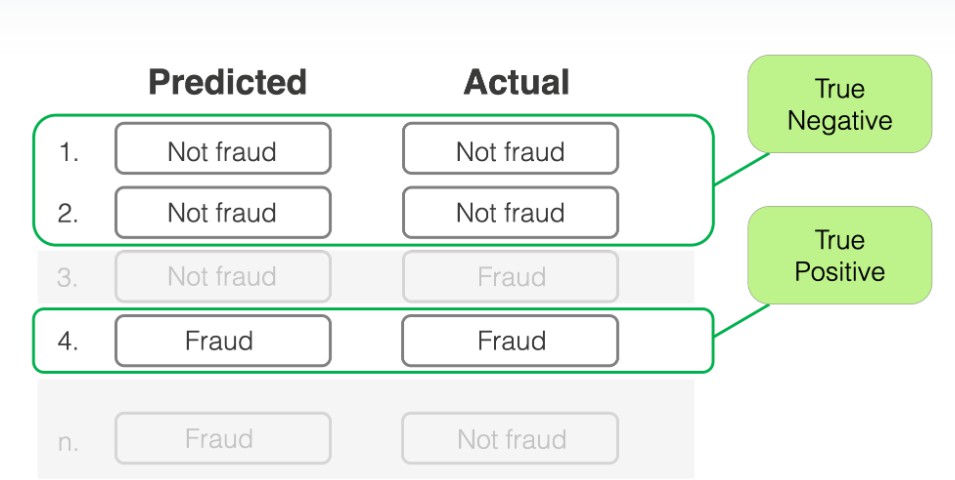
**Confusion Matrix**

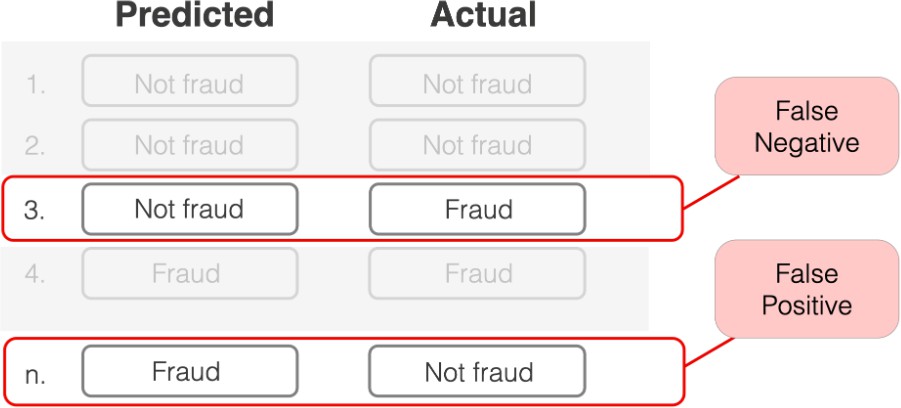
A **confusion matrix** is a table that summarizes the performance of a classification model by

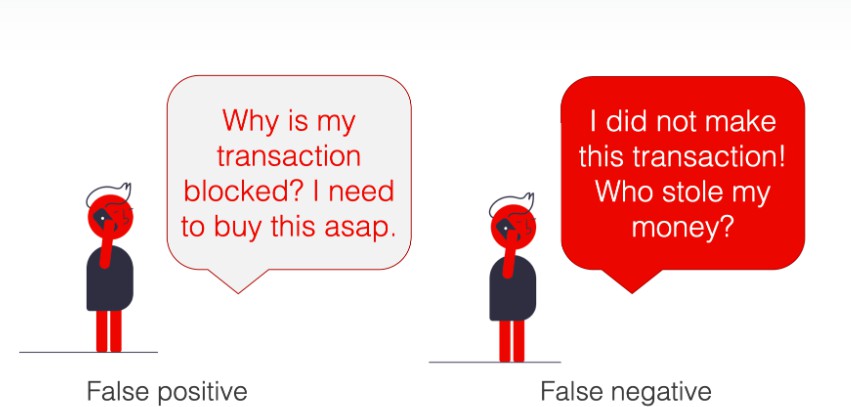
comparing predicted and actual values. It is especially useful for binary and multiclass classification problems.



* + **True Positive (TP):** Cases where the model correctly predicts the positive class.
  + **False Positive (FP):** Cases where the model incorrectly predicts the positive class (Type I error).
  + **False Negative (FN):** Cases where the model incorrectly predicts the negative class (Type II error).
  + **True Negative (TN):** Cases where the model correctly predicts the negative class.







1. **Accuracy**

**Definition:** Measures the proportion of correctly classified instances out of the total instances.

**Formula:**

****

**When to Use:** Suitable when classes are balanced.

**Limitation:** Misleading for imbalanced datasets (e.g., detecting rare diseases).

1. **Precision (Positive Predictive Value)**

**Definition:** Measures the percentage of positive predictions that are correct.

**Formula:**

****

**When to Use:** Important when false positives need to be minimized (e.g., spam detection).

1. **Recall (Sensitivity, True Positive Rate)**

**Definition:** Measures the percentage of actual positive instances that are correctly predicted.

**Formula:**

****

**When to Use:** Crucial when false negatives need to be minimized (e.g., cancer detection).

1. **F1 Score**

**Definition:** Harmonic mean of precision and recall, balancing their trade-off.

**Formula:**

****

**When to Use:** Useful when you need a balance between precision and recall, especially in imbalanced datasets.

1. **Specificity (True Negative Rate)**

**Definition:** Measures the percentage of actual negatives correctly predicted.

**Formula:**

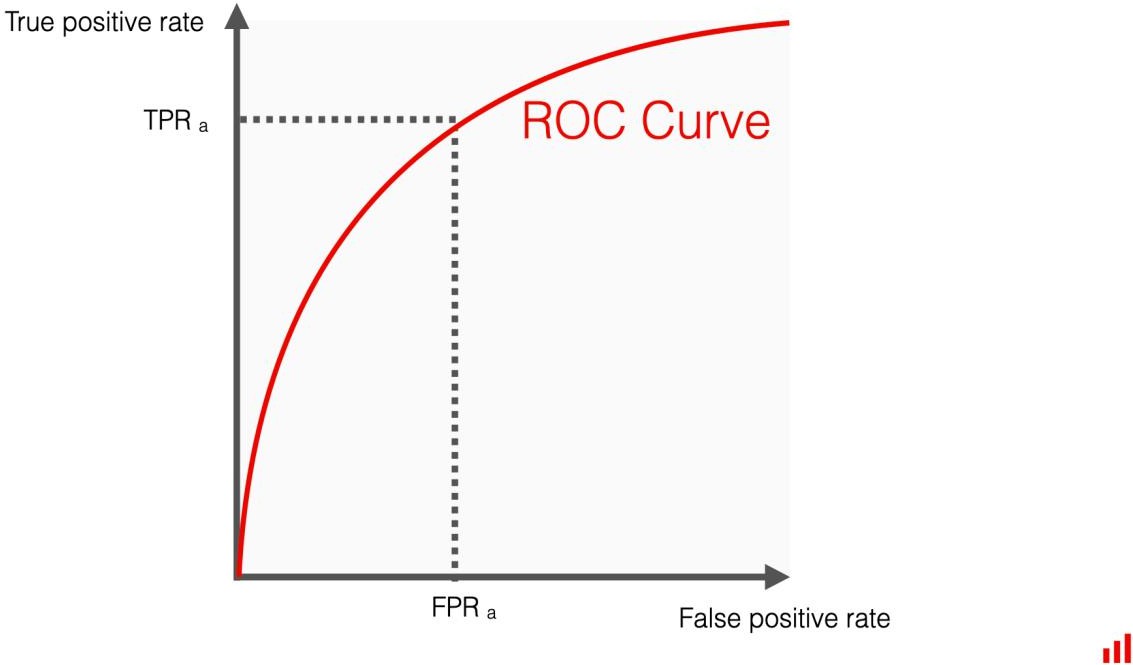


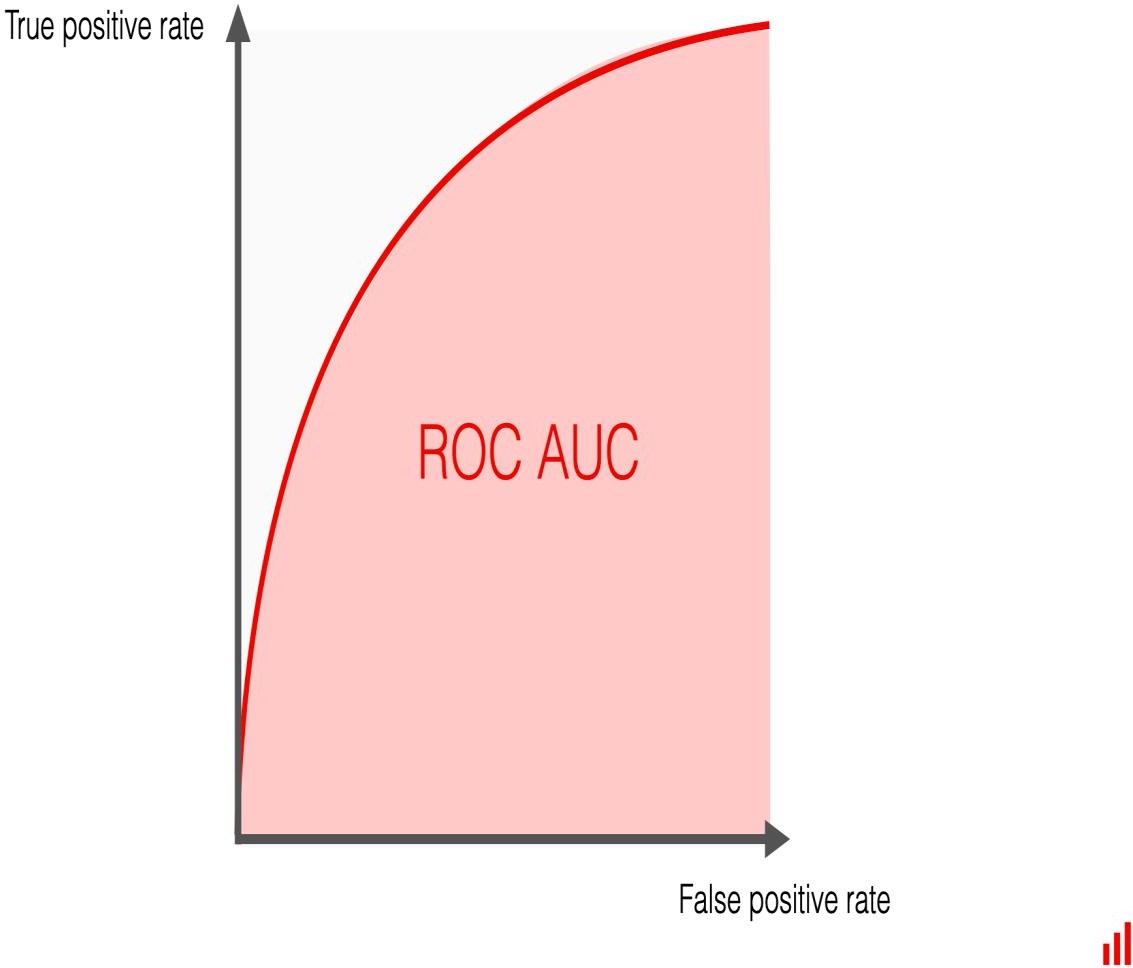
**When to Use:** Relevant when correctly identifying negatives is as important as identifying positives.

1. **ROC-AUC (Receiver Operating Characteristic - Area Under Curve)**
   * **Definition:** Evaluates the trade-off between true positive rate (recall) and false positive rate across various thresholds.
   * **Components:**
     + **ROC Curve:** Plots TPR (y-axis) vs. FPR (x-axis).
     + **AUC:** Area under the ROC curve; a higher value indicates better model performance.
   * **When to Use:** Suitable for binary classification, regardless of class balance.

**Classification threshold**

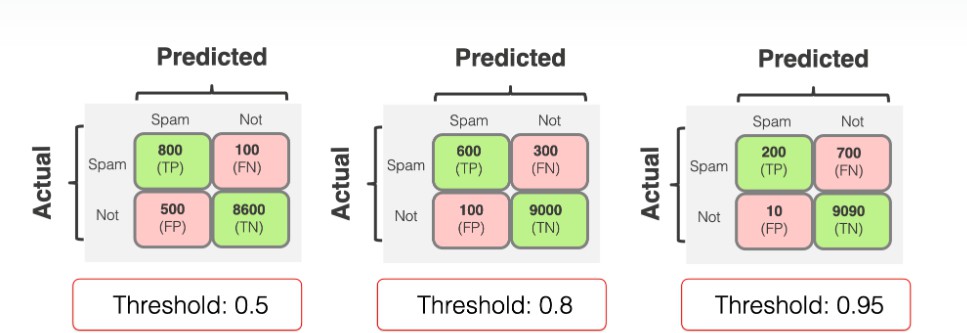
You might ask, what do "different" TPR and FPR values mean? Did we not just calculate them once and for all?

* + In fact, we calculated the values for a given confusion matrix **at a given decision threshold**. But for a probabilistic classification model, these TPR and FPR values are not set in stone.
  + You can vary the decision threshold that defines how to convert the model predictions into labels. This, in turn, can change the number of errors the model makes.

A probabilistic classification model returns a number from 0 to 1 for each object. For

example, for each email, it predicts how likely this email is spam. For a given email, it can be 0.1, 0.55, 0.99, or any other number.

You then have to decide at which probability you **convert this prediction to a label**. For instance, you can label all emails with a predicted probability of over 0.5 as spam. Or, you can only apply this decision when the score is 0.8 or higher.

This choice is what sets the classification threshold.

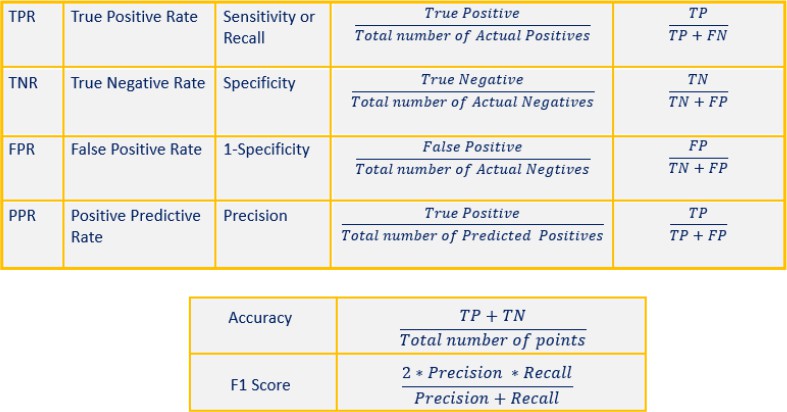
When you **set the threshold higher**, you make the model "more conservative." It assigns

the *True* label when it is "more confident." But as a consequence, you typically lower recall: you detect fewer examples of the target class overall.

When you **set the threshold lower**, you make the model "less strict." It assigns the *True* label more often, even when "less confident." Consequently, you increase recall: you will detect more examples

of the target class. However, this may also lead to lower precision, as the model may make more False Positive predictions.

TPR and FPR change in the same direction. The higher the recall (TPR), the higher the rate of false positive errors (FPR). The lower the recall, the fewer false alarms the model gives.



**Choosing Metrics**

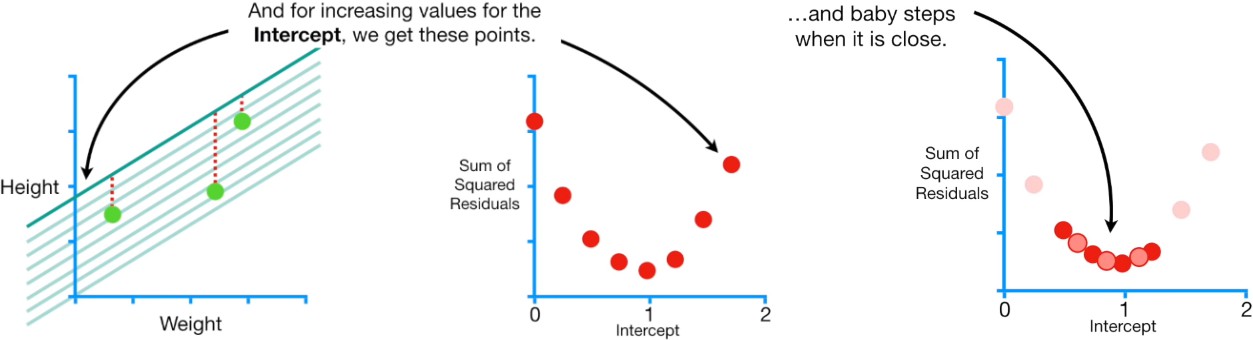
* + **Balanced Data:** Accuracy, F1 Score, Precision, Recall.
  + **Imbalanced Data:** F1 Score, Precision, Recall, MCC, Specificity, ROC-AUC.
  + **Probability-Based Models:** Log Loss, ROC-AUC.

**Supervised Learning -Classification**

**Introduction to Gradient Descent**

Gradient Descent is an optimization algorithm widely used in machine learning and deep learning to minimize a cost function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. It is a core concept for training machine learning models, particularly in supervised learning tasks like regression and classification.

In statistics, Machine Learning and other Data Science fields, we optimize a lot of stuff. For example in linear regression, we optimize the Intercept and Slope, or when we use Logistic Regression we optimize the squiggle. A good example is the Sum of the Squared Residuals in Regression: in Machine Learning lingo this is a type of Loss Function. The Residual is the difference between the Observed Value and the Predicted Value.



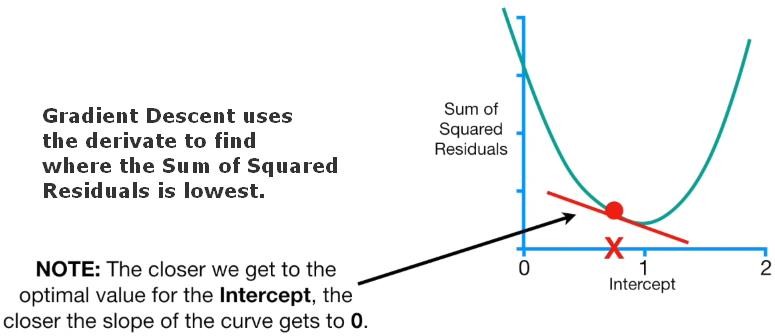
The figure above shows on the y-axis the sum of the squared residuals and the x-axis different value for the intercept. The first point on the y-axis represent the sum of the squared residuals when the intercept is equal to zero. We continue to plot point on the graph based on different value of the intercept. The lowest point in the graph has the lowest sum of the squared residuals. Gradient descent identifies the optimal value by taking big steps when we are far away to the optimal sum of the squared residual, and start to make many steps when it is close to the best solution. Than we can calculate the **derivate d** of each point of the function created by the points. In other words, we are taking the derivative of the **Loss Function**.

# Why Gradient Descent?

When training machine learning models, the goal is to minimize a loss function, which quantifies the difference between the predicted and actual outputs. Examples of loss functions include:

* + **Mean Squared Error (MSE):** Used in regression.
  + **Cross-Entropy Loss:** Used in classification.

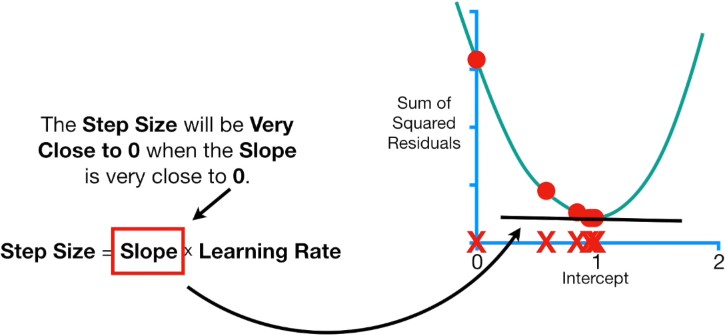
Gradient Descent helps find the set of parameters (weights and biases) that minimize this loss function.



Gradient Descent uses derivative in order to find where the Sum of the Squared Residuals is lowest. The closer we get to the optimal value for the intercept, the closer the slope of the curve gets to zero.

# HOW DOES GRADIENT DESCENT KNOW TO STOP TAKING STEPS?

Gradient Descent stops when the step size is very close to zero, and the step size is very close to zero qhen the **slop size** is close to zero.



* + In practice, the **Minimum Step Size** is equal to **0.001** or smaller. Moreover, Gradient Descent includes a limit on the number of steps it will take before giving up. In practice, the **Maximum Number of Steps** is equal to **1000** or greater.
  + We can also estimate the intercept and the slope simultaneously. We use the Sum of the Squared Residuals as the Loss Function, and we can represent a **3D graph of the Loss Function** for different values of intercept and the slope.
  + We want to find the values for the intercept and slope that give us the minumum Sum of the Squared Residuals.

# Basic Concept

The process of Gradient Descent involves:

1. Calculating the gradient (partial derivative) of the loss function with respect to the model's parameters.
2. Updating the parameters in the opposite direction of the gradient to reduce the loss.

Mathematically, the update rule for a parameter θ

Where:

* + α: Learning rate (step size for updates).
  + J(θ): Loss function.
  + ∂J(θ)/∂θ: Gradient of the loss function with respect to θ.

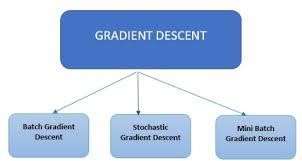
# Steps in Gradient Descent

1. - Take the derivative of the Loss Function for each parameter in it. i.e take the Gradient of the Loss Function.
   1. - Pick random values for the parameters.
   2. - Plug the paramenter values into the derivatives (Gradient).
   3. - Calculate the Step Sizes: Step Size = Slope \* Learning Rate.
   4. - Calculate the New Parameters (e.g. intercept): New Parameter = Old Parameter - Step Size.
   5. - Go back to step 3 and repeat untill Step Size is very small, or when the Maximum Number of Steps is reached.

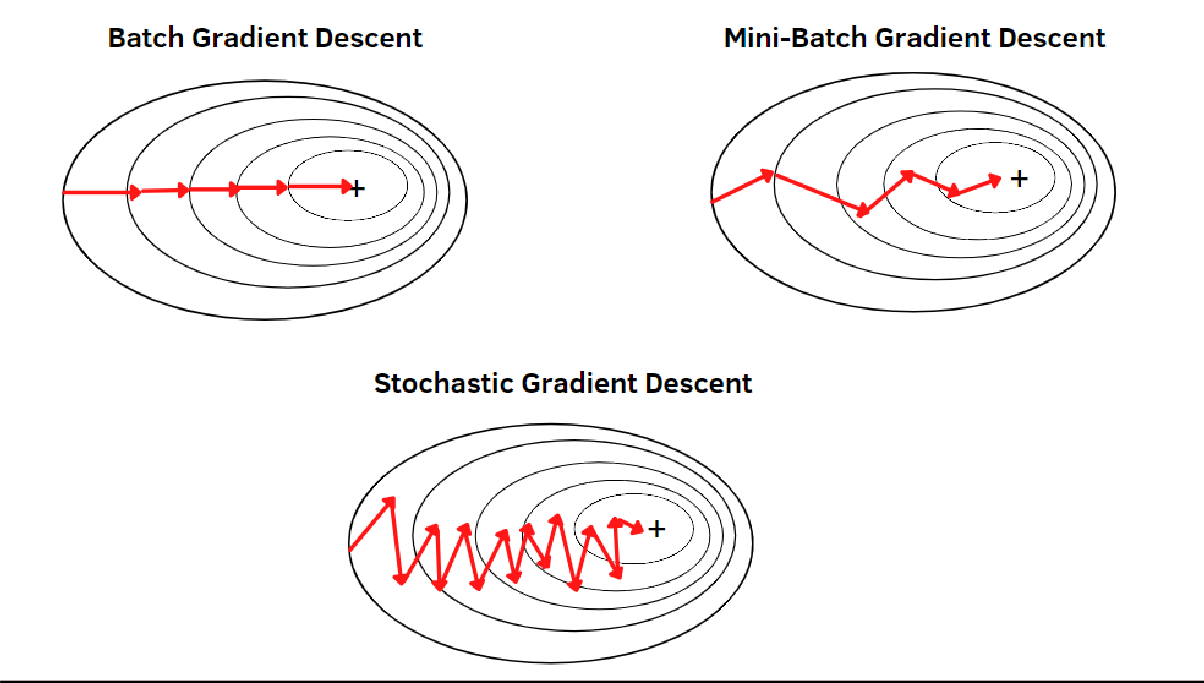
**Learning rate**, used in the calculation of the Step Size is a hyper-parameter that controls how much we are adjusting the weights with respect of the Loss Function. The lower the values, the slower we travel along the downward slope. This make sure that we do not miss any **local minima**. Moreover, the learning rate affects how quickly our model can coverge to a local minina (aka arrive at the best accuracy).

# Types of Gradient Descent

There are three main types of Gradient Descent based on how the data is used:



1. Batch Gradient Descent
   * Uses the entire dataset to compute the gradient.
   * Pros: Converges steadily and guarantees convergence to a minimum.
   * Cons: Computationally expensive for large datasets.
2. Stochastic Gradient Descent (SGD)
   * Uses a single data point (example) to compute the gradient.
   * Pros: Faster updates, suitable for large datasets.
   * Cons: Noisy updates can lead to fluctuation around the minimum.
3. Mini-Batch Gradient Descent
   * Uses a small batch of data (a subset of the dataset) to compute the gradient.
   * Pros: Balances between batch and stochastic methods, commonly used in practice.
   * Cons: Still computationally demanding but efficient.



**Supervised Learning -Classification**

K-Nearest Neighbors (KNN) is a **supervised machine learning algorithm** used for **classification**

and **regression** tasks but mostly it is used for the Classification problems.

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

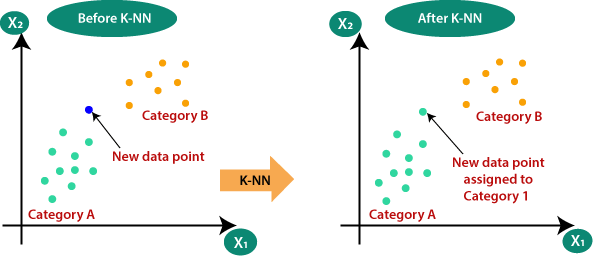
It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.



* Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Why do we need a K-NN Algorithm?

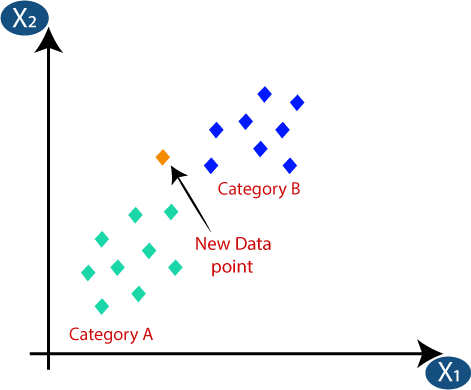
* Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



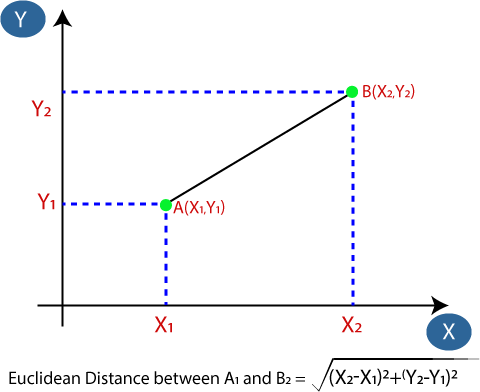
**How does K-NN work?**

The K-NN working can be explained on the basis of the below algorithm:

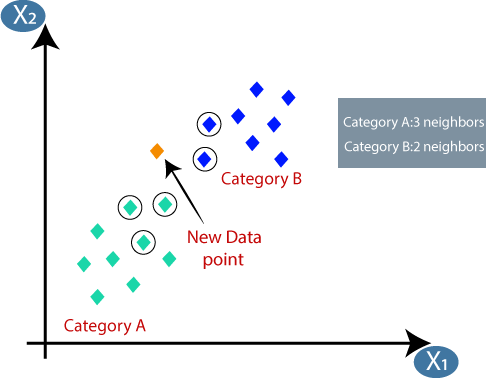
* Step-1: Select the number K of the neighbors
* Step-2: Calculate the Euclidean distance of K number of neighbors
* Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
* Step-4: Among these k neighbors, count the number of the data points in each category.
* Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
* Step-6: Our model is ready.



* Firstly, we will choose the number of neighbors, so we will choose the k=5.
* Next, we will calculate the Euclidean distance between the data points.
* The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image



As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

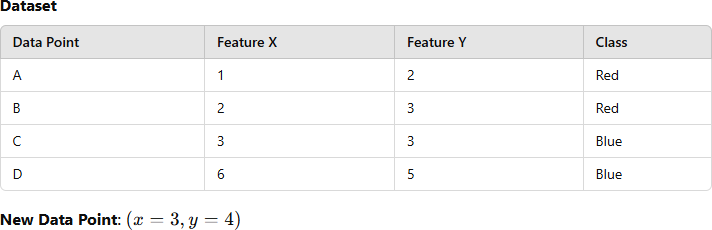
How to select the value of K in the K-NN Algorithm?

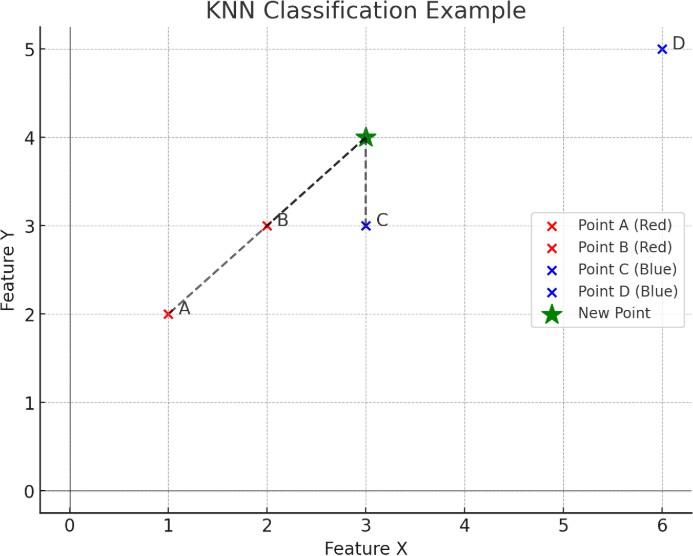
Below are some points to remember while selecting the value of K in the K-NN algorithm:

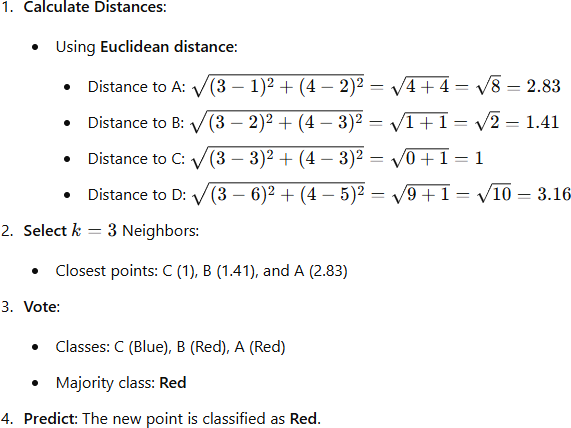
* There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
* A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
* Large values for K are good, but it may find some difficulties.

Example: Classification with KNN

* We want to classify a new data point into one of two classes: *Red* or *Blue*.







# Advantages of KNN

1. Simple and Intuitive: Easy to implement and understand.
2. Non-parametric: Makes no assumptions about data distribution.
3. Versatile: Works for both classification and regression.

# Disadvantages

1. Computationally Expensive: Distance calculations for all training points can be slow for large datasets.
2. Memory-intensive: Requires storing the entire training dataset.
3. Sensitive to Irrelevant Features: Can be affected by noise and irrelevant features.
4. Choice of k: Selection of k can significantly affect performance

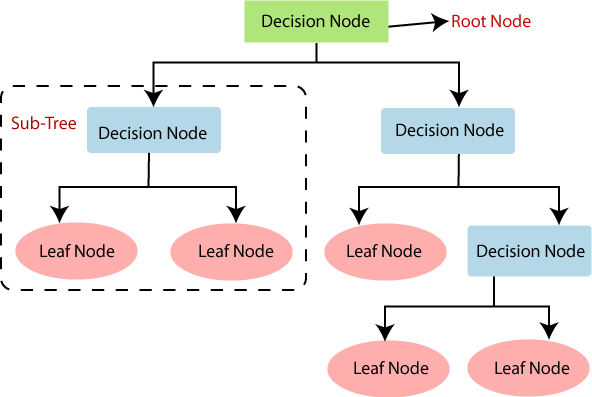
# Applications

* + Handwritten digit recognition
  + Recommender systems
  + Disease classification in healthcare
  + Fraud detection in banking

**Supervised Learning -Classification**

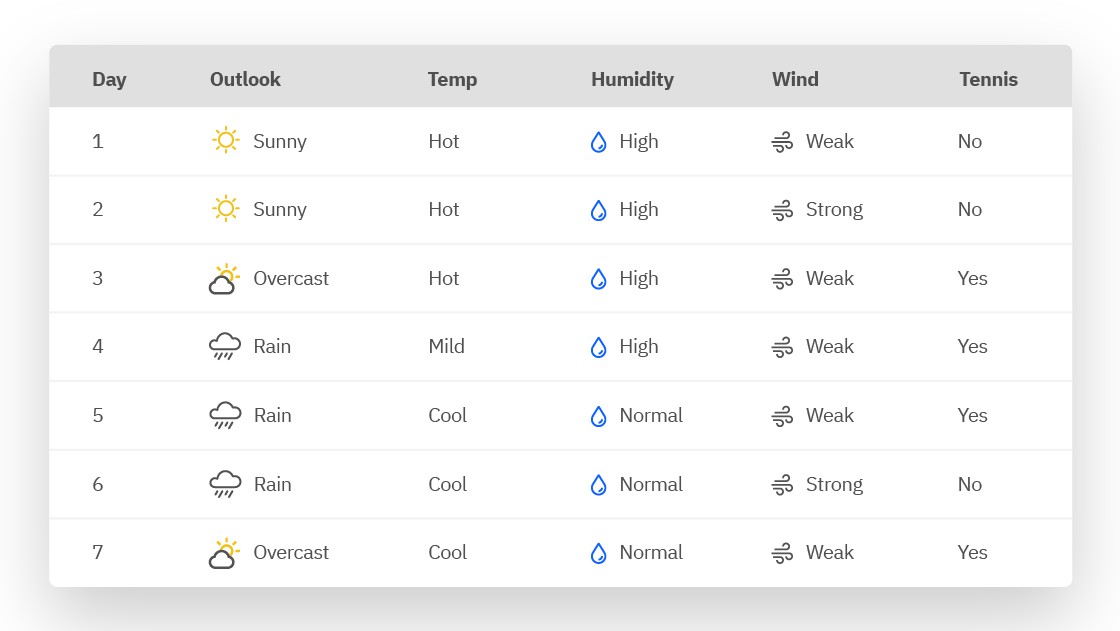
**Classification Algorithm-Decision trees**

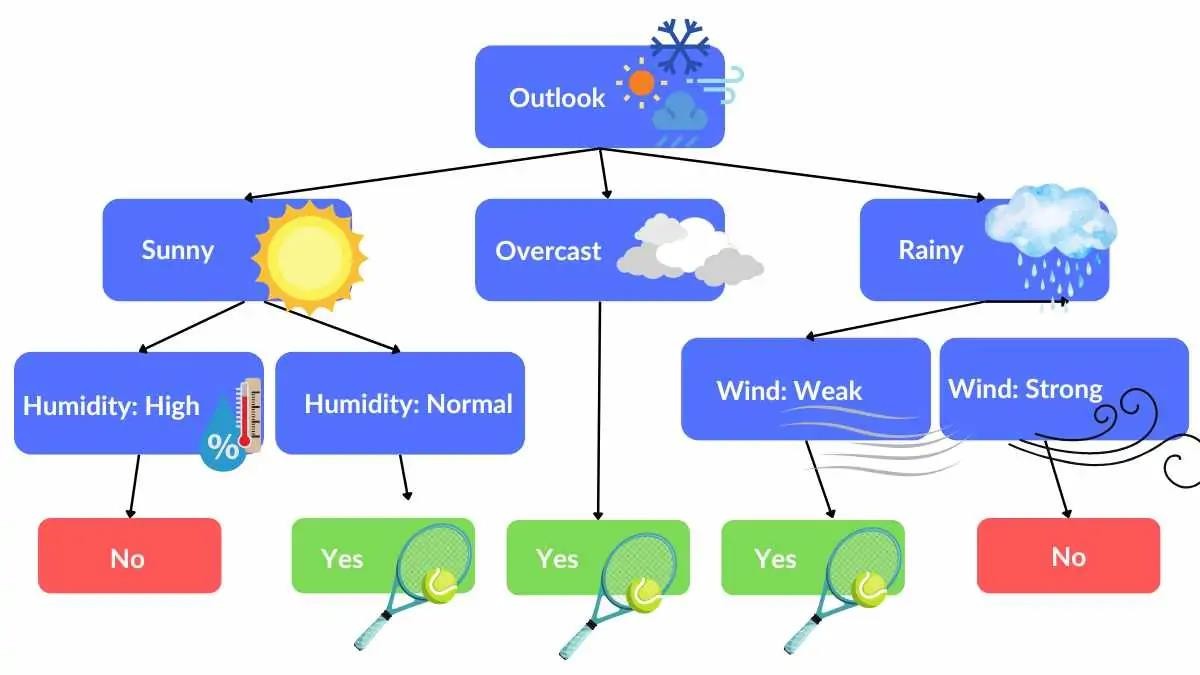
A **Decision Tree** is a supervised machine learning algorithm used for **classification** and **regression** tasks. It is structured like a flowchart, where each internal node represents a decision based on a feature, each branch represents an outcome of the decision, and each leaf node represents a class label (classification) or a value (regression).

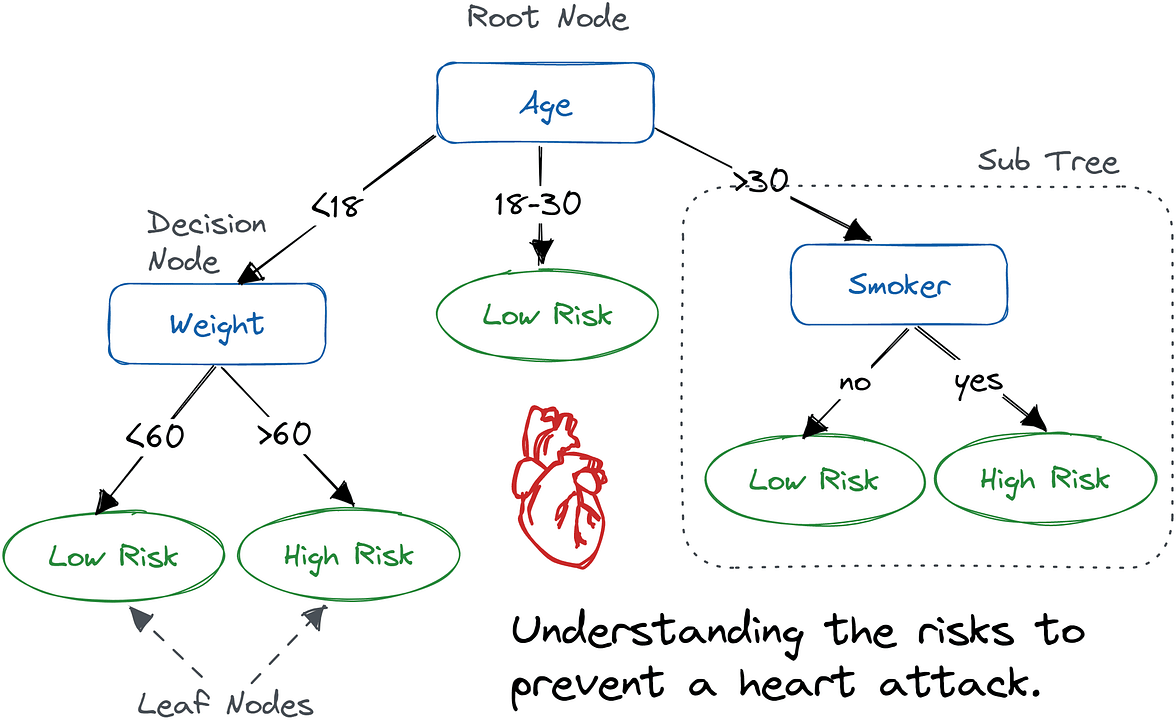


**Structure of a Decision Tree**

1. **Root Node**:
   1. The starting point of the tree.
   2. Represents the entire dataset and splits it based on the best decision rule.
2. **Decision Nodes**:
   1. Intermediate nodes where features are evaluated, and the dataset is split into subsets.
   2. Each split corresponds to a condition on a feature.
   3. **Leaf Nodes**:
      1. Terminal nodes that provide the output (class label or value).
   4. **Branches**:
      1. Pathways connecting nodes, representing the decisions or outcomes of a condition.







**How Decision Trees Work**

1. **Select the Best Feature to Split**:
   1. Use a splitting criterion like **Gini Index**, **Information Gain**.
   2. Choose the feature that best separates the data into pure subsets.
2. **Split the Dataset**:
   1. Divide the dataset based on the selected feature and its values.
3. **Repeat Recursively**:
   1. For each subset, repeat the process until:
      1. The tree reaches a maximum depth.
      2. The dataset in a node is "pure" (only one class remains).
      3. A minimum number of samples per leaf node is reached.
      4. **Output**:
         1. For classification: Assign the majority class in the leaf node.
         2. For regression: Assign the average value in the leaf node.

**Splitting Criteria**

1. **Gini Index (Used in CART):**
   * Measures impurity of a dataset. Formula:



* + where pi is the proportion of samples belonging to class i.

1. **Information Gain (Used in ID3):**
   * Measures the reduction in entropy after a split. Formula:



**Advantages of Decision Trees**

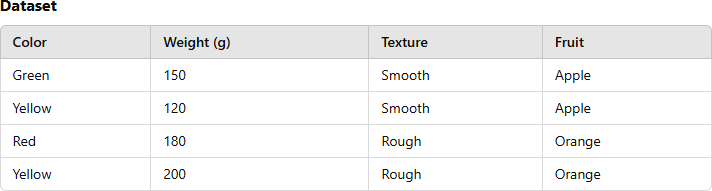
1. **Easy to Understand**: Visual representation is intuitive and interpretable.
2. **Handles Non-linear Relationships**: Can model complex decision boundaries.
3. **No Assumptions About Data**: Works well for both categorical and numerical data.
4. **Feature Importance**: Identifies the most important features.

**Disadvantages**

1. **Overfitting**: Deep trees can model noise, reducing generalization.
2. **Instability**: Small changes in data can lead to a completely different tree.
3. **Bias**: Trees tend to favor features with more splits (numerical features).

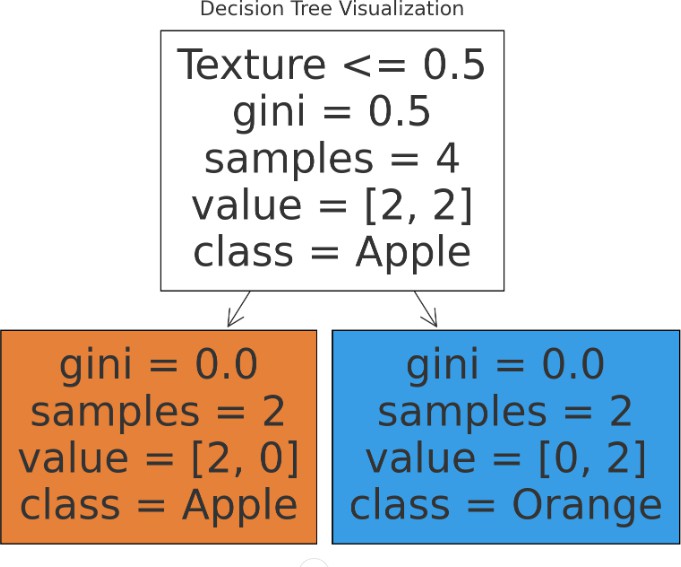
**Example: Classification with Decision Tree**

1. **Scenario**
2. We want to classify fruits based on their attributes.



**Tree Construction**

1. **Root Node**: Split based on "Color".
2. **Decision Nodes**: Further split based on "Weight" or "Texture".
3. **Leaf Nodes**: Classify as Apple or Orange.



**Types of Decision Trees**

* **ID3:** Ross Quinlan is credited within the development of ID3, which is shorthand for “Iterative Dichotomiser 3.” This algorithm leverages entropy and information gain as metrics to evaluate candidate splits.
* **C4.5:** This algorithm is considered a later iteration of ID3, which was also developed by Quinlan. It can use information gain or gain ratios to evaluate split points within the decision trees.
* **CART:** The term, CART, is an abbreviation for “classification and regression trees” and was introduced by Leo Breiman. This algorithm typically utilizes Gini impurity to identify the ideal

attribute to split on. Gini impurity measures how often a randomly chosen attribute is misclassified. When evaluating using Gini impurity, a lower value is more ideal.

**Supervised Learning -Classification**

**Classification Algorithm: Naïve bayes Classifier**

The Naive Bayes classifier is a probabilistic machine learning algorithm based on **Bayes' Theorem**. It is particularly suited for classification problems and is widely used in text classification (e.g., spam detection, sentiment analysis) and other areas requiring a simple yet effective classifier.

**"Naive" Assumption**

The term "naive" comes from the assumption that all features are **independent** given the class label. In reality, this assumption is often not true, but the classifier still works well in many applications.

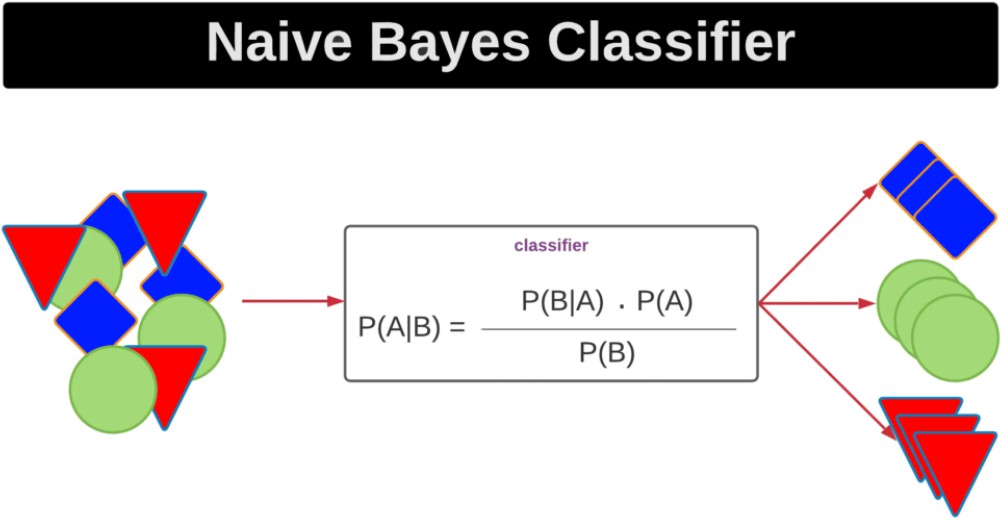
1. **Bayes' Theorem**

Bayes' Theorem provides a way to calculate the probability of a hypothesis (class) based on prior knowledge and evidence:



Where:

* + P(C∣X): Posterior probability — the probability of class C given the features X.
  + P(X∣C): Likelihood — the probability of features X given the class C.
  + P(C): Prior probability — the initial probability of class C (before seeing the features).
  + P(X): Marginal probability — the total probability of the features X.



**How Naive Bayes Works**

The classifier uses the probabilities from Bayes' Theorem to predict the most likely class for a given instance. Here's the step-by-step process:

**Training Phase**:

* + Compute the **prior probability** P(C) for each class C from the training data.
  + Compute the **likelihood** P(X∣C) for each feature X given the class C.
  + If the data is continuous, probabilities are computed using distributions like Gaussian. For categorical data, probabilities are calculated as frequencies.



**Prediction Phase**:

* + Given a new instance X={x1,x2,…,xn} = {x1,x2,…,xn}, calculate the posterior probability for each class C:



* + Assign the class with the highest posterior probability:

**Types of Naive Bayes Classifiers**

1. **Gaussian Naive Bayes**:
   1. Assumes the data follows a Gaussian (normal) distribution.
   2. Used for continuous data.
2. **Multinomial Naive Bayes**:
   1. Assumes feature frequencies are drawn from a multinomial distribution.
   2. Commonly used for document classification (e.g., bag-of-words model).
   3. **Bernoulli Naive Bayes**:
      1. Features are binary (presence/absence of a feature).
      2. Suitable for binary data.

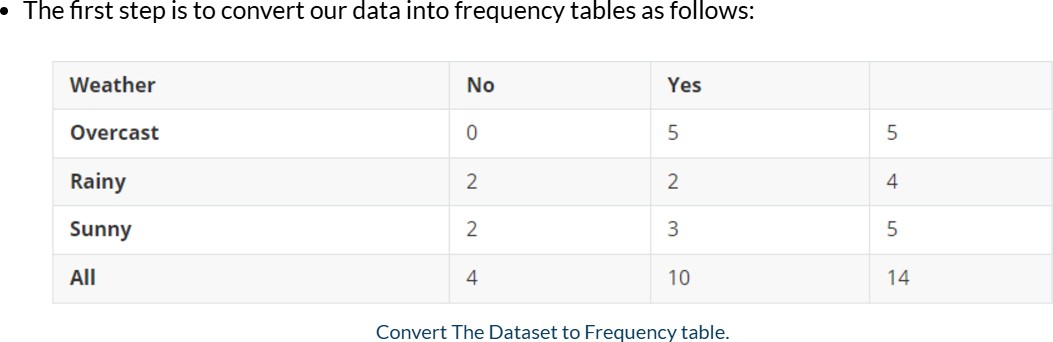
**Example: Spam Email Classification**

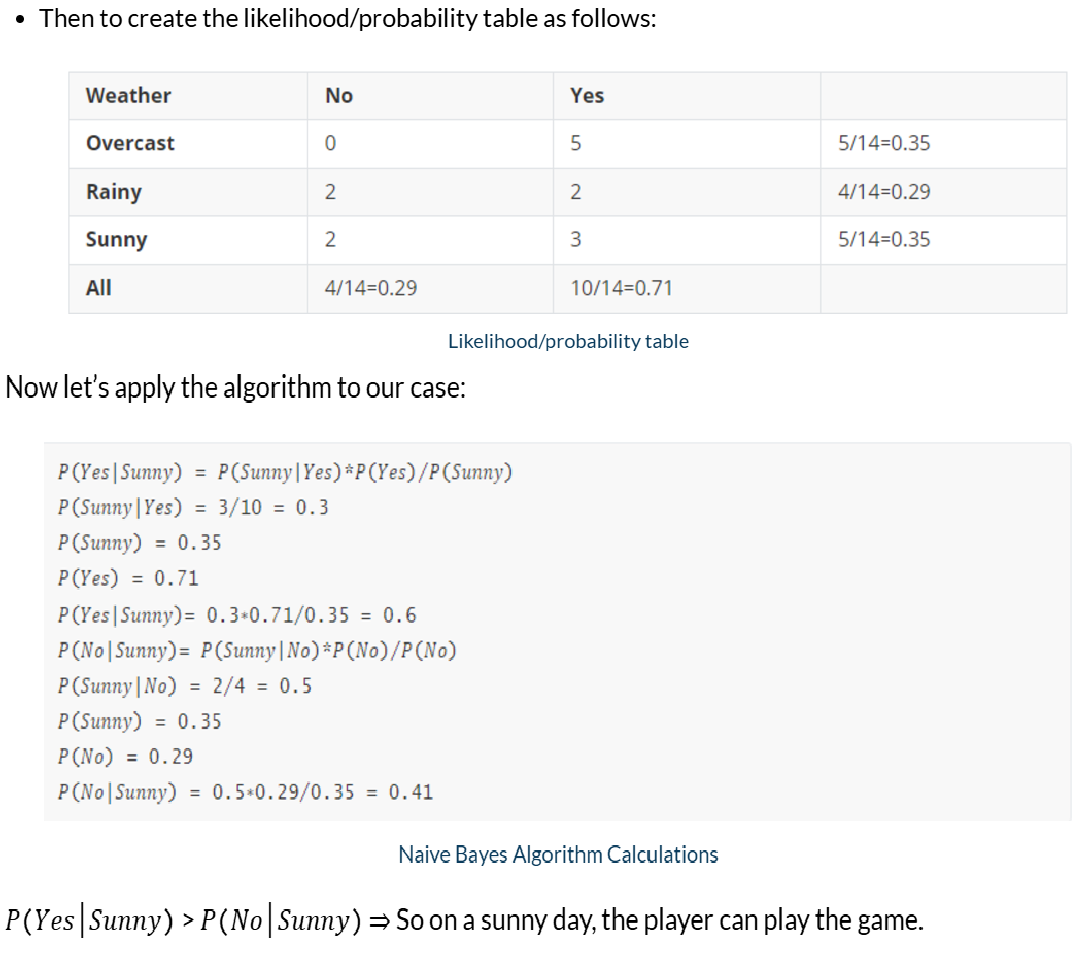
1. **Training**:
   1. Classify emails into "Spam" or "Not Spam".
   2. Compute P(Spam)and P(NotSpam).
   3. Calculate the likelihood P(word∣Spam) and P(word∣NotSpam) for all words.
2. **Prediction**:
   1. For a new email, calculate P(Spam∣Email)and P(NotSpam∣Email).
   2. Classify the email based on the higher posterior probability.
   3. The Naive Bayes consists of two words: 1- Naive: As it assumes the independency between traits or features. 2- Bayes: Based on Bayes’ theorem.
   4. To use the algorithm:
      1. We must convert the presented data set into frequency tables
      2. Then create a probability table by finding the probabilities of certain features.
      3. Then use Bayes’ theorem in order to calculate the posterior probability.

Given the following dataset:

* For example, let’s solve the following problem: If the weather is sunny, then the Player should play or not?







**Advantages**

1. **Simple and Fast**: Easy to implement and computationally efficient.
2. **Scalable**: Handles large datasets well.
3. **Performs Well with Small Data**: Works well even with limited training data.
4. **Handles High-Dimensional Data**: Often used for text classification with thousands of features.

**Disadvantages**

1. **Independence Assumption**: Assumes features are independent, which is rarely true in real- world data.
2. **Continuous Variables**: May not perform well if the data distribution doesn't match the assumed Gaussian distribution in the case of Gaussian Naive Bayes.

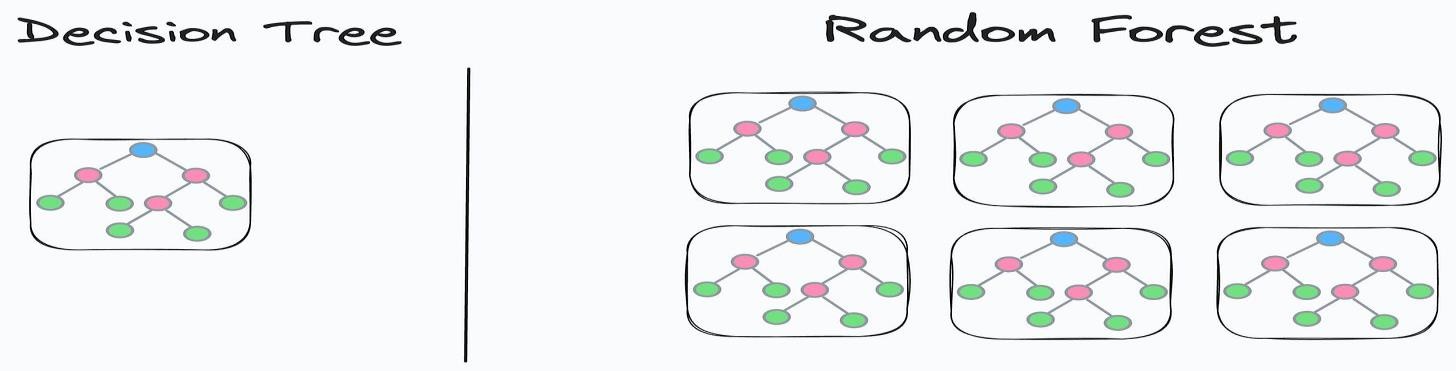
**Applications**

* + Text classification (e.g., sentiment analysis, spam detection).
  + Medical diagnosis.
  + Recommendation systems.
  + Face recognition.

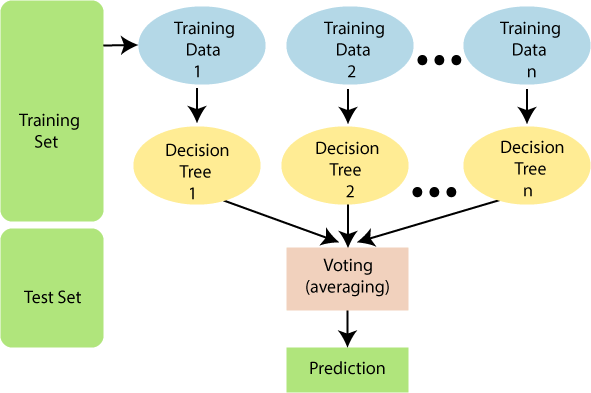
**Supervised Learning -Classification**

**Classification Algorithm: Random Forest What is the Random Forest Algorithm?**

Random Forest algorithm is a powerful tree learning technique in Machine Learning. It works by creating a number of Decision Trees during the training phase. Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition. This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance**.**

****

In prediction, the algorithm aggregates the results of all trees, either by voting (for classification tasks) or by averaging (for regression tasks) This collaborative decision-making process, supported by multiple trees with their insights, provides an example stable and precise results. Random forests are widely used for classification and regression functions, which are known for their ability to handle complex data, reduce overfitting, and provide reliable forecasts in different environments.



# What are Ensemble Learning models?

Ensemble learning models work just like a group of diverse experts teaming up to make decisions – think of them as a bunch of friends with different strengths tackling a problem together.

Similarly, in ensemble learning, different models, often of the same type or different types, team up to enhance predictive performance. It's all about leveraging the collective wisdom of the group to overcome individual limitations and make more informed decisions in various machine learning tasks. Some popular ensemble models

include- XGBoost, AdaBoost, LightGBM, Random Forest, Bagging, Voting etc.

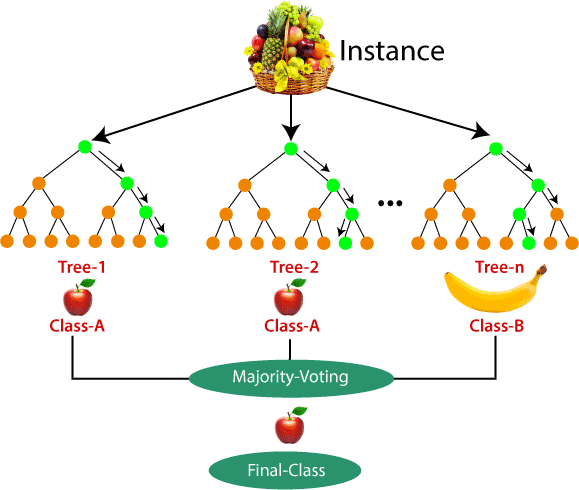
# What is Bagging and Boosting?

Bagging is an ensemble learning model, where multiple week models are trained on different subsets of the training data. Each subset is sampled with replacement and prediction is made by averaging the prediction of the week models for regression problem and considering majority vote for classification problem.

Boosting trains multiple based models sequentially. In this method, each model tries to correct the errors made by the previous models. Each model is trained on a modified version of the dataset, the instances that were misclassified by the previous models are given more weight. The final prediction is made by weighted voting.

# How Does Random Forest Work?

* **Ensemble of Decision Trees:** Random Forest leverages the power of ensemble learning by constructing an army of Decision Trees. These trees are like individual experts, each specializing in a particular aspect of the data. Importantly, they operate independently, minimizing the risk of the model being overly influenced by the nuances of a single tree.
* **Random Feature Selection:** To ensure that each decision tree in the ensemble brings a unique perspective, Random Forest employs random feature selection. During the training of each tree, a random subset of features is chosen. This randomness ensures that each tree focuses on different aspects of the data, fostering a diverse set of predictors within the ensemble.
* **Bootstrap Aggregating or Bagging:** The technique of bagging is a cornerstone of Random Forest's training strategy which involves creating multiple bootstrap samples from the original dataset, allowing instances to be sampled with replacement. This results in different subsets of data for each decision tree, introducing variability in the training process and making the model more robust.
* **Decision Making and Voting:** When it comes to making predictions, each decision tree in the Random Forest casts its vote. For classification tasks, the final prediction is determined by the mode (most frequent prediction) across all the trees. In regression tasks, the average of the individual tree predictions is taken. This internal voting mechanism ensures a balanced and collective decision-making process.



# Key Features of Random Forest

* **Handles Missing Values**: Random Forest can handle missing values effectively by using surrogate splits.
* **Handles High Dimensionality**: It works well even with a large number of features.
* **Feature Importance**: It provides a measure of feature importance, helping to understand which variables contribute most to predictions.
* **Robustness to Noise**: Randomness in data and feature selection reduces the impact of noisy or irrelevant features.
* **Overfitting Prevention**: Ensemble methods and random feature selection reduce overfitting compared to a single decision tree.

Let’s consider a **real-world example** where a Random Forest model is used to predict whether a loan applicant will default based on their financial and demographic

information. Below is an explanation, followed by a flowchart to visualize the Random Forest process.

# Steps in Random Forest Process

1. **Data Preprocessing**:
   * Clean the data, handle missing values, and normalize numerical features.

# Bootstrapping:

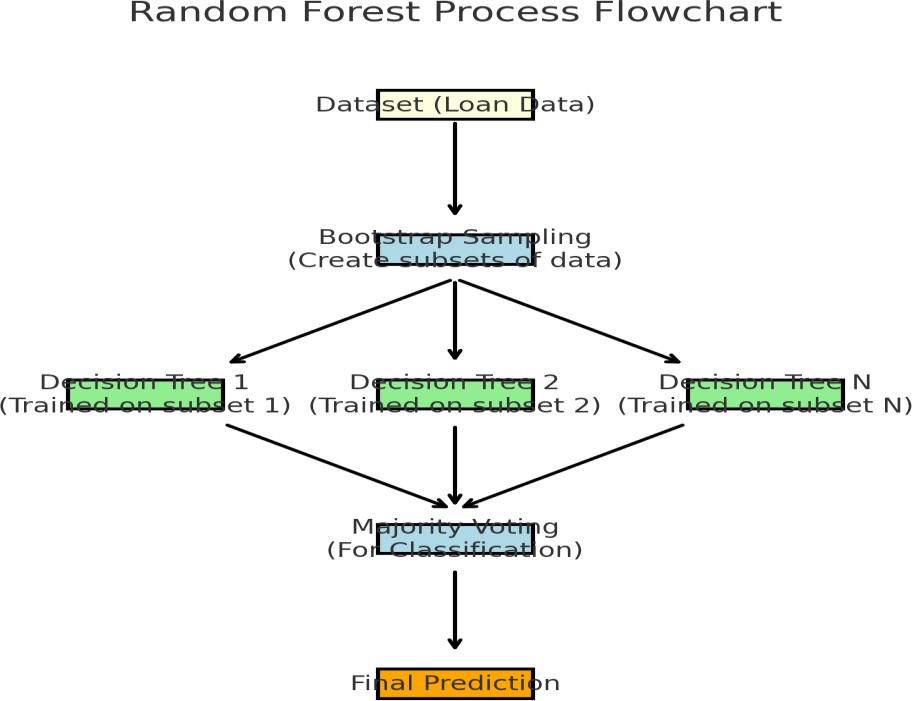
* + Random subsets of the training data are created using sampling with replacement.

# Training Individual Trees:

* + For each bootstrap sample:
    - A decision tree is trained using a random subset of features (e.g., "Income" and "Credit Score").
    - The tree predicts whether an applicant defaults.

# Ensemble Prediction:

* + For classification:
    - Each tree in the forest votes for a class (Yes or No).
    - The class with the majority votes is selected.
  + For regression:
    - Average the predictions of all trees.



# Dataset Description

We have a dataset with the following features:

1. Age: Age of the applicant.
2. Income: Monthly income of the applicant.
3. Credit Score: The applicant's credit score.
4. Loan Amount: The amount of loan applied for.
5. Loan Tenure: Duration of the loan in years. The target variable is:
   * Loan Default: Whether the applicant defaulted on the loan (Yes or No).

# Advantages

* + Random Forest is capable of performing both Classification and Regression tasks.
  + It is capable of handling large datasets with high dimensionality.
  + It enhances the accuracy of the model and prevents the overfitting issue.

# Disadvantages

* + **Computational Cost**: Training a large number of trees can be computationally expensive and memory-intensive.
  + **Interpretability**: While individual trees are interpretable, the overall Random Forest model is more complex and harder to interpret.
  + **Bias with Small Data**: It may not perform as well with very small datasets compared to simpler models.

# Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

**Supervised Learning -Classification**

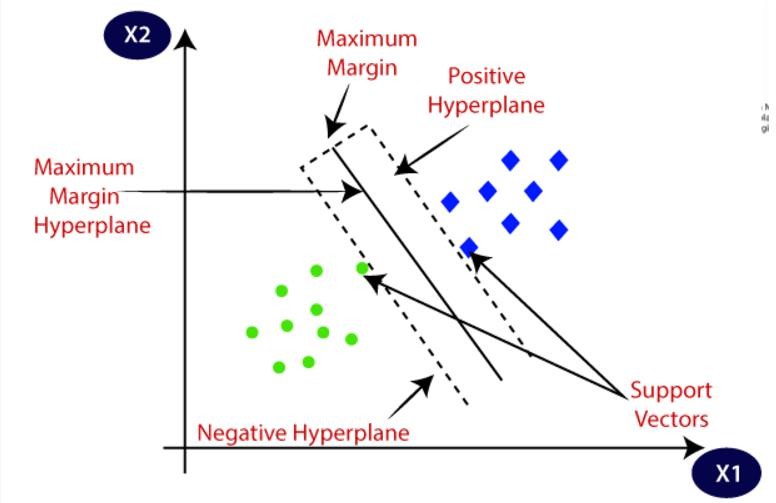
**Classification Algorithm: Support Vector machine**

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-

dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



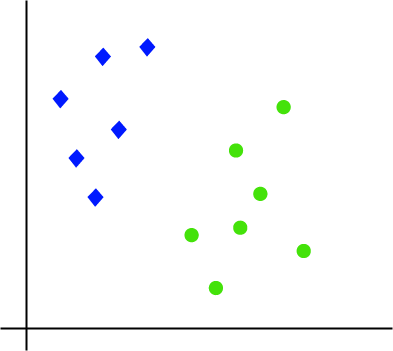
**Types of SVM**

**SVM can be of two types:**

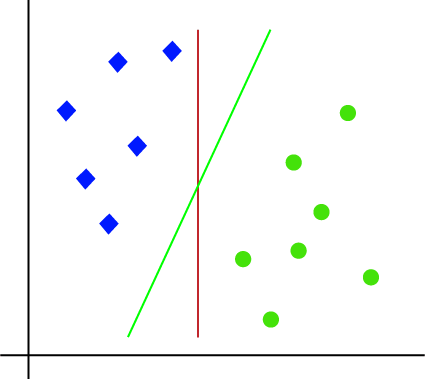
* **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
* **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

**How does SVM works?**

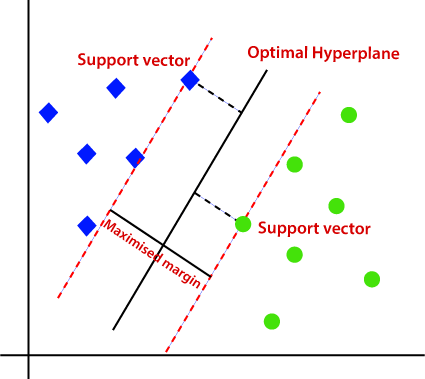
* **Linear SVM:**
* The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image

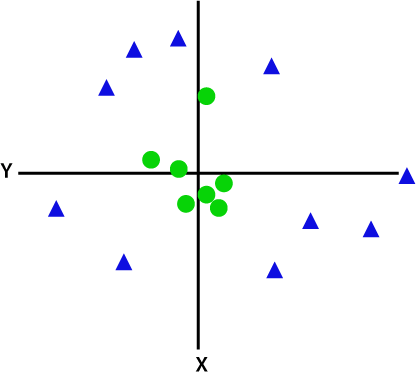


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



**Non-Linear SVM:**

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:

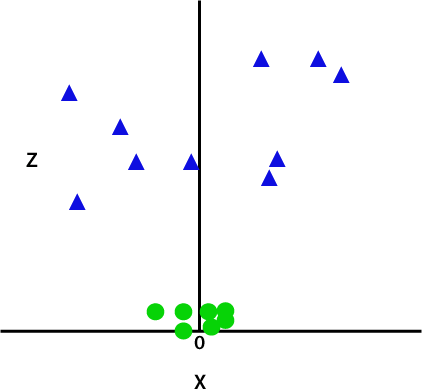


So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated

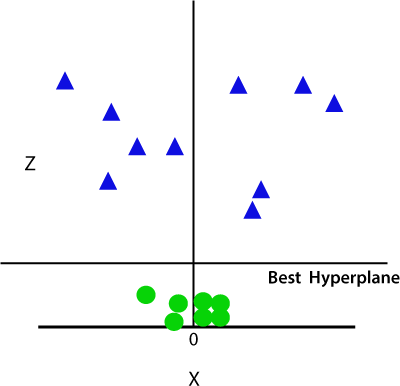
as:

*z=x2 +y2*

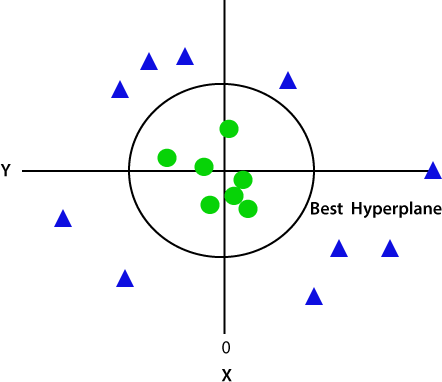
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

**Dataset Example**

* **Features**:
* Number of spam words (f1)
* Frequency of specific keywords (f2)
* Length of the email (f3)
* Other metrics.
* **Labels**:
  + Spam (1)
  + Not Spam (0)

**Training Process**

1. Input features and labels into the SVM algorithm.
2. The SVM calculates the hyperplane to separate Spam from Not Spam.
3. Support Vectors (emails close to the boundary) define the margin.

**Prediction**

Given a new email, SVM calculates its position relative to the hyperplane and predicts whether it’s Spam or Not Spam.

**Advantages**

1. **Effective in High-Dimensional Spaces**
   1. SVM performs well in cases where the number of dimensions (features) is greater than the number of samples.
   2. **Works Well for Clear Margin of Separation**
      1. It is highly effective for datasets with a clear boundary between classes.
   3. **Versatile with Kernel Functions**
      1. Using different kernel functions, SVM can handle both linear and non-linear classification tasks.
   4. **Robust to Overfitting (Especially in High Dimensions)**
      1. SVM maximizes the margin, reducing the chance of overfitting, particularly when the number of features is high.
   5. **Effective in Both Classification and Regression Tasks**
      1. While primarily used for classification, it can also perform regression (Support Vector Regression - SVR).

**Disadvantages**

1. **Computationally Intensive**
   1. Training an SVM can be slow for large datasets, especially with complex kernel functions.
   2. **Not Ideal for Large Datasets**
      1. When the number of data points is very large, SVM struggles due to its computational complexity (O(n2) or more).
   3. **Requires Careful Selection of Kernel and Parameters**
      1. The performance depends heavily on choosing the right kernel and tuning parameters like C (regularization) and γ (for RBF kernel).
   4. **Less Effective with Overlapping Classes**
      1. SVM struggles when classes are not well-separated or have significant overlap.

**Applications**

1. **Healthcare**
   * **Disease Diagnosis**: Predicting diseases like cancer (e.g., breast cancer diagnosis using medical imaging data).
   * **Genomics**: Classifying gene expression data.
2. **Finance**
   * **Fraud Detection**: Identifying fraudulent transactions in banking.
   * **Credit Scoring**: Determining credit risk for loans.
3. **Text and Image Recognition**
   * **Spam Filtering**: Classifying emails as spam or not spam.
   * **Handwritten Digit Recognition**: Identifying handwritten characters (e.g., in the MNIST dataset).
   * **Face Detection**: Recognizing and classifying faces in images.