Supervised Learning

Logistic Regression, Multiple Regression, Polynomial Regression, Decision Tree, K-Nearest Neighbor (KNN), Support Vector Machine, Naive Bayes' Classifier

---

❖ SUPERVISED LEARNING :

Supervised learning is a machine learning approach that trains models using labeled data to predict outcomes or classify data. It involves providing the algorithm with input data and the corresponding correct output (the "label"), allowing it to learn the relationship between them. This enables the model to make accurate predictions when presented with new, unseen data.

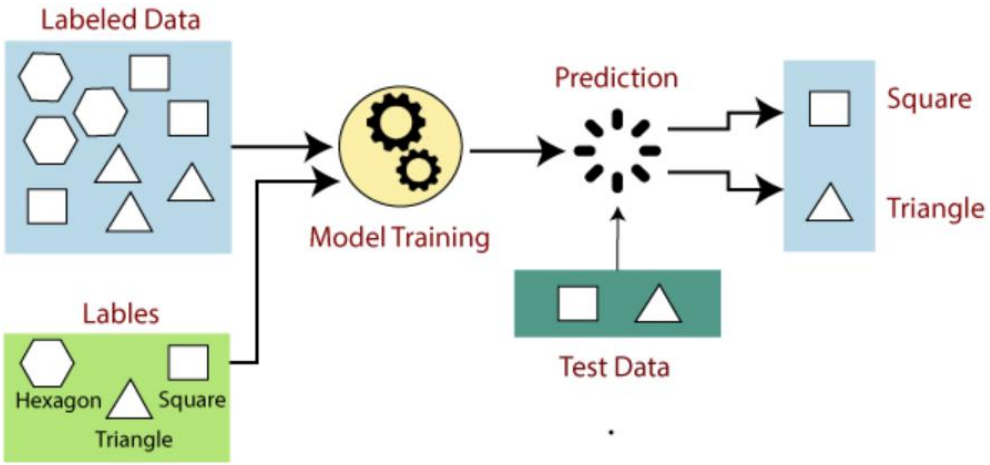
Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y)**.

In the real-world, supervised learning can be used for **Risk Assessment, Image classification, Fraud Detection, spam filtering**, etc.

How Supervised Learning Works?

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram:



Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

- If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
- If the given shape has three sides, then it will be labelled as a **triangle**.
- If the given shape has six equal sides then it will be labelled as **hexagon**.

Now, after training, we test our model using the test set, and the task of the model is to identify the shape.

The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

Detailed explanation

- **Labeled Data:**

Supervised learning relies on datasets where each input data point is paired with its correct output or label. For example, in image classification, the dataset would include images of objects (like cats and dogs) along with labels indicating which image contains a cat and which contains a dog.

- **Training the Model:**

The algorithm learns from this labeled data by identifying patterns and relationships between the input features and the corresponding output labels. This process is often iterative, with the model adjusting its internal parameters to improve accuracy.

- **Making Predictions:**

Once trained, the model can be used to predict outputs for new, unseen data. For example, it can classify new images it has not seen before based on the patterns it learned from the training data.

Steps involved in supervised learning

1. Data Collection and Preparation:

- **Gather labeled data:**

This is the foundation of supervised learning. You need a dataset where each data point has both input features and the corresponding correct output (label).

- **Data preprocessing:**

Clean the data by handling missing values, scaling features, and potentially encoding categorical variables to make it suitable for the chosen algorithm.

- **Split the data:**

Divide the dataset into training, testing, and sometimes validation sets. The training set is used to train the model, the test set to evaluate its performance on unseen data, and the validation set to fine-tune the model.

2. Model Selection:

- **Choose an appropriate algorithm:**

Select a supervised learning algorithm based on the type of problem (e.g., classification, regression) and the characteristics of your data. Common algorithms include linear regression, logistic regression, decision trees, support vector machines, and neural networks.

- **Consider factors:**

When choosing an algorithm, consider the bias-variance tradeoff, function complexity, dimensionality of the input space, and noise in the output values.

3. Model Training:

- **Feed the training data to the model:**

The algorithm learns the relationship between input features and output labels by adjusting its internal parameters.

- **Iterative process:**

Model training is often an iterative process, where the model adjusts its parameters based on the training data to minimize errors.

4. Model Evaluation:

- **Test the model:**

Use the test set (data not used in training) to evaluate the model's performance.

- **Select appropriate metrics:**

Choose metrics relevant to the problem type (e.g., accuracy, precision, recall for classification; mean squared error for regression) to quantify the model's performance.

5. Fine-tuning (Optional):

- **Hyperparameter tuning:**

If the initial evaluation is not satisfactory, adjust the model's hyperparameters (settings that control the learning process) using techniques like grid search or cross-validation.

- **Retrain and re-evaluate:**

Retrain the model with the best hyperparameters and re-evaluate its performance on the test set.

6. Model Deployment:

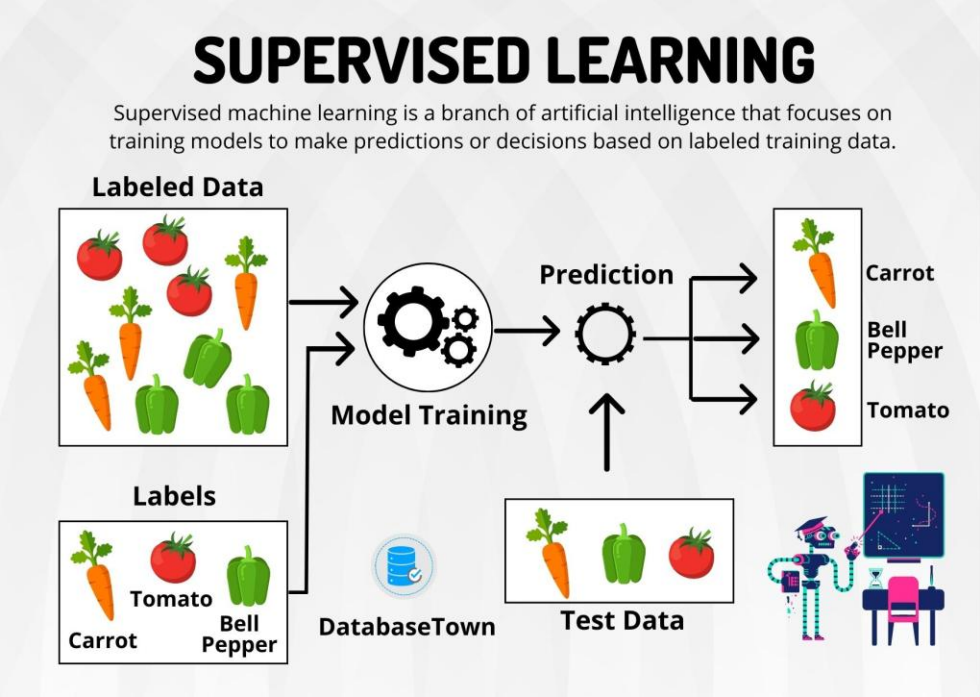
- **Ready for predictions:**

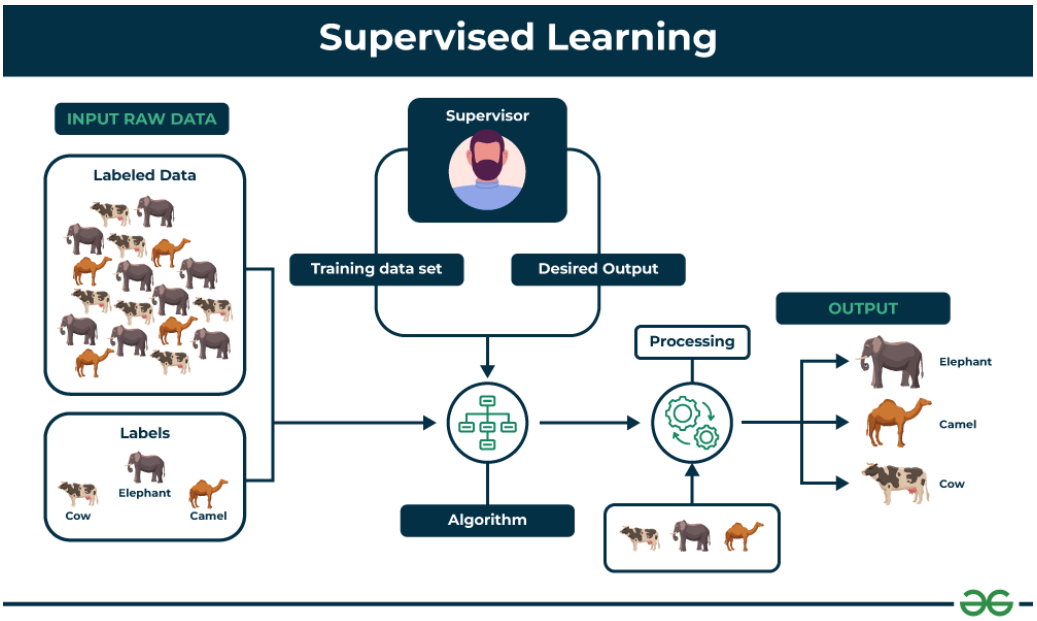
Once the model is trained and evaluated to your satisfaction, it can be deployed to make predictions on new, unseen data.

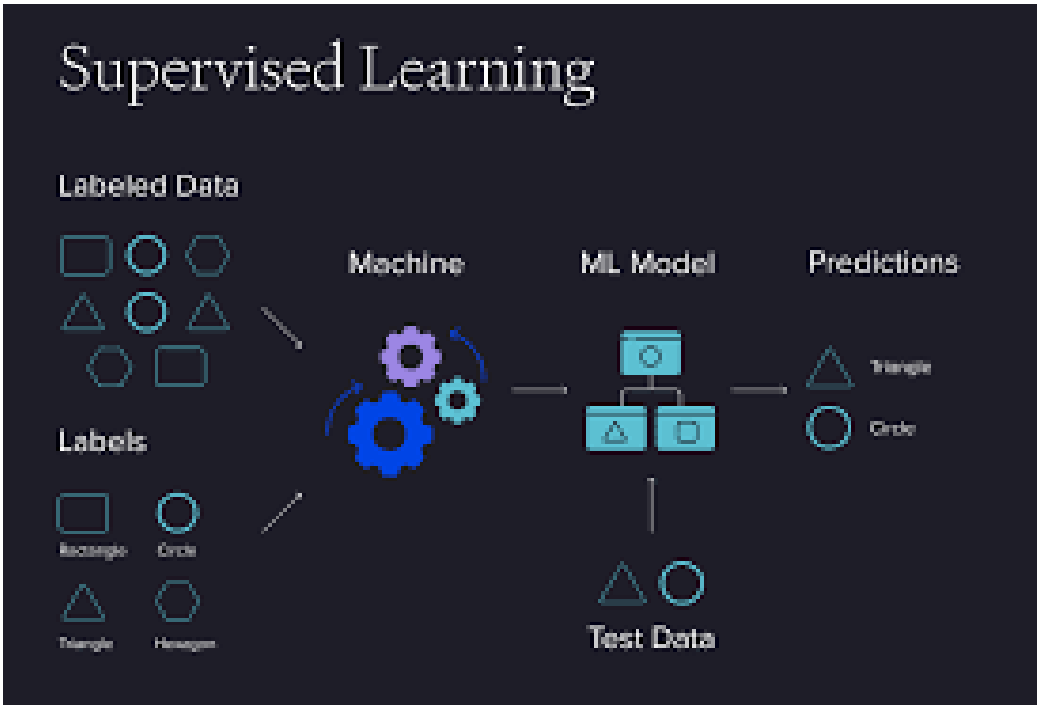
- **Continuous monitoring:**

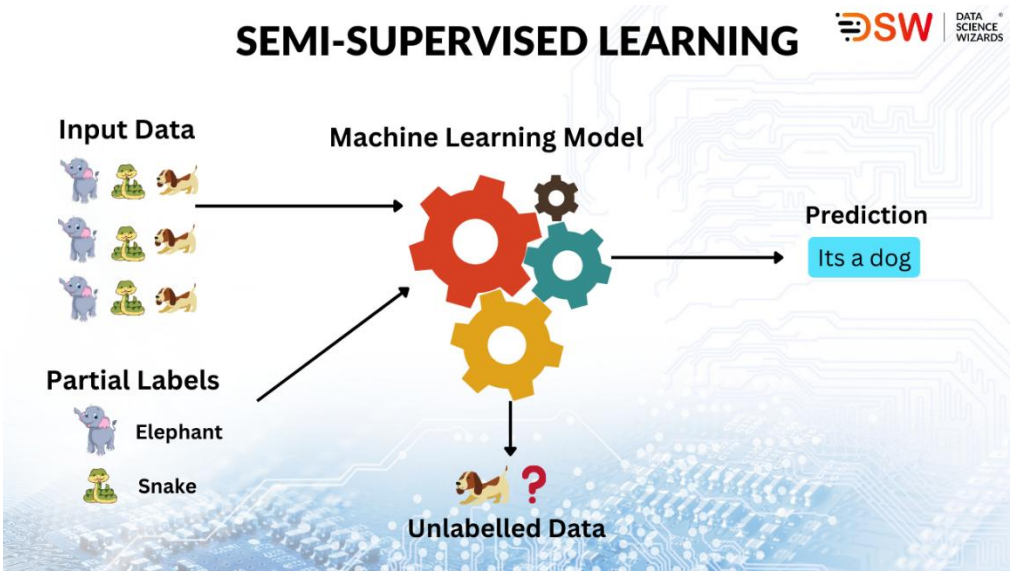
Continuously monitor the model's performance in the real world and retrain it with new data as needed to maintain accuracy and relevance.

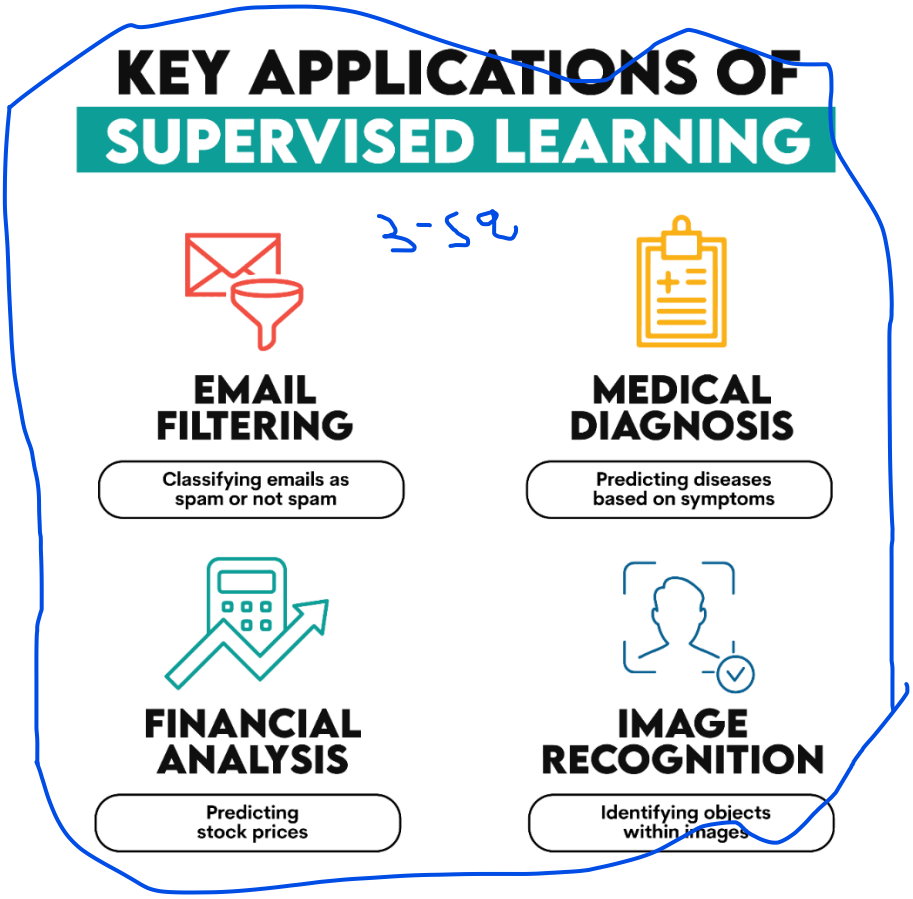
- **Examples**




Supervised Learning
INPUT RAW DATA
Labeled Data
Labels
Cow
Elephant
Camel
Supervisor
Training data set
Desired Output
Algorithm
Processing
OUTPUT
Elephant
Camel
Cow


Supervised Learning
Labeled Data
Labels
Machine
ML Model
Predictions
Test Data


SEMI-SUPERVISED LEARNING
DSW
DATA SCIENCE WIZARDS
Input Data
Machine Learning Model
Prediction
Its a dog
Partial Labels
Elephant
Snake
Unlabelled Data


KEY APPLICATIONS OF
SUPERVISED LEARNING
3-59
EMAIL FILTERING
Classifying emails as spam or not spam
MEDICAL DIAGNOSIS
Predicting diseases based on symptoms
FINANCIAL ANALYSIS
Predicting stock prices
IMAGE RECOGNITION
Identifying objects within images

TYPES OF SUPERVISED LEARNING

Supervised learning, a type of machine learning, primarily involves two main categories: classification and regression. Classification algorithms predict a categorical outcome (e.g., spam/not spam), while regression algorithms predict a continuous numerical value (e.g., house price). Common examples of supervised learning algorithms include linear regression, logistic regression, decision trees, support vector machines (SVM), and neural networks.

Classification:

- **Definition:** Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc.
- Involves assigning input data to predefined categories or classes.
- **Examples:**
  - Spam detection (classifying emails as spam or not spam).
  - Image recognition (classifying images of cats, dogs, etc.).
  - Medical diagnosis (classifying diseases based on symptoms).
- **Algorithms:** Logistic regression, SVM, decision trees, k-nearest neighbors, Naive Bayes, neural networks.

Regression:

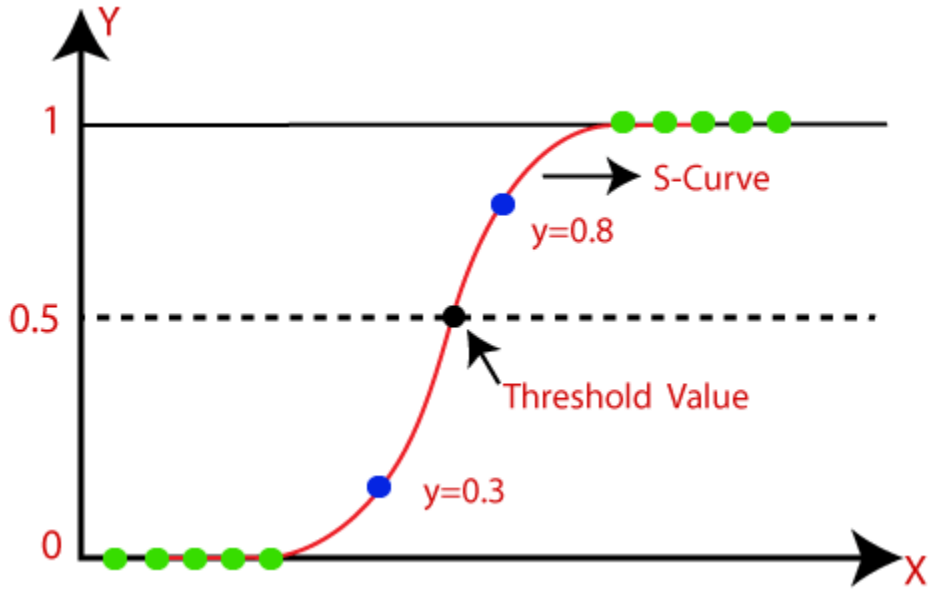
Definition: Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

- Involves predicting a continuous numerical value.
- **Examples:**
  - Predicting house prices based on features like size, location, etc.
  - Forecasting stock prices.
  - Predicting customer churn.
- **Algorithms:** Linear regression, polynomial regression, support vector regression, decision trees, neural networks.

❖ **Logistic Regression:**

Logistic Regression is a supervised machine learning algorithm used for classification problems. Unlike linear regression which predicts continuous values it predicts the probability that an input belongs to a specific class. It is used for binary classification where the output can be one of two possible categories such as Yes/No, True/False or 0/1. It uses sigmoid function to convert inputs into a probability value between 0 and 1

- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.



Understanding Sigmoid Function

1. The sigmoid function is a important part of logistic regression which is used to convert the raw output of the model into a probability value between 0 and 1.
2. This function takes any real number and maps it into the range 0 to 1 forming an "S" shaped curve called the sigmoid curve or logistic curve. Because probabilities must lie between 0 and 1, the sigmoid function is perfect for this purpose.
3. In logistic regression, we use a threshold value usually 0.5 to decide the class label.

- If the sigmoid output is same or above the threshold, the input is classified as Class 1.
- If it is below the threshold, the input is classified as Class 0.

Assumptions of Logistic Regression

Understanding the assumptions behind logistic regression is important to ensure the model is applied correctly, main assumptions are:

1. **Independent observations**: Each data point is assumed to be independent of the others means there should be no correlation or dependence between the input samples.
2. **Binary dependent variables**: It takes the assumption that the dependent variable must be binary, means it can take only two values.
3. **Linearity relationship between independent variables and log odds**: The model assumes a linear relationship between the independent variables and the log odds of the dependent variable which means the predictors affect the log odds in a linear way.
4. **No outliers**: The dataset should not contain extreme outliers as they can distort the estimation of the logistic regression coefficients.
5. **Large sample size**: It requires a sufficiently large sample size to produce reliable and stable results.

Logistic Regression Equation:

- The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:
- We know the equation of the straight line can be written as:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

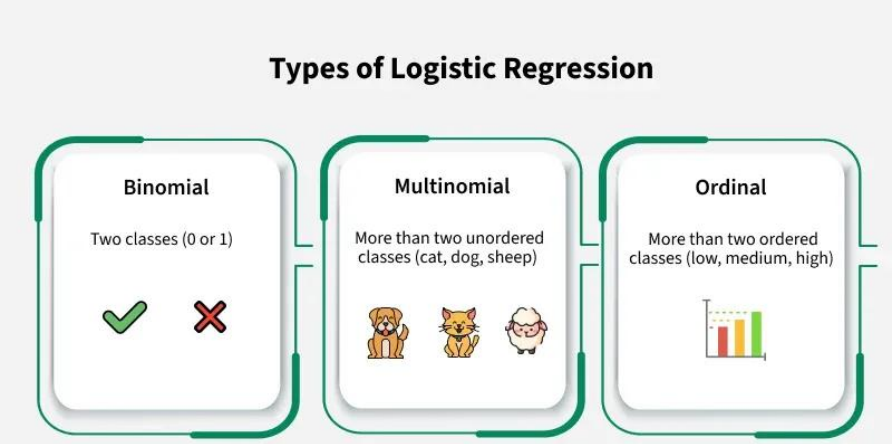
$$\frac{y}{1-y}\text{; 0 for y= 0, and infinity for y=1}$$

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Types of Logistic Regression

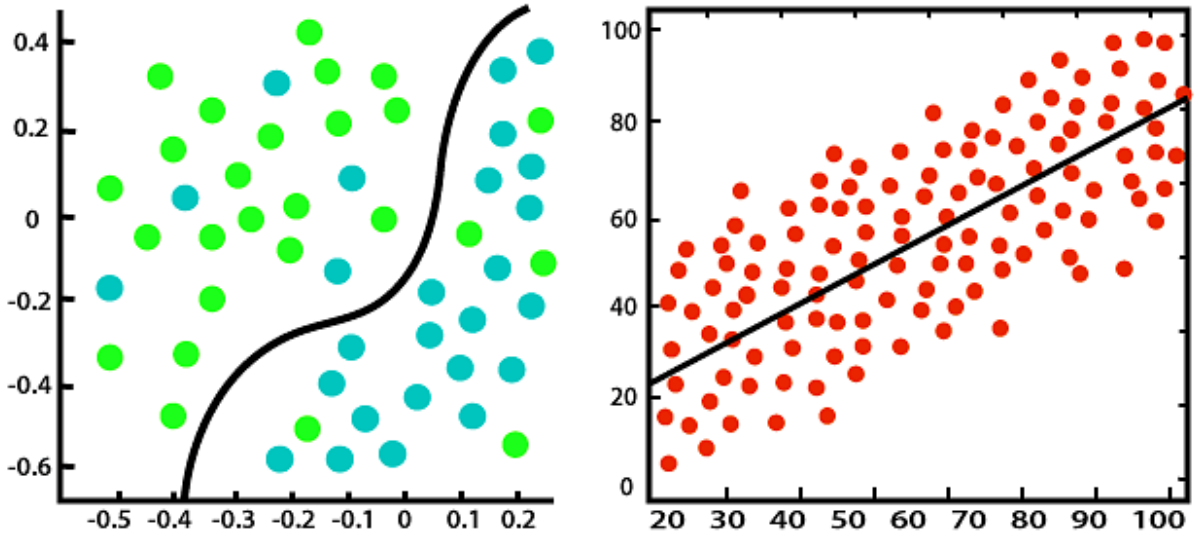


Logistic regression can be classified into three main types based on the nature of the dependent variable:

1. **Binomial Logistic Regression**: This type is used when the dependent variable has only two possible categories. Examples include Yes/No, Pass/Fail or 0/1. It is the most common form of logistic regression and is used for binary classification problems.
2. **Multinomial Logistic Regression**: This is used when the dependent variable has three or more possible categories that are not ordered. For example, classifying animals into categories like "cat," "dog" or "sheep." It extends the binary logistic regression to handle multiple classes.
3. **Ordinal Logistic Regression**: This type applies when the dependent variable has three or more categories with a natural order or ranking. Examples include ratings like "low," "medium" and "high." It takes the order of the categories into account when modeling.

Classification vs Regression in Machine Learning

Classification and regression are two primary tasks in supervised machine learning, where ***key difference lies in the nature of the output: classification deals with discrete outcomes (e.g., yes/no, categories), while regression handles continuous values (e.g., price, temperature).*** *Both approaches* ***require labeled data for training but differ in their objectives****—classification aims to find decision boundaries that separate classes, whereas regression focuses on finding the best-fitting line to predict numerical outcomes. Understanding these distinctions helps in selecting the right approach for specific machine learning tasks.*



For example, it can determine whether an email is spam or not, classify images as "cat" or "dog," or predict weather conditions like "sunny," "rainy," or "cloudy." with decision boundary and regression models are used to predict house prices based on features like size and location, or forecast stock prices over time with straight fit line

What is Regression in Machine Learning?

Regression algorithms predict a continuous value based on input data. This is used when you want to predict numbers such as income, height, weight, or even the probability of something happening (like the chance of rain). Some of the most common types of regression are:

1. **Simple Linear Regression**: Models the relationship between one independent variable and a dependent variable using a straight line.
2. **Multiple Linear Regression**: Predicts a dependent variable based on two or more independent variables.
3. **Polynomial Regression**: Models nonlinear relationships by fitting a curve to the data.

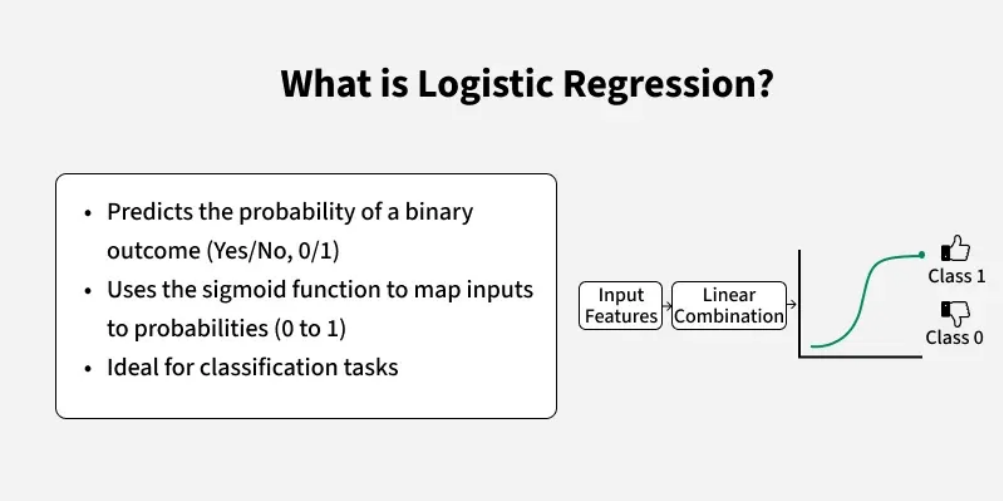
What is Classification in Machine Learning?

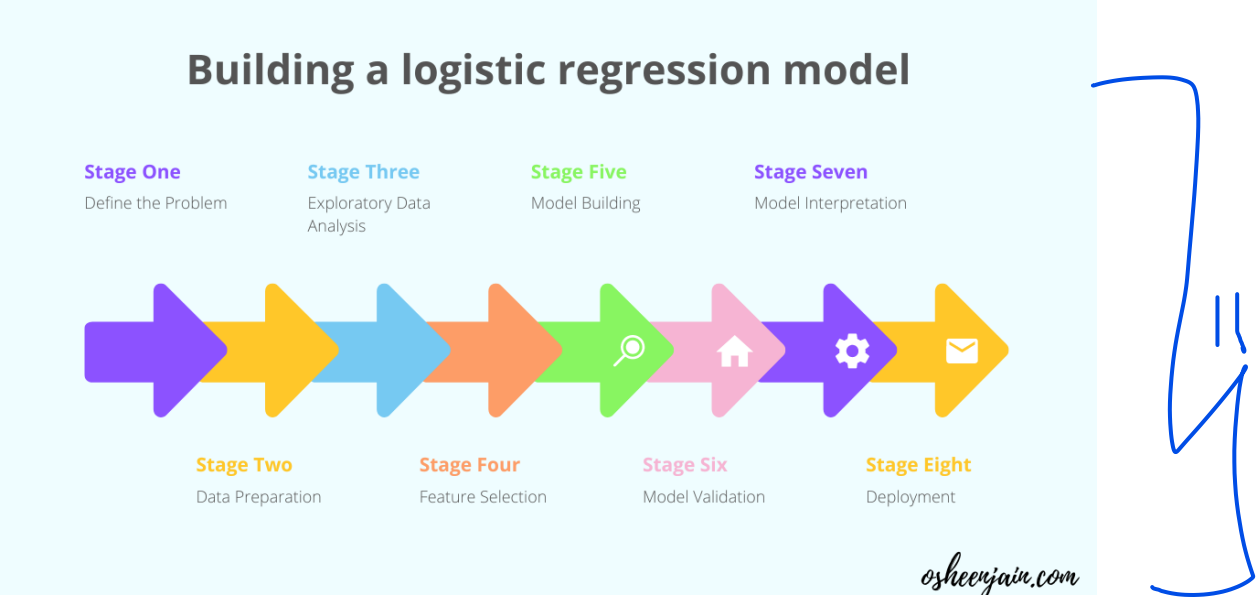
Classification is used when you want to categorize data into different classes or groups. For example, classifying emails as "spam" or "not spam" or predicting whether a patient has a certain disease based on their symptoms. Here are some common types of classification models:

1. **Decision Tree Classification**: Builds a tree where each node represents a test case for an attribute, and branches represent possible outcomes.
2. **Random Forest Classification**: Uses an ensemble of decision trees to make predictions, improving accuracy by averaging the results from multiple trees.
3. **K-Nearest Neighbor (KNN)**: Classifies data points based on the 'k' nearest neighbors using feature similarity.

Logistic Regression

Logistic Regression is a supervised machine learning algorithm used for classification problems. Unlike linear regression which predicts continuous values it predicts the probability that an input belongs to a specific class. It is used for binary classification where the output can be one of two possible categories such as Yes/No, True/False or 0/1. It uses sigmoid function to convert inputs into a probability value between 0 and 1





- ***Python Implementation of Logistic Regression (Binomial)***

data.py*

```python
import numpy
from sklearn import linear_model

# Feature: Chemical concentration (in arbitrary units)
X = numpy.array([1.2, 0.5, 0.8, 1.0, 1.4, 0.6, 3.0, 2.8, 3.2, 3.5, 2.9, 3.1]).reshape(-1, 1)

# Label: 0 = Non-toxic, 1 = Toxic
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

# Train logistic regression model
logr = linear_model.LogisticRegression()
logr.fit(X, y)

# Predict if a plant with chemical concentration 4.9.0 is toxic
test_value = numpy.array([2.0]).reshape(-1, 1)
predicted = logr.predict(test_value)
probability = logr.predict_proba(test_value)

print("Predicted class (0 = non-toxic, 1 = toxic):", predicted[0])
print("Probability of toxicity:", probability[0][1])

```

```
In [8]: runfile('E:/cvmu/np/data.py', wdir='E:/cvmu/np')
Predicted class (0 = non-toxic, 1 = toxic): 1
Probability of toxicity: 0.6353403903447106
```

Steps of Logistic Regression (for the chemical concentration example)

1. **1. Import Libraries**

You import the necessary libraries to work with arrays and logistic regression:

```
import numpy
from sklearn import linear_model
```

---

2. **2. Prepare the Data**

Define your **feature (X)** and **target label (y)**:

```
# Feature: chemical concentration
X = numpy.array([...]).reshape(-1, 1)
```

```
# Label: 0 = non-toxic, 1 = toxic
y = numpy.array([...])
```

- Reshaping with .reshape(-1, 1) ensures it's in the correct shape for scikit-learn.
- X is a 1D array of values like [1.2, 0.5, ...]
- y contains the corresponding classification: 0 or 1

◇ 3. Create and Train the Model

You initialize and train a logistic regression model on the dataset:

```
logr = linear_model.LogisticRegression()
logr.fit(X, y)
```

- The .fit() method learns the coefficients and intercept from the data using **maximum likelihood estimation**.

◇ 4. Make a Prediction

You predict whether a new sample (e.g., concentration = 2.0) is toxic or not:

Steps in building and applying a logistic regression model

1. Data collection and preparation: This involves loading the dataset, exploring its structure and potential missing values, and preparing the data through steps like feature scaling and splitting into training and testing sets.

2. Model building and training: Initializing and training the `LogisticRegression` model from Scikit-learn on the training data.

3. Prediction: Using the trained model to make predictions on the test set, including predicting both class labels and probabilities.

4. Evaluation and Interpretation: Evaluating the model's performance using metrics such as accuracy, confusion matrix, and classification report. The code also demonstrates how to generate and interpret the Receiver Operating Characteristic (ROC) curve and calculate the Area Under the Curve (AUC).

5. Hyperparameter tuning (optional): An example using `GridSearchCV` is provided to demonstrate how to find the best hyperparameters for the model.

❖ Multiple Regression :

predicting an outcome with multiple factors.

Multiple regression analysis, also known as multiple linear regression (MLR), is a statistical technique used to understand and model the relationship between a single dependent variable and two or more independent variables. It extends the concepts of simple linear regression by allowing for the inclusion of multiple predictors, which often leads to more accurate and comprehensive insights.

When to use multiple regression

You should consider using multiple regression when you want to:

- **Predict a continuous outcome variable** based on the combined influence of several independent variables.
- **Analyze the strength and direction of the relationship** between each independent variable and the dependent variable, while controlling for the effects of other predictors in the model.
- **Understand the relative contribution of each independent variable** in explaining the variation in the dependent variable.
- **Determine whether a particular predictor variable is still able to predict an outcome** when the effects of another variable are controlled for.

How it works

Multiple regression aims to find a linear equation that best represents the relationship between the variables, minimizing the differences between the observed and predicted values of the dependent variable.
The general formula for multiple linear regression is:

$$Y=B_0+B_1X+B_2X_2 + \ldots\ldots\ldots +B_nX_n + e$$

Where:

- Y is the predicted value of the dependent variable.
- β_0 is the y-intercept, representing the value of Y when all independent variables are zero.
- $X_1, X_2, \ldots, X_n$ are the independent variables.
- $\beta_1, \beta_2, \ldots, \beta_n$ are the regression coefficients (slopes), representing the change in Y for a one-unit increase in the corresponding independent variable, holding other variables constant.

- e: is the error term, accounting for the unexplained variation in Y.

Assumptions of multiple regression

Before performing multiple regression, it's essential to check the following assumptions:

- **Linearity:** A linear relationship should exist between the dependent variable and the independent variables.
- **Homoscedasticity:** The variance of the errors (residuals) should be constant across all levels of the independent variables.
- **Independence of Observations:** The observations in the dataset should be independent of each other.
- **No Multicollinearity:** The independent variables should not be highly correlated with each other.
- **Normality of Residuals:** The residuals should be approximately normally distributed.

Real-world applications

Multiple regression is widely used across various fields, including:

- Business & Marketing: Predicting sales, optimizing marketing campaigns.
- Healthcare: Analyzing the impact of factors like age, weight, and lifestyle on disease risk, predicting patient outcomes.
- Finance: Forecasting stock prices, assessing risk in portfolio management.
- Real Estate: Predicting property prices based on size, location, and amenities.
- Environmental Science: Estimating air pollution levels based on traffic volume, weather, and industrial activity.
- Social Sciences: Understanding factors influencing employee performance, consumer behavior.

Steps in building and applying a multiple regression Model

1. Prepare your data

- Gather data: Collect your dependent variable and multiple independent variables.
- Clean and organize: Handle missing values, outliers, and potentially transform or standardize features as necessary.

- Split data: Separate your dataset into training and testing sets for model development and evaluation.

2. Build and fit the model

- Initialize the Model: Use appropriate libraries (e.g., scikit-learn's LinearRegression in Python) to create the regression model object.
- Train the Model: Fit the model to your training data to estimate the regression coefficients (β values) that best describe the relationship between the independent and dependent variables.

3. Evaluate the model

- Make predictions: Use the trained model to predict values on the unseen test data.
- Assess model performance: Evaluate the model's accuracy and effectiveness using metrics like R-squared, Adjusted R-squared, Mean Squared Error (MSE), and examine residuals for assumptions.
- Interpret Coefficients: Understand the influence of each independent variable by analyzing the regression coefficients and their statistical significance.

4. Refine and validate (optional but recommended)

- Address issues: Handle issues like multicollinearity or violations of assumptions by techniques such as removing or combining variables, or using transformations.
- Validate: Use cross-validation or other methods to ensure the model generalizes well to new data.

❖ **Polynomial regression:**

modeling nonlinear relationships

Polynomial regression is a type of regression analysis that allows us to model non-linear relationships between variables. While traditional linear regression fits a straight line to the data, polynomial regression uses an nth-degree polynomial function to capture curves and more intricate patterns within the data points.

Use polynomial regression when you suspect a curvilinear relationship between your variables. For instance, consider the relationship between temperature and ice cream sales. Sales might rise with increasing temperature up to a certain point, then decline as it gets too hot for outdoor activities. A linear model would struggle to represent this curved behavior, whereas a polynomial regression model can capture more effectively.

How it works

Polynomial regression extends the linear regression model by introducing polynomial terms. For example, a quadratic polynomial regression (degree 2) would involve terms like X, X2 allowing the model to fit a parabolic curve to the data. The equation for a polynomial regression of degree *n* is:

$$Y=B_0+B_1X+B_2X^2 + \ldots\ldots\ldots +B_nX^n + e$$

Where:

Y: represents the dependent variable.
X: represents the independent variable.

$B_0, B_1, \ldots B_n$ are the coefficients (parameters) estimated from the data

n : is the degree of the polynomial.

e: is the error term, capturing the unexplained variation.

- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.

Need for Polynomial Regression:

- Nonlinear relationships: When the relationship between the variables isn't a simple straight line, linear regression will likely provide an inaccurate or underfit model. Polynomial regression, by introducing higher-degree terms, can capture these curves and better represent the underlying relationship.

- Capturing complex trends: In fields like physics, engineering, biology, economics, and finance, data often exhibits acceleration, deceleration, or other changing rates of growth or decline that a linear model cannot handle. Polynomial regression's ability to model curves helps capture these intricate trends and patterns.

- Improved accuracy: By fitting a curve to the data, polynomial regression can often achieve a better fit and more accurate predictions compared to linear regression, especially in situations with non-linear relationships. In some instances, it has been shown to improve the accuracy of a linear model by a significant margin.

- Flexibility: It offers greater flexibility in model fitting as the polynomial degree can be adjusted to match the complexity of the data, making it adaptable across various industries.

Assumptions of polynomial regression
Like other regression techniques, polynomial regression relies on certain assumptions for accurate and reliable results. These include:

- **Relationship type:** The relationship between the dependent variable and independent variables is linear or curvilinear (specifically polynomial).
- **Independent variables:** The independent variables are independent of each other (no multicollinearity).
- **Errors:** Errors are independent, normally distributed with a mean of zero, and have a constant variance.

Advantages

- Can model a wide range of nonlinear relationships.
- Offers greater flexibility compared to linear regression.
- Can improve accuracy in predictions when the underlying relationship is non-linear.

Disadvantages

- **Overfitting:** High-degree polynomials can overfit the data, capturing noise rather than the true underlying pattern, leading to poor generalization to new data.
- **Sensitivity to outliers:** Outliers can have a significant impact on the estimated coefficients and the shape of the curve.
- **Multicollinearity:** As the degree increases, the terms(e.g. X, X^2) can become highly correlated, leading to instability in coefficient estimates.

Applications

- Growth rates: Modeling the growth of tissues or populations over time.

- Disease progression: Tracking the progression of disease outbreaks.

- Environmental studies: Analyzing the distribution of isotopes in lake sediments.
- Economics and finance: Analyzing trends and fluctuations in financial markets.
- Engineering: Modeling material behavior under different conditions.

Steps

1. Prepare data

- Gather & Clean Data: Collect and pre-process data. Handle missing values and outliers.
- Split Data: Divide into training and testing sets.

2. Generate polynomial features

- Choose Degree: Select the polynomial degree (e.g., 2, 3) for the non-linear relationship based on the data.

  Transform Features: Create new features by raising the original features to the chosen power. For instance, if degree 2 is chosen, a feature X would become X and X^2.

3. Train the model

- Initialize & Train Model: Fit a linear regression model to the transformed polynomial features and the dependent variable.

4. Evaluate and interpret

- Predict and Assess: Generate predictions on test data and evaluate using metrics like R-squared (R^2) and Mean Squared Error (MSE).

- Visualize the Fit: Plot the regression curve to visualize how well the model captures the data's non-linear patterns.

❖ Decision Tree:

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**

Why use Decision Trees?

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.

The logic behind the decision tree can be easily understood because it shows a tree-like structure

Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Attribute Selection Measures

Attribute selection measures (ASMs) are crucial in decision tree algorithms for identifying the optimal features or attributes to split the data at each node. These measures help in determining which feature, when used for splitting, will lead to the purest (most homogeneous) child nodes, ultimately improving the accuracy and efficiency of the decision tree model.

1. Information Gain:

- ✓ Measures the reduction in entropy (uncertainty or disorder) of the target variable after splitting the data based on a specific attribute.
- ✓ A higher information gain indicates that an attribute is more effective in separating the data into distinct classes, making it a strong candidate for splitting.
- ✓ It is used by algorithms like ID3 and C4.5.
- ✓ Formula: Information Gain = Entropy(Parent) - Weighted Average Entropy(Children).

2. Gini Index (or Gini Impurity):

- ✓ Measures the probability of misclassifying a randomly chosen element if it were classified according to the class distribution in a given subset.
- ✓ A lower Gini index indicates greater purity or homogeneity within a node.
- ✓ It is used by the CART (Classification and Regression Trees) algorithm.
- ✓ Formula: Gini Impurity = 1 - Σ(P_i)^2, where P_i is the probability of an instance belonging to class i.

3. Gain Ratio:

- ✓ An extension of Information Gain that addresses its bias towards attributes with a large number of distinct values.
- ✓ It normalizes Information Gain by considering the "splitting information," which measures how broadly and uniformly an attribute splits the data.

- ✓ Gain Ratio helps prevent the selection of attributes that may have high information gain simply because they create many small, pure partitions.

4. Reduction in Variance:

- ✓ Used specifically in regression trees, where the target variable is continuous.
- ✓ Measures the reduction in variance of the target variable after splitting the data based on an attribute.
- ✓ The attribute leading to the greatest reduction in variance is chosen for the split.

Pruning: obtaining an optimal decision tree

1. Preventing overfitting

- Addressing model complexity: Decision trees can become overly complex by creating too many branches and nodes, essentially memorizing the training data, including noise and outliers.
- Simplifying the structure: Pruning removes these unnecessary branches and nodes, simplifying the tree and allowing it to focus on the essential features and patterns in the data, rather than the idiosyncrasies of the training set.

2. Improving generalization

- Focusing on relevant patterns: By simplifying the tree, pruning encourages the model to learn more generalizable patterns, making it perform better on unseen data.
- Discarding noise: It helps in discarding noise or outliers that the unpruned tree might have picked up, which would otherwise hinder its ability to generalize.

3. Reducing model complexity and enhancing interpretability

- Simpler models: Pruning results in smaller, more concise decision trees that are easier to understand and interpret.
- Facilitating human understanding: This simplified structure can be especially valuable in domains where model interpretability and human insight into the decision-making process are important, such as in medicine or finance.

4. Faster training and inference

- Resource efficiency: A smaller, pruned decision tree requires fewer computational resources for both training and making predictions (inference).
- Efficiency and scalability: This can be beneficial when dealing with large datasets or when deploying models in resource-constrained environments.

5. Facilitating model maintenance

- Adaptability: Pruned decision trees, being less complex, are easier to update and adapt to new data or evolving problem domains compared to complex, unpruned trees.

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

Steps for implementing a decision tree

1. Defining the problem and preparing the data

- Problem Definition: Clearly identify whether your problem requires a classification (predicting categories) or a regression (predicting continuous values) model. This influences your choice of evaluation metrics later.
- Data Collection & Preparation: Gather your dataset, ensuring it includes both features (independent variables) and a target variable (dependent variable). Clean the data by handling missing values, outliers, and errors.

- Engineering: Select relevant features that are most likely to influence the target variable. Transform and scale the data as needed, ensuring all features are in the appropriate format for the algorithm.

- Data Splitting: Divide the dataset into a training set (70-80% of the data) and a testing set (20-30%), the training set is used to build the model, while the testing set is used to evaluate its performance on unseen data.

2. Building the decision tree

- Attribute Selection Measure (ASM): Choose an appropriate ASM like Gini Impurity or Information Gain (Entropy) to guide the splitting process. This criterion will determine which feature best separates the data into homogeneous groups.

- Recursive Splitting: Start at the root node with the entire training dataset. Recursively split the data at each node based on the feature that maximizes information gain or minimizes Gini impurity. This process creates a hierarchical structure of nodes and branches.

- Stopping Criteria: Define termination conditions to prevent overfitting, such as reaching a maximum tree depth, having a minimum number of samples in a node or leaf, or encountering nodes where further splitting doesn't significantly improve impurity reduction.

- Leaf Nodes: Once the stopping criteria are met, the nodes become leaf nodes, each assigned the most common class label (for classification) or the mean/median of the target values (for regression).

3. Pruning (optional but recommended)

- Addressing Overfitting: After building the tree, consider pruning to prevent overfitting, which occurs when the tree becomes overly complex and memorizes the training data instead of learning generalized patterns.

- Post-pruning or Pre-pruning: Choose either pre-pruning (setting constraints during tree growth) or post-pruning (removing branches after the tree is fully grown). Pruning simplifies the model and improves its ability to generalize.

4. Making predictions and evaluating the model

- Prediction: To make predictions for new data points, traverse the tree from the root to a leaf node by following the decision path based on the input features.
- Evaluation: Assess the model's performance using appropriate metrics. For classification tasks, use accuracy, precision, recall, or F1-score. For regression tasks, use metrics like Root Mean Squared Error (RMSE) or Mean Absolute Error (MAE).

5. Iteration and refinement

- Analyze Results: Interpret the decision tree, understanding the decision rules learned and the importance of various features.
- Hyperparameter Tuning: Fine-tune hyperparameters (like `max_depth`, `min_samples_split`, `min_samples_leaf`) to optimize the model's performance on your specific dataset. Cross-validation is often used to evaluate different hyperparameter settings.

❖ K – Nearest Neighbor (KNN) Algorithm for Machine Learning :

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for both classification and regression tasks. It works by finding the k nearest data points (neighbors) to a new, unseen data point and then classifying or predicting based on the majority class or average value of those neighbors.

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification 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.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **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.

- ***detailed explanation:***

- **Supervised Learning:**

  KNN is a supervised learning algorithm, meaning it learns from labeled training data where both inputs and desired outputs are provided.

- **Non-parametric:**

  KNN is a non-parametric method, meaning it doesn't make assumptions about the underlying data distribution. This makes it versatile for various datasets.

- **Distance Calculation:**

The algorithm relies on calculating distances between data points. Common distance metrics include Euclidean distance, Manhattan distance, and Hamming distance.

- **Classification:**

In classification, KNN assigns the class label of the majority of the k nearest neighbors to the new data point.

- **Regression:**

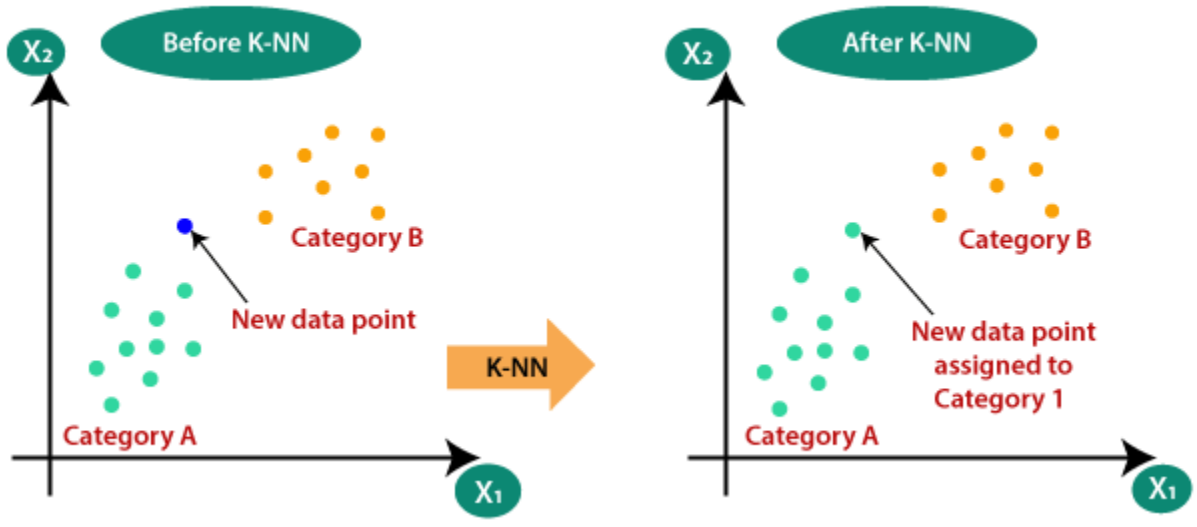
In regression, KNN predicts a continuous value by averaging the values of the k nearest neighbors.

- **Choosing 'k':**

The value of k (number of neighbors) is a crucial parameter. A smaller k can lead to overfitting, while a larger k can smooth out the decision boundaries but might miss local patterns.

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:

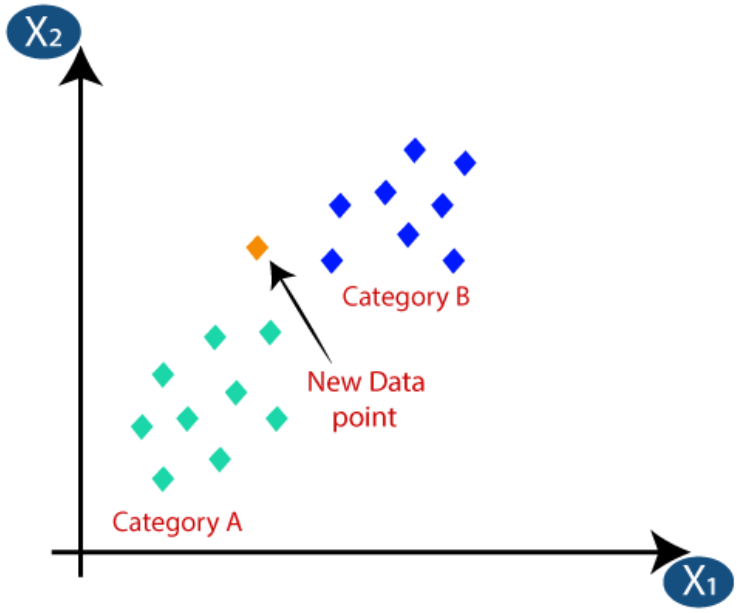


How does K-NN work?

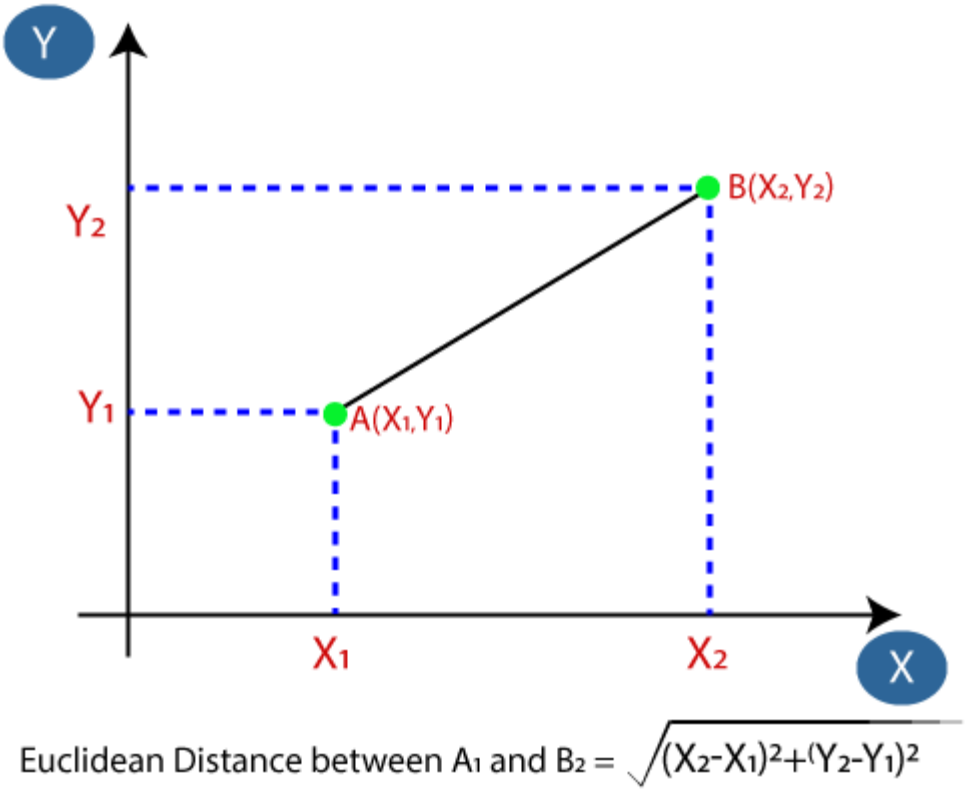
The K-NN working can be explained on the basis of the below algorithm:

- **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.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

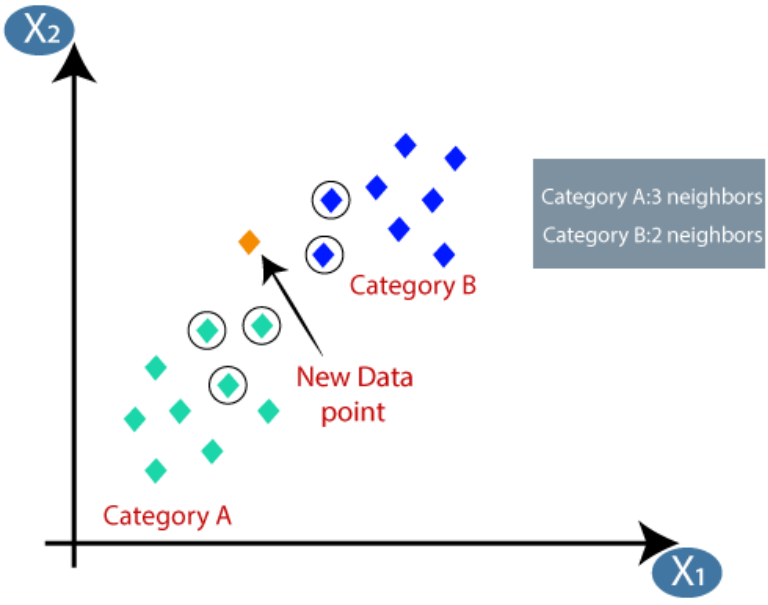


- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



Euclidean Distance between A_1 and $B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

- 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.

Advantages

- Simple to understand and implement.
- Adaptable to data changes.
- Handles both classification and regression.
- Non-parametric.

Disadvantages

- Can be computationally expensive for large datasets.
- Sensitive to irrelevant features and data scale.
- Performance can degrade in high-dimensional spaces.
- May struggle with imbalanced classes.

Applications

KNN is used in text classification, image recognition, recommendation systems, and medical diagnosis.

KNN is a simple and versatile algorithm, good for understanding machine learning concepts, but its limitations should be considered for large and complex datasets

Building a K Nearest Neighbors Model

We can follow the below steps to build a KNN model −

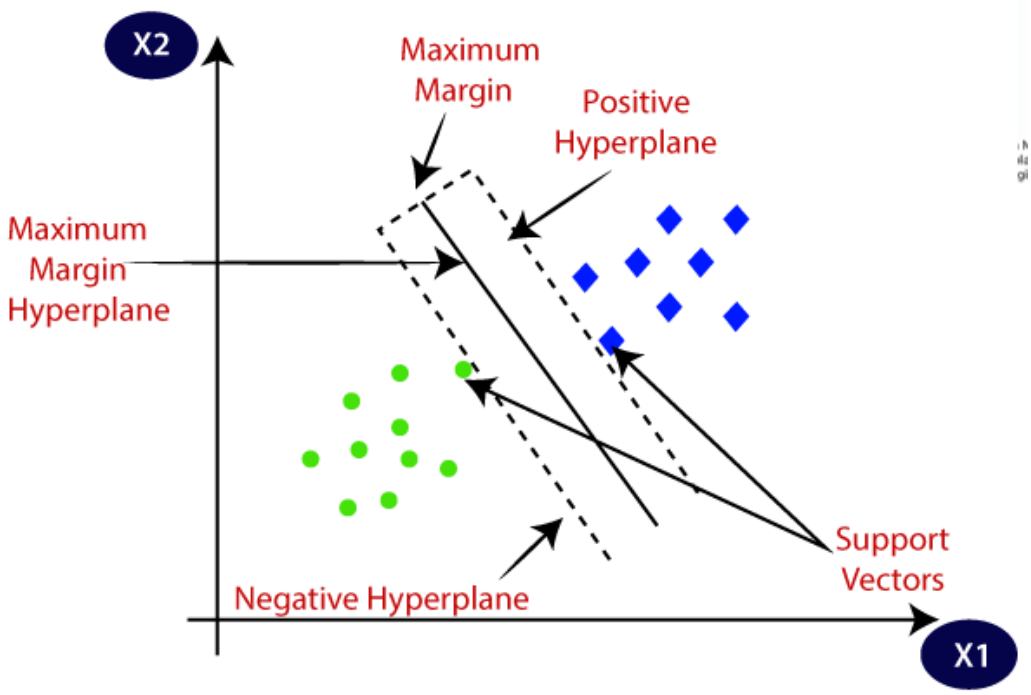
- **Load the data** − The first step is to load the dataset into memory. This can be done using various libraries such as pandas or numpy.
- **Split the data** − The next step is to split the data into training and test sets. The training set is used to train the KNN algorithm, while the test set is used to evaluate its performance.
- **Normalize the data** − Before training the KNN algorithm, it is essential to normalize the data to ensure that each feature contributes equally to the distance metric calculation.
- **Calculate distances** − Once the data is normalized, the KNN algorithm calculates the distances between the test data point and each data point in the training set.
- **Select k-nearest neighbors** − The KNN algorithm selects the k-nearest neighbors based on the distances calculated in the previous step.
- **Make a prediction** − For classification problems, the KNN algorithm assigns the test data point to the class that appears most frequently among the k-nearest neighbors. For regression problems, the KNN algorithm assigns the test data point the average of the k-nearest neighbors' values.
- **Evaluate performance** − Finally, the KNN algorithm's performance is evaluated using various metrics such as accuracy, precision, recall, and F1-score.

❖ **Support Vector Machine :**

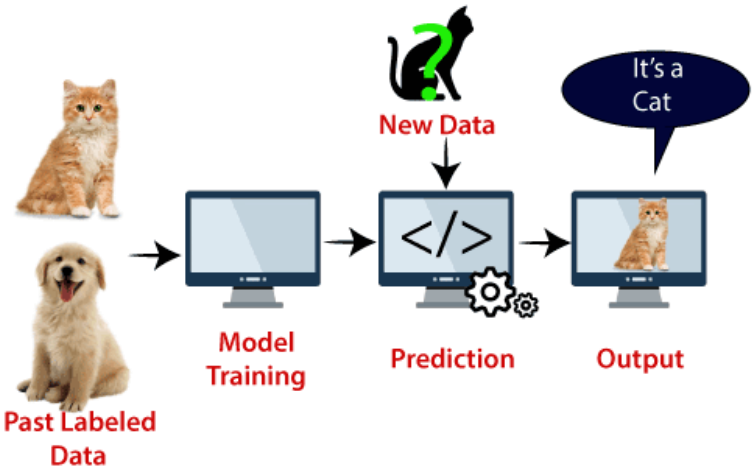
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

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:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

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.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

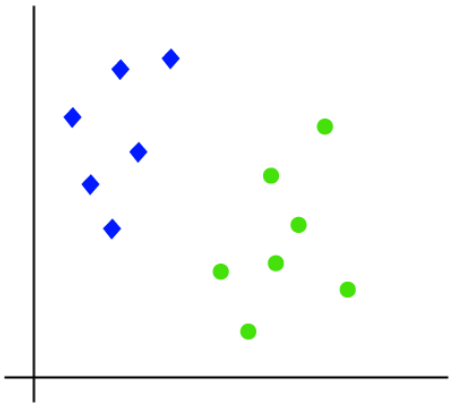
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

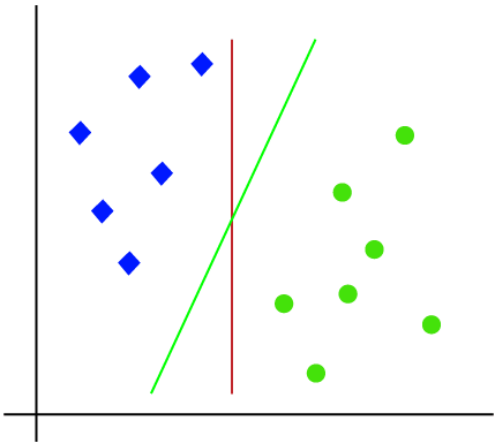
How does SVM works?

Linear SVM:

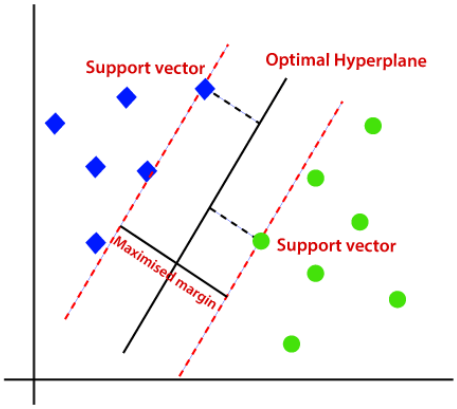
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



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:

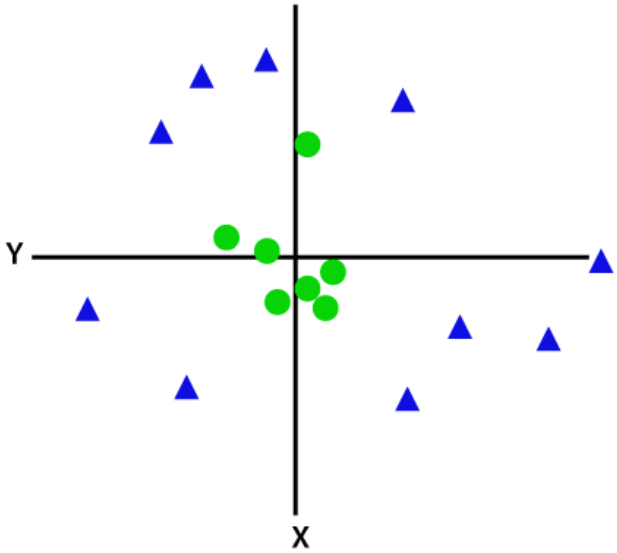


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane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

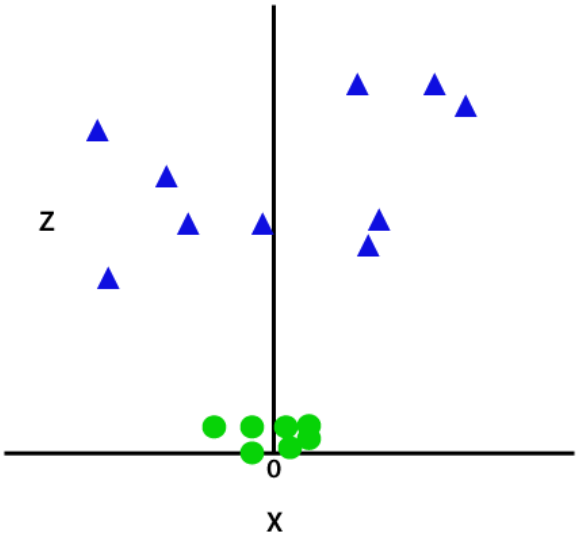
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



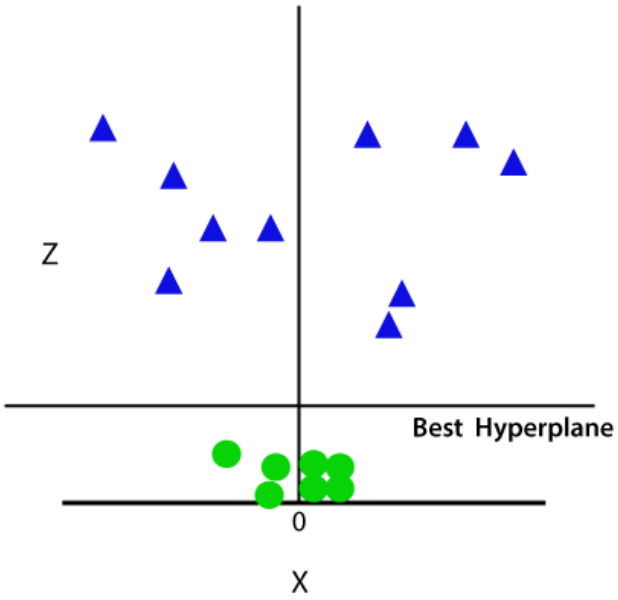
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$$z=x^2+y^2$$

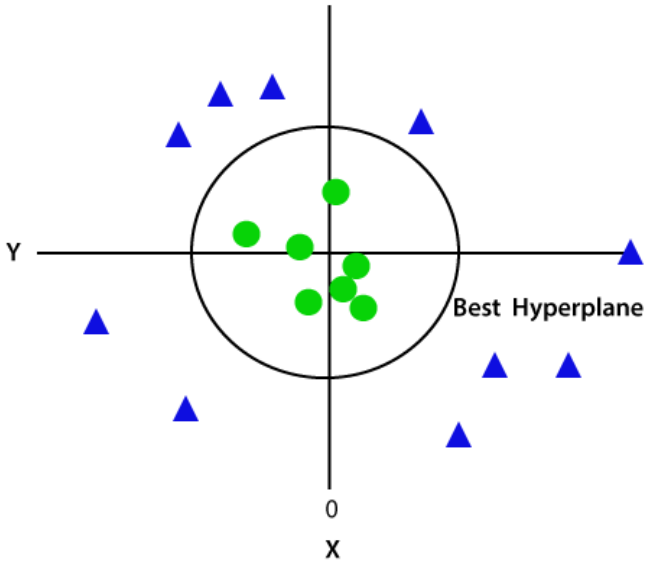
By adding the third dimension, the sample space will become as below image:



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.

❖ Naïve Bayes Classifier :

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

- ***Bayes' Theorem:***

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

- ***Working of Naïve Bayes' Classifier:***

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

Outlook	**Play**	
0	Rainy	Yes

1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

- ***Frequency table for the Weather Conditions:***

Weather	Yes	No
Overcast	5	0
Rainy	2	2

Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

Types of Naïve Bayes Model:

` There are three types of Naive Bayes Model, which are given below:

- **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc.
  The classifier uses the frequency of words for the predictors.
- **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.