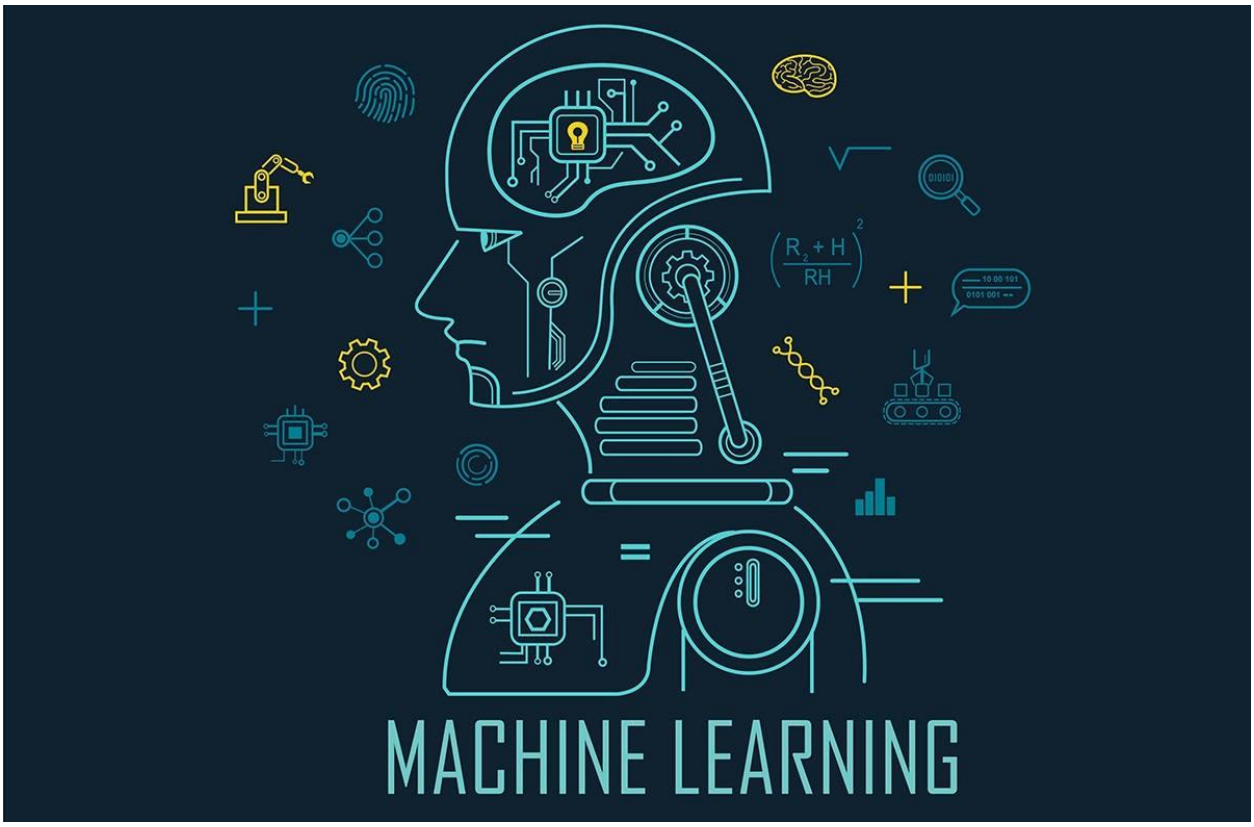
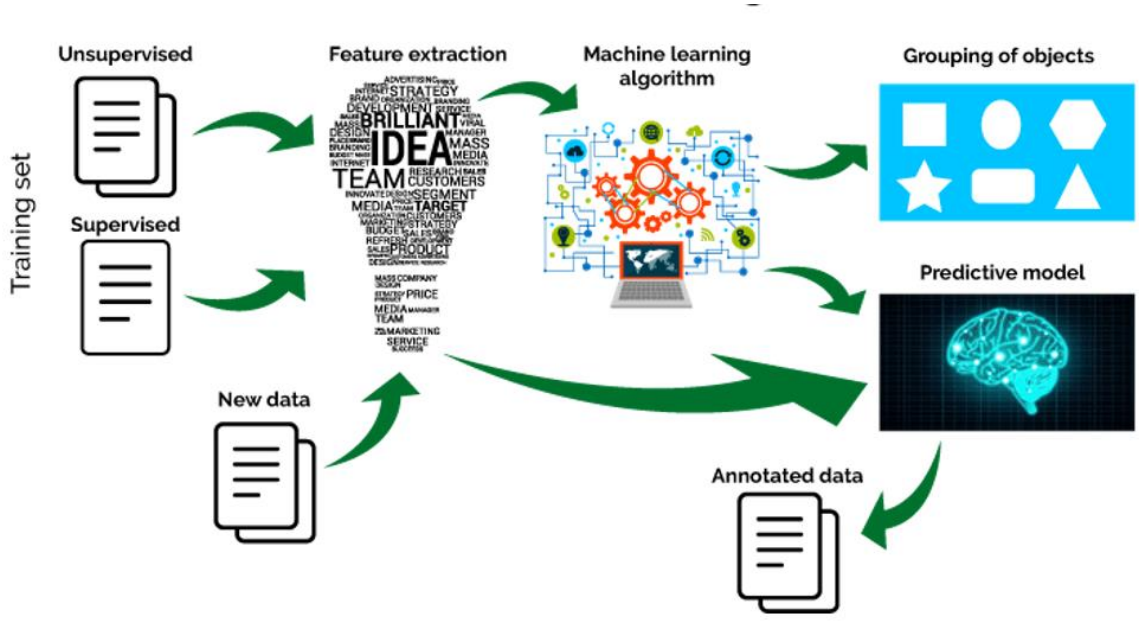

MACHINE LEARNING


Training set
Unsupervised
Supervised
New data
Feature extraction
Machine learning algorithm
Grouping of objects
Predictive model
Annotated data

Chapter 1

Introduction to Machine Learning

1.1 Definition and Evolution of Machine Learning

Definition:
Machine Learning (ML) is a method of data analysis that automates model building. It's a branch of artificial intelligence (AI) based on the idea that systems can learn from data, identify patterns, and make decisions with minimal human intervention.

Evolution Timeline:

1950s: Alan Turing introduces the "Turing Test"

1957: Perceptron - the first neural network

1990s: Rise of Support Vector Machines and ensemble learning

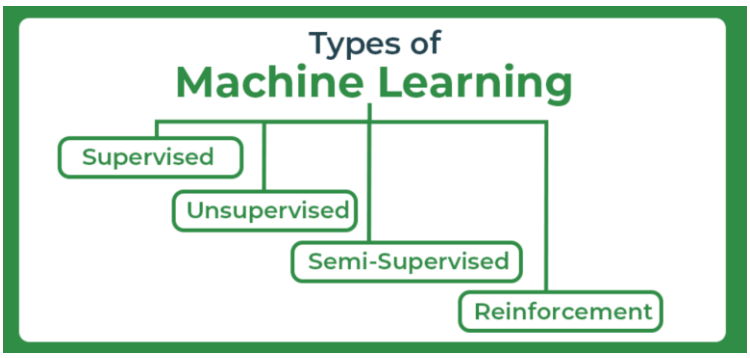
2010s–present: Deep learning revolution with frameworks like TensorFlow and PyTorch

Example:

Netflix uses ML to recommend shows based on your watch history.

Email services like Gmail use ML to filter spam.

1.2 Types of Machine Learning



1.2.1 Supervised Learning

Definition: Learning from labeled data (i.e., the outcome is known).
Example:

- Predicting house prices based on features like size and location.
- Input: Features of houses
- Output: Price (label)

1.2.2 Unsupervised Learning

Definition: Finding hidden patterns in unlabeled data.
Example:

- Market segmentation: Grouping customers based on purchasing behavior using clustering algorithms.

1.2.3 Reinforcement Learning

Definition: Learning by interacting with an environment and receiving feedback in the form of rewards or penalties.
Example:

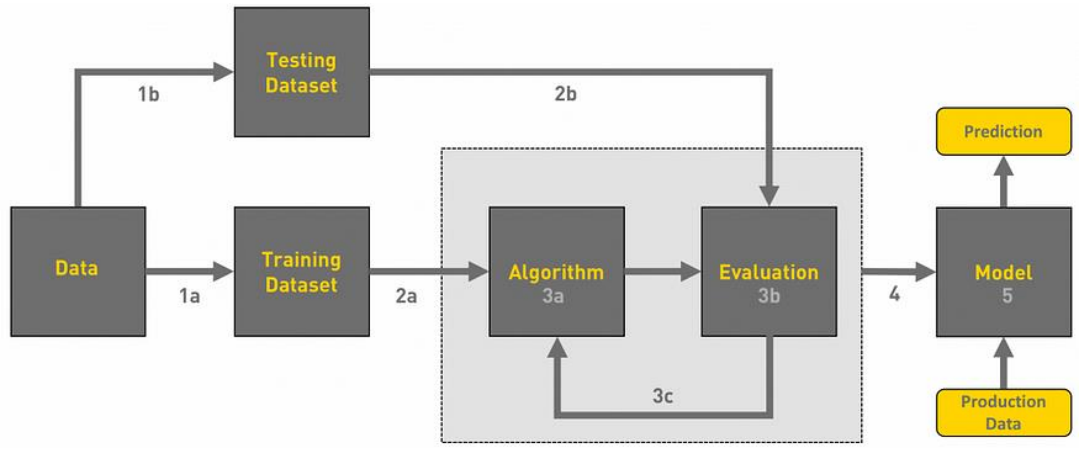
- A robot learning to walk through trial and error.
- Games like AlphaGo use RL to improve strategies by playing millions of games.

1.2.4 Active Learning

Definition: The algorithm selects the most useful data points to be labeled.
Example:

- A medical diagnosis system asking doctors to label only ambiguous X-rays.

1.3 ML Workflow



We can define the machine learning workflow in following stages.

1. **Gathering data**
2. **Data pre-processing**
3. **Researching the model that will be best for the type of data**
4. **Training and testing the model**
5. **Evaluation**

1. Gathering Data

The process of gathering data depends on the type of project we desire to make, if we want to make an ML project that uses real-time data, then we can build an IoT system that using different sensors data. The data set can be collected from various sources such as a file, database, sensor and many other such sources but the collected data cannot be used directly for performing the analysis process as there might be a lot of missing data, extremely large values, unorganized text data or noisy data. Therefore, to solve this problem Data Preparation is done.

2. Data pre-processing

Data pre-processing is one of the most important steps in machine learning. It is the most important step that helps in building machine learning models more accurately. In machine learning, there is an 80/20 rule. Every data scientist should spend 80% time for data pre-processing and 20% time to actually perform the analysis.

What is data pre-processing?

Data pre-processing is a process of cleaning the raw data i.e. the data is collected in the real world and is converted to a clean data set. In other words, whenever the data is gathered from different sources it is collected in a raw format and this data isn't feasible for the analysis. Therefore, certain steps are executed to convert the data into a small clean data set, this part of the process is called as data pre-processing.

Why do we need it?

As we know that data pre-processing is a process of cleaning the raw data into clean data, so that can be used to train the model. So, we definitely need data pre-processing to achieve good results from the applied model in machine learning and deep learning projects.

Most of the real-world data is messy, some of these types of data are:

1. **Missing data:** Missing data can be found when it is not continuously created or due to technical issues in the application (IOT system).

2. **Noisy data:** This type of data is also called outliners, this can occur due to human errors (human manually gathering the data) or some technical problem of the device at the time of collection of data.

3. **Inconsistent data:** This type of data might be collected due to human errors (mistakes with the name or values) or duplication of data.

Three Types of Data
1. Numeric e.g. income, age
2. Categorical e.g. gender, nationality
3. Ordinal e.g. low/medium/high

How can data pre-processing be performed?

These are some of the basic pre — processing techniques that can be used to convert raw data.

1. **Conversion of data:** As we know that Machine Learning models can only handle numeric features, hence categorical and ordinal data must be somehow converted into numeric features.

2. **Ignoring the missing values:** Whenever we encounter missing data in the data set then we can remove the row or column of data depending on our need. This method is known to be efficient but it shouldn't be performed if there are a lot of missing values in the dataset.

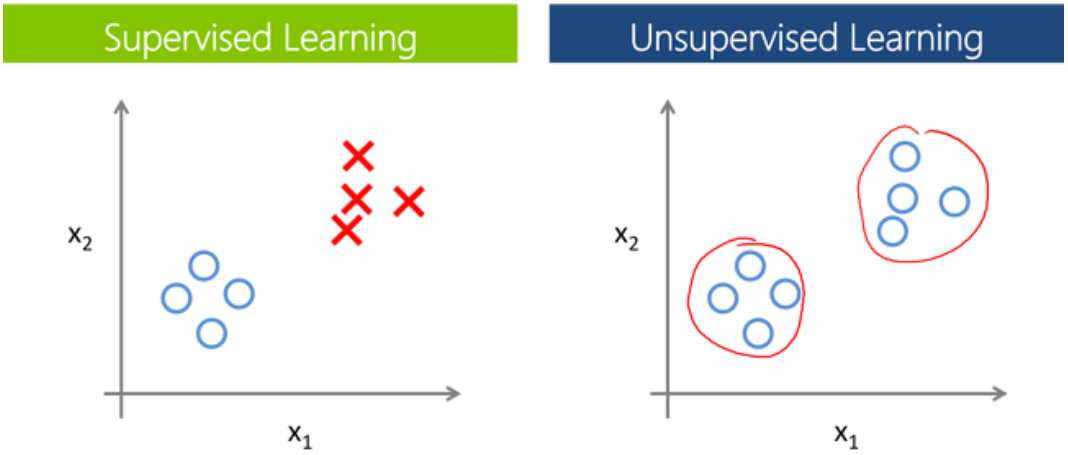
3. **Filling the missing values:** Whenever we encounter missing data in the data set then we can fill the missing data manually, most commonly the mean, median or highest frequency value is used.

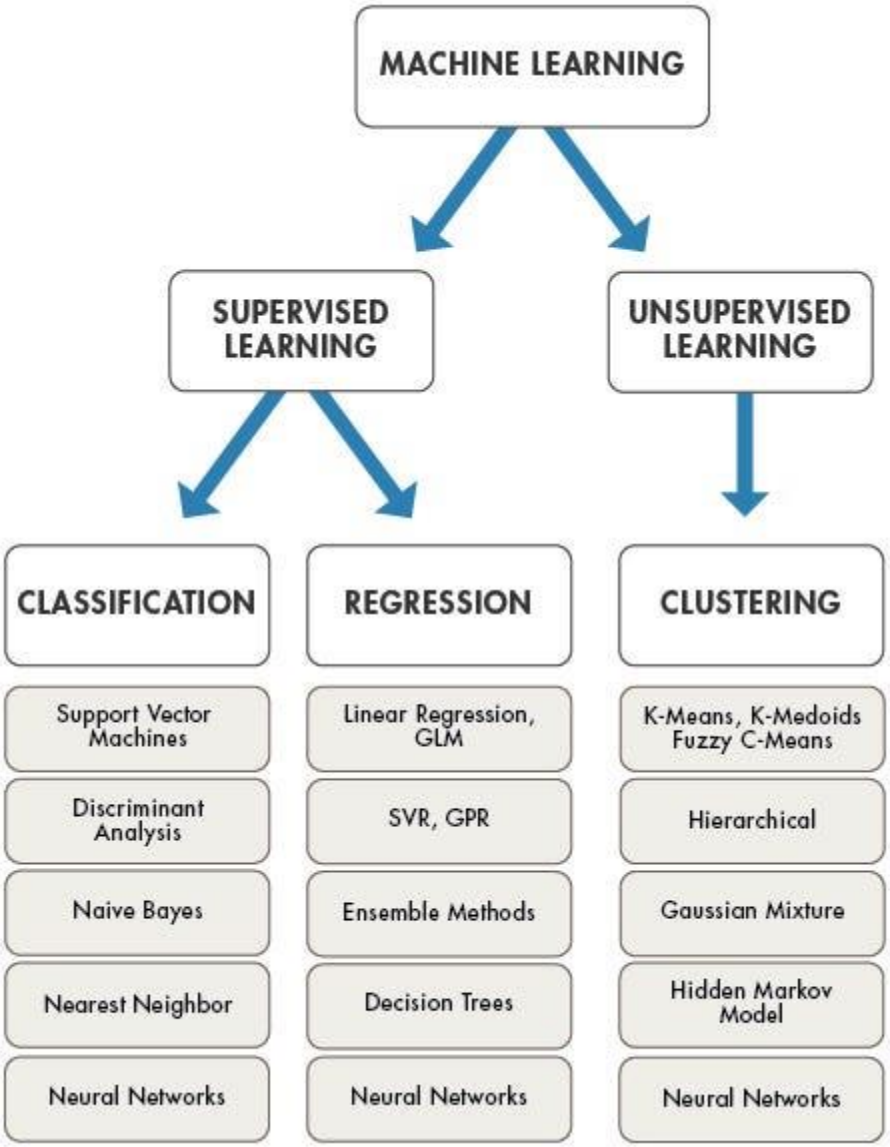
4. **Machine learning:** If we have some missing data then we can predict what data shall be present at the empty position by using the existing data.

5. **Outlier's detection:** There are some error data that might be present in our data set that deviates drastically from other observations in a data set. [Example: human weight = 800 Kg; due to mistyping of extra 0]

3. Researching the model that will be best for the type of data

Our main goal is to train the best performing model possible, using the pre-processed data.





Supervised Learning:

In Supervised learning, an AI system is presented with data which is labelled, which means that each data tagged with the correct label.

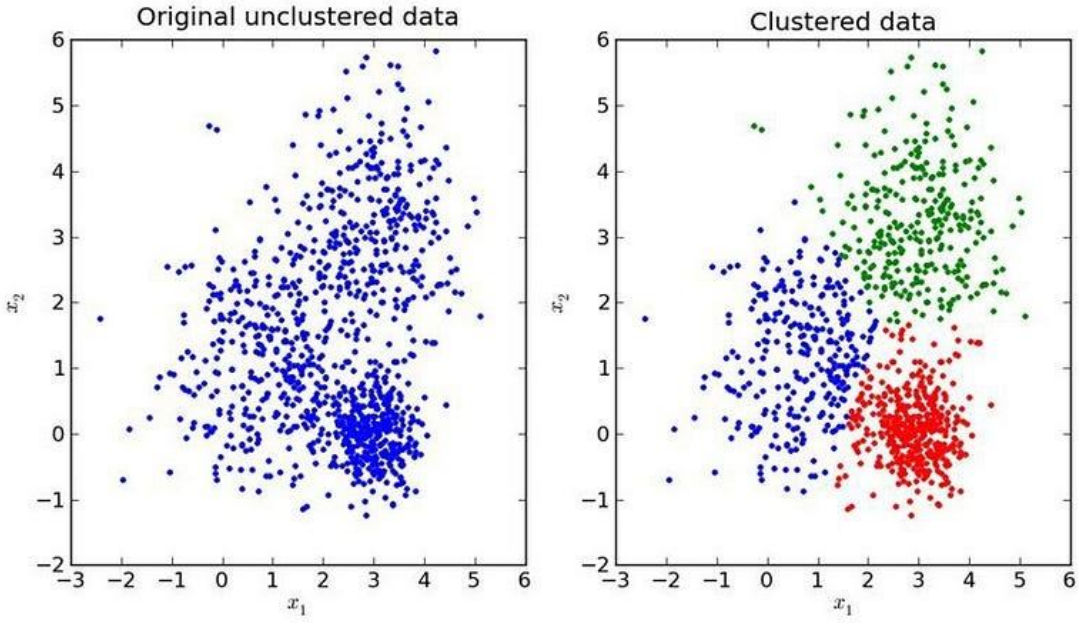
The supervised learning is categorized into 2 other categories which are "**Classification**" and "**Regression**".

Classification:

Classification problem is when the target variable is **categorical** (i.e. the output could be classified into classes — it belongs to either Class A or B or something else).

A classification problem is when the output variable is a category, such as "red" or "blue", "disease" or "no disease" or "spam" or "not spam".

Unsupervised Learning:

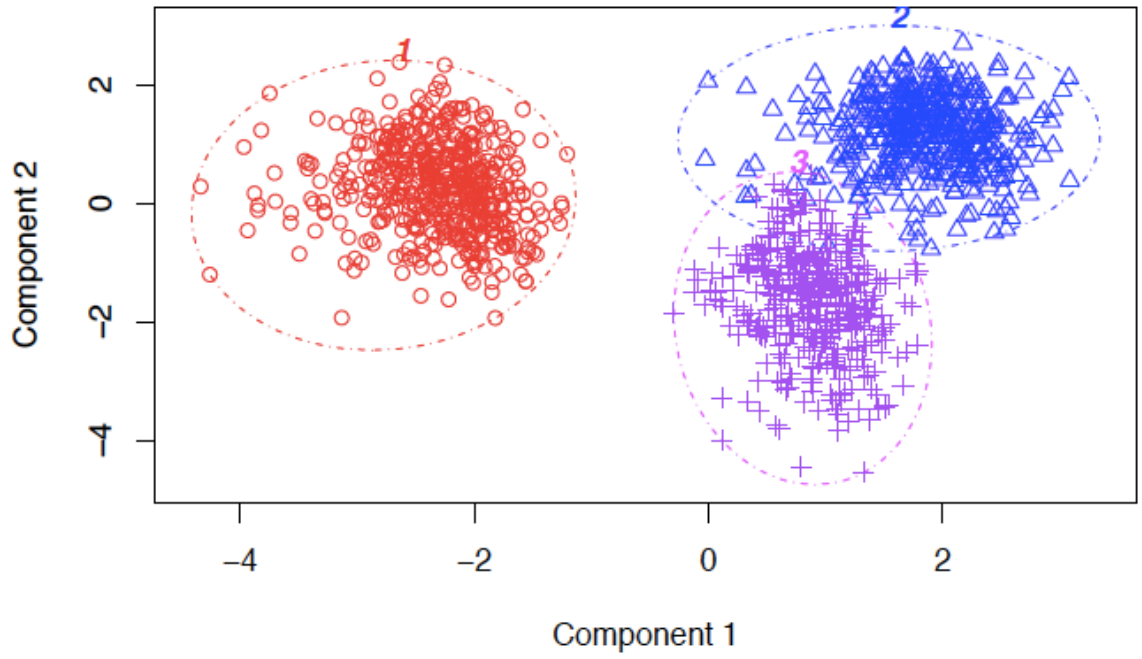


In unsupervised learning, an AI system is presented with unlabeled, un-categorized data and the system's algorithms act on the data without prior training. The output is dependent upon the coded algorithms. Subjecting a system to unsupervised learning is one way of testing AI.

The unsupervised learning is categorized into 2 other categories which are "**Clustering**" and "**Association**".

Clustering:

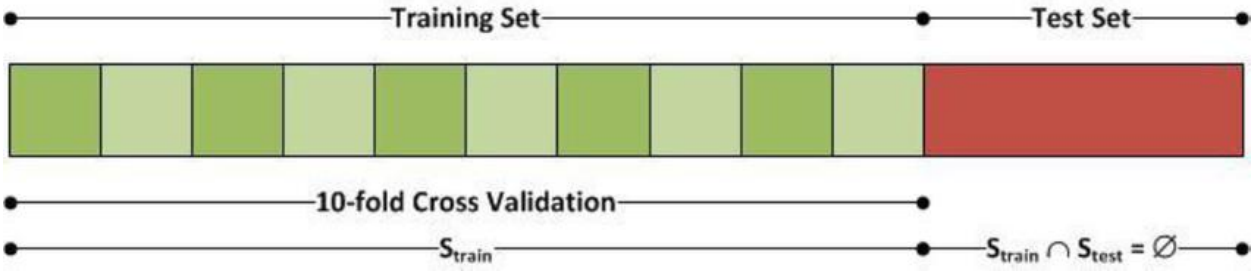
A set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.



4. Training and testing the model on data

For training a model we initially split the model into 3 three sections which are '**Training data**', **'Validation data**' and '**Testing data**'.

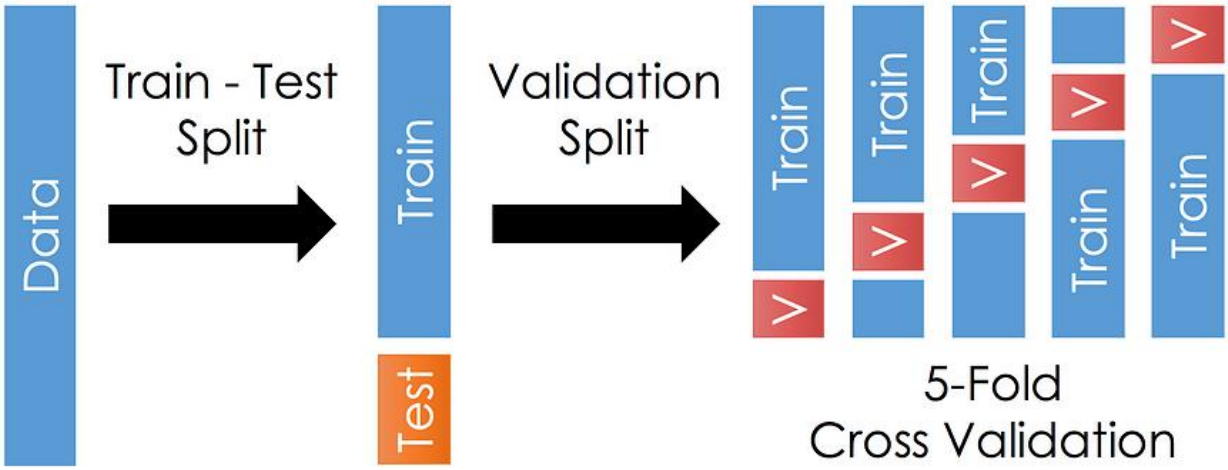
You train the classifier using '**training data set**', tune the parameters using '**validation set**' and then test the performance of your classifier on unseen '**test data set**'. An important point to note is that during training the classifier only the training and/or validation set is available. The test data set must not be used during training the classifier. The test set will only be available during testing the classifier.



Training set: The training set is the material through which the computer learns how to process information. Machine learning uses algorithms to perform the training part. A set of data used for learning, that is to fit the parameters of the classifier.

Validation set: Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. A set of unseen data is used from the training data to tune the parameters of a classifier.

Test set: A set of unseen data used only to assess the performance of a fully-specified classifier.



Once the data is divided into the 3 given segments, we can start the training process.

The model uses any one of the models that we had chosen in step 3/ point 3. Once the model is trained, we can use the same trained model to predict using the testing data i.e. the unseen data. Once this is done, we can develop a confusion matrix, this tells us how well our model is trained. A confusion matrix has 4 parameters, which are '**True positives'**, **'True Negatives'**, **'False Positives'** and '**False Negative'**. We prefer that we get more values in the True negatives and true positives to get a more accurate model. The size of the Confusion matrix completely depends upon the number of classes.

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

- **True positives:** These are cases in which we predicted TRUE and our predicted output is correct.
- **True negatives:** We predicted FALSE and our predicted output is correct.
- **False positives:** We predicted TRUE, but the actual predicted output is FALSE.
- **False negatives:** We predicted FALSE, but the actual predicted output is TRUE.

We can also find out the accuracy of the model using the confusion matrix.

Accuracy = (True Positives +True Negatives) / (Total number of classes)

i.e., for the above example:

Accuracy = (100 + 50) / 165 = 0.9090 (90.9% accuracy)

5. Evaluation

Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future.

To improve the model, we might tune the hyper-parameters of the model and try to improve the accuracy and also looking at the confusion matrix to try to increase the number of true positives and true negatives.

1.4 Challenges in ML

Data Quality Issues

- Missing values, imbalanced classes, noise
- *Example:* A cancer dataset where most patients are healthy causes bias.

Computational Complexity

- Training deep learning models can require GPUs and long runtimes.
- *Example:* Training GPT-style models takes weeks.

Interpretability

- Understanding how the model made a decision
- *Example:* Doctors need to know why a model predicted a disease.

Ethical Considerations

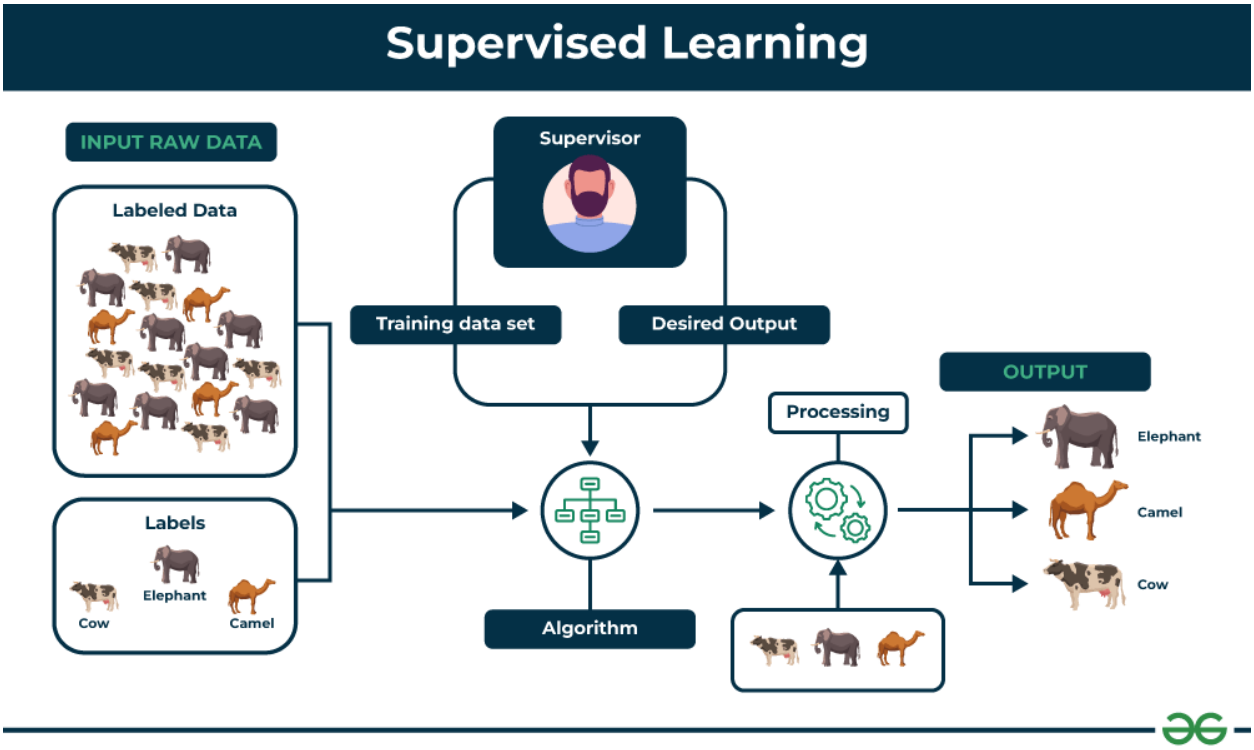
- Privacy, bias, misuse of ML
- *Example:* Facial recognition systems misidentifying people of certain ethnicities.

Chapter 2

Supervised Learning

Supervised learning algorithms are generally categorized into **two main types:**

- **Classification** - where the goal is to predict discrete labels or categories
- **Regression** - where the aim is to predict continuous numerical values.



There are many algorithms used in supervised learning each suited to different types of problems. Some of the most commonly used supervised learning algorithms are:

1. Linear Regression

This is one of the simplest ways to predict numbers using a straight line. It helps find the relationship between input and output.

2. Logistic Regression

Used when the output is a "yes or no" type answer. It helps in predicting categories like pass/fail or spam/not spam.

3. **Decision Trees**

A model that makes decisions by asking a series of simple questions, like a flowchart. Easy to understand and use.

4. Support Vector Machines (SVM)

A bit more advanced—it tries to draw the best line (or boundary) to separate different categories of data.

5. k-Nearest Neighbors (k-NN)

This model looks at the closest data points (neighbors) to make predictions. Super simple and based on similarity.

6. Naïve Bayes
A quick and smart way to classify things based on probability. It works well for text and spam detection.
7. Random Forest (Bagging Algorithm)
A powerful model that builds lots of decision trees and combines them for better accuracy and stability.

2.2. Regression

2.2.1 Linear Regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It provides valuable insights for prediction and data analysis. This article will explore its types, assumptions, implementation, advantages and evaluation metrics.

Linear regression is a supervised machine learning algorithm used to model the relationship between a dependent variable and one or more independent variables. It aims to find the line or surface that best fits the data, allowing for predictions of the dependent variable based on the independent variables. This algorithm is a fundamental tool in statistical analysis and machine learning, used for tasks like predicting house prices, forecasting sales, and analyzing the impact of different factors on a target variable.

Key Concepts and Components:

Dependent Variable (y): The variable that is being predicted or explained.

Independent Variables (x): The input variables or features used to predict the dependent variable.

Linear Equation: The algorithm uses a linear equation ($Y = mx + c$, where m is the slope and c are the y-intercept) to model the relationship between variables. In machine learning, this is often represented as $y' = b + w1*x1$, where y' is the predicted value, b is the bias, w1 is the weight, and x1 is the input feature.

Least Squares: A common method used to find the best-fit line by minimizing the sum of the squared differences between the predicted values and the actual values.

Model Parameters (Weights and Bias): The algorithm learns the optimal values for these parameters during training, which define the relationship between the variables.

Steps in Linear Regression:

1. Data Collection and Preparation:

Gather and prepare the data, including identifying relevant variables and addressing missing values or outliers.

2. Model Fitting:

Train the linear regression model using the data, typically using least squares or other optimization techniques to find the best-fit line or surface.

3. Model Evaluation:

Evaluate the performance of the model using metrics like Mean Squared Error (MSE) or R-squared to assess how well it fits the data.

4. Prediction:

Use the trained model to predict the value of the dependent variable for new, unseen data.

Types of Linear Regression:

Simple Linear Regression: Relates a single independent variable to a dependent variable.

Multiple Linear Regression: Relates multiple independent variables to a dependent variable.

Applications of Linear Regression:

Predictive Modeling: Forecasting future outcomes based on historical data.

Statistical Analysis: Identifying relationships between variables and understanding the impact of different factors.

Business and Finance: Predicting sales, stock prices, or customer behavior.

Social Sciences: Analyzing the impact of social factors on various outcomes.

Understanding Linear Regression

Linear regression is also a type of **supervised machine-learning algorithm** that learns from the labelled datasets and maps the data points with most optimized linear functions which can be used for prediction on new datasets. It computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation with observed data. It predicts the continuous output variables based on the independent input variable.

For example, if we want to predict house price, we consider various factor such as house age, distance from the main road, location, area and number of rooms, linear regression uses all these parameters to predict house price as it considers a linear relation between all these features and price of house.

Why Linear Regression is Important?

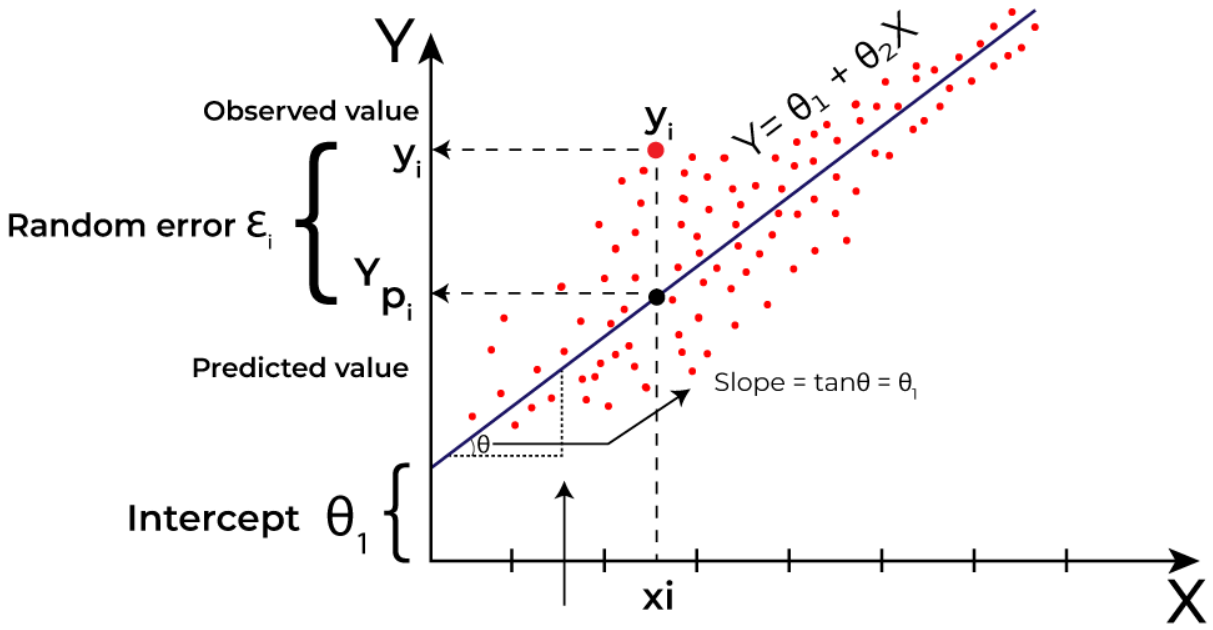
The interpretability of linear regression is one of its greatest strengths. The model's equation offers clear coefficients that illustrate the influence of each independent variable on the dependent variable, enhancing our understanding of the underlying relationships. Its simplicity is a significant advantage; linear regression is transparent, easy to implement, and serves as a foundational concept for more advanced algorithms.

Now that we have discussed why linear regression is important now we will discuss its working based on best fit line in regression.

What is the best Fit Line?

Our primary objective while using linear regression is to locate the best-fit line, which implies that the error between the predicted and actual values should be kept to a minimum. There will be the least error in the best-fit line.

The best Fit Line equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).



Linear Regression

Here Y is called a dependent or target variable and X is called an independent variable also known as the predictor of Y. There are many types of functions or modules that can be used for regression. A linear function is the simplest type of function. Here, X may be a single feature or multiple features representing the problem.

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x)). Hence, the name is Linear Regression. In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best-fit line for our model.

In linear regression some hypotheses are made to ensure reliability of the model's results.

Hypothesis function in Linear Regression

Assumptions are:

- **Linearity:** It assumes that there is a linear relationship between the independent and dependent variables. This means that changes in the independent variable lead to proportional changes in the dependent variable.
- **Independence**: The observations should be independent from each other that is the errors from one observation should not influence other.

As we have discussed that our independent feature is the experience i.e X and the respective salary Y is the dependent variable. Let's assume there is a linear relationship between X and Y then the salary can be predicted using:

$$\hat{Y} = \theta_1 + \theta_2 X$$

OR

$$\hat{y}_i = \theta_1 + \theta_2 x_i$$

Here,

- $y_i \epsilon Y \ \ (i = 1, 2, \cdots, n)$ are labels to data (Supervised learning)
- $x_i \epsilon X \ \ (i = 1, 2, \cdots, n)$ are the input independent training data (univariate – one input variable(parameter))
- $\hat{y}_i \epsilon \hat{Y} \ \ (i = 1, 2, \cdots, n)$ are the predicted values.

The model gets the best regression fit line by finding the best θ_1 and θ_2 values.

- **θ_1:** intercept
- **θ_2:** coefficient of x

Once we find the best θ_1 and θ_2 values, we get the best-fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

How to update θ1 and θ2 values to get the best-fit line?

To achieve the best-fit regression line, the model aims to predict the target value $\hat{Y}$ such that the error difference between the predicted value $\hat{Y}$ and the true value Y is minimum. So, it is very important to update the θ_1 and θ_2 values, to reach the best value that minimizes the error between the predicted y value (pred) and the true y value (y).

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

Types of Linear Regression

When there is only one independent feature it is known as **Simple Linear Regression** or Univariate Linear Regression and when there are more than one feature it is known as **Multiple Linear Regression** or Multivariate Regression.

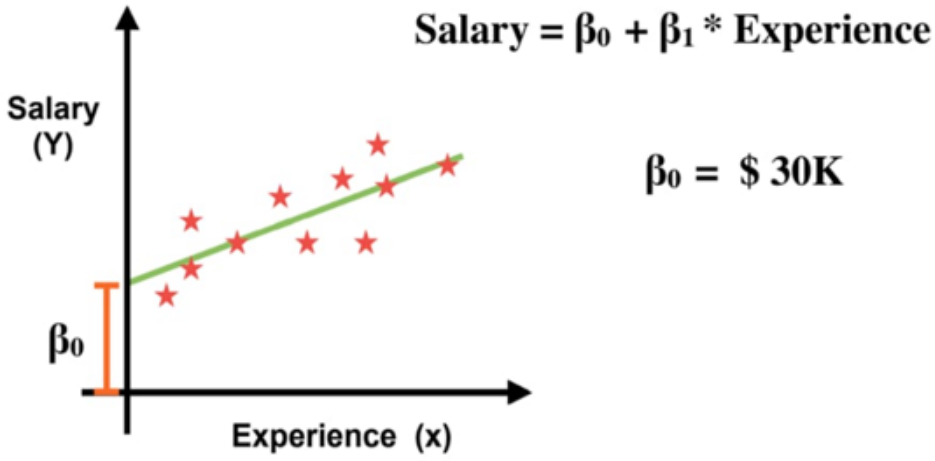
1. Simple Linear Regression

Simple linear regression is the simplest form of linear regression and it involves only one independent variable and one dependent variable. The equation for simple linear regression is:

$$y = \beta_0 + \beta_1 X$$

where:

- Y is the dependent variable
- X is the independent variable
- β0 is the intercept
- β1 is the slope



Simple linear regression is a statistical method used to model the relationship between two continuous variables, typically an independent (predictor) variable and a dependent (response) variable. It aims to find the best-fitting straight line that represents the relationship between the variables, often used for prediction or understanding how the independent variable affects the dependent variable.

Key Concepts and Steps:

1. Data Collection:

Gather data points for both the independent (x) and dependent (y) variables.

2. Linear Relationship Assumption:

Assume that a linear relationship exists between x and y, meaning that the change in y is proportional to the change in x.

3. Ordinary Least Squares (OLS):

The most common method for fitting a linear regression line is the ordinary least squares (OLS) method. This method minimizes the sum of the squared differences (residuals) between the observed y values and the values predicted by the regression line.

4. Finding the Slope and Intercept:

OLS finds the values of the slope (b1) and intercept (b0) that minimize the total error.

5. Regression Equation:

*The resulting regression equation takes the form: y = b0 + b1*x, where:*

y is the predicted value of the dependent variable.

b0 is the y-intercept (the value of y when x is zero).

b1 is the slope (the change in y for every unit change in x).

6. Prediction:

Use the regression equation to predict the value of y for a given value of x.

Example:

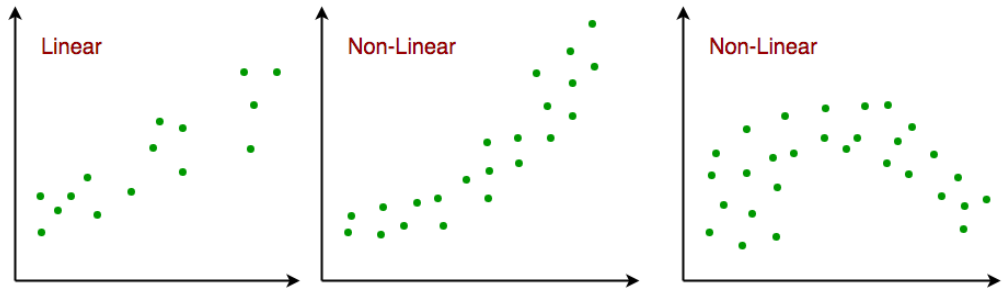
If you are trying to predict the sales of a product based on the amount spent on advertising, you could use simple linear regression to find the linear relationship between advertising spending (x) and sales (y). The regression equation would then allow you to estimate the sales for a given advertising budget.

In essence, simple linear regression provides a simple yet powerful way to model and understand relationships between two variables, making it a fundamental tool in various fields like statistics, machine learning, and data analysis.

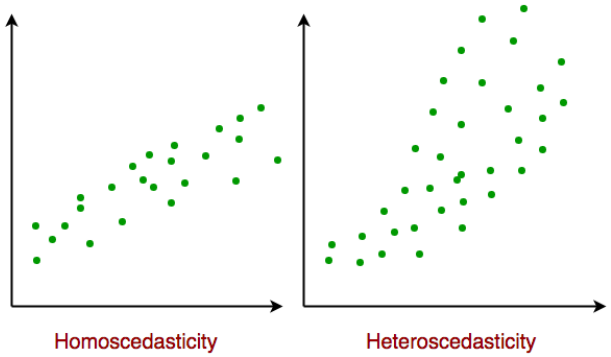
Assumptions of Simple Linear Regression

Linear regression is a powerful tool for understanding and predicting the behavior of a variable, however, it needs to meet a few conditions in order to be accurate and dependable solutions.

1. **Linearity**: The independent and dependent variables have a linear relationship with one another. This implies that changes in the dependent variable follow those in the independent variable(s) in a linear fashion. This means that there should be a straight line that can be drawn through the data points. If the relationship is not linear, then linear regression will not be an accurate model.



2. **Independence**: The observations in the dataset are independent of each other. This means that the value of the dependent variable for one observation does not depend on the value of the dependent variable for another observation. If the observations are not independent, then linear regression will not be an accurate model.
3. **Homoscedasticity**: Across all levels of the independent variable(s), the variance of the errors is constant. This indicates that the amount of the independent variable(s) has no impact on the variance of the errors. If the variance of the residuals is not constant, then linear regression will not be an accurate model.



Homoscedasticity in Linear Regression

4. **Normality**: The residuals should be normally distributed. This means that the residuals should follow a bell-shaped curve. If the residuals are not normally distributed, then linear regression will not be an accurate model.

2. Multiple Linear Regression

Multiple Linear Regression is an extension of this concept that allows us to model the relationship between a dependent variable and two or more independent variables. This

technique is used to understand how multiple features collectively affect the outcome, fitting a linear equation to predict the dependent variable based on the values of these features.

Steps for Multiple Linear Regression

Steps to perform multiple linear Regression are almost similar to that of simple linear Regression difference lies in the evaluation. We can use it to find out which factor has the highest impact on the predicted output and how different variables relate to each other.

Equation for multiple linear regression is:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

- $y$is the dependent variable
- $X_1, X_2, \cdots X_n$ are the independent variables
- β_0 is the intercept
- $\beta_1, \beta_2, \cdots \beta_n$are the slopes

The goal of the algorithm is to find the best fit line equation that can predict the values based on the independent variables. Regression model learns a function from the dataset (with known X and Y values) and uses it to predict Y values for unknown X.

$y = \beta_0 + \beta_1 X1 + \beta_2 X2 + \ldots\ldots\ldots \beta_n Xn$

where:

- Y is the dependent variable
- X1, X2, ..., Xn are the independent variables
- β0 is the intercept
- β1, β2, ..., βn are the slopes

The goal of the algorithm is to find the best Fit Line equation that can predict the values based on the independent variables.

In regression set of records are present with X and Y values and these values are used to learn a function so if you want to predict Y from an unknown X this learned function can be used. In regression we have to find the value of Y, So, a function is required that predicts continuous Y in the case of regression given X as independent features.

Handling Categorical Data with Dummy Variables

In multiple regression model, we often encounter **categorical data**, such as **gender** (male/female), **location** (urban/rural) etc. Since regression models typically expect numerical inputs, categorical data must be transformed into a usable form.

This is where **Dummy Variables** come into play. Dummy variables are binary variables (0 or 1) that represent the presence or absence of each category.

For example:

- **Male**: 1 if male, 0 otherwise
- **Female**: 1 if female, 0 otherwise

GENDER	MALE	FEMALE
Male	1	0
Male	1	0
Female	0	1
Female	0	1
Male	1	0
Female	0	1
Male	1	0

Assumptions of Multiple Linear Regression

For Multiple Linear Regression, all four of the assumptions from Simple Linear Regression apply. In addition to this, below are few more:

1. **No multicollinearity**: There is no high correlation between the independent variables. This indicates that there is little or no correlation between the independent variables. Multicollinearity occurs when two or more independent variables are highly correlated with each other, which can make it difficult to determine the individual effect of each variable on the dependent variable. If there is multicollinearity, then multiple linear regression will not be an accurate model.
2. **Additivity:** The model assumes that the effect of changes in a predictor variable on the response variable is consistent regardless of the values of the other variables. This assumption implies that there is no interaction between variables in their effects on the dependent variable.
3. **Feature Selection:** In multiple linear regression, it is essential to carefully select the independent variables that will be included in the model. Including irrelevant or redundant variables may lead to overfitting and complicate the interpretation of the model.
4. **Overfitting:** Overfitting occurs when the model fits the training data too closely, capturing noise or random fluctuations that do not represent the true underlying relationship between variables. This can lead to poor generalization performance on new, unseen data.

Numerical Example:

We have a small dataset:

x	y
1	2
2	3
4	5

We want to fit a line:

$$y = a + bx$$

Where:

- a: intercept
- b: slope

📐 Step 1: Compute Means

$$\bar{x} = \frac{1+2+4}{3} = \frac{7}{3} \approx 2.33$$
$$\bar{y} = \frac{2+3+5}{3} = \frac{10}{3} \approx 3.33$$

📉 Step 2: Compute Slope b

$$b = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

Compute each term:

x_i	y_i	$x_i - \bar{x}$	$y_i - \bar{y}$	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \bar{x})^2$
1	2	-1.33	-1.33	1.77	1.77
2	3	-0.33	-0.33	0.11	0.11
4	5	1.67	1.67	2.78	2.78

$$\text{Numerator } = 1.77 + 0.11 + 2.78 = 4.66$$
$$\text{Denominator } = 1.77 + 0.11 + 2.78 = 4.66 \Rightarrow b = \frac{4.66}{4.66} = 1.0$$

🧮 Step 3: Compute Intercept a

$$a = \bar{y} - b\bar{x} = 3.33 - 1 \cdot 2.33 = 1.0$$

✅ Final Model

$$\hat{y} = 1.0 + 1.0 \cdot x$$

🔍 Check Predictions

x	Actual y	Predicted y
1	2	2
2	3	3
4	5	5

Multicollinearity in Multiple Linear Regression

When building a multiple linear regression model multicollinearity can arise. It occurs when two or more independent variables are highly correlated with each other. This can make it difficult to evaluate the individual contribution of each variable to the dependent variable.

Detecting Multicollinearity includes two techniques:

1. **Correlation Matrix:** Examining the correlation matrix among the independent variables is a common way to detect multicollinearity. High correlations (close to 1 or -1) indicate potential multicollinearity.
2. **VIF (Variance Inflation Factor):** VIF is a measure that quantifies how much the variance of an estimated regression coefficient increases if your predictors are correlated. A high VIF (typically above 10) suggests multicollinearity.

Assumptions of Multiple Regression Model

Just like simple linear regression we have use some assumptions in multiple linear regression also:

1. **Linearity**: Relationship between dependent and independent variables should be linear.
2. **Homoscedasticity**: Variance of errors should remain constant across all levels of independent variables.
3. **Multivariate Normality**: Residuals should follow a normal distribution.
4. **No Multicollinearity**: Independent variables should not be highly correlated.

Numerical Example:

We use the model:

$$y = a + b_1 x_1 + b_2 x_2$$

📋 Dataset:

x_1	x_2	y
1	2	6
2	1	8
3	3	11

We want to find coefficients: a, b_1, b_2

🧠 Matrix Form of Multiple Linear Regression

$$\mathbf{y} = \mathbf{X} \cdot \mathbf{w}$$

Where:

- $\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_2 \end{bmatrix}$
- $\mathbf{w} = \begin{bmatrix} a \\ b_1 \\ b_2 \end{bmatrix}$

So for our data:

$$X = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 1 \\ 1 & 3 & 3 \end{bmatrix}, \quad y = \begin{bmatrix} 6 \\ 8 \\ 11 \end{bmatrix}$$

🔢 Step: Solve Normal Equation

$$\mathbf{w} = (X^T X)^{-1} X^T y$$

🧠 Step-by-step Calculation

Step 1: Compute X^TX

$$X^T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 2 & 1 & 3 \end{bmatrix}$$

Now compute X^TX:

$$X^TX = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 2 & 1 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 1 \\ 1 & 3 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 6 \\ 6 & 14 & 13 \\ 6 & 13 & 14 \end{bmatrix}$$

Step 2: Compute X^Ty

$$X^Ty = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 2 & 1 & 3 \end{bmatrix} \cdot \begin{bmatrix} 6 \\ 8 \\ 11 \end{bmatrix} = \begin{bmatrix} 6+8+11=25 \\ 6\times1+8\times2+11\times3=6+16+33=55 \\ 6\times2+8\times1+11\times3=12+8+33=53 \end{bmatrix} = \begin{bmatrix} 25 \\ 55 \\ 53 \end{bmatrix}$$

Step 3: Solve $w = (X^TX)^{-1}X^Ty$

We already have:

$$X^TX = \begin{bmatrix} 3 & 6 & 6 \\ 6 & 14 & 13 \\ 6 & 13 & 14 \end{bmatrix}, \quad X^Ty = \begin{bmatrix} 25 \\ 55 \\ 53 \end{bmatrix}$$

Use a tool (like NumPy or calculator) to compute the inverse:

$$(X^TX)^{-1} \approx \begin{bmatrix} 4.25 & -2.50 & -2.50 \\ -2.50 & 1.50 & 1.00 \\ -2.50 & 1.00 & 1.50 \end{bmatrix}$$

Now multiply:

$$w = (X^TX)^{-1}X^Ty = \begin{bmatrix} 4.25 & -2.50 & -2.50 \\ -2.50 & 1.50 & 1.00 \\ -2.50 & 1.00 & 1.50 \end{bmatrix} \cdot \begin{bmatrix} 25 \\ 55 \\ 53 \end{bmatrix}$$

$$w_0 = 4.25 \times 25 + (-2.5) \times 55 + (-2.5) \times 53 = 106.25 - 137.5 - 132.5 = 2.25$$

$$w_1 = -2.5 \times 25 + 1.5 \times 55 + 1 \times 53 = -62.5 + 82.5 + 53 = 73$$

$$w_2 = -2.5 \times 25 + 1 \times 55 + 1.5 \times 53 = -62.5 + 55 + 79.5 = 72.$$

$$:Now divide these by 36 (due to rounding difference from inverse approximation if done by hand) Final approximate solution : \[w = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

✅ Final Regression Equation:

$$\hat{y} = 2 + 2x_1 + 1x_2$$

Polynomial Regression

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modelled as an *nth-degree* polynomial. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y | x). In this article, we'll go in-depth about polynomial regression.

What is a Polynomial Regression?

- There are some relationships that a researcher will hypothesize is curvilinear. Clearly, such types of cases will include a polynomial term.
- Inspection of residuals. If we try to fit a linear model to curved data, a scatter plot of residuals (Y-axis) on the predictor (X-axis) will have patches of many positive residuals in the middle. Hence in such a situation, it is not appropriate.
- An assumption in the usual multiple linear regression analysis is that all the independent variables are independent. In the polynomial regression model, this assumption is not satisfied.

Why Polynomial Regression?

Polynomial regression is a type of regression analysis used in statistics and machine learning when the relationship between the independent variable (input) and the dependent variable (output) is not linear. While simple linear regression models the relationship as a straight line, polynomial regression allows for more flexibility by fitting a polynomial equation to the data.
When the relationship between the variables is better represented by a curve rather than a straight line, polynomial regression can capture the non-linear patterns in the data.

How does a Polynomial Regression work?

If we observe closely then we will realize that to evolve from linear regression to polynomial regression. We are just supposed to add the higher-order terms of the dependent features in the feature space. This is sometimes also known as feature engineering but not exactly.
When the relationship is non-linear, a polynomial regression model introduces higher-degree polynomial terms.
The general form of a polynomial regression equation of degree n is:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \epsilon$$

where,

- y is the dependent variable.
- x is the independent variable.
- $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients of the polynomial terms.
- n is the degree of the polynomial.
- ϵ represents the error term.

The basic goal of regression analysis is to model the expected value of a dependent variable y in terms of the value of an independent variable x. In simple linear regression, we used the following equation –

y = a + bx + e

Here y is a dependent variable, a is the y-intercept, b is the slope and e is the error rate. In many cases, this linear model will not work out For example if we analyze the production of chemical synthesis in terms of the temperature at which the synthesis takes place in such cases we use a quadratic model.

$$a + b_1 x + b_2 x^2 + e$$

Here,

- y is the dependent variable on x
- a is the y-intercept and e is the error rate.

In general, we can model it for the nth value. $y = a + b_1 x + b_2 x^2 + \ldots. + b_n x^n$

Since the regression function is linear in terms of unknown variables, hence these models are linear from the point of estimation. Hence through the Least Square technique, response value (y) can be computed.

By including higher-degree terms (quadratic, cubic, etc.), the model can capture the non-linear patterns in the data.

1. **The choice of the polynomial degree (*n*)** is a crucial aspect of polynomial regression. A higher degree allows the model to fit the training data more closely, but it may also lead to overfitting, especially if the degree is too high. Therefore, the degree should be chosen based on the complexity of the underlying relationship in the data.

2. The polynomial regression model is trained to **find the coefficients** that minimize the difference between the predicted values and the actual values in the training data.
3. Once the model is trained, it can be used to make predictions on new, unseen data. The polynomial equation captures the non-linear patterns observed in the training data, allowing the model to generalize to non-linear relationships.

Polynomial Regression Real-Life Example

Let's consider a real-life example to illustrate the application of polynomial regression. Suppose you are working in the field of finance, and you are analyzing the relationship between the years of experience (in years) an employee has and their corresponding salary (in dollars). You suspect that the relationship might not be linear and that higher degrees of the polynomial might better capture the salary progression over time.

Years of Experience	Salary (in dollars)
1	50,000
2	55,000
3	65,000
4	80,000
5	110,000
6	150,000
7	200,000

Now, let's apply polynomial regression to model the relationship between years of experience and salary. We'll use a quadratic polynomial (degree 2) for this example.

The quadratic polynomial regression equation is:

$$\text{Salary}=\beta_0 + \beta_1 \times\text{Experience}+\beta_2\times\text{Experience}\text{^}2+\epsilon$$

Now, to find the coefficients that minimize the difference between the predicted salaries and the actual salaries in the dataset we can use a method of least squares. The objective is to minimize the sum of squared differences between the predicted values and the actual values.

Polynomial Regression Algorithm (Step-by-Step)

Start with your dataset

- Input features: $X = [x_1, x_2, ..., x_m]$
- Output values: $y = [y_1, y_2, ..., y_m]$

Transform input features

- For degree n, transform each x into:

$$[1, x, x^2, x^3, ..., x^n]$$

This creates a new **feature matrix** X_{poly} of shape $m \times (n+1)$.

Fit the linear model

- Use **Linear Regression** on the transformed data:

$$\hat{y} = b_0 + b_1 x + b_2 x^2 + ... + b_n x^n$$

The model minimizes **Mean Squared Error (MSE)**:

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

- Solve using **Normal Equation** or **Gradient Descent**:

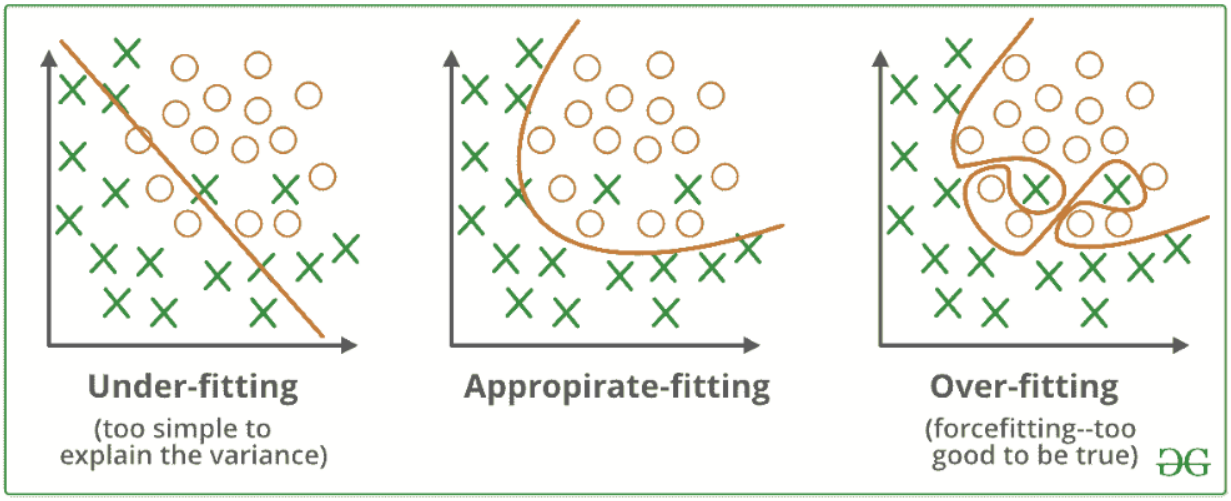
$$\theta = (X_{poly}^T X_{poly})^{-1} X_{poly}^T y$$

Predict on new input

- Transform new input x_{new} into polynomial form.
- Apply the learned model to get $\hat{y}_{new}$.

Step	Description
Feature Mapping	Add $x^2, x^3, \ldots, x^n$ to features
Linear Fit	Apply standard linear regression
Output	Predicted curve that fits the data

2.2.2 Regularization Techniques



In the previous session, we learned how to implement linear regression. Now, we'll move on to regularization, which helps prevent overfitting and makes our models work better with new data. While developing machine learning models we may encounter a situation where model is overfitted. To avoid such issues, we *use regularization techniques to improve model accuracy*.

What is Regularization?

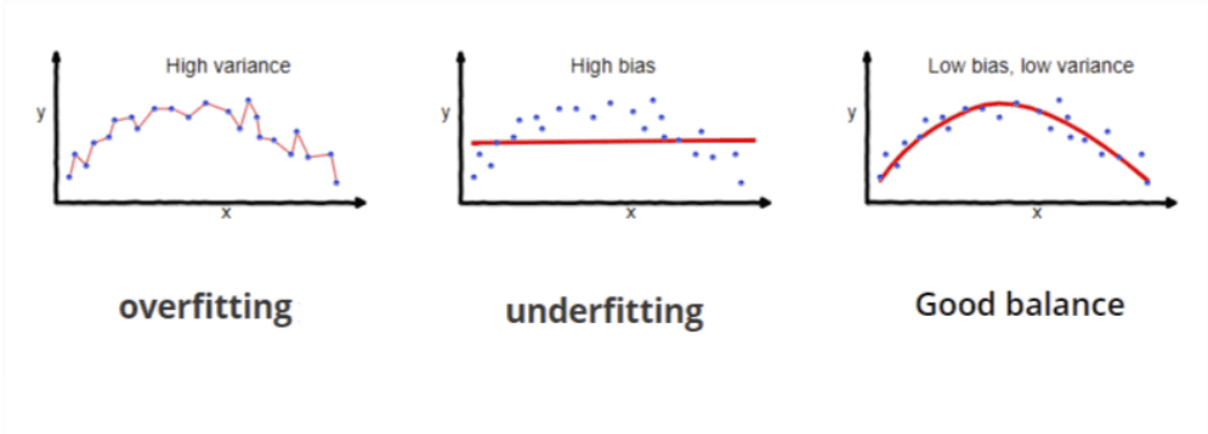
Regularization is a technique used in **machine learning** to prevent **overfitting**. Overfitting happens when a model learns the training data too well, including the noise and outliers, which causes it to perform poorly on new data. In simple terms, regularization adds a penalty to the model for being too complex, encouraging it to stay simpler and more general. This way, it's less likely to make extreme predictions based on the noise in the data.
The commonly used regularization techniques are:

📌 Types of Regularization:

Type	Penalty Term	Description
L1 (Lasso)	`λ * sum(	weights
L2 (Ridge)	`λ * sum(weights²)`	Penalizes large weights, leading to smoother models.
Elastic Net	Combination of L1 and L2	Balances sparsity and smoothness.

> Where `λ` (lambda) is the **regularization parameter** that controls the strength of the penalty.

1. **Lasso Regularization** – (L1 Regularization) (L1 regularization **removing less important features**.)
2. **Ridge Regularization** – (L2 Regularization)
3. **Elastic Net Regularization** – (L1 and L2 Regularization combined)



The image illustrates three scenarios in model performance:

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

Benefits:

- Reduces **overfitting**
- Improves **generalization**
- Can lead to **simpler, more interpretable models**

Types of Regularization

1. Lasso Regression

A regression model which uses the **L1 Regularization** technique is called **LASSO (Least Absolute Shrinkage and Selection Operator)** regression. **Lasso Regression** adds the *"absolute value of magnitude"* of the coefficient as a penalty term to the loss function(L). Lasso regression also helps us achieve feature selection by penalizing the weights to approximately equal to zero if that feature does not serve any purpose in the model.

Lasso minimizes the following cost function:

$$J(w) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T w)^2 + \alpha\sum_{j=1}^{p}|w_j|$$

Where:

- w = weights (coefficients)
- α = regularization strength (tunable hyperparameter)
- n = number of samples
- p = number of features

OR

$$\text{Loss} = \text{MSE} + \lambda\sum|w_j|$$

Where:

- $\text{MSE} = \frac{1}{n}\sum(y_i - \hat{y}_i)^2$
- w_j = coefficient for feature j
- λ = regularization parameter (controls the strength of penalty)

Basic Lasso Algorithm (Coordinate Descent)

Lasso is often solved using **coordinate descent**, an efficient iterative method:

Algorithm Steps:

1. **Initialize** all coefficients $w_j = 0$
2. Repeat until convergence:
   - For each coefficient w_j (while keeping others fixed):
     1. Compute the **partial residual** (excluding x_j):

$$r_j = y - \sum_{k \neq j} x_k w_k$$

     2. Compute:

$$\rho_j = \sum_{i=1}^{n} x_{ij} \cdot r_{ij}$$

     3. Update w_j using **soft thresholding**:

$$w_j = \frac{S(\rho_j, \alpha)}{\sum_{i=1}^{n} x_{ij}^2}$$

        Where:

$$S(\rho, \alpha) = \begin{cases} \rho - \alpha & \text{if } \rho > \alpha \\ 0 & \text{if } |\rho| \leq \alpha \\ \rho + \alpha & \text{if } \rho < -\alpha \end{cases}$$

3. Stop when changes in w are smaller than a threshold.

Key Features

- Encourages **sparsity**: Some coefficients become exactly zero.
- Useful for **feature selection** in high-dimensional datasets.
- Works well when only a few features are truly important.

Example:

Sample data:

x	y
1	2
2	3
3	4

Let's try to find the best weight w using Lasso.

📉 Step 1: Mean Squared Error (MSE)

Prediction:

$$\hat{y}_i = w \cdot x_i$$

MSE:

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

Try different values of w and compute loss for $\lambda = 1$:

🔬 Try $w = 1.0$

x	y	$\hat{y} = w \cdot x$	$(y - \hat{y})^2$
1	2	1	1
2	3	2	1
3	4	3	1

MSE = (1 + 1 + 1) / 3 = **1.0**
L1 penalty = $\lambda \cdot |w| = 1 \cdot 1.0 = 1.0$

Total Loss = 1.0 + 1.0 = 2.0

🔬 Try $w = 1.5$

x	y	$\hat{y}$	$(y - \hat{y})^2$
1	2	1.5	0.25
2	3	3.0	0.00
3	4	4.5	0.25

MSE = (0.25 + 0 + 0.25) / 3 = **0.167**

L1 penalty = $1 \cdot 1.5 = 1.5$

Total Loss = 0.167 + 1.5 = 1.667

🔬 Try $w = 2.0$

x	y	$\hat{y}$	$(y - \hat{y})^2$
1	2	2.0	0
2	3	4.0	1
3	4	6.0	4

MSE = (0 + 1 + 4) / 3 = **1.667**

L1 penalty = $1 \cdot 2.0 = 2.0$

Total Loss = 1.667 + 2.0 = 3.667

Result:

w	MSE	L1 Penalty	Total Loss
1.0	1.0	1.0	2.0
1.5	0.167	1.5	**1.667** ✅ (best)
2.0	1.667	2.0	3.667

- Lasso prefers **smaller weights** even if MSE slightly increases.
- It balances **fit quality** and **simplicity** (low weights).

💡 Summary

This simple example shows how Lasso:

- Iteratively updates weights using **coordinate descent**.
- Applies **soft thresholding** to introduce **sparsity**.
- Keeps relevant features (like x_1) and drops irrelevant ones (like x_2).

2. Ridge Regression

A regression model that uses the **L2 regularization** technique is called **Ridge regression**. **Ridge regression** adds the "*squared magnitude*" of the coefficient as a penalty term to the loss function(L).

$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m} w_i^2$$

Where,

- n = Number of examples (data points)
- m = Number of features (predictor variables)
- y_i = Actual target value for the iii-th example
- $\hat{y}_i$ = Predicted target value for the i-th example
- w_i = Coefficients of the features
- λ= Regularization parameter (controls the strength of regularization)

Or

$$\text{Loss} = \text{MSE}(y, \hat{y}) + \lambda \sum_{j=1}^{n} w_j^2$$

Where:

- y is the actual value
- $\hat{y}$ is the predicted value
- w_j is each model coefficient
- λ is the **regularization parameter** (controls the strength of the penalty)

The output shows the MSE, indicating model performance. Lower MSE means better accuracy. The coefficients **reflect the regularized feature weights.**

🔍 Comparison: Lasso vs Ridge

Feature	Lasso (L1)	Ridge (L2)
Coefficients	Can become **zero**	**Shrink**, but rarely zero
Feature selection	Yes	No
Use case	Sparse feature sets	Many correlated features

Ridge is preferred when:

- All features are expected to contribute.
- Data has **multicollinearity**.
- You want to **reduce model variance** without eliminating features.

Example:

$$y = w \cdot x$$

Sample data:

x	y
1	2
2	3
3	4

Try different values of w, compute:

- **MSE**
- **L2 penalty**
- **Total Ridge Loss**

Use $\lambda = 1$

🔬 Try $w = 1.5$

x	y	$\hat{y}$	$(y - \hat{y})^2$
1	2	1.5	0.25
2	3	3.0	0.0
3	4	4.5	0.25

MSE = (0.25 + 0 + 0.25) / 3 = 0.167

L2 Penalty = 1 × (1.5^2) = 2.25

Total Loss = 0.167 + 2.25 = 2.417

🔬 Try $w = 2.0$

x	y	$\hat{y}$	$(y - \hat{y})^2$
1	2	2.0	0.0
2	3	4.0	1.0
3	4	6.0	4.0

MSE = (0 + 1 + 4) / 3 = 1.667

L2 Penalty = 1 × (2.0^2) = 4.0

Total Loss = 1.667 + 4.0 = 5.667

Summary:

Weight w	MSE	L2 Penalty	Total Loss
1.0	1.000	1.000	2.000
1.5	0.167	2.250	2.417
2.0	1.667	4.000	5.667

Interpretation:

- Ridge regression **penalizes large weights**, even if they reduce the MSE.
- The model with **small weights + low penalty** is preferred.
- Ridge tends to **keep all features**, but shrinks their influence (unlike Lasso which can zero out weights).

3. Elastic Net Regression

Elastic Net Regression is a combination of both **L1 as well as L2 regularization.** That implies that we add the absolute norm of the weights as well as the squared measure of the weights. With the help of an extra hyperparameter that controls the ratio of the L1 and L2 regularization.

$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\left((1-\alpha)\sum_{i=1}^{m}|w_i| + \alpha\sum_{i=1}^{m}w_i^2\right)$$

Where,

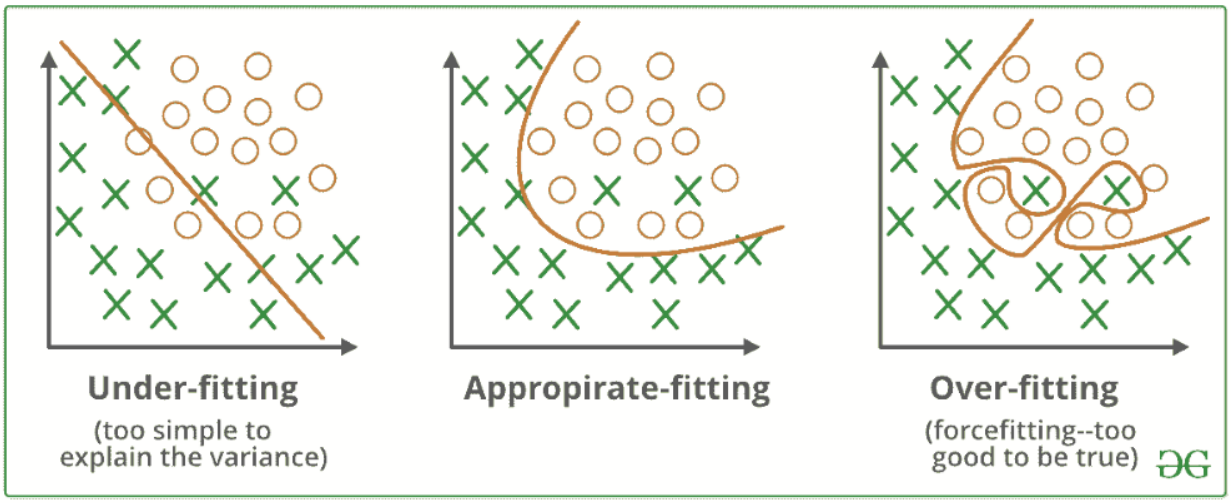
- n = Number of examples (data points)
- m = Number of features (predictor variables)
- y_i = Actual target value for the i-th example
- $\hat{y}_i$ = Predicted target value for the i-th example
- w_i = Coefficients of the features
- λ = Regularization parameter that controls the strength of regularization
- α = Mixing parameter (where $0 \leq \alpha \leq 1$):
    - $\alpha = 1$ corresponds to Lasso (L_1) regularization
    - $\alpha = 0$ corresponds to Ridge (L_2) regularization
    - Values between 0 and 1 provide a balance of both L_1 and L_2 regularization

MSE: Measures how far off predictions are from actual values. Lower is better.

Coefficients: Show feature importance. Elastic Net combines L1 and L2 to select features and prevent overfitting.

What is Overfitting and Underfitting?

Overfitting and **underfitting** are terms used to describe the performance of machine learning models in relation to their ability to generalize from the training data to unseen data.



Overfitting is a phenomenon that occurs when a machine learning model is constrained to training set and not able to perform well on unseen data. That is when our model learns the noise in the training data as well. Here, our model memorizes the training data instead of learning the patterns in it.

Imagine you study only last week's weather to predict tomorrows. Your model might predict tomorrow's weather based on irrelevant details, like a one-time rainstorm, which won't help for future predictions.

Underfitting on the other hand is the case when our model is not able to learn even the basic patterns available in the dataset. In the case of underfitting model is unable to perform well even on the training data hence we cannot expect it to perform well on the validation data. This is the case when we are supposed to increase the complexity of the model or add more features to the feature set.

Now, if you only use the average temperature of the year to predict tomorrow's weather, your model is too simple and misses important patterns, like seasonal changes, leading to poor predictions.

Bias and Variance

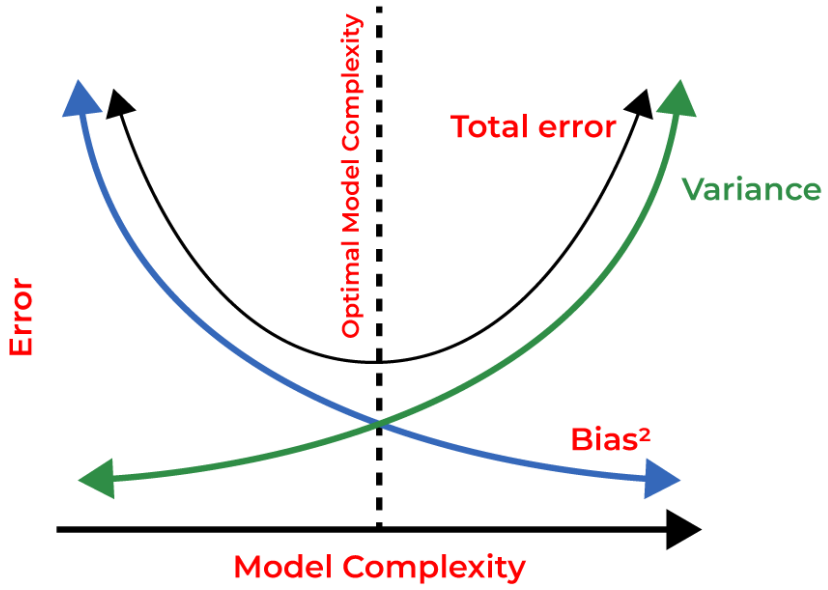
What are Bias and Variance?

- **Bias** refers to the errors which occur when we try to fit a statistical model on real-world data which does not fit perfectly well on some mathematical model. If we use a way too simplistic a model to fit the data then we are more probably face the situation of **High Bias** (underfitting) refers to the case when the model is unable to learn the patterns in the data at hand and perform poorly.
- **Variance** implies the error value that occurs when we try to make predictions by using data that is not previously seen by the model. There is a situation known as **high variance** (overfitting) that occurs when the model learns noise that is present in the data.

Finding a proper balance between the two is also known as the **Bias-Variance Tradeoff** which helps us to design an accurate model.

The **Bias-Variance Tradeoff** is a fundamental concept in machine learning. It refers to the balance between bias and variance, which affect predictive model performance. Finding the right tradeoff is crucial for creating models that generalize well to new data.

- The **bias-variance tradeoff** demonstrates the inverse relationship between bias and variance. When one decreases, the other tends to increase, and vice versa.
- Finding the right balance is crucial. An overly simple model with high bias won't capture the underlying patterns, while an overly complex model with high variance will fit the noise in the data.

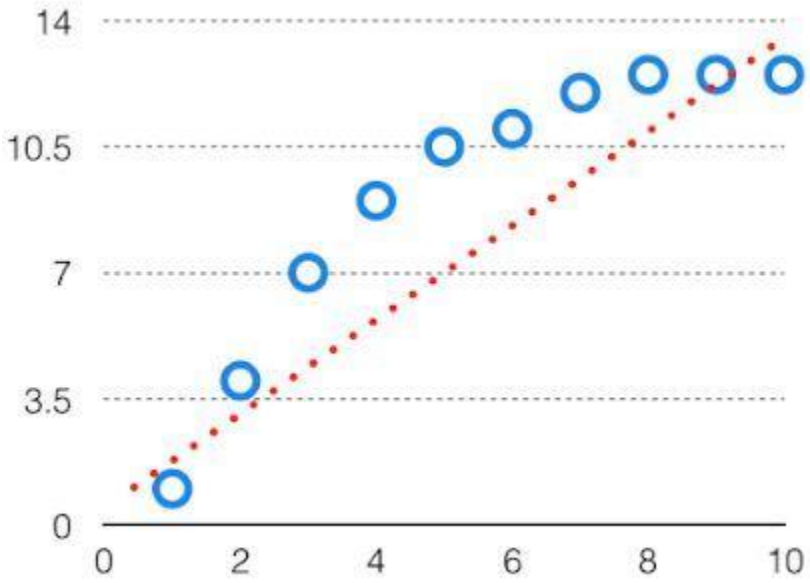


Bias-Variance Trade Off

It is important to understand prediction errors (bias and variance) when it comes to accuracy in any machine-learning algorithm. There is a tradeoff between a model's ability to minimize bias and variance which is referred to as the best solution for selecting a value of **Regularization** constant. A proper understanding of these errors would help to avoid the overfitting and underfitting of a data set while training the algorithm.

What is Bias?

The bias is known as the difference between the prediction of the values by the Machine Learning model and the correct value. Being high in biasing gives a large error in training as well as testing data. It recommended that an algorithm should always be low-biased to avoid the problem of underfitting. By high bias, the data predicted is in a straight-line format, thus not fitting accurately in the data in the data set. Such fitting is known as the **Underfitting of Data**. This happens when the hypothesis is too simple or linear in nature. Refer to the graph given below for an example of such a situation.



High Bias in the Model

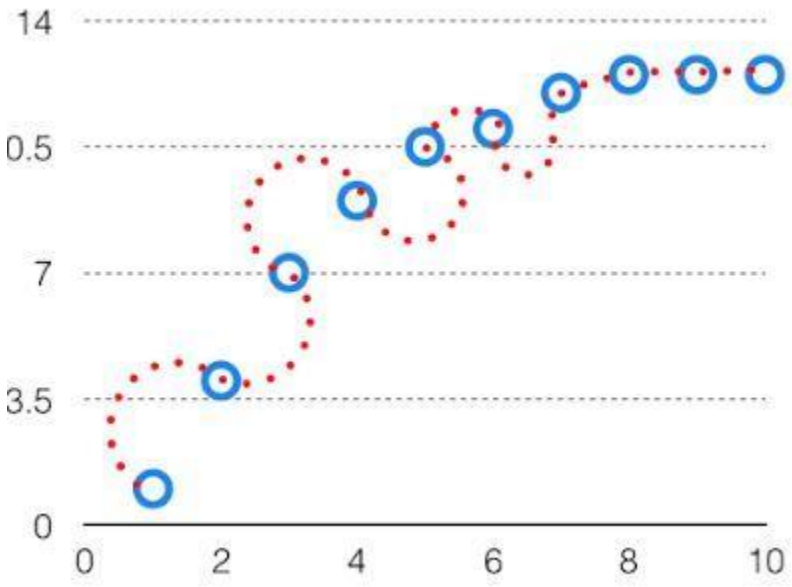
In such a problem, a hypothesis looks like follows.

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

What is Variance?

The variability of model prediction for a given data point which tells us the spread of our data is called the variance of the model. The model with high variance has a very complex fit to the training data and thus is not able to fit accurately on the data which it hasn't seen before. As a

result, such models perform very well on training data but have high error rates on test data. When a model is high on variance, it is then said to as **Overfitting of Data**. Overfitting is fitting the training set accurately via complex curve and high order hypothesis but is not the solution as the error with unseen data is high. While training a data model variance should be kept low. The high variance data looks as follows.

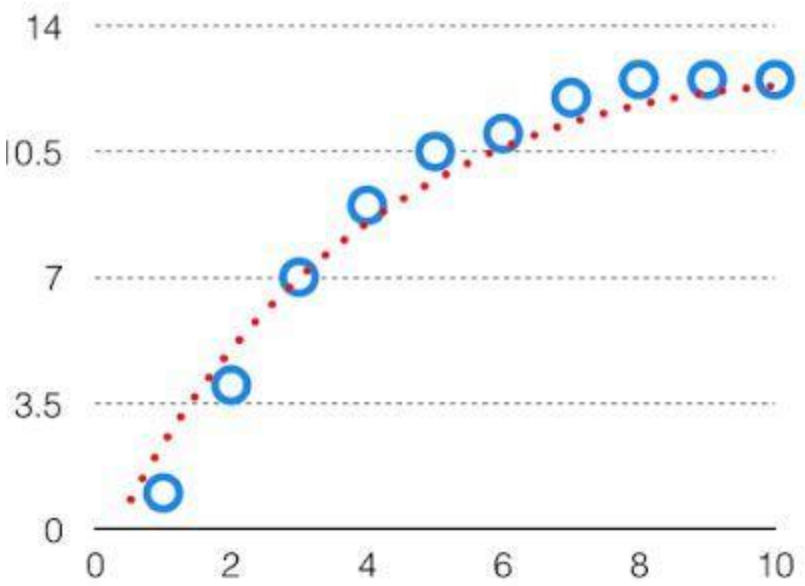


High Variance in the Model

In such a problem, a hypothesis looks like follows.

$$h_\theta(x) = g\left(\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4\right)$$

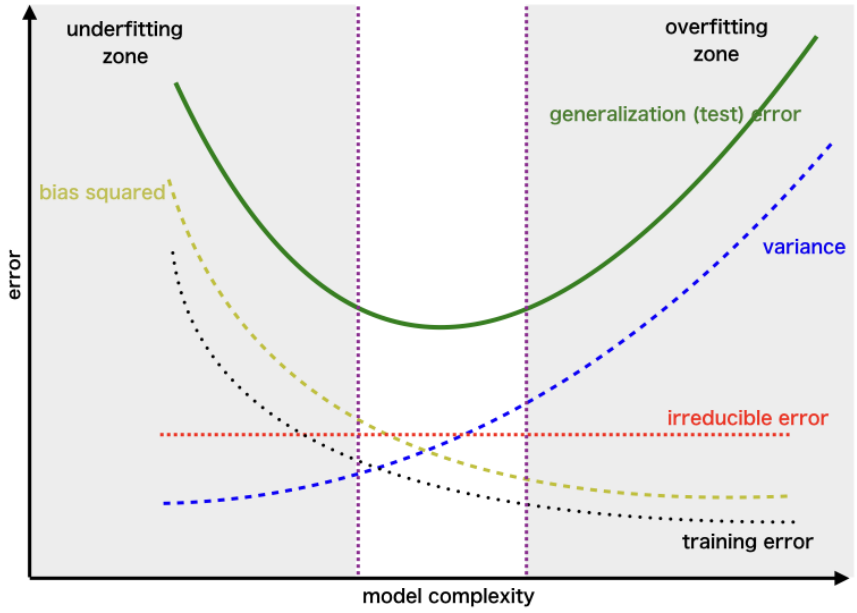
If the algorithm is too simple (hypothesis with linear equation) then it may be on high bias and low variance condition and thus is error-prone. If algorithms fit too complex (hypothesis with high degree equation) then it may be on high variance and low bias. In the latter condition, the new entries will not perform well. Well, there is something between both of these conditions, known as a Trade-off or Bias Variance Trade-off. This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time. For the graph, the perfect tradeoff will be like this.



We try to optimize the value of the total error for the model by using the Bias-Variance Tradeoff.

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

The best fit will be given by the hypothesis on the tradeoff point. The error to complexity graph to show trade-off is given as



Region for the Least Value of Total Error

This is referred to as the best point chosen for the training of the algorithm which gives low error in training as well as testing data.

Benefits of Regularization

Till now, we have understood about the Overfitting and Underfitting technique but lets have an understanding of the benefits of regularization.

- **Prevents Overfitting:** Regularization helps models focus on underlying patterns instead of memorizing noise in the training data.
- **Improves Interpretability:** L1 (Lasso) regularization simplifies models by reducing less important feature coefficients to zero.
- **Enhances Performance:** Prevents excessive weighting of outliers or irrelevant features, improving overall model accuracy.
- **Stabilizes Models:** Reduces sensitivity to minor data changes, ensuring consistency across different data subsets.
- **Prevents Complexity:** Keeps models from becoming too complex, which is crucial for limited or noisy data.
- **Handles Multicollinearity:** Reduces the magnitudes of correlated coefficients, improving model stability.
- **Allows Fine-Tuning:** Hyperparameters like alpha and lambda control regularization strength, balancing bias and variance.
- **Promotes Consistency:** Ensures reliable performance across different datasets, reducing the risk of large performance shifts.

Regularization techniques like L1 (Lasso), L2 (Ridge) and Elastic Net **play a important role in improving model performance by reducing overfitting and ensuring better generalization. By controlling complexity, selecting important features and stabilizing models these techniques help making more accurate predictions especially in large datasets.**

2.2.3 Support Vector Regression

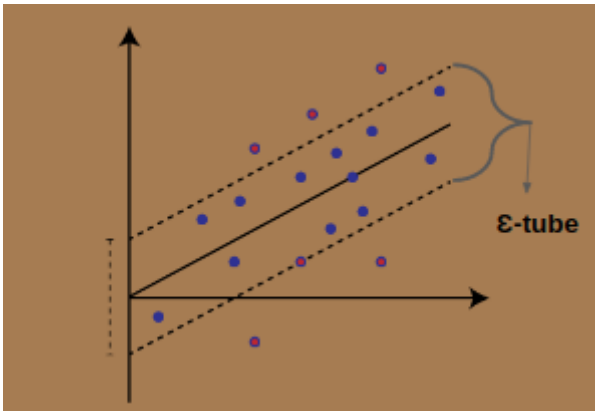
Support vector regression (SVR) is a type of support vector machine (SVM) that is used for regression tasks. It tries to find a function that best predicts the continuous output value for a given input value.

Support vector regression can solve both linear and non-linear models. SVM uses non-linear kernel functions (such as polynomial) to find the optimal solution for non-linear models. The main idea of SVR is to minimize error, individualizing the hyperplane which maximizes the margin.

SVR can use both linear and non-linear kernels. A linear kernel is a simple dot product between two input vectors, while a non-linear kernel is a more complex function that can capture more intricate patterns in the data. The choice of kernel depends on the data's characteristics and the task's complexity.

In scikit-learn package for Python, you can use the ***'SVR'*** class to perform SVR with a linear or non-linear ***'kernel'***. To specify the kernel, you can set the kernel parameter to ***'linear'*** or ***'RBF'*** (radial basis function).

SVM generalization to SVR is accomplished by introducing an Ɛ-insensitive region around the function, called the **Ɛ-tube**. This tube reformulates the optimization problem to find the tube that best approximates the continuous-valued function. The tube tries to balance model **complexity** and **prediction error**. To be more specific about creation of tube, SVR is formulated as an optimization problem by first defining a convex Ɛ-insensitive loss function to be minimized and finding the flattest tube that contains most of the training instances. The hyperplane is represented in terms of support vectors, which are training samples that lie outside the boundary of the tube. As in SVM, the support vectors in SVR are the most influential instances that affect the **shape of the tube**, and the **training** and **test** data are assumed to be **independent and identically distributed**.



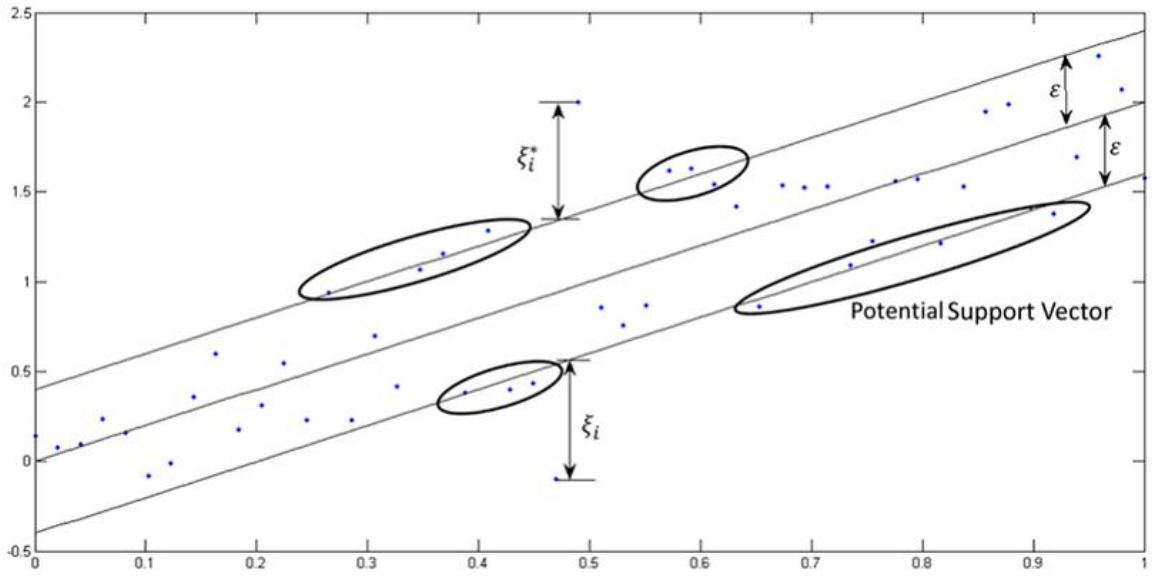
SVR: Mathematical and Graphical Representation:

SVR problem formulation is often best derived from a geometrical perspective. Dealing with two types of Data:

One Dimensional Data: Mathematical Implementation of continuous-valued function being approximated can be written as in Equation below-

$$y = f(x) = < w, x > + b = \sum_{j=1}^{M} w_j x_j + b,\ y,\ b \in \mathbb{R}, x, w \in \mathbb{R}^M$$

Equation representing One-dimensional formulation.



One- Dimensional Linear SVR. Source: Google Images.

The former condition produces the objective function in Equation, where ||***w***|| is the magnitude of the normal vector to the surface that is being approximated:

$$\min_w \frac{1}{2}\|w\|^2.$$

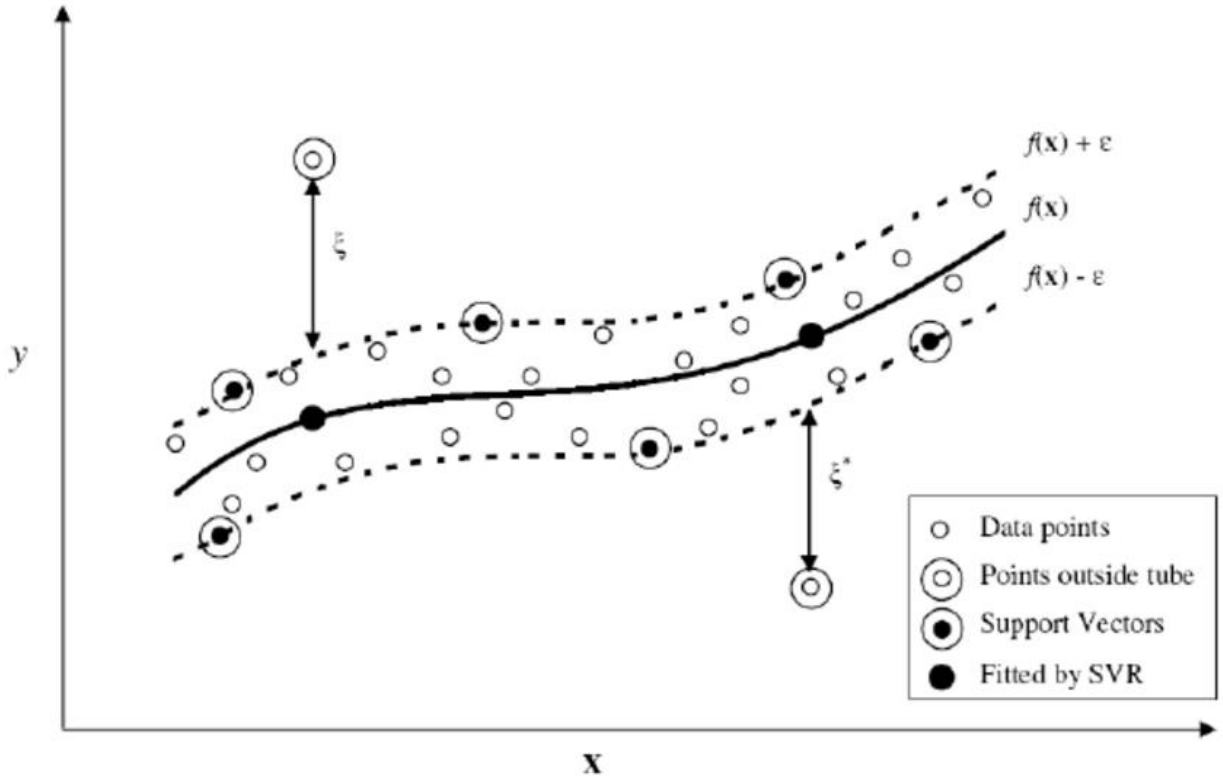
Formula to minimize the distance.

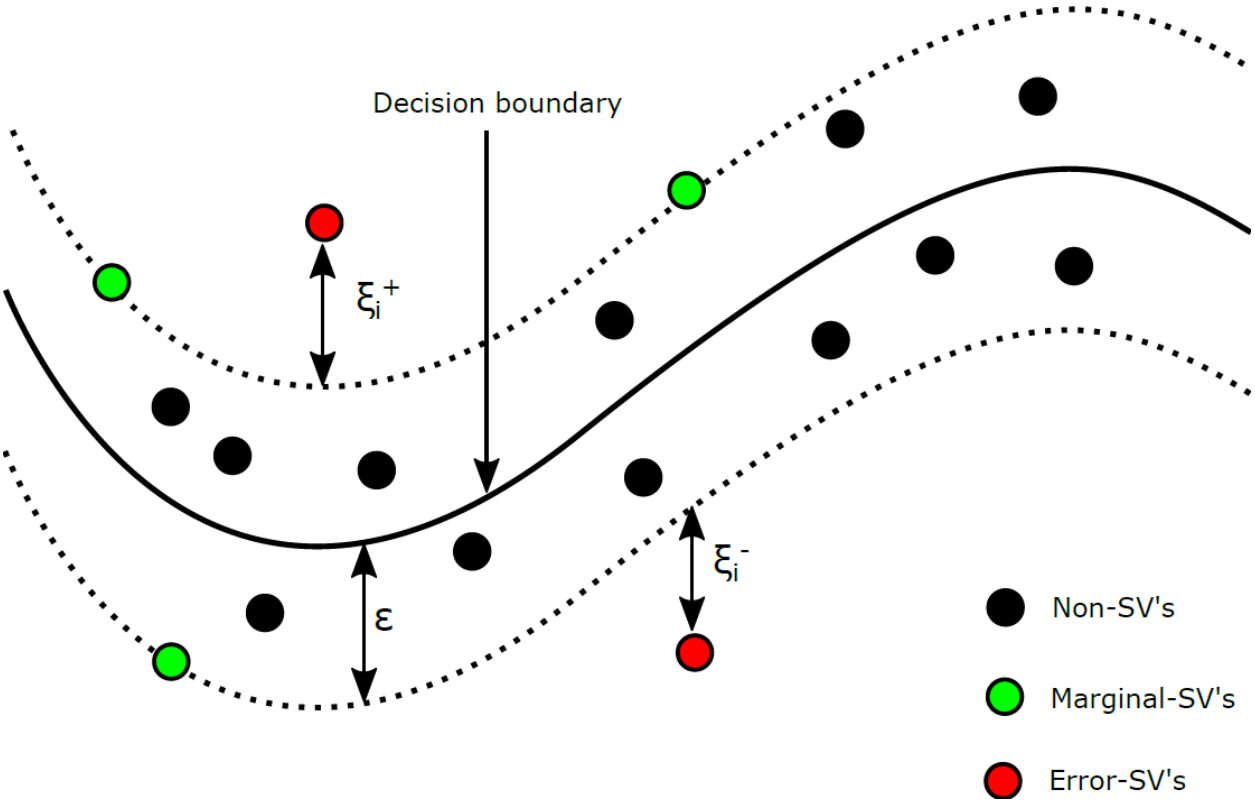
Multidimensional Data: We augment x by one and include b in the w vector to simply the mathematical notation, and obtain the multivariate regression in Equation below-

$$f(x) = \begin{bmatrix} w \\ b \end{bmatrix}^T \begin{bmatrix} x \\ 1 \end{bmatrix} = w^T x + b \quad x, w \in \mathbb{R}^{M+1}$$

Equation representing Multidimensional formulation.

Here, M is the order of the polynomial used to approximate a function. As the magnitude of the vector w increases, a greater number of ***wi*** are nonzero, resulting in higher-order solutions. The horizontal line is a 0th-order polynomial solution and has a very large deviation from the desired outputs, and thus, a large error. The linear function, a 1st-order polynomial, produces better approximations for a portion of the data but still underfits the training data. The 6th-order solution produces the best tradeoff between function flatness and prediction error. The highest-order solution has zero error but a high complexity and will most likely overfit the solution on yet to be seen data. The magnitude of w acts as a regularizing term and provides optimization problem control over the flatness of the solution.



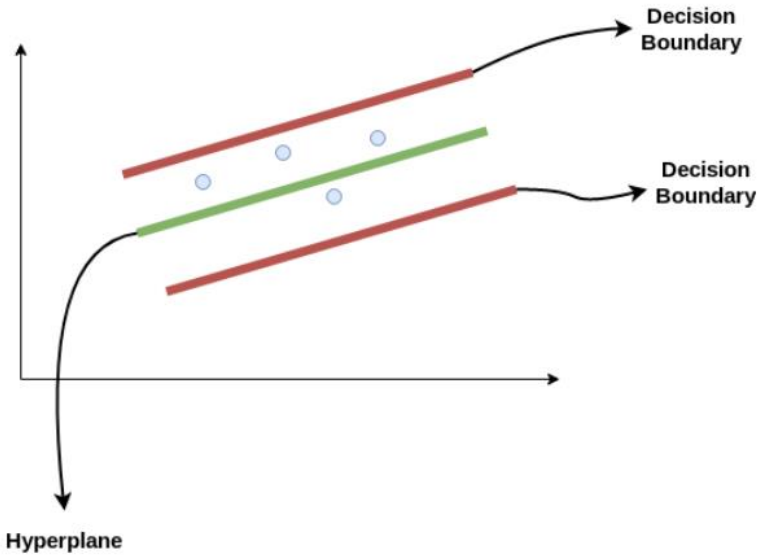


Concepts related to the Support vector regression (SVR):

There are several concepts related to support vector regression (SVR) that you may want to understand in order to use it effectively. Here are a few of the most important ones:

- **Support vector machines (SVMs):** SVR is a type of support vector machine (SVM), a supervised learning algorithm that can be used for classification or regression tasks. SVMs try to find the hyperplane in a high-dimensional space that maximally separates different classes or output values.
- **Kernels:** SVR can use different types of kernels, which are functions that determine the similarity between input vectors. A linear kernel is a simple dot product between two input vectors, while a non-linear kernel is a more complex function that can capture more intricate patterns in the data. The choice of kernel depends on the data's characteristics and the task's complexity.
- **Hyperparameters:** SVR has several hyperparameters that you can adjust to control the behavior of the model. For example, the **'C'** parameter controls the trade-off between the insensitive loss and the sensitive loss. A larger value of **'C'** means that the model will try to minimize the insensitive loss more, while a smaller value of C means that the model will be more lenient in allowing larger errors.
- **Model evaluation:** Like any machine learning model, it's important to evaluate the performance of an SVR model. One common way to do this is to split the data into a

training set and a test set, and use the training set to fit the model and the test set to evaluate it. You can then use metrics like mean squared error (MSE) or mean absolute error (MAE) to measure the error between the predicted and true output values.



2.3. Classification

The classifier algorithms are designed to indicate whether a new data point belongs to one or another among several predefined classes. Imagine when you are organizing emails into spam or inbox, categorizing images as cat or dog, or predicting whether a loan applicant is a credible borrower.

Common Classification Algorithms:

- **Logistic Regression:** A very efficient technique for the classification problems of binary nature (two types, for example, spam/not spam).
- **Support Vector Machine (SVM):** Good for tasks like classification, especially when the data has a large number of features.
- **Decision Tree:** Constructs a decision tree having branches and proceeds to the class predictions through features.
- **Random Forest:** The model generates an "ensemble" of decision trees that ultimately raise the accuracy and avoid overfitting (meaning that the model performs great on the training data but lousily on unseen data).
- **K-Nearest Neighbors (KNN):** Assigns a label of the nearest neighbors for a given data point.

2.3.1. Logistic Regression

Logistic regression is a **supervised machine learning algorithm** used for **classification tasks** where the goal is to predict the probability that an instance belongs to a given class or not. Logistic regression is a statistical algorithm which analyze the relationship between two data factors. The article explores the fundamentals of logistic regression, it's types and implementations.

Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1.

For example, we have two classes Class 0 and Class 1 if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 otherwise it belongs to Class 0. It's referred to as regression because it is the extension of linear regression but is mainly used for classification problems.

Key Points:

- Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value.
- It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

Types of Logistic Regression

On the basis of the categories, Logistic Regression can be classified into three types:

1. **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
2. **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
3. **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Assumptions of Logistic Regression

We will explore the assumptions of logistic regression as understanding these assumptions is important to ensure that we are using appropriate application of the model. The assumption include:

1. Independent observations: Each observation is independent of the other. meaning there is no correlation between any input variables.

2. Binary dependent variables: It takes the assumption that the dependent variable must be binary or dichotomous, meaning it can take only two values. For more than two categories SoftMax functions are used.

3. Linearity relationship between independent variables and log odds: The relationship between the independent variables and the log odds of the dependent variable should be linear.

4. No outliers: There should be no outliers in the dataset.

5. Large sample size: The sample size is sufficiently large

Understanding Sigmoid Function

So far, we've covered the basics of logistic regression, but now let's focus on the most important function that forms the core of logistic regression**.**

- The **sigmoid function** is a mathematical function used to map the predicted values to probabilities.

- It maps any real value into another value within a range of **0 and 1**. The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "**S**" form.

- The S-form curve is called the **Sigmoid function or the logistic function.**

- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

How does Logistic Regression work?

The logistic regression model transforms the linear regression function continuous value output into categorical value output using a sigmoid function, which maps any real-valued set of independent variables input into a value between 0 and 1. This function is known as the logistic function.

Let the independent input features be:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$$

and the dependent variable is Y having only binary value i.e., 0 or 1.

$$Y = \begin{cases} 0 & \text{if } Class\ 1 \\ 1 & \text{if } Class\ 2 \end{cases}$$

then, apply the multi-linear function to the input variables X.

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

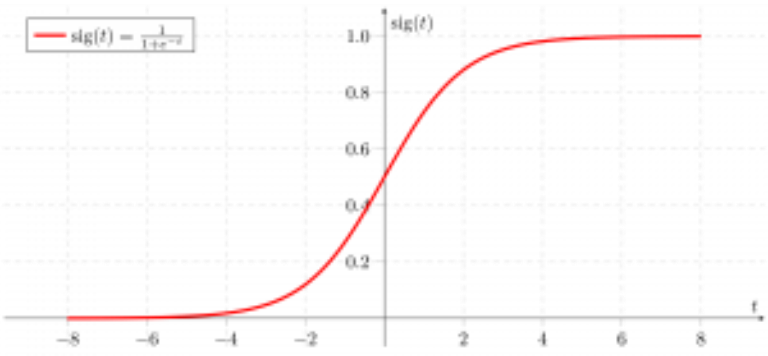
Here x_i is the ith observation of X, $w_i = [w_1, w_2, w_3, \cdots, w_m]$ is the weights or Coefficient, and b is the bias term also known as intercept. simply this can be represented as the dot product of weight and bias.

$$z = w \cdot X + b$$

Sigmoid Function

Now we use the sigmoid function where the input will be z and we find the probability between 0 and 1. i.e. predicted y.

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



Sigmoid function

As shown above, the figure sigmoid function converts the continuous variable data into the probability i.e. between 0 and 1.

- σ(z) $\sigma(z)$ tends towards 1 as z→∞z→∞
- σ(z) $\sigma(z)$ tends towards 0 as z→−∞z→−∞
- σ(z) $\sigma(z)$ is always bounded between 0 and 1

where the probability of being a class can be measured as:

$$P(y=1)=\sigma(z)$$

$$P(y=0)=1-\sigma(z)$$

The odd is the ratio of something occurring to something not occurring. it is different from probability as the probability is the ratio of something occurring to everything that could possibly occur. so odd will be:

Types of Classification

There are different types of classification problems depending on how many categories (or classes) we are working with and how they are organized. There are two main classification types in machine learning:

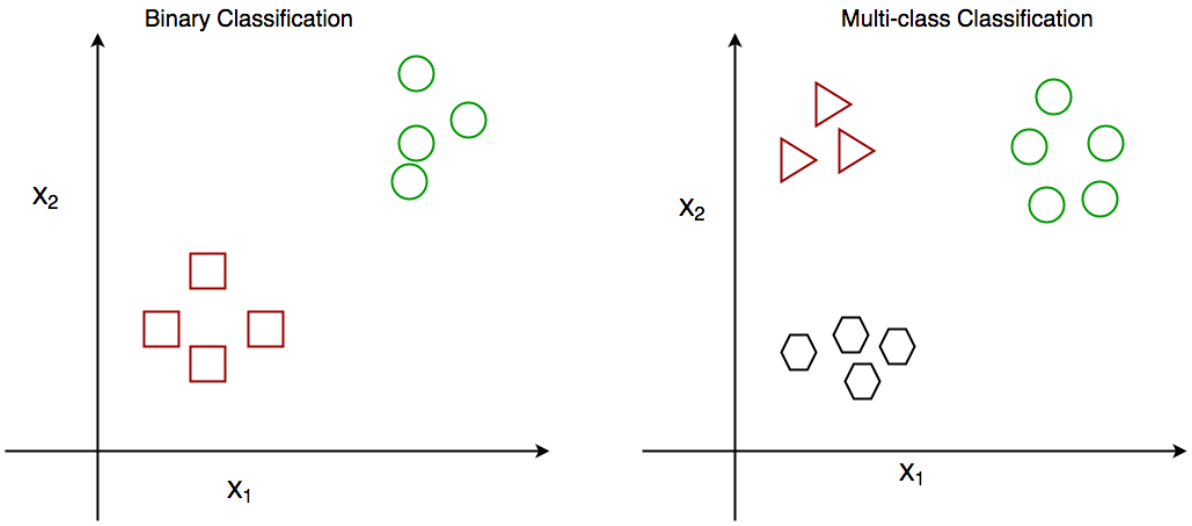
1. Binary Classification

This is the simplest kind of classification. In binary classification, the goal is to sort the data into two distinct categories. Think of it like a simple choice between two options. Imagine a system that sorts emails into either spam or not spam. It works by looking at different features of the email like certain keywords or sender details, and decides whether it's spam or not. It only chooses between these two options.

2. Multiclass Classification

Here, instead of just two categories, the data needs to be sorted into more than two categories. The model picks the one that best matches the input. Think of an image recognition system that sorts pictures of animals into categories like cat, dog, and bird.

Basically, machine looks at the features in the image (like shape, color, or texture) and chooses which animal the picture is most likely to be based on the training it received.

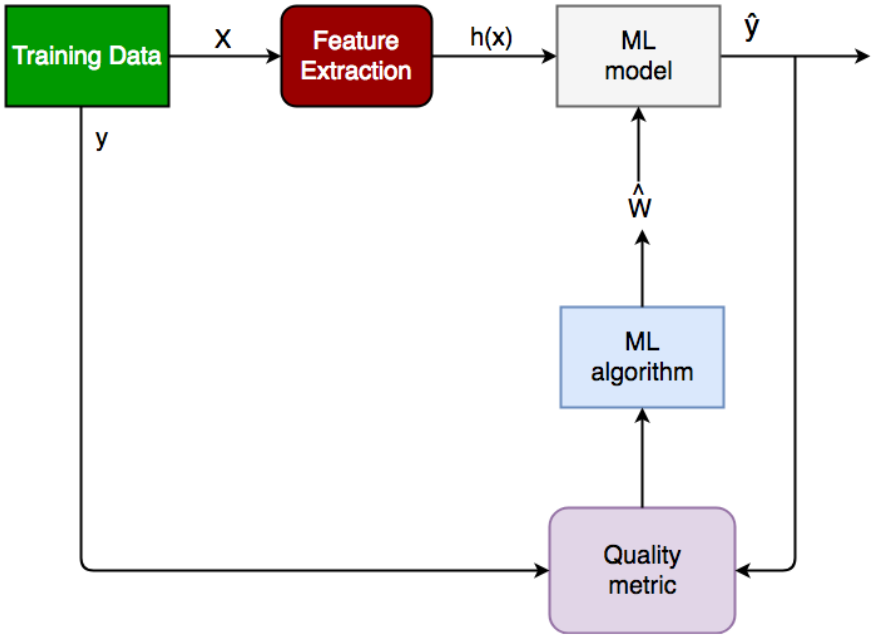


Binary classification vs Multi class classification

3. Multi-Label Classification

In multi-label classification single piece of data can belong to multiple categories at once. Unlike multiclass classification where each data point belongs to only one class, multi-label classification allows datapoints to belong to multiple classes. A movie recommendation system could tag a movie as both action and comedy. The system checks various features (like movie

plot, actors, or genre tags) and assigns multiple labels to a single piece of data, rather than just one.



Examples of Machine Learning Classification in Real Life

Classification algorithms are widely used in many real-world applications across various domains, including:

- **Email spam filtering**
- **Credit risk assessment**: Algorithms predict whether a loan applicant is likely to default by analyzing factors such as credit score, income, and loan history. This helps banks make informed lending decisions and minimize financial risk.
- **Medical diagnosis**: Machine learning models classify whether a patient has a certain condition (e.g., cancer or diabetes) based on medical data such as test results, symptoms, and patient history. This aids doctors in making quicker, more accurate diagnoses, improving patient care.
- **Image classification**: Applied in fields such as facial recognition, autonomous driving, and medical imaging.
- Sentiment analysis: Determining whether the sentiment of a piece of text is positive, negative, or neutral. Businesses use this to understand customer opinions, helping to improve products and services.

- **Fraud detection**: Algorithms detect fraudulent activities by analyzing transaction patterns and identifying anomalies crucial in protecting against credit card fraud and other financial crimes.
- **Recommendation systems**: Used to recommend products or content based on past user behavior, such as suggesting movies on Netflix or products on Amazon. This personalization boosts user satisfaction and sales for businesses.

Classification Modeling in Machine Learning

Now that we understand the fundamentals of classification, it's time to explore how we can use these concepts to build classification models. Classification modeling refers to the process of using machine learning algorithms to categorize data into predefined classes or labels. These models are designed to handle both binary and multi-class classification tasks, depending on the nature of the problem. Let's see key characteristics of Classification Models:

1. Class Separation: Classification relies on distinguishing between distinct classes. The goal is to learn a model that can separate or categorize data points into predefined classes based on their features.
2. Decision Boundaries: The model draws decision boundaries in the feature space to differentiate between classes. These boundaries can be linear or non-linear.
3. Sensitivity to Data Quality: Classification models are sensitive to the quality and quantity of the training data. Well-labeled, representative data ensures better performance, while noisy or biased data can lead to poor predictions.
4. Handling Imbalanced Data: Classification problems may face challenges when one class is underrepresented. Special techniques like resampling or weighting are used to handle class imbalances.
5. Interpretability: Some classification algorithms, such as Decision Trees, offer higher interpretability, meaning it's easier to understand why a model made a particular prediction.

2.3.2 Support Vector Regression

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. While it can handle regression problems, SVM is particularly well-suited for classification tasks.

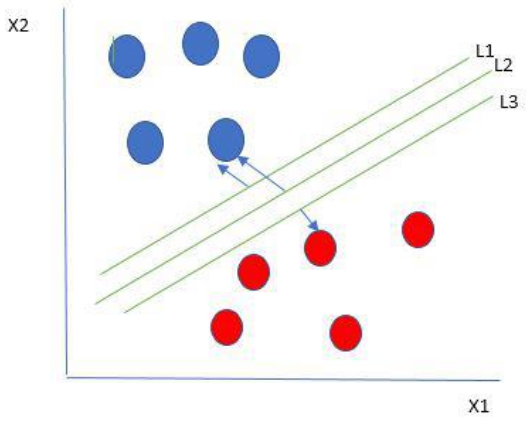
SVM aims to find the optimal hyperplane in an N-dimensional space to separate data points into different classes. The algorithm maximizes the margin between the closest points of different classes.

Support Vector Machine (SVM) Terminology

- **Hyperplane**: A decision boundary separating different classes in feature space, represented by the equation **wx + b = 0** in linear classification.
- **Support Vectors**: The closest data points to the hyperplane, crucial for determining the hyperplane and margin in SVM.
- **Margin**: The distance between the hyperplane and the support vectors. SVM aims to maximize this margin for better classification performance.
- **Kernel**: A function that maps data to a higher-dimensional space, enabling SVM to handle non-linearly separable data.
- **Hard Margin**: A maximum-margin hyperplane that perfectly separates the data without misclassifications.
- **Soft Margin**: Allows some misclassifications by introducing slack variables, balancing margin maximization and misclassification penalties when data is not perfectly separable.
- **C**: A regularization term balancing margin maximization and misclassification penalties. A higher C value enforces a stricter penalty for misclassifications.
- **Hinge Loss**: A loss function penalizing misclassified points or margin violations, combined with regularization in SVM.
- **Dual Problem**: Involves solving for Lagrange multipliers associated with support vectors, facilitating the kernel trick and efficient computation.

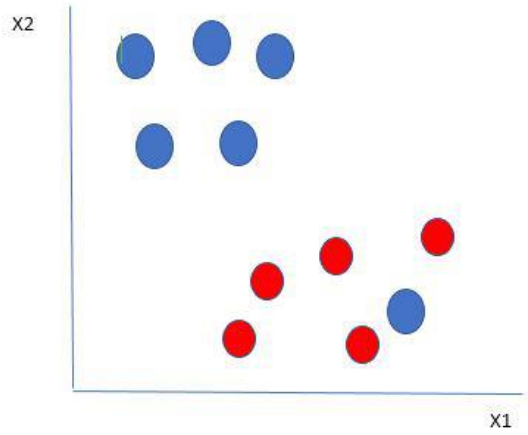
How does Support Vector Machine Algorithm Work?

The key idea behind the SVM algorithm is to find the hyperplane that best separates two classes by maximizing the margin between them. This margin is the distance from the hyperplane to the nearest data points (**support vectors**) on each side.



Multiple hyperplanes separate the data from two classes

The best hyperplane, also known as the **"hard margin,"** is the one that maximizes the distance between the hyperplane and the nearest data points from both classes. This ensures a clear separation between the classes. So, from the above figure, we choose L2 as hard margin.
Let's consider a scenario like shown below:

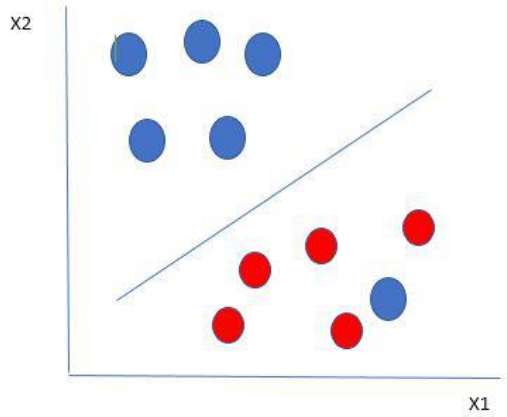


Selecting hyperplane for data with outlier

Here, we have one blue ball in the boundary of the red ball.

How does SVM classify the data?

It's simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.



Hyperplane which is the most optimized one

A soft margin allows for some misclassifications or violations of the margin to improve generalization. The SVM optimizes the following equation to balance margin maximization and penalty minimization:

$$\text{Objective Function} = \left(\frac{1}{\text{margin}}\right) + \lambda \sum \text{penalty}$$

The penalty used for violations is often **hinge loss**, which has the following behavior:

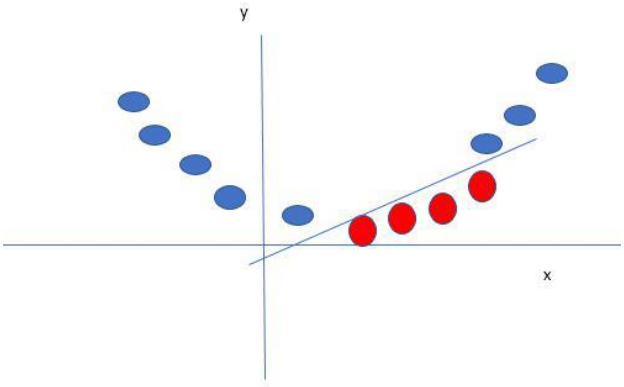
- If a data point is correctly classified and within the margin, there is no penalty (loss = 0).
- If a point is incorrectly classified or violates the margin, the hinge loss increases proportionally to the distance of the violation.

Till now, we were talking about linearly separable data (the group of blue balls and red balls are separable by a straight line/linear line).

A **kernel** is a function that maps data points into a higher-dimensional space without explicitly computing the coordinates in that space. This allows SVM to work efficiently with non-linear data by implicitly performing the mapping.

For example, consider data points that are not linearly separable. By applying a kernel function, SVM transforms the data points into a higher-dimensional space where they become linearly separable.

- **Linear Kernel**: For linear separability.
- **Polynomial Kernel**: Maps data into a polynomial space.
- **Radial Basis Function (RBF) Kernel**: Transforms data into a space based on distances between data points.



Mapping 1D data to 2D to become able to separate the two classes

In this case, the new variable y is created as a function of distance from the origin.

Mathematical Computation: SVM

Consider a binary classification problem with two classes, labeled as +1 and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y.

The equation for the linear hyperplane can be written as:

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

Where:

- w is the normal vector to the hyperplane (the direction perpendicular to it).
- b is the offset or bias term, representing the distance of the hyperplane from the origin along the normal vector w.

Distance from a Data Point to the Hyperplane

The distance between a data point x_i and the decision boundary can be calculated as:

$$d_i = \frac{w^T x_i + b}{||w||}$$

where ||w|| represents the Euclidean norm of the weight vector w. Euclidean norm of the normal vector W

Types of Support Vector Machine

Based on the nature of the decision boundary, Support Vector Machines (SVM) can be divided into two main parts:

- **Linear SVM:** Linear SVMs use a linear decision boundary to separate the data points of different classes. When the data can be precisely linearly separated, linear SVMs are very suitable. This means that a single straight line (in 2D) or a hyperplane (in higher dimensions) can entirely divide the data points into their respective classes. A hyperplane that maximizes the margin between the classes is the decision boundary.
- **Non-Linear SVM:** Non-Linear SVM can be used to classify data when it cannot be separated into two classes by a straight line (in the case of 2D). By using kernel functions, nonlinear SVMs can handle nonlinearly separable data. The original input data is transformed by these kernel functions into a higher-dimensional feature space, where the data points can be linearly separated. A linear SVM is used to locate a nonlinear decision boundary in this modified space.

Advantages of Support Vector Machine (SVM)

1. **High-Dimensional Performance**: SVM excels in high-dimensional spaces, making it suitable for **image classification** and **gene expression analysis**.
2. **Nonlinear Capability**: Utilizing **kernel functions** like **RBF** and **polynomial**, SVM effectively handles **nonlinear relationships**.
3. **Outlier Resilience**: The **soft margin** feature allows SVM to ignore outliers, enhancing robustness in **spam detection** and **anomaly detection**.
4. **Binary and Multiclass Support**: SVM is effective for both **binary classification** and **multiclass classification**, suitable for applications in **text classification**.
5. **Memory Efficiency**: SVM focuses on **support vectors**, making it memory efficient compared to other algorithms.

Disadvantages of Support Vector Machine (SVM)

1. **Slow Training**: SVM can be slow for large datasets, affecting performance in **SVM in data mining** tasks.

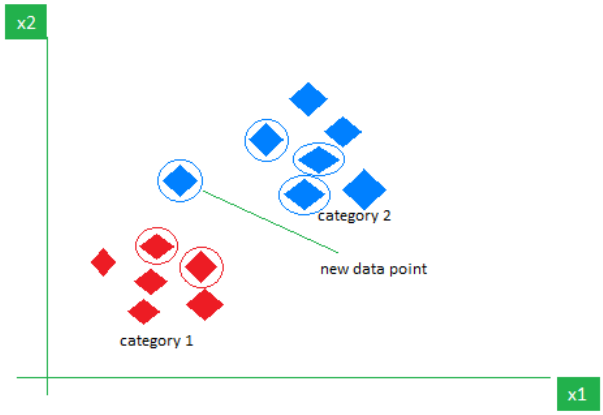
2. **Parameter Tuning Difficulty**: Selecting the right **kernel** and adjusting parameters like **C** requires careful tuning, impacting **SVM algorithms**.
3. **Noise Sensitivity**: SVM struggles with noisy datasets and overlapping classes, limiting effectiveness in real-world scenarios.
4. **Limited Interpretability**: The complexity of the **hyperplane** in higher dimensions makes SVM less interpretable than other models.
5. **Feature Scaling Sensitivity**: Proper **feature scaling** is essential; otherwise, SVM models may perform poorly.

2.3.3 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm generally used for classification but can also be used for regression tasks. It works by finding the "k" closest data points (neighbors) to a given input and makesa predictions based on the majority class (for classification) or the average value (for regression). Since KNN makes no assumptions about the underlying data distribution it makes it a non-parametric and instance-based learning method.

For example

consider the following table of data points containing two features:



KNN Algorithm working visualization

The new point is classified as Category 2 because most of its closest neighbors are blue squares. KNN assigns the category based on the majority of nearby points. The image shows how KNN predicts the category of a new data point based on its closest neighbours.

- The red diamonds represent Category 1 and the blue squares represent Category 2.
- The new data point checks its closest neighbors (circled points).
- Since the majority of its closest neighbors are blue squares (Category 2) KNN predicts the new data point belongs to Category 2.

KNN works by using proximity and majority voting to make predictions.

What is 'K' in K Nearest Neighbor?

In the k-Nearest Neighbors algorithm k is just a number that tells the algorithm how many nearby points or neighbors to look at when it makes a decision.

Example: Imagine you're deciding which fruit it is based on its shape and size. You compare it to fruits you already know.

- If k = 3, the algorithm looks at the 3 closest fruits to the new one.

- If 2 of those 3 fruits are apples and 1 is a banana, the algorithm says the new fruit is an apple because most of its neighbors are apples.

How to choose the value of k for KNN Algorithm?

- The value of k in KNN decides how many neighbors the algorithm looks at when making a prediction.
- Choosing the right k is important for good results.
- If the data has lots of noise or outliers, using a larger k can make the predictions more stable.
- But if k is too large the model may become too simple and miss important patterns and this is called underfitting.
- So k should be picked carefully based on the data.

Statistical Methods for Selecting k

- **Cross-Validation**: Cross-Validation is a good way to find the best value of k is by using k-fold cross-validation. This means dividing the dataset into k parts. The model is trained on some of these parts and tested on the remaining ones. This process is repeated for each part. The k value that gives the highest average accuracy during these tests is usually the best one to use.
- **Elbow Method**: In Elbow Method we draw a graph showing the error rate or accuracy for different k values. As k increases the error usually drops at first. But after a certain point error stops decreasing quickly. The point where the curve changes direction and looks like an "elbow" is usually the best choice for k.
- **Odd Values for k**: It's a good idea to use an odd number for k especially in classification problems. This helps avoid ties when deciding which class is the most common among the neighbors.

Distance Metrics Used in KNN Algorithm

KNN uses distance metrics to identify nearest neighbor, these neighbors are used for classification and regression task. To identify nearest neighbor, we use below distance metrics:

1. Euclidean Distance

Euclidean distance is defined as the straight-line distance between two points in a plane or space. You can think of it like the shortest path you would walk if you were to go directly from one point to another.

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^{d} (x_j - X_{i_j})^2]}$$

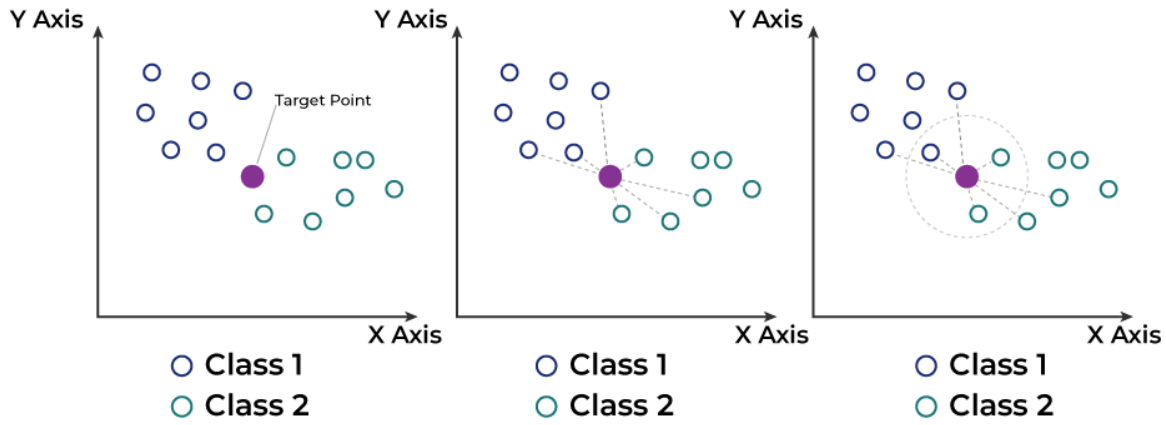
2. Manhattan Distance

This is the total distance you would travel if you could only move along horizontal and vertical lines like a grid or city streets. It's also called "taxicab distance" because a taxi can only drive along the grid-like streets of a city.

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Working of KNN algorithm

The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Step 1: Selecting the optimal value of K

- K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

- To measure the similarity between target and training data points Euclidean distance is used. Distance is calculated between data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

- The k data points with the smallest distances to the target point are nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

- When you want to classify a data point into a category like spam or not spam, the KNN algorithm looks at the K closest points in the dataset. These closest points are called neighbors. The algorithm then looks at which category the neighbors belong to and picks the one that appears the most. This is called majority voting.
- In regression, the algorithm still looks for the K closest points. But instead of voting for a class in classification, it takes the average of the values of those K neighbors. This average is the predicted value for the new point for the algorithm.

It shows how a test point is classified based on its nearest neighbors. As the test point moves the algorithm identifies the closest 'k' data points i.e., 5 in this case and assigns test point the majority class label that is grey label class here.

Applications of KNN

- **Recommendation Systems**: Suggests items like movies or products by finding users with similar preferences.
- **Spam Detection**: Identifies spam emails by comparing new emails to known spam and non-spam examples.
- **Customer Segmentation**: Groups customers by comparing their shopping behavior to others.
- **Speech Recognition**: Matches spoken words to known patterns to convert them into text.

Advantages of KNN

- **Simple to use**: Easy to understand and implement.
- **No training step**: No need to train as it just stores the data and uses it during prediction.

- **Few parameters**: Only needs to set the number of neighbors (k) and a distance method.
- **Versatile**: Works for both classification and regression problems.

Disadvantages of KNN

- **Slow with large data**: Needs to compare every point during prediction.
- **Struggles with many features**: Accuracy drops when data has too many features.
- **Can Overfit**: It can overfit especially when the data is high-dimensional or not clean.

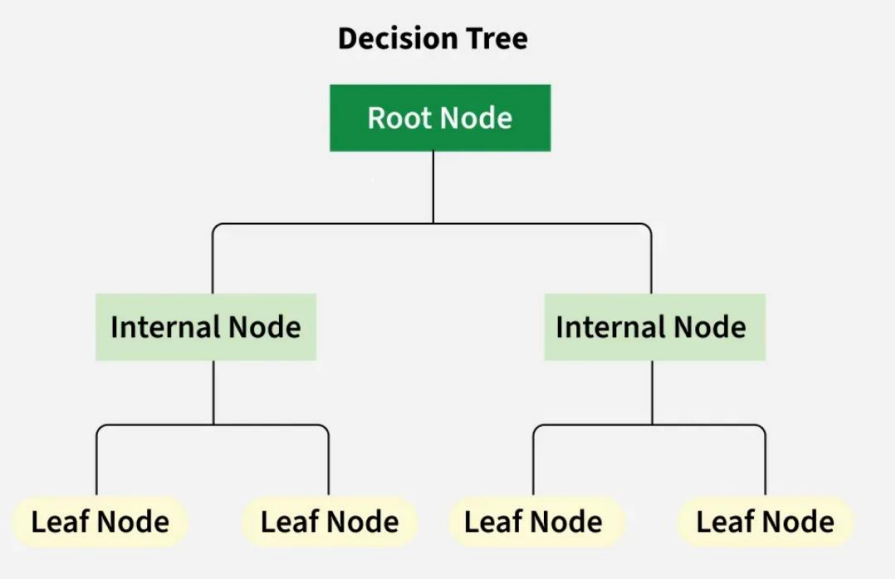
2.3.4. Decision Trees

Decision tree is a simple diagram that shows different choices and their possible results helping you make decisions easily. This article is all about what decision trees are, how they work, their advantages and disadvantages and their applications.

Understanding Decision Tree

A decision tree is a graphical representation of different options for solving a problem and show how different factors are related. It has a hierarchical tree structure starts with one main question at the top called a node which further branches out into different possible outcomes where:

- **Root Node** is the starting point that represents the entire dataset.
- **Branches**: These are the lines that connect nodes. It shows the flow from one decision to another.
- **Internal Nodes** are Points where decisions are made based on the input features.
- **Leaf Nodes**: These are the terminal nodes at the end of branches that represent final outcomes or predictions



Classification of Decision Tree

We have mainly two types of decision tree based on the nature of the target variable: **classification trees** and **regression trees**.

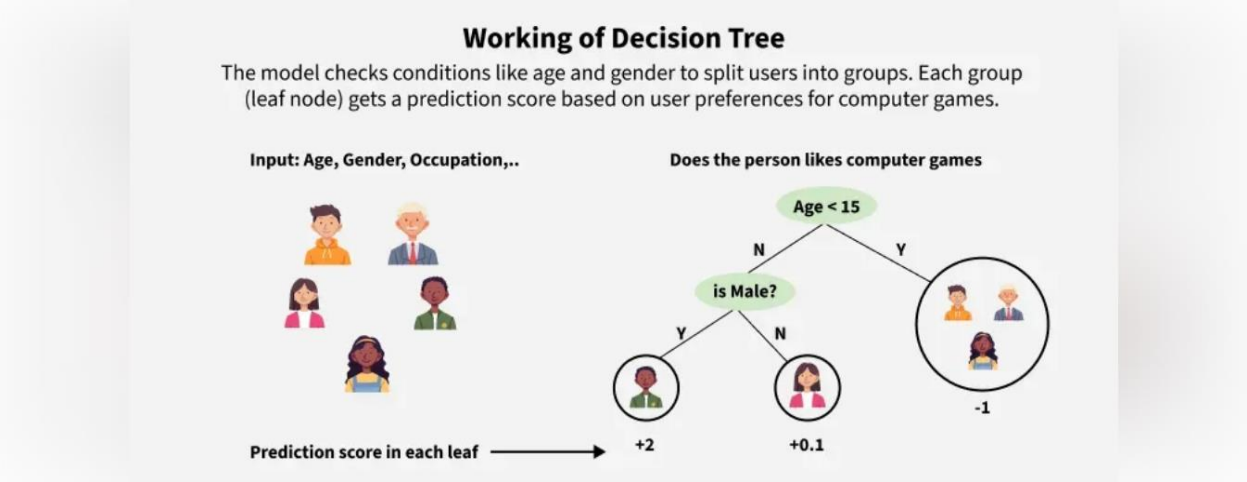
- **Classification trees:** They are designed to predict categorical outcomes means they classify data into different classes. They can determine whether an email is "spam" or "not spam" based on various features of the email.
- **Regression trees**: These are used when the target variable is continuous It predict numerical values rather than categories. For example a regression tree can estimate the price of a house based on its size, location, and other features.

How Decision Trees Work?

A decision tree working starts with a main question known as the **root node.** This question is derived from the features of the dataset and serves as the starting point for decision-making.

From the root node, the tree asks a series of yes/no questions. Each question is designed to split the data into subsets based on specific attributes. For example if the first question is "Is it raining?", the answer will determine which branch of the tree to follow. Depending on the response to each question you follow different branches. If your answer is "Yes," you might proceed down one path if "No," you will take another path.

This branching continues through a sequence of decisions. As you follow each branch, you get more questions that break the data into smaller groups. This step-by-step process continues until you have no more helpful questions.



You reach at the end of a branch where you find the final outcome or decision. It could be a classification (like "spam" or "not spam") or a prediction (such as estimated price).

Example:

Let's consider a **decision tree** for predicting whether a customer will buy a product based on **age**, **income** and **previous purchases**: Here's how the decision tree works:

1. Root Node (Income)

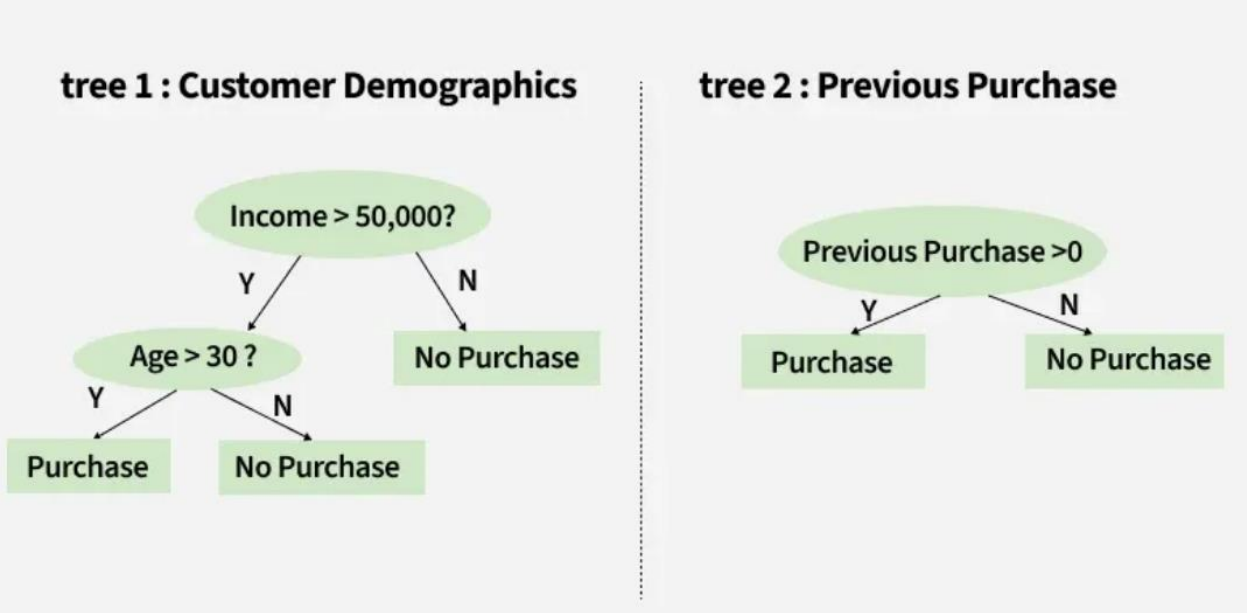
- **First Question**: **"Is the person's income greater than $50,000?"**
- **If Yes**, proceed to the next question.
- **If No**, predict "No Purchase" (leaf node).

2. Internal Node (Age):

- **If the person's income is greater than $50,000**, ask: **"Is the person's age above 30?"**
- **If Yes**, proceed to the next question.
- **If No**, predict "No Purchase" (leaf node).

3. Internal Node (Previous Purchases):

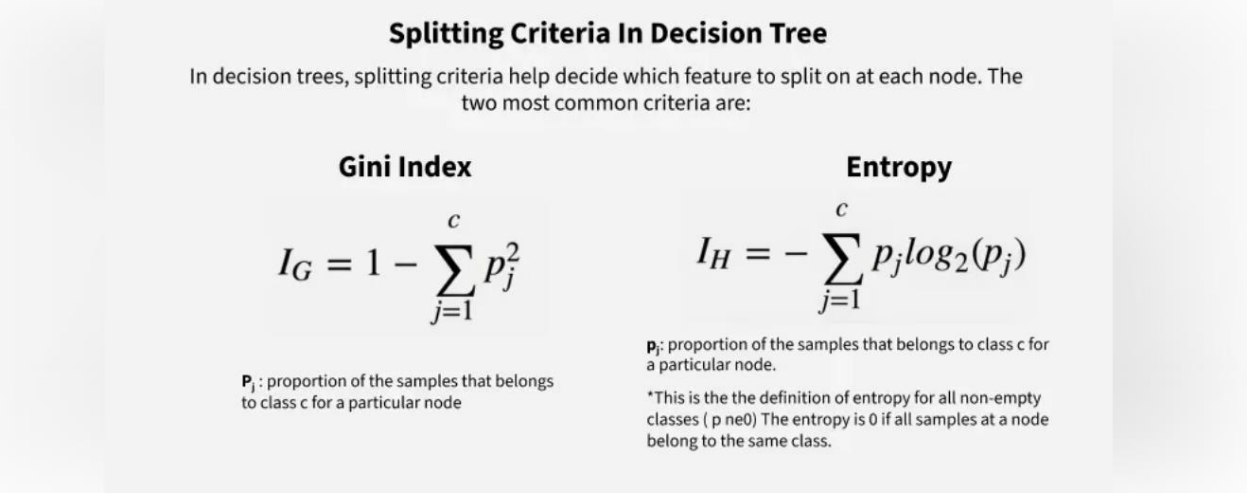
- If the person is above 30 and has made previous purchases, predict "Purchase" (leaf node).
- If the person is above 30 and has not made previous purchases, predict "No Purchase" (leaf node).



Information Gain and Gini Index in Decision Tree

Till now we have discovered the basic intuition and approach of how decision tree works, so let's just move to the attribute selection measure of decision tree. We have two popular attribute selection measures used:

1. **Information Gain**
2. **Gini Index**



1. Information Gain

Information Gain tells us how useful a question (or feature) is for splitting data into groups. It measures how much the uncertainty decreases after the split. A good question will create clearer groups and the feature with the highest Information Gain is chosen to make the decision.

For example, if we split a dataset of people into "**Young**" and "**Old**" based on age, and all young people bought the product while all old people did not, the Information Gain would be high because the split perfectly separates the two groups with no uncertainty left.

- Suppose S is a set of instances $A$is an attribute, $Sv$is the subset of S, $v$represents an individual value that the attribute Acan take and Values (A) is the set of all possible values of A then

$$Gain(S, A) = Entropy(S) - \sum_v^A \frac{|S_v|}{|S|}.Entropy(S_v)$$

  - **Entropy:** is the measure of uncertainty of a random variable it characterizes the impurity of an arbitrary collection of examples. The higher the entropy more the information content.

For example, if a dataset has an equal number of "**Yes**" and "**No**" outcomes (like 3 people who bought a product and 3 who didn't), the entropy is high because it's uncertain which outcome to predict. But if all the outcomes are the same (all "Yes" or all "No"), the entropy is 0 meaning there is no uncertainty left in predicting the outcome.

Suppose $S$is a set of instances, $A$is an attribute, $Sv$is the subset of $S$with A= v, and Values (A) is the set of all possible values of A, then

$$Gain(S, A) = Entropy(S) - \sum_{v \epsilon Values(A)} \frac{|S_v|}{|S|}.Entropy(S_v)$$

Example:

For the set X = {a,a,a,b,b,b,b,b}
Total instances: 8
Instances of b: 5
Instances of a: 3

$$\begin{aligned} \text{Entropy } H(X) &= \left[\left(\frac{3}{8}\right)\log_2\frac{3}{8} + \left(\frac{5}{8}\right)\log_2\frac{5}{8}\right] \\ &= -[0.375(-1.415) + 0.625(-0.678)] \\ &= -(-0.53 - 0.424) \\ &= 0.954 \end{aligned}$$

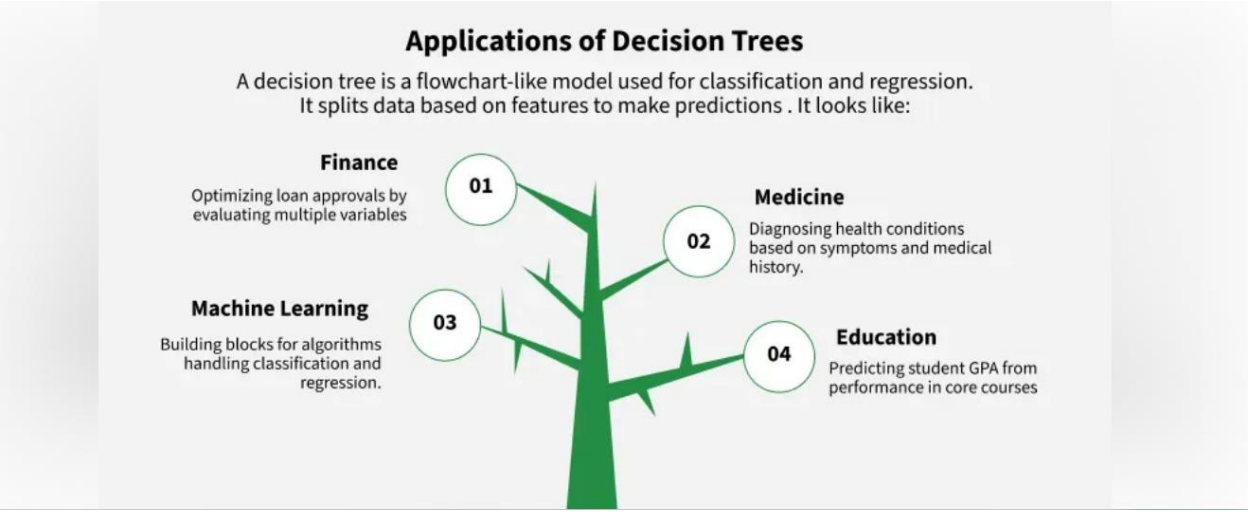
Advantages of Decision Trees

- **Simplicity and Interpretability**: Decision trees are straightforward and easy to understand. You can visualize them like a flowchart which makes it simple to see how decisions are made.
- **Versatility**: It means they can be used for different types of tasks can work well for both **classification** and **regression**
- **No Need for Feature Scaling**: They don't require you to normalize or scale your data.
- **Handles Non-linear Relationships**: It is capable of capturing non-linear relationships between features and target variables.

Disadvantages of Decision Trees

- **Overfitting**: Overfitting occurs when a decision tree captures noise and details in the training data and it perform poorly on new data.
- **Instability**: instability means that the model can be unreliable slight variations in input can lead to significant differences in predictions.
- **Bias towards Features with More Levels**: Decision trees can become biased towards features with many categories focusing too much on them during decision-making. This can cause the model to miss out other important features led to less accurate predictions.

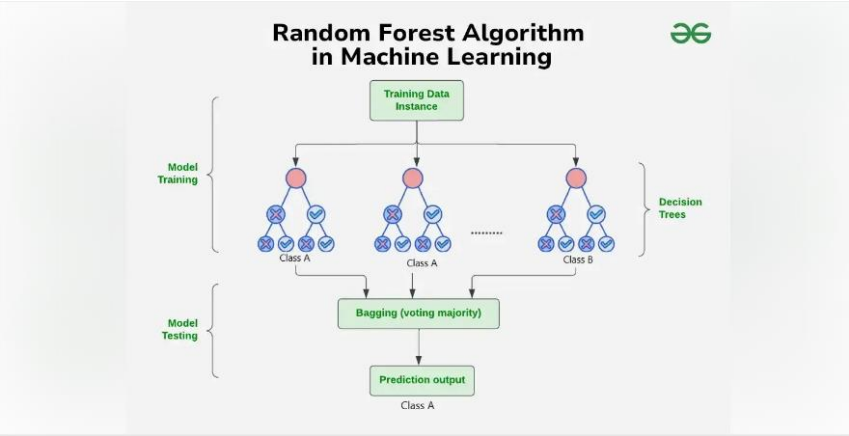
Applications of Decision Trees



- **Loan Approval in Banking**: A bank needs to decide whether to approve a loan application based on customer profiles.
    - Input features include income, credit score, employment status, and loan history.
    - The decision tree predicts loan approval or rejection, helping the bank make quick and reliable decisions.
- **Medical Diagnosis:** A healthcare provider wants to predict whether a patient has diabetes based on clinical test results.
    - Features like glucose levels, BMI, and blood pressure are used to make a decision tree.
    - Tree classifies patients into diabetic or non-diabetic, assisting doctors in diagnosis.
- **Predicting Exam Results in Education: S**chool wants to predict whether a student will pass or fail based on study habits.
    - Data includes attendance, time spent studying, and previous grades.
    - The decision tree identifies at-risk students, allowing teachers to provide additional support.

Random Forest Algorithm

Random Forest is a machine learning algorithm that uses many decision trees to make better predictions. Each tree looks at different random parts of the data and their results are combined by voting for classification or averaging for regression. This helps in improving accuracy and reducing errors.



Working of Random Forest Algorithm

- **Create Many Decision Trees:** The algorithm makes many decision trees each using a random part of the data. So every tree is a bit different.
- **Pick Random Features:** When building each tree it doesn't look at all the features (columns) at once. It picks a few at random to decide how to split the data. This helps the trees stay different from each other.
- **Each Tree Makes a Prediction:** Every tree gives its own answer or prediction based on what it learned from its part of the data.
- **Combine the Predictions:**
    - For **classification** we choose a category as the final answer is the one that most trees agree on i.e majority voting.
    - For **regression** we predict a number as the final answer is the average of all the trees predictions.
- **Why It Works Well:** Using random data and features for each tree helps avoid overfitting and makes the overall prediction more accurate and trustworthy.

Key Features of Random Forest

- **Handles Missing Data:** It can work even if some data is missing so you don't always need to fill in the gaps yourself.
- **Shows Feature Importance:** It tells you which features (columns) are most useful for making predictions which helps you understand your data better.
- **Works Well with Big and Complex Data:** It can handle large datasets with many features without slowing down or losing accuracy.
- **Used for Different Tasks:** You can use it for both **classification** like predicting types or labels and **regression** like predicting numbers or amounts.

Assumptions of Random Forest

- **Each tree makes its own decisions**: Every tree in the forest makes its own predictions without relying on others.
- **Random parts of the data are used**: Each tree is built using random samples and features to reduce mistakes.

- **Enough data is needed**: Sufficient data ensures the trees are different and learn unique patterns and variety.
- **Different predictions improve accuracy**: Combining the predictions from different trees leads to a more accurate final result.

Advantages of Random Forest

- Random Forest provides very accurate predictions even with large datasets.
- Random Forest can handle missing data well without compromising with accuracy.
- It doesn't require normalization or standardization on dataset.
- When we combine multiple decision trees it reduces the risk of overfitting of the model.

Limitations of Random Forest

- It can be computationally expensive especially with a large number of trees.
- It's harder to interpret the model compared to simpler models like decision trees.

Bagging and **Boosting**

As we know, Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. Basic idea is to learn a set of classifiers (experts) and to allow them to vote. **Bagging** and **Boosting** are two types of **Ensemble Learning**. These two decrease the variance of a single estimate as they combine several estimates from different models. So, the result may be a model with higher stability. Let's understand these two terms in a glimpse.

1. **Bagging**: It is a homogeneous weak learners' model that learns from each other independently in parallel and combines them for determining the model average.
2. **Boosting**: It is also a homogeneous weak learners' model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

Let's look at both of them in detail and understand the Difference between Bagging and Boosting.

Bagging

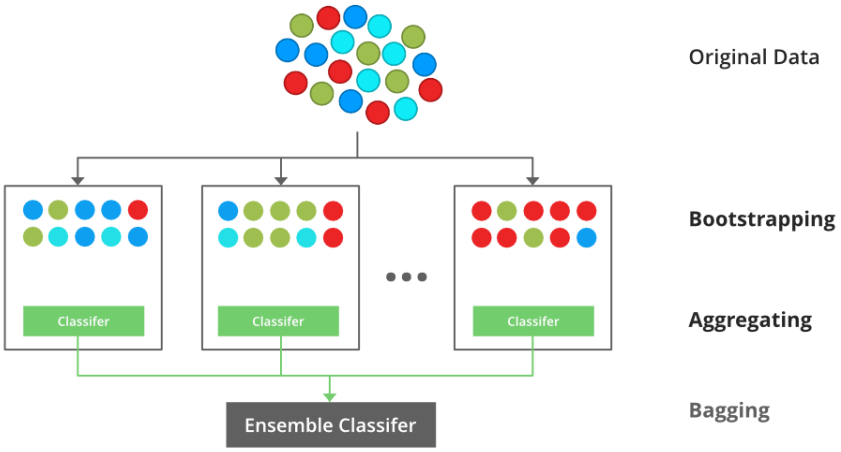
Bootstrap Aggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It decreases the variance and helps to avoid overfitting. It is usually applied to decision tree methods. Bagging is a special case of the model averaging approach.

Description of the Technique

Suppose a set D of d tuples, at each iteration i, a training set Di of d tuples is selected via row sampling with a replacement method (i.e., there can be repetitive elements from different d tuples) from D (i.e., bootstrap). Then a classifier model Mi is learned for each training set D<i. Each classifier Mi returns its class prediction. The bagged classifier M* counts the votes and assigns the class with the most votes to X (unknown sample).

Implementation Steps of Bagging

- **Step 1:** Multiple subsets are created from the original data set with equal tuples, selecting observations with replacement.
- **Step 2:** A base model is created on each of these subsets.
- **Step 3:** Each model is learned in parallel with each training set and independent of each other.
- **Step 4:** The final predictions are determined by combining the predictions from all the models.



Example of Bagging

The Random Forest model uses Bagging, where decision tree models with higher variance are present. It makes random feature selection to grow trees. Several random trees make a Random Forest.

Boosting

Boosting is an ensemble modeling technique designed to create a strong classifier by combining multiple weak classifiers. The process involves building models sequentially, where each new model aims to correct the errors made by the previous ones.

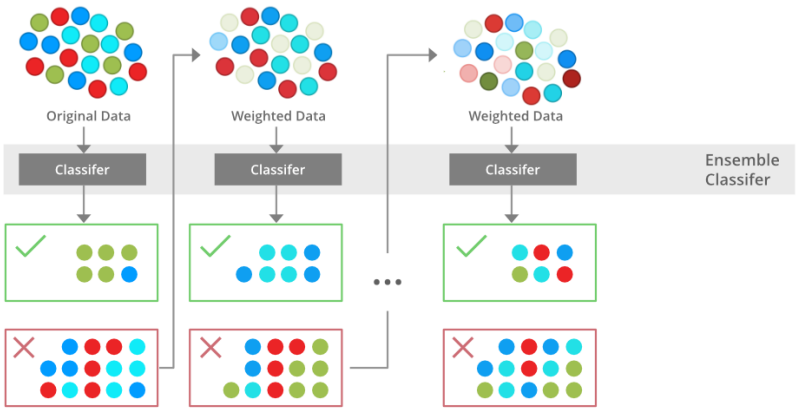
- Initially, a model is built using the training data.
- Subsequent models are then trained to address the mistakes of their predecessors.
- boosting assigns weights to the data points in the original dataset.
- Higher weights: Instances that were misclassified by the previous model receive higher weights.
- Lower weights: Instances that were correctly classified receive lower weights.
- Training on weighted data: The subsequent model learns from the weighted dataset, focusing its attention on harder-to-learn examples (those with higher weights).
- This iterative process continues until:
    - The entire training dataset is accurately predicted, or
    - A predefined maximum number of models is reached.

Boosting Algorithms

There are several boosting algorithms. The original ones, proposed by **Robert Schapire** and **Yoav Freund** were not adaptive and could not take full advantage of the weak learners. Schapire and Freund then developed AdaBoost, an adaptive boosting algorithm that won the prestigious Gödel Prize. AdaBoost was the first really successful boosting algorithm developed for the purpose of binary classification. AdaBoost is short for Adaptive Boosting and is a very popular boosting technique that combines multiple “weak classifiers” into a single “strong classifier”.

Algorithm:

1. *Initialize the dataset and assign equal weight to each of the data point.*
2. *Provide this as input to the model and identify the wrongly classified data points.*
3. *Increase the weight of the wrongly classified data points and decrease the weights of correctly classified data points. And then normalize the weights of all data points.*
4. *if (got required results)*
   Goto step 5
   else
   Goto step 2
5. *End*



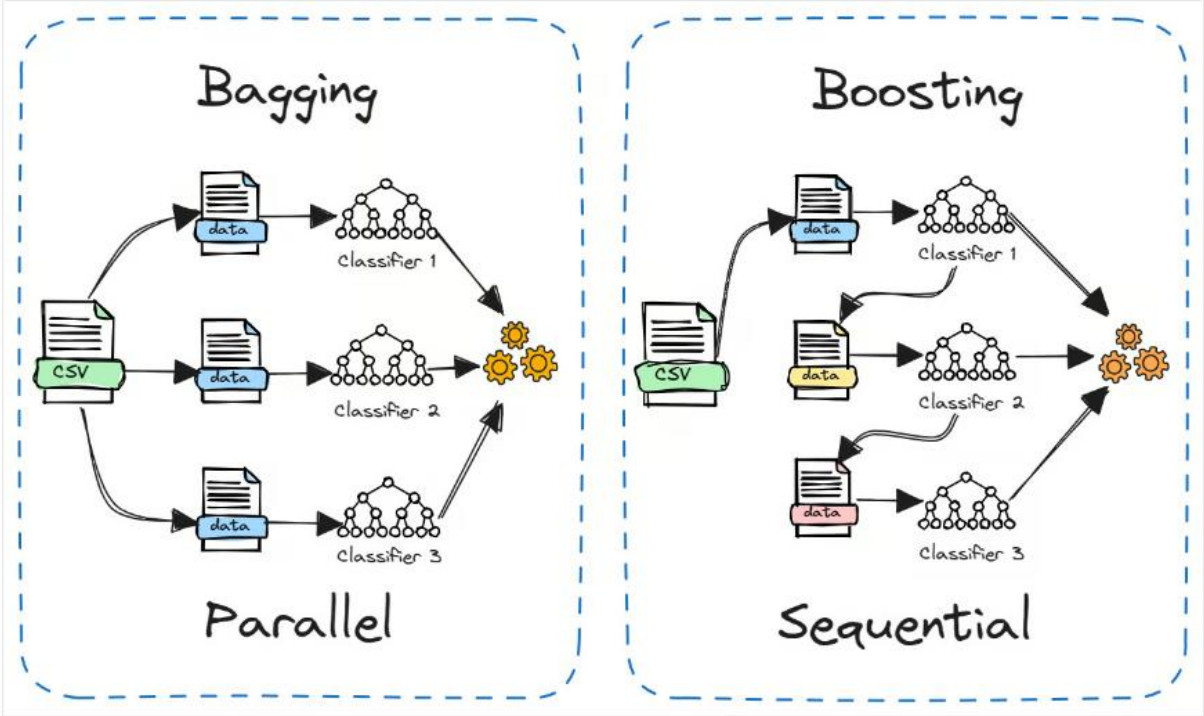
Similarities Between Bagging and Boosting

Bagging and boosting, both being the commonly used methods, have a universal similarity of being classified as ensemble methods. Here we will explain the similarities between them.

1. Both are ensemble methods to get N learners from 1 learner.
2. Both generate several training data sets by random sampling.
3. In Bagging, the final decision is made by averaging predictions or majority voting. In Boosting, new models focus on correcting previous errors, and the final decision is made by a weighted majority vote.
4. Both are good at reducing variance and provide higher stability.

Differences Between Bagging and Boosting

Bagging	Boosting
The simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
Each model receives equal weight.	Models are weighted according to their performance.
Each model is built independently.	New models are influenced by the performance of previously built models.
Different training data subsets are selected using row sampling with replacement and random sampling methods from the entire training dataset.	Iteratively train models, with each new model focusing on correcting the errors (misclassifications or high residuals) of the previous models
Bagging tries to solve the over-fitting problem.	Boosting tries to reduce bias.
If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.
In these base classifiers are trained parallelly.	In these base classifiers are trained sequentially.
Example: The Random Forest model uses Bagging.	Example: The AdaBoost uses Boosting techniques


Bagging
data
Classifier 1
CSV
data
Classifier 2
data
Classifier 3
Parallel
Boosting
data
Classifier 1
CSV
data
Classifier 2
data
Classifier 3
Sequential

Numerical Problems:

Linear Regression Numerical:

SUBJECT	AGE X	GLUCOSE LEVEL Y
1	43	99
2	21	65
3	25	79
4	42	75
5	57	87
6	59	81
7	55	?

Regression analysis is used to:

- Predict the value of a dependent variable based on the value of at least one independent variable.
- Explain the impact of changes in an independent variable on the dependent variable.

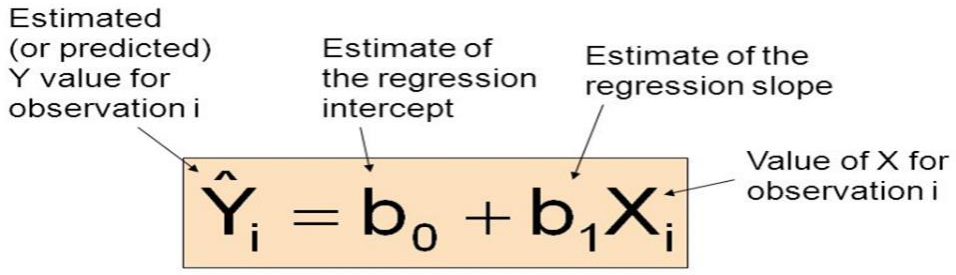
The key steps for regression are simple:

1. List all the variables available for making the model.
2. Establish a dependent variable of interest.
3. Examine visual (if possible) relationships between variables of interest.
4. Find a way to predict the dependent variables using the other variables.

The regression model is described as a linear equation that follows.

- *y* is the dependent variable, that is, the variable being predicted.
- *x* is the independent variable or the predictor variable.

There could be many predictor variables (such as *x*1, *x*2, . . .) in a regression equation.



Here we need to find the value of b_0, b_1 using the following equation.

$$b_0 = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n(\sum x^2) - (\sum x)^2}$$

$$b_1 = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$

Step 1: *Make a chart of your data, filling in the columns in the same way as you would fill in the chart if you were finding the Pearson's Correlation Coefficient*

SUBJECT	AGE X	GLUCOSE LEVEL Y	XY	X^2	Y^2
1	43	99	4257	1849	9801
2	21	65	1365	441	4225
3	25	79	1975	625	6241
4	42	75	3150	1764	5625
5	57	87	4959	3249	7569
6	59	81	4779	3481	6561
Σ	**247**	**486**	**20485**	**11409**	**40022**

Step 2: Use the following equations to find b_0 and b_1.

Find b_0:

$$b_0 = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n(\sum x^2) - (\sum x)^2}$$

$$b_0 = \frac{(486)(11409) - (247)(20485)}{6\,(11409) - (247)^2}$$

$$b_0 = \frac{4848979}{7445} = 65.14$$

Find b_1:

$$b_1 = \frac{n\,(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$

$$b_1 = \frac{6\,(20485) - (247)(486)}{6(11409) - (247)^2}$$

$$b_1 = \frac{2868}{7445} = 0.385335$$

Step 3: *Insert the values into the equation.*

$y' = b_o + b_1 * x$

$y' = 65.14 + (0.385225 * x)$

Step 4: *Prediction – the value of y for the given value of x = 55*

$y' = 65.14 + (0.385225 * 55)$

$y' = 86.327$

Summary:
In this tutorial, we understood, how to use a Linear regression with One Independent Variable to predict the glucose level given the age

Multiple Linear Regression

Suppose we have the following dataset with one response variable *y* and two predictor variables X_1 and X_2:

y	X_1	X_2
140	60	22
155	62	25
159	67	24
179	70	20
192	71	15
200	72	14
212	75	14
215	78	11

Use the following steps to fit a multiple linear regression model to this dataset.

Step 1: Calculate X_1^2, X_2^2, X_1y, X_2y and X_1X_2.

	y	X_1	X_2
	140	60	22
	155	62	25
	159	67	24
	179	70	20
	192	71	15
	200	72	14
	212	75	14
	215	78	11
Mean	181.5	69.375	18.125
Sum	1452	555	145

X_1^2	X_2^2	X_1y	X_2y	X_1X_2
3600	484	8400	3080	1320
3844	625	9610	3875	1550
4489	576	10653	3816	1608
4900	400	12530	3580	1400
5041	225	13632	2880	1065
5184	196	14400	2800	1008
5625	196	15900	2968	1050
6084	121	16770	2365	858
Sum 38767	2823	101895	25364	9859

Step 2: Calculate Regression Sums.

Next, make the following regression sum calculations:

$\Sigma x_1^2 = \Sigma X_1^2 - (\Sigma X_1)^2 / n = 38{,}767 - (555)^2 / 8 =$ **263.875**
$\Sigma x_2^2 = \Sigma X_2^2 - (\Sigma X_2)^2 / n = 2{,}823 - (145)^2 / 8 =$ **194.875**
$\Sigma x_1 y = \Sigma X_1 y - (\Sigma X_1 \Sigma y) / n = 101{,}895 - (555*1{,}452) / 8 =$ **1,162.5**
$\Sigma x_2 y = \Sigma X_2 y - (\Sigma X_2 \Sigma y) / n = 25{,}364 - (145*1{,}452) / 8 =$ **-953.5**
$\Sigma x_1 x_2 = \Sigma X_1 X_2 - (\Sigma X_1 \Sigma X_2) / n = 9{,}859 - (555*145) / 8 =$ **-200.375**

	y	X_1	X_2		X_1^2	X_2^2	X_1y	X_2y	X_1X_2
	140	60	22		3600	484	8400	3080	1320
	155	62	25		3844	625	9610	3875	1550
	159	67	24		4489	576	10653	3816	1608
	179	70	20		4900	400	12530	3580	1400
	192	71	15		5041	225	13632	2880	1065
	200	72	14		5184	196	14400	2800	1008
	212	75	14		5625	196	15900	2968	1050
	215	78	11		6084	121	16770	2365	858
Mean	181.5	69.375	18.125	**Sum**	38767	2823	101895	25364	9859
Sum	1452	555	145						
				Reg Sums	263.875	194.875	1162.5	-953.5	-200.375

Step 3: Calculate b_0, b_1, and b_2.

The formula to calculate b_1 is:

$$[(\Sigma x_2^2)(\Sigma x_1 y) - (\Sigma x_1 x_2)(\Sigma x_2 y)] / [(\Sigma x_1^2)(\Sigma x_2^2) - (\Sigma x_1 x_2)^2]$$

Thus,

$\mathbf{b_1} = [(194.875)(1162.5) - (-200.375)(-953.5)] / [(263.875)(194.875) - (-200.375)^2] = \mathbf{3.148}$

The formula to calculate b_2 is:

$$[(\Sigma x_1^2)(\Sigma x_2 y) - (\Sigma x_1 x_2)(\Sigma x_1 y)] / [(\Sigma x_1^2)(\Sigma x_2^2) - (\Sigma x_1 x_2)^2]$$

Thus,

$\mathbf{b_2} = [(263.875)(-953.5) - (-200.375)(1152.5)] / [(263.875)(194.875) - (-200.375)^2]$

$= \mathbf{-1.656}$

The formula to calculate b_0 is:

$$\mathbf{y - b_1X_1 - b_2X_2}$$

Thus,

$$\mathbf{b_0} = 181.5 - 3.148(69.375) - (-1.656)(18.125) = \mathbf{-6.867}$$

Step 4: Place b_0, b_1, and b_2 in the estimated linear regression equation.

The estimated linear regression equation is:

$$\hat{y} = b_0 + b_1 * x_1 + b_2 * x_2$$

In our example, it is

$$\hat{y} = -6.867 + 3.148x_1 - 1.656x_2$$

How to Interpret a Multiple Linear Regression Equation

Here is how to interpret this estimated linear regression equation:

$\hat{y} = -6.867 + 3.148x_1 - 1.656x_2$

$b_0 = -6.867$

When both predictor variables are equal to zero, the mean value for y is -6.867.

$b_1 = 3.148$

A one unit increase in x_1 is associated with a 3.148 unit increase in y, on average, assuming x_2 is held constant.

$b_2 = -1.656$

A one unit increase in x_2 is associated with a 1.656 unit decrease in y, on average, assuming x_1 is held constant.

Example

Y	X1	X2
240	25	24
236	31	21
290	45	24
274	60	25
316	72	26
300	80	25
296	84	25
267	75	24
276	60	25
288	50	25
261	38	23

Ans : Y=9.27+0.49·X1+9.88·X2

y	x_1	x_2	x_3
85.1	8.5	5.1	4.7
106.3	12.9	5.8	8.8
50.2	5.2	2.1	15.1
130.6	10.7	8.4	12.2
54.8	3.1	2.9	10.6
30.3	3.5	1.2	3.5
79.4	9.2	3.7	9.7
91.0	9.0	7.6	5.9
135.4	15.1	7.7	20.8
89.3	10.2	4.5	7.9

Predicting House Prices using Linear Regression

Dataset: Suppose we have data on house sizes (in square feet) and their prices (in thousands of Rs):

House Size (x (sq ft))	Price (y (Rs1000))
1000	300
1500	400
2000	500
2500	600
3000	700

Predict the price for a house of 2200 sq ft.

KNN Problem

You have the following training data:

Point	X1	X2	Class
A	2	4	0
B	4	2	0
C	4	4	1
D	6	2	1

Now, classify a **new point**:
X = (5, 3) using **K = 3** and **Euclidean Distance**.

Step 1: Calculate Euclidean Distance

Formula:

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

Compute distances from point X = (5, 3) to all training points:

- Distance to A (2, 4):

$$\sqrt{(5-2)^2 + (3-4)^2} = \sqrt{9+1} = \sqrt{10} \approx 3.16$$

- Distance to B (4, 2):

$$\sqrt{(5-4)^2 + (3-2)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$$

- Distance to C (4, 4):

$$\sqrt{(5-4)^2 + (3-4)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$$

- Distance to D (6, 2):

$$\sqrt{(5-6)^2 + (3-2)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$$

Step 2: Sort by Distance

Point	Distance	Class
B	1.41	0
C	1.41	1
D	1.41	1
A	3.16	0

Step 3: Pick K = 3 Nearest Neighbors

B (Class 0), C (Class 1), D (Class 1)

Step 4: Majority Voting

Class 1: 2 votes (C and D)

Class 0: 1 vote (B)

Result: The new point X = (5, 3) is classified as Class 1

KNN Problem 2

You have the following training data:

Point	X1	X2	Class
P	1	2	0
Q	2	3	0
R	3	3	1
S	6	5	1
T	7	7	1

Now classify a **new point**:
X = (4, 4) using **K = 3** and **Euclidean Distance**.

Step 1: Calculate Euclidean Distances

- To P (1, 2):

$$\sqrt{(4-1)^2+(4-2)^2} = \sqrt{9+4} = \sqrt{13} \approx 3.61$$

- To Q (2, 3):

$$\sqrt{(4-2)^2+(4-3)^2} = \sqrt{4+1} = \sqrt{5} \approx 2.24$$

- To R (3, 3):

$$\sqrt{(4-3)^2+(4-3)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$$

- To S (6, 5):

$$\sqrt{(4-6)^2+(4-5)^2} = \sqrt{4+1} = \sqrt{5} \approx 2.24$$

- To T (7, 7):

$$\sqrt{(4-7)^2+(4-7)^2} = \sqrt{9+9} = \sqrt{18} \approx 4.24$$

Step 2: Sort by Distance

Point	Distance	Class
R	1.41	1
Q	2.24	0
S	2.24	1
P	3.61	0
T	4.24	1

Step 3: Pick K = 3 Nearest Neighbors

R (Class 1), Q (Class 0), S (Class 1)

Step 4: Majority Voting

Class 1: 2 votes (R and S)

Class 0: 1 vote (Q)

Result: The new point X = (4, 4) is classified as Class 1

Decision Tree

following dataset to classify whether a student will **Pass** (Yes/No) based on two attributes: **Study (Yes/No)** and **Sleep (Yes/No)**.

Student	Study	Sleep	Pass
1	Yes	Yes	Yes
2	Yes	No	Yes
3	No	Yes	No
4	No	No	No
5	Yes	Yes	Yes

Step 1: Calculate the Entropy of the Target (Pass)

There are 5 samples:

- Pass = Yes: 3
- Pass = No: 2

$$Entropy(S) = -\left(\frac{3}{5}\log_2\frac{3}{5} + \frac{2}{5}\log_2\frac{2}{5}\right) \approx -(0.6 \cdot -0.737 + 0.4 \cdot -1.322) \approx 0.971$$

Step 2: Calculate Information Gain for "Study"

Study	Count	Yes	No
Yes	3	3	0
No	2	0	2

- For **Study = Yes**, all 3 → Pass = Yes → Entropy = 0
- For **Study = No**, all 2 → Pass = No → Entropy = 0

$$IG(Study) = Entropy(S) - \left(\frac{3}{5}\cdot 0 + \frac{2}{5}\cdot 0\right) = 0.971 - 0 = 0.971$$

Step 3: Calculate Information Gain for "Sleep"

Sleep	Count	Yes	No
Yes	3	2	1
No	2	1	1

Entropy for Sleep = Yes:

$$-\left(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right) \approx 0.918$$

Entropy for Sleep = No:

$$-\left(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right) = 1.0$$

$$IG(Sleep) = 0.971 - \left(\frac{3}{5}\cdot 0.918 + \frac{2}{5}\cdot 1.0\right) = 0.971 - (0.5508 + 0.4) = 0.971 - 0.9508 = 0.0202$$

Step 4: Select the Best Attribute

- IG(Study) = **0.971**
- IG(Sleep) = 0.0202

Best attribute = Study

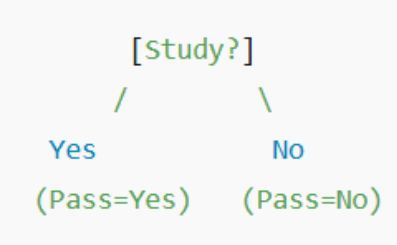
Step 5: Build the Tree

- **Root Node**: Study
  - **Yes** → Pass = Yes (Leaf)
  - **No**:
    - We now look at remaining attribute **Sleep** for 2 samples where Study = No:

Student	Sleep	Pass
3	Yes	No
4	No	No

    - Both outcomes → Pass = No → Entropy = 0

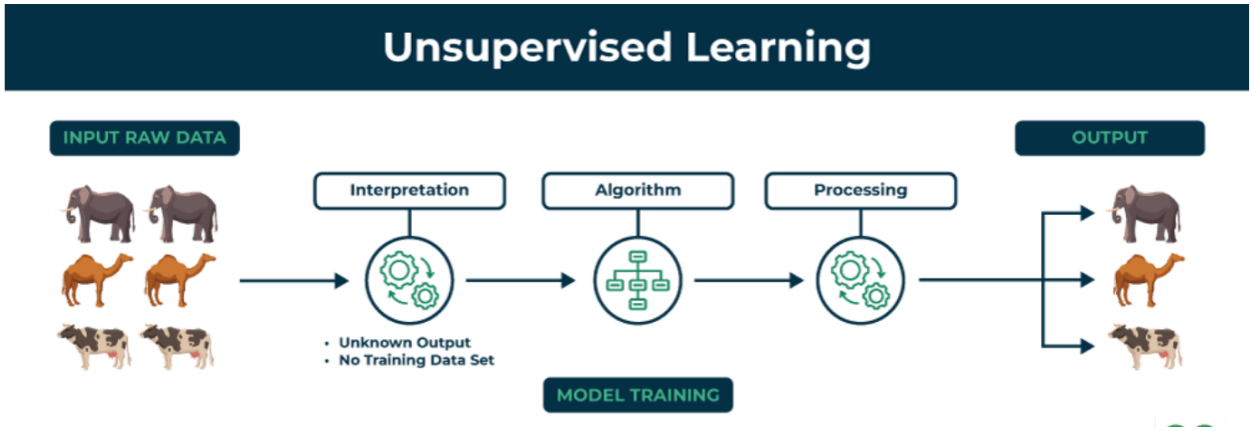
Step 6: Final Tree



Chapter 3

Unsupervised Learning

1.0 Basic Concept of Unsupervised Learning



Unsupervised learning is a branch of machine learning that deals with unlabeled data. Unlike supervised learning, where the data is labeled with a specific category or outcome, unsupervised learning algorithms are tasked with finding patterns and relationships within the data without any prior knowledge of the data's meaning.

Unsupervised machine learning algorithms find hidden patterns and data without any human intervention, i.e., we don't give output to our model. The training model has only input parameter values and discovers the groups or patterns on its own.

The image shows set of animals: elephants, camels, and cows that represents raw data that the unsupervised learning algorithm will process.

The **"Interpretation"** stage signifies that the algorithm doesn't have predefined labels or categories for the data. It needs to figure out how to group or organize the data based on inherent patterns.

Algorithm represents the core of unsupervised learning process using techniques like clustering, dimensionality reduction, or anomaly detection to identify patterns and structures in the data.

Processing stage shows the algorithm working on the data.

The input to the unsupervised learning models is as follows:

- Unstructured data: May contain noisy(meaningless) data, missing values, or unknown data

- Unlabeled data: Data only contains a value for input parameters, there is no targeted value(output). It is easy to collect as compared to the labeled one in the Supervised approach.

3.2. Types of Unsupervised Learning

There are mainly 3 types of Algorithms which are used for Unsupervised dataset.

- **Clustering**
- **Association Rule Learning**
- **Dimensionality Reduction**

1. Clustering Algorithms

Clustering in unsupervised machine learning is the process of grouping unlabeled data into clusters based on their similarities. The goal of clustering is to identify patterns and relationships in the data without any prior knowledge of the data's meaning.

Broadly this technique is applied to group data based on different patterns, such as similarities or differences, our machine model finds. These algorithms are used to process raw, unclassified data objects into groups. For example, in the above figure, we have not given output parameter values, so this technique will be used to group clients based on the input parameters provided by our data.

Some common clustering algorithms:

- ***K-means Clustering:*** *Groups data into K clusters based on how close the points are to each other.*
- ***Hierarchical Clustering:*** *Creates clusters by building a tree step-by-step, either merging or splitting groups.*
- ***Density-Based Clustering (DBSCAN):*** *Finds clusters in dense areas and treats scattered points as noise.*
- ***Mean-Shift Clustering:*** *Discovers clusters by moving points toward the most crowded areas.*
- ***Spectral Clustering:*** *Groups data by analyzing connections between points using graphs.*

2. Association Rule Learning

Association rule learning is also known as association rule mining is a common technique used to discover associations in unsupervised machine learning. This technique is a rule-based ML technique that finds out some very useful relations between parameters of a large data set. This technique is basically used for market basket analysis that helps to better understand the relationship between different products.

For e.g., shopping stores use algorithms based on this technique to find out the relationship between the sale of one product w.r.t to another's sales based on customer behavior. **Like if a customer buys milk, then he may also buy bread, eggs, or butter**. Once trained well, such models can be used to increase their sales by planning different offers.

Some common Association Rule Learning algorithms:

- ***Apriori Algorithm:*** *Finds patterns by exploring frequent item combinations step-by-step.*
- ***FP-Growth Algorithm:*** *An Efficient Alternative to Apriori. It quickly identifies frequent patterns without generating candidate sets.*
- ***Eclat Algorithm:*** *Uses intersections of itemsets to efficiently find frequent patterns.*
- ***Efficient Tree-based Algorithms:*** *Scales to handle large datasets by organizing data in tree structures.*

3. Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of features in a dataset while preserving as much information as possible. This technique is useful for improving the performance of machine learning algorithms and for data visualization.

Imagine a dataset of 100 features about students (height, weight, grades, etc.). To focus on key traits, you reduce it to just 2 features: height and grades, making it easier to visualize or analyze the data.

Here are some popular Dimensionality Reduction algorithms*:*

- ***Principal Component Analysis (PCA):*** *Reduces dimensions by transforming data into uncorrelated principal components.*
- ***Linear Discriminant Analysis (LDA):*** *Reduces dimensions while maximizing class separability for classification tasks.*

- ***Non-negative Matrix Factorization (NMF):*** *Breaks data into non-negative parts to simplify representation.*
- ***Locally Linear Embedding (LLE):*** *Reduces dimensions while preserving the relationships between nearby points.*
- ***Isomap:*** *Captures global data structure by preserving distances along a manifold.*

Challenges of Unsupervised Learning

Here are the key challenges of unsupervised learning:

- **Noisy Data**: Outliers and noise can distort patterns and reduce the effectiveness of algorithms.
- **Assumption Dependence**: Algorithms often rely on assumptions (e.g., cluster shapes), which may not match the actual data structure.
- **Overfitting Risk**: Overfitting can occur when models capture noise instead of meaningful patterns in the data.
- **Limited Guidance**: The absence of labels restricts the ability to guide the algorithm toward specific outcomes.
- **Cluster Interpretability**: Results, such as clusters, may lack clear meaning or alignment with real-world categories.
- **Sensitivity to Parameters**: Many algorithms require careful tuning of hyperparameters, such as the number of clusters in k-means.
- **Lack of Ground Truth**: Unsupervised learning lacks labeled data, making it difficult to evaluate the accuracy of results.

Applications of Unsupervised learning

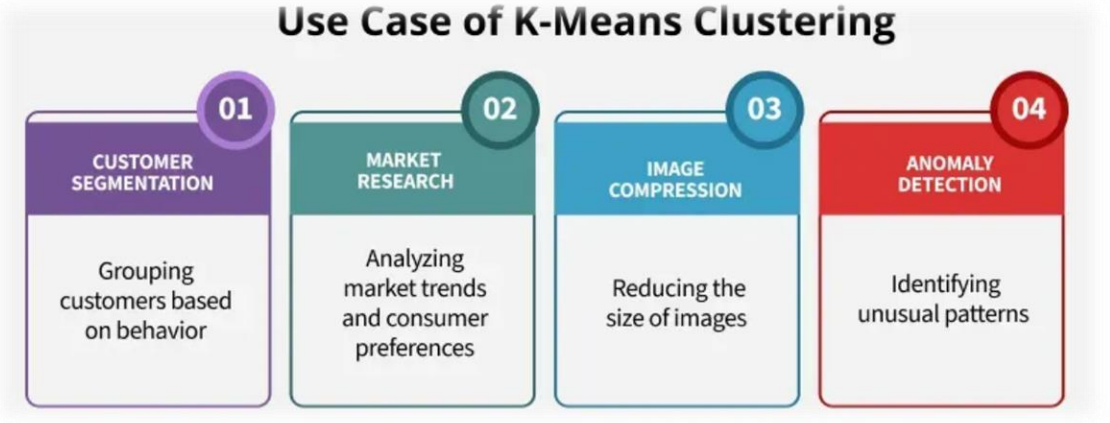
Unsupervised learning has diverse applications across industries and domains. Key applications include:

- **Customer Segmentation:** Algorithms cluster customers based on purchasing behavior or demographics, enabling targeted marketing strategies.
- **Anomaly Detection:** Identifies unusual patterns in data, aiding fraud detection, cybersecurity, and equipment failure prevention.
- **Recommendation Systems**: Suggests products, movies, or music by analyzing user behavior and preferences.
- **Image and Text Clustering**: Groups similar images or documents for tasks like organization, classification, or content recommendation.
- **Social Network Analysis**: Detects communities or trends in user interactions on social media platforms.
- **Astronomy and Climate Science:** Classifies galaxies or groups weather patterns to support scientific research

3.2. Clustering

3.2.1. K-Means Clustering

K-Means Clustering is an Unsupervised Machine Learning algorithm which groups unlabeled dataset into different clusters. It is used to organize data into **groups based on their similarity**.



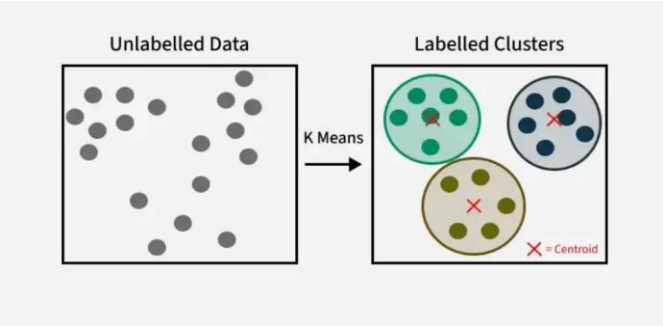
For example, online store uses K-Means to group customers based on purchase frequency and spending creating segments like Budget Shoppers, Frequent Buyers and Big Spenders for personalised marketing.

The algorithm works by first randomly picking some central points called centroids and each data point is then assigned to the closest centroid forming a cluster. After all the points are assigned to a cluster the centroids are updated by finding the average position of the points in each cluster. This process repeats until the centroids stop changing forming clusters. The goal of clustering is to divide the data points into clusters so that similar data points belong to same group.

How k-means clustering works?

We are given a data set of items with certain features and values for these features like a vector. The task is to categorize those items into groups. To achieve this, we will use the K-means

algorithm. 'K' in the name of the algorithm represents the number of groups/clusters we want to classify our items into.



K means Clustering

The algorithm will categorize the items into k groups or clusters of similarity. To calculate that similarity, we will use the **Euclidean distance** as a measurement. The algorithm works as follows:

1. First, we randomly initialize k points called means or cluster centroids.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that cluster so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

The "points" mentioned above are called means because they are the mean values of the items categorized in them. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set. For example, for a feature *x* the items have values in [0,3] we will initialize the means with values for x at [0,3].

Steps:

Step-01:

- Choose the number of clusters K.

Step-02:

- Randomly select any K data points as cluster centers.
- Select cluster centers in such a way that they are as farther as possible from each other.

Step-03:

- Calculate the distance between each data point and each cluster center.
- The distance may be calculated either by using given distance function or by using Euclidean distance formula.

Step-04:

- Assign each data point to some cluster.

- A data point is assigned to that cluster whose center is nearest to that data point.

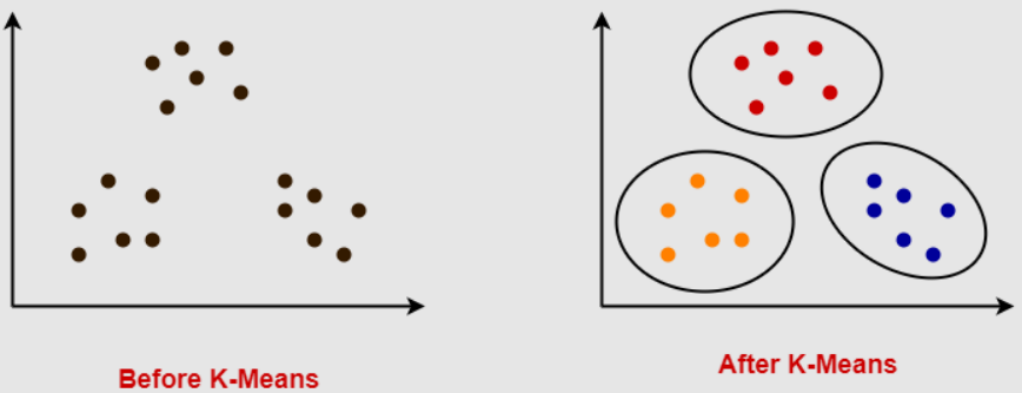
Step-05:

- Re-compute the center of newly formed clusters.
- The center of a cluster is computed by taking mean of all the data points contained in that cluster.

Step-06:

Keep repeating the procedure from *Step-03* to *Step-05* until any of the following stopping criteria is met-

- Center of newly formed clusters do not change
- Data points remain present in the same cluster
- Maximum number of iterations are reached



Applications of K-means Clustering in Machine Learning

K-means clustering algorithm finds its applications in various domains. Following are some of the popular applications of k-means clustering.

- **Document Classification**: Using k-means clustering, we can divide documents into various clusters based on their content, topics, and tags.
- **Customer segmentation**: Supermarkets and e-commerce websites divide their customers into various clusters based on their transaction data and demography. This helps the business to target appropriate customers with relevant products to increase sales.
- **Cyber profiling**: In cyber profiling, we collect data from individuals as well as groups to identify their relationships. With k-means clustering, we can easily make clusters of people based on their connection to each other to identify any available patterns.
- **Image segmentation**: We can use k-means clustering to perform image segmentation by grouping similar pixels into clusters.
- **Fraud detection in banking and insurance**: By using historical data on frauds, banks and insurance agencies can predict potential frauds by the application of k-means clustering.

Advantages of K-means Clustering Algorithm

- **Easy to implement**: K-means clustering is an iterable algorithm and a relatively simple algorithm. In fact, we can also perform k-means clustering manually as we did in the numerical example.
- **Scalability**: We can use k-means clustering for even 10 records or even 10 million records in a dataset. It will give us results in both cases.
- **Convergence**: The k-means clustering algorithm is guaranteed to give us results. It guarantees convergence. Thus, we will get the result of the execution of the algorithm for sure.
- **Generalization**: K-means clustering doesn't apply to a specific problem. From numerical data to text documents, you can use the k-means clustering algorithm on any dataset to perform clustering. It can also be applied to datasets of different sizes having entirely different distributions in the dataset. Hence, this algorithm is completely generalized.
- **Choice of centroids**: You can warm-start the choice of centroids in an easy manner. Hence, the algorithm allows you to choose and assign centroids that fit well with the dataset.

Disadvantages of K-means Clustering Algorithm

- **Deciding the number of clusters**: In k-means clustering, you need to decide the number of clusters by using the elbow method.
- **Choice of initial centroids**: The number of iterations in the clustering process completely depends on the choice of centroids. Hence, you need to properly choose the centroids in the initial step for maximizing the efficiency of the algorithm.
- **Effect of outliers**: In the execution of the k-means clustering algorithm, we use all the points in a cluster to determine the centroids for the next iteration. If there are outliers in the dataset, they highly affect the position of the centroids. Due to this, the clustering becomes inaccurate. To avoid this, you can try to identify outliers and remove them in the data cleaning process.
- **Curse of Dimensionality**: If the number of dimensions in the dataset increases, the distance of the data points from a given point starts converging to a specific value. Due to this, k-means clustering that calculates the clusters based on the distance between the points becomes inefficient. To overcome this problem, you can use advanced clustering algorithms like spectral clustering. Alternatively, you can also try to reduce the dimensionality of the dataset while data preprocessing.

Numerical Problem 1:

A1(2, 10), A2(2, 5), A3(8, 4), A4(5, 8), A5(7, 5), A6(6, 4), A7(1, 2), A8(4, 9)

Initial cluster centers are: A1(2, 10), A4(5, 8) and A7(1, 2).

Initial Cluster Centers:

- Cluster 1 center: A1 (2, 10)
- Cluster 2 center: A4 (5, 8)
- Cluster 3 center: A7 (1, 2)

We will perform 1 full iteration of K-means clustering (assignment + recompute centroids).

Step 1: Assign Points to the Nearest Cluster

We compute the Euclidean distance from each point to the 3 cluster centers.

Distance Formula:

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

Point	Distance to A1 (2,10)	Distance to A4 (5,8)	Distance to A7 (1,2)	Assigned Cluster
A1	0.00	3.61	8.06	Cluster 1
A2	5.00	3.61	3.16	Cluster 3
A3	7.21	3.61	7.81	Cluster 2
A4	3.61	0.00	6.71	Cluster 2
A5	5.83	2.24	6.08	Cluster 2
A6	5.66	2.83	5.39	Cluster 2
A7	8.06	6.71	0.00	Cluster 3
A8	2.24	1.41	7.21	Cluster 2

New Cluster Assignments

Cluster 1 (A1): A1

Cluster 2 (A4): A3, A4, A5, A6, A8

Cluster 3 (A7): A2, A7

Recompute Cluster Centers

Cluster 1:

Points: A1 → (2, 10)

New center: (2, 10)

Cluster 2:

Points: A3(8,4), A4(5,8), A5(7,5), A6(6,4), A8(4,9)

Average:

x=(8+5+7+6+4)/5=30/5=6

y=(4+8+5+4+9)/5

New center: (6, 6)

Cluster 3:

Points: A2(2,5), A7(1,2)

Average:

x=(2+1)/2=1.5x = (2+1)/2 = 1.5x=(2+1)/2=1.5

y=(5+2)/2=3.5y = (5+2)/2 = 3.5y=(5+2)/2=3.5
New center: (1.5, 3.5)

After One Iteration

Cluster 1 (center: (2, 10)): A1
Cluster 2 (center: (6, 6)): A3, A4, A5, A6, A8
Cluster 3 (center: (1.5, 3.5)): A2, A7

Numerical Exercise:

Given Points:

B1 (3, 7), B2 (6, 2), B3 (7, 4), B4 (4, 6), B5 (5, 3), B6 (2, 8), B7 (1, 1), B8 (8, 2)

Initial Cluster Centers:

Cluster 1: B1 (3, 7)
Cluster 2: B2 (6, 2)
Cluster 3: B7 (1, 1)

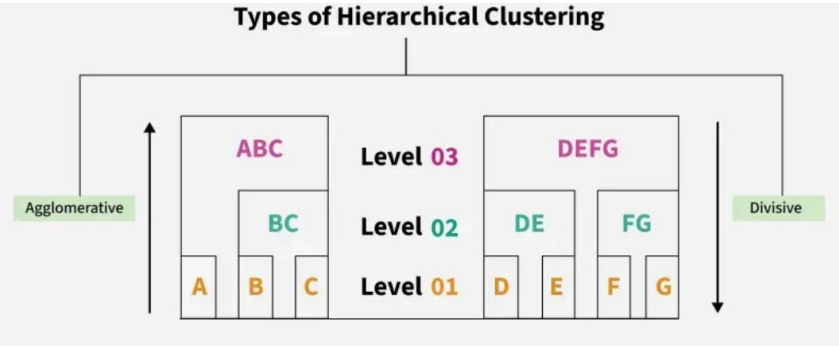
3.2.2. Hierarchical Clustering

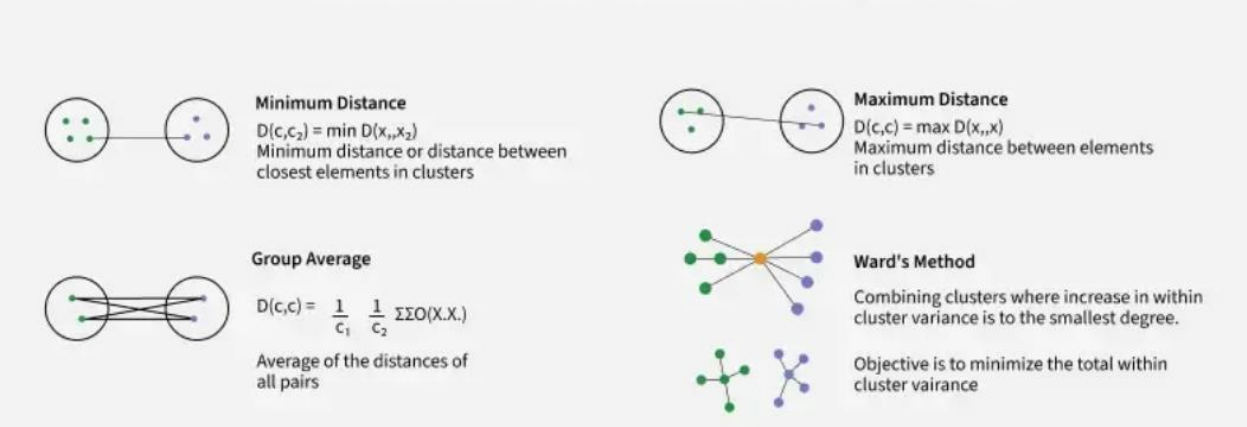
Hierarchical clustering is used to group similar data points together based on their similarity creating a **hierarchy or tree-like structure**. The key idea is to begin with each data point as its own separate cluster and then progressively merge or split them based on their similarity. Let's understand this with the help of an example

*Imagine you have four fruits with different weights: an **apple (100g)**, **a banana (120g), a cherry (50g) and a grape (30g)**. Hierarchical clustering starts by treating each **fruit as its own group**.*

- It then merges the closest groups based on their weights.
- First the cherry and grape are grouped together because they are the lightest.
- Next the apple and banana are grouped together.

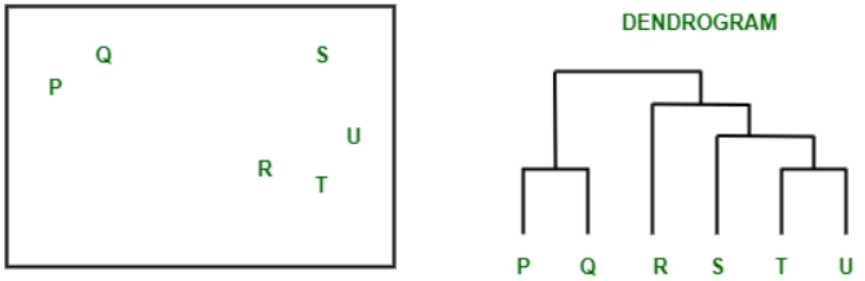
Finally, all the fruits are merged into one large group, showing how hierarchical clustering progressively combines the most similar data points.





Dendogram

A **dendrogram** is like a family tree for clusters. It shows how individual data points or groups of data merge together. The bottom shows each data point as its own group, and as you move up, similar groups are combined. The lower the merge point, the more similar the groups are. It helps you see how things are grouped step by step. The working of the dendrogram can be explained using the below diagram:



In the above image on the left side there are five points labeled P, Q, R, S and T. These represent individual data points that are being clustered. On the right side there's a **dendrogram** which show how these points are grouped together step by step.

- At the bottom of the dendrogram the points P, Q, R, S and T are all separate.
- As you move up, the closest points are merged into a single group.
- The lines connecting the points show how they are progressively merged based on similarity.
- The height at which they are connected shows how similar the points are to each other; the shorter the line the more similar they are

Types of Hierarchical Clustering

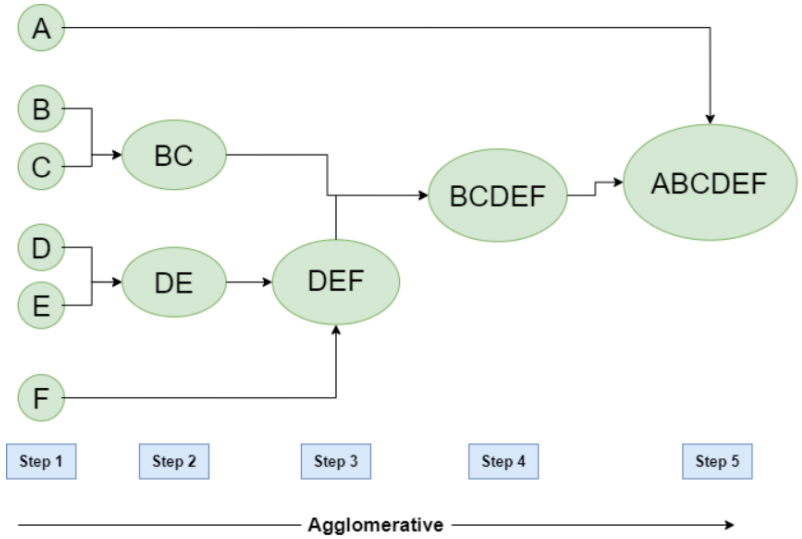
Now we understand the basics of hierarchical clustering. There are two main types of hierarchical clustering.

1. Agglomerative Clustering
2. Divisive clustering

3.2.2.1. Hierarchical Agglomerative Clustering

It is also known as the **bottom-up approach** or **hierarchical agglomerative clustering (HAC)**. Unlike flat clustering hierarchical clustering provides a structured way to group data. This

clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data.



Workflow for Hierarchical Agglomerative clustering

1. **Start with individual points**: Each data point is its own cluster. For example, if you have 5 data points you start with 5 clusters each containing just one data point.
2. **Calculate distances between clusters**: Calculate the distance between every pair of clusters. Initially since each cluster has one point this is the distance between the two data points.
3. **Merge the closest clusters**: Identify the two clusters with the smallest distance and merge them into a single cluster.
4. **Update distance matrix**: After merging you now have one less cluster. Recalculate the distances between the new cluster and the remaining clusters.
5. **Repeat steps 3 and 4**: Keep merging the closest clusters and updating the distance matrix until you have only one cluster left.
6. **Create a dendrogram**: As the process continues you can visualize the merging of clusters using a tree-like diagram called a **dendrogram**. It shows the hierarchy of how clusters are merged.

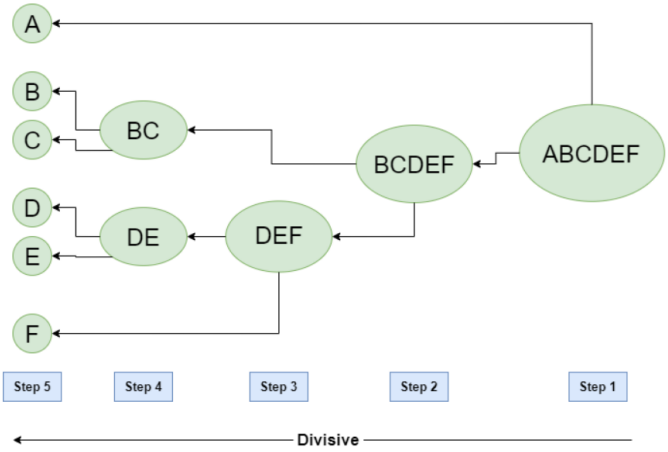
3.2.2.2. Hierarchical Divisive clustering

It is also known as a **top-down approach**. This algorithm also does not require to prespecify the number of clusters. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been split into singleton clusters.

Workflow for Hierarchical Divisive clustering:

1. **Start with all data points in one cluster**: Treat the entire dataset as a single large cluster.

2. **Split the cluster**: Divide the cluster into two smaller clusters. The division is typically done by finding the two most dissimilar points in the cluster and using them to separate the data into two parts.
3. **Repeat the process**: For each of the new clusters, repeat the splitting process:
    1. Choose the cluster with the most dissimilar points.
    2. Split it again into two smaller clusters.
4. **Stop when each data point is in its own cluster**: Continue this process until every data point is its own cluster, or the stopping condition (such as a predefined number of clusters) is met.



Application Of Hierarchical Clustering

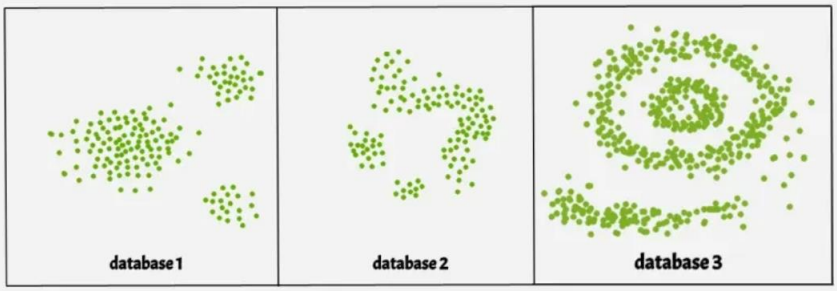
3.3.3. Density-based Clustering

3.3.3.1. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN is a **density-based clustering algorithm that groups data points that are closely packed together and marks outliers as noise based on their density** in the feature space. It identifies clusters as dense regions in the data space separated by areas of lower density. Unlike K-Means or hierarchical clustering which assumes clusters are **compact and spherical,** DBSCAN perform well in handling real-world data irregularities such as:

- **Arbitrary-Shaped Clusters**: Clusters can take any shape not just circular or convex.
- **Noise and Outliers**: It effectively identifies and handles noise points without assigning them to any cluster.

The figure above shows a data set with clustering algorithms: K-Means and Hierarchical handling compact, spherical clusters with varying noise tolerance while DBSCAN manages arbitrary-shaped clusters and noise handling.



Key Parameters in DBSCAN

1. eps: This defines the radius of the neighborhood around a data point. If the distance between two points is less than or equal to eps they are considered neighbors. A common method to determine eps is by analyzing the k-distance graph. Choosing the right eps is important:

- If eps are too small most points will be classified as noise.
- If eps are too large clusters may merge and the algorithm may fail to distinguish between them.

2. MinPts: This is the minimum number of points required within the **eps** radius to form a dense region. A general rule of thumb is to set MinPts >= D+1 where **D** is the number of dimensions in the dataset.

*For most cases a minimum value of **MinPts = 3** is recommended.*

How Does DBSCAN Work?

DBSCAN works by categorizing data points into three types:

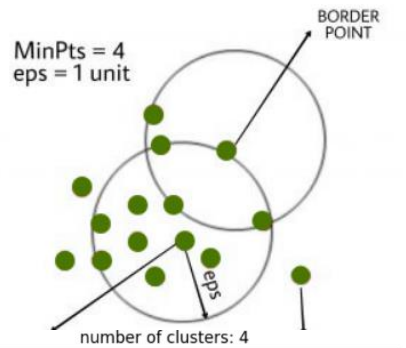
1. Core points which have a sufficient number of neighbors within a specified radius (eplison)
2. Border points which are near core points but lack enough neighbors to be core points themselves
3. Noise points which do not belong to any cluster.

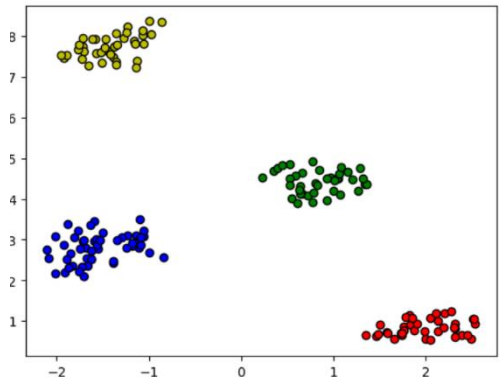
By iteratively expanding clusters from core points and connecting density-reachable points, DBSCAN forms clusters without relying on rigid assumptions about their shape or size.

Steps in the DBSCAN Algorithm

1. **Identify Core Points**: For each point in the dataset count the number of points within its eps neighborhood. If the count meets or exceeds MinPts mark the point as a core point.

2. **Form Clusters**: For each core point that is not already assigned to a cluster create a new cluster. Recursively find all density-connected points i.e points within the eps radius of the core point and add them to the cluster.
3. **Density Connectivity**: Two points a and b are density-connected if there exists a chain of points where each point is within the eps radius of the next and at least one point in the chain is a core point. This chaining process ensures that all points in a cluster are connected through a series of dense regions.
4. **Label Noise Points**: After processing all points any point that does not belong to a cluster is labeled as noise.

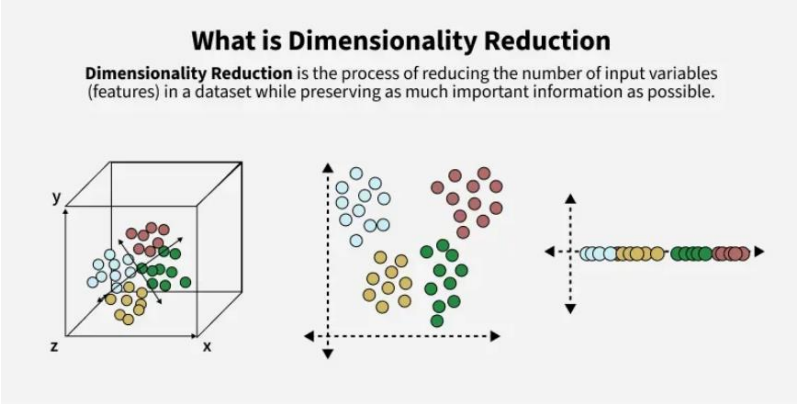




When Should We Use DBSCAN Over K-Means Clustering?

No.	K-Means Clustering	DBSCAN Clustering
1.	Clusters are typically spherical or convex and require similar feature sizes.	Clusters can have arbitrary shapes and varying feature sizes.
2.	Requires the number of clusters (**K**) to be specified in advance.	Does **not require** the number of clusters to be pre-specified.
3.	Performs efficiently on large datasets.	Less efficient on high-dimensional data.
4.	Sensitive to **outliers** and **noisy data**.	Handles outliers and noisy data effectively.
5.	May wrongly include anomalies within clusters, making it less effective for **anomaly detection**.	Identifies **dense regions**, separating them from low-density (possibly anomalous) regions.
6.	Requires **1 parameter**: Number of clusters (**K**).	Requires **2 parameters**: Radius (**R**) and Minimum Points (**M**).• **R** defines the neighborhood radius.• **M** sets the min points for a dense cluster.
7.	Less affected by **varying densities** of data points.	Performs poorly on data with **varying density** or **sparse distributions**.

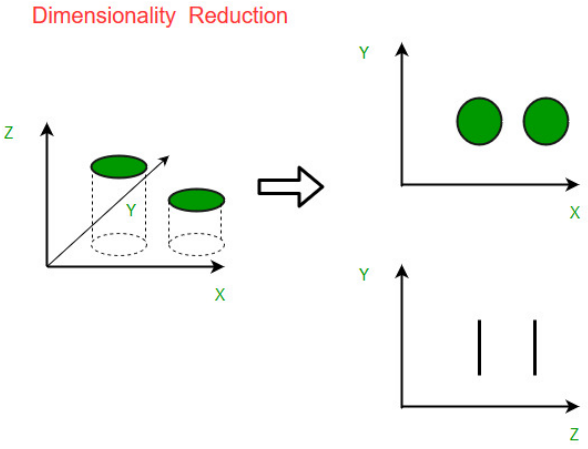
3.4. Dimensionality Reduction



When working with machine learning models, datasets with too many features can cause issues like slow computation and overfitting. Dimensionality reduction helps to reduce the number of features while retaining key information. Techniques like **principal component analysis (PCA)**, **singular value decomposition (SVD)** and **linear discriminant analysis (LDA)** convert data into a lower-dimensional space while preserving important details.

How Dimensionality Reduction Works?

Let's understand how dimensionality Reduction is used with the help of example. Imagine a dataset where each data point exists in a 3D space defined by axes X, Y and Z. If most of the data variance occurs along X and Y then the Z-dimension may contribute very little to understanding the structure of the data.



- Before Reduction You can see that Data exist in 3D (X,Y,Z). It has high redundancy and Z contributes little meaningful information
- On the right after reducing the dimensionality the data is represented in **lower-dimensional spaces**. The top plot (X-Y) maintains the meaningful structure while the bottom plot (Z-Y) shows that the Z-dimension contributed little useful information.

This process makes data analysis more efficient, improving computation speed and visualization while minimizing redundancy.

Dimensionality Reduction Techniques

Dimensionality reduction techniques can be broadly divided into two categories:

1. Feature Selection

Feature selection chooses the most relevant features from the dataset without altering them. It helps remove redundant or irrelevant features, improving model efficiency. Some common methods are:

- Filter methods rank the features based on their relevance to the target variable.
- Wrapper methods use the model performance as the criteria for selecting features.
- Embedded methods combine feature selection with the model training process.

2. Feature Extraction

Feature extraction involves creating new features by combining or transforming the original features. These new features retain most of the dataset's important information in fewer dimensions. Common feature extraction methods are:

1. **Principal Component Analysis (PCA):** Converts correlated variables into uncorrelated 'principal components, reducing dimensionality while maintaining as much variance as possible enabling more efficient analysis.
2. **Missing Value Ratio:** Variables with missing data beyond a set threshold are removed, improving dataset reliability.
3. **Backward Feature Elimination**: Starts with all features and removes the least significant ones in each iteration. The process continues until only the most impactful features remain, optimizing model performance.
4. **Forward Feature Selection:** Forward Feature Selection Begins with one feature, adds others incrementally and keeps those improving model performance.
5. **Random Forest**: Random Forest Uses decision trees to evaluate feature importance, automatically selecting the most relevant features without the need for manual coding, enhancing model accuracy.
6. **Factor Analysis**: Groups variables by correlation and keeps the most relevant ones for further analysis.
7. **Independent Component Analysis (ICA)**: Identifies statistically independent components, ideal for applications like 'blind source separation' where traditional correlation-based methods fall short.

Dimensionality Reduction Real World Examples

Dimensionality reduction plays an important role in many real-world applications such as text categorization, image retrieval, gene expression analysis and more. Here are a few examples:

1. **Text Categorization:** With vast amounts of online data dimensionality reduction helps classify text documents into predefined categories by reducing the feature space like word or phrase features while maintaining accuracy.
2. **Image Retrieval**: As image data grows indexing based on visual content like color, texture, shape rather than just text descriptions have become essential. This allows for better retrieval of images from large databases.
3. **Gene Expression Analysis**: Dimensionality reduction accelerates gene expression analysis help to classify samples like leukemia by identifying key features, improve both speed and accuracy.
4. **Intrusion Detection**: In cybersecurity dimensionality reduction helps analyze user activity patterns to detect suspicious behaviors and intrusions by identifying optimal features for network monitoring.

Advantages of Dimensionality Reduction

As seen earlier high dimensionality makes models inefficient. Let's now summarize the key advantages of reducing dimensionality.

- **Faster Computation**: With fewer features machine learning algorithms can process data more quickly. This results in faster model training and testing which is particularly useful when working with large datasets.
- **Better Visualization**: As we saw in the earlier figure reducing dimensions makes it easier to visualize data and reveal hidden patterns.
- **Prevent Overfitting**: With few features models are less likely to memorize the training data and overfit. This helps the model generalize better to new, unseen data improve its ability to make accurate predictions.

Disadvantages of Dimensionality Reduction

- **Data Loss & Reduced Accuracy:** Some important information may be lost during dimensionality reduction and affect model performance.
- **Choosing the Right Components:** Deciding how many dimensions to keep is difficult as keeping too few may lose valuable information while keeping too many can led to overfitting.

3.4.1. Principal Component Analysis (PCA)

PCA (Principal Component Analysis) is a **dimensionality reduction** technique used in data analysis and machine learning. **It helps you to reduce the number of features in a dataset while keeping the most important information.** It changes your original features into new features these new features don't overlap with each other and the first few keep most of the important differences found in the original data.

PCA is commonly used for data preprocessing for use with machine learning algorithms. It helps to remove redundancy, improve computational efficiency and make data easier to visualize and analyze especially when dealing with high-dimensional data.

How Principal Component Analysis Works

PCA uses linear algebra to transform data into new features called principal components. It finds these by calculating eigenvectors (directions) and eigenvalues (importance) from the covariance matrix. PCA selects the top components with the highest eigenvalues and projects the data onto them simplify the dataset.

Note: *It prioritizes the directions where the data varies the most because more variation = more useful information.*

Imagine you're looking at a messy cloud of data points like stars in the sky and want to simplify it. PCA helps you find the "most important angles" to view this cloud so you don't miss the big patterns. Here's how it works step by step:

Step 1: Standardize the Data

Different features may have different units and scales like salary vs. age. To compare them fairly PCA first standardizes the data by making each feature have:

- A mean of 0
- A standard deviation of 1

$$Z = \frac{X - \mu}{\sigma}$$

where:

- μ is the mean of independent features
  $\mu=\{\mu_1,\mu_2,\cdots,\mu m\}\mu=\{\mu_1,\mu_2,\cdots,\mu_m\}$
- σ is the standard deviation of independent features
  $\sigma=\{\sigma_1,\sigma_2,\cdots,\sigma m\}\sigma=\{\sigma_1,\sigma_2,\cdots,\sigma_m\}$

Step 2: Calculate Covariance Matrix

Next PCA calculates the **covariance matrix** to see how features relate to each other whether they increase or decrease together. The covariance between two features x1 and x2 is:

$$cov(x1, x2) = \frac{\sum_{i=1}^{n}(x1_i - \bar{x1})(x2_i - \bar{x2})}{n-1}$$

Where:

- $\bar{x}_1 \, and \, \bar{x}_2$ are the mean values of features $x_1 \, and \, x_2$
- n is the number of data points

The value of covariance can be positive, negative or zeros.

Step 3: Find the Principal Components

PCA identifies **new axes** where the data spreads out the most:

- **1st Principal Component (PC1):** The direction of maximum variance (most spread).
- **2nd Principal Component (PC2):** The next best direction, *perpendicular to PC1* and so on.

These directions come from the **eigenvectors** of the covariance matrix and their importance is measured by **eigenvalues**. For a square matrix A an **eigenvector** X (a non-zero vector) and its corresponding **eigenvalue** λ satisfy:

$$AX=\lambda X$$

This means:

- When *A* acts on X it only stretches or shrinks X by the scalar λ.
- The direction of X remains unchanged hence eigenvectors define "stable directions" of A.

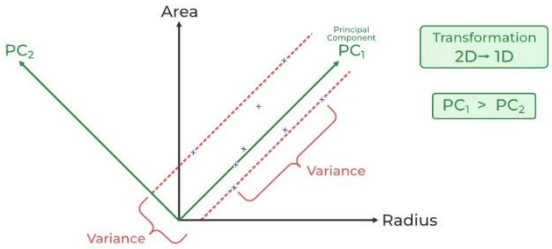
Eigenvalues help rank these directions by importance.

Step 4: Pick the Top Directions & Transform Data

After calculating the eigenvalues and eigenvectors PCA ranks them by the amount of information they capture. We then:

1. **Select the top k components** hat capture most of the variance like 95%.
2. **Transform the original dataset** by projecting it onto these top components.

This means we reduce the number of features (dimensions) while keeping the important patterns in the data.



Advantages of Principal Component Analysis

1. **Multicollinearity Handling:** Creates new, uncorrelated variables to address issues when original features are highly correlated.
2. **Noise Reduction:** Eliminates components with low variance enhance data clarity.
3. **Data Compression:** Represents data with fewer components reduce storage needs and speeding up processing.
4. **Outlier Detection:** Identifies unusual data points by showing which ones deviate significantly in the reduced space.

Disadvantages of Principal Component Analysis

1. **Interpretation Challenges:** The new components are combinations of original variables which can be hard to explain.
2. **Data Scaling Sensitivity:** Requires proper scaling of data before application or results may be misleading.
3. **Information Loss:** Reducing dimensions may lose some important information if too few components are kept.
4. **Assumption of Linearity:** Works best when relationships between variables are linear and may struggle with non-linear data.
5. **Computational Complexity:** Can be slow and resource-intensive on very large datasets.
6. **Risk of Overfitting:** Using too many components or working with a small dataset might lead to models that don't generalize well.

Numerical for PCA :

Consider the following dataset

x1	**2.5**	**0.5**	**2.2**	**1.9**	**3.1**	**2.3**	**2.0**	**1.0**	**1.5**	**1.1**
x2	**2.4**	**0.7**	**2.9**	**2.2**	**3.0**	**2.7**	**1.6**	**1.1**	**1.6**	**0.9**

Step 1: Standardize the Dataset

Mean for x_1 $= 1.81 = x_{1mean}$

Mean for x_2 $= 1.91 = x_{2mean}$

We will change the dataset.

$x_{1new} = x_1 - x_{1mean}$	0.69	-1.31	0.39	0.09	1.29	0.49	0.19	-0.81	-0.31	-0.71
$x_{2new} = x_2 - x_{2mean}$	0.49	-1.21	0.99	0.29	1.09	0.79	-0.31	-0.81	-0.31	-1.01

Step 1: Standardize the Dataset

Mean for x_1 $= 1.81 = x_{1mean}$

Mean for x_2 $= 1.91 = x_{2mean}$

We will change the dataset.

$x_{1new} = x_1 - x_{1mean}$	0.69	-1.31	0.39	0.09	1.29	0.49	0.19	-0.81	-0.31	-0.71
$x_{2new} = x_2 - x_{2mean}$	0.49	-1.21	0.99	0.29	1.09	0.79	-0.31	-0.81	-0.31	-1.01

Step 2: Find the Eigenvalues and eigenvectors

$$\textbf{Correlation Matrix c} = C = \left(\frac{X \cdot X^T}{N-1}\right)$$

where, X is the Dataset Matrix (In this numerical, it is a 10 X 2 matrix)

where, X is the Dataset Matrix (In this numerical, it is a 10 X 2 matrix)

X^T is the transpose of the X (In this numerical, it is a 2 X 10 matrix) and N is the number of elements = 10

So, $C = \left(\frac{X \cdot X^T}{10-1}\right) = \left(\frac{X \cdot X^T}{9}\right)$

{So in order to calculate the Correlation Matrix, we have to do the multiplication of the Dataset Matrix with its transpose}

$$C = \begin{bmatrix} 0.616556 & 0.615444 \\ 0.615444 & 0.716556 \end{bmatrix}$$

Using the equation, | **C** - λ **I** | = **0- equation (i)** where { \lambda is the eigenvalue and I is the Identity Matrix }

So solving equation (i)

$$\begin{bmatrix} 0.616556 & 0.615444 \\ 0.615444 & 0.716556 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{vmatrix} 0.616556 - \lambda & 0.615444 \\ 0.615444 & 0.716556 - \lambda \end{vmatrix} = 0$$

Taking the determinant of the left side, we get

$$0.44180 - 0.616556\lambda - 0.716556\lambda + \lambda^2 - 0.37877 = 0$$

$$\lambda^2 - 1.33311\lambda + 0.06303 = 0$$

We get two values for λ , that are (λ_1) = **1.28403 and** (λ_2) = **0.0490834**. Now we have to find the eigenvectors for the eigenvalues λ_1 **and** λ_2

To find the eigenvectors from the eigenvalues, we will use the following approach:

First, we will find the eigenvectors for the eigenvalue 1.28403 by using the equation $C \cdot X = \lambda \cdot X$

$$\begin{bmatrix} 0.616556 & 0.615444 \\ 0.615444 & 0.716556 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = 1.28403 \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} 0.616556x + 0.615444y \\ 0.615444x + 0.716556y \end{bmatrix} = \begin{bmatrix} 1.28403x \\ 1.28403y \end{bmatrix}$$

Solving the matrices, we get

0.616556x + 0.615444y = 1.28403x ; **x = 0.922049 y**

(x and y belongs to the matrix X) so if we put y = 1, x comes out to be 0.922049. So now the updated X matrix will look like:

$$X = \begin{bmatrix} 0.922049 \\ 1 \end{bmatrix}$$

IMP: Till now we haven't reached to the eigenvectors, we have to a bit of modifications in the X matrix. They are as follows:

A. Find the square root of the sum of the squares of the element in X matrix i.e.

$$\sqrt{0.922049^2 + 1^2} = \sqrt{0.850174 + 1} = \sqrt{1.850174} = 1.3602$$

B. Now divide the elements of the X matrix by the number 1.3602 (just found that)

$$\begin{bmatrix} \frac{0.922049}{1.3602} \\ \\ \frac{1}{1.3602} \end{bmatrix} = \begin{bmatrix} 0.67787 \\ 0.73518 \end{bmatrix}$$

So now we found the eigenvectors for the eigenvalue λ_1 , they are 0.67787 and 0.73518

Secondly, we will find the eigenvectors for the eigenvalue 0.0490834 by using the equation {Same approach as of previous step)

$$\begin{bmatrix} 0.616556 & 0.615444 \\ 0.615444 & 0.716556 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = 0.0490834 \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} 0.616556x + 0.615444y \\ 0.615444x + 0.716556y \end{bmatrix} = \begin{bmatrix} 0.0490834x \\ 0.0490834y \end{bmatrix}$$

Solving the matrices, we get

0.616556x + 0.615444y = 0.0490834x; **y = -0.922053**

(x and y belongs to the matrix X) so if we put x = 1, y comes out to be -0.922053 So now the updated X matrix will look like:

$$X = \begin{bmatrix} 1 \\ -0.922053 \end{bmatrix}$$

IMP: Till now we haven't reached to the eigenvectors, we have to a bit of modifications in the X matrix. They are as follows:

A. Find the square root of the sum of the squares of the elements in X matrix i.e.

$$\sqrt{1^2 + (-0.922053)^2} = \sqrt{1 + 0.85018} = \sqrt{1.85018} = 1.3602$$

B. Now divide the elements of the X matrix by the number 1.3602 (just found that)

$$\begin{bmatrix} \frac{1}{1.3602} \\ \\ \frac{-0.922053}{1.3602} \end{bmatrix} = \begin{bmatrix} 0.735179 \\ 0.677873 \end{bmatrix}$$

So now we found the eigenvectors for the eigenvector \lambda_2, they are 0.735176 and 0.677873

Sum of eigenvalues (λ_1) and (λ_2) = 1.28403 + 0.0490834 = 1.33 = Total Variance {Majority of variance comes from λ_1 }

Step 3: Arrange Eigenvalues

The eigenvector with the highest eigenvalue is the Principal Component of the dataset. So in this case, eigenvectors of lambda1 are the principal components.

{Basically in order to complete the numerical we have to only solve till this step, but if we have to prove why we have chosen that particular eigenvector we have to follow the steps from 4 to 6}

Step 4: Form Feature Vector

$$\begin{bmatrix} 0.677873 & 0.735179 \\ 0.735179 & -0.677879 \end{bmatrix}$$ This is the FEATURE VECTOR for Numerical

Step 5: Transform Original Dataset

Use the equation Z = X V

$$\begin{bmatrix} 0.69 & 0.49 \\ -1.31 & -1.21 \\ 0.39 & 0.99 \\ 0.09 & 0.29 \\ 1.29 & 1.09 \\ 0.49 & 0.79 \\ 0.19 & -0.31 \\ -0.81 & -0.81 \\ -0.31 & -0.31 \\ -0.71 & -1.01 \end{bmatrix} \cdot \begin{bmatrix} 0.677873 & 0.735179 \\ 0.735179 & -0.677879 \end{bmatrix} = \begin{bmatrix} 0.8297008 & 0.17511574 \\ -1.77758022 & -0.14285816 \\ 0.99219768 & -0.38437446 \\ 0.27421048 & -0.13041706 \\ 1.67580128 & 0.20949934 \\ 0.91294918 & -0.17528196 \\ -1.14457212 & -0.04641786 \\ -0.43804612 & -0.01776486 \\ -1.22382.62 & 0.16267464 \end{bmatrix} = Z$$

$$\begin{bmatrix} 0.8297008 & 0.17511574 \\ -1.77758022 & -0.14285816 \\ 0.99219768 & -0.38437446 \\ 0.27421048 & -0.13041706 \\ 1.67580128 & 0.20949934 \\ 0.91294918 & -0.17528196 \\ -1.14457212 & -0.04641786 \\ -0.43804612 & -0.01776486 \\ -1.22382.62 & 0.16267464 \end{bmatrix} \cdot \begin{bmatrix} 0.677873 & 0.735179 \\ 0.735176 & -0.677879 \end{bmatrix} = \begin{bmatrix} 0.6899999766573 & 0.4899999834233 \\ -1.3099999556827 & -1.2099999590657 \\ 0.389999968063 & 0.9899999665083 \\ 0.0899999969553 & 0.2899999901893 \\ 0.61212695653593 & 0.35482096313253 \\ 0.4899999834233 & 0.7899999732743 \\ 0.189999935723 & -0.309999995127 \\ -0.8099999725977 & -0.8099999725977 \\ -0.3099999895127 & -0.3099999895127 \\ -0.7099999759807 & -1.0099999658317 \end{bmatrix}$$

So in order to reconstruct the original data, we follow:

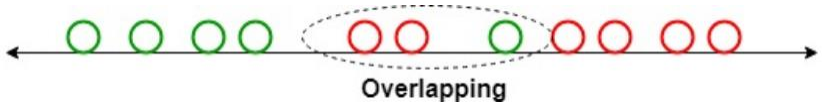
Row Original DataSet = Row Zero Mean Data + Original Mean

$$\begin{bmatrix} 0.6899999766573 & 0.4899999834233 \\ -1.3099999556827 & -1.2099999590657 \\ 0.389999968063 & 0.9899999665083 \\ 0.0899999969553 & 0.2899999901893 \\ 0.61212695653593 & 0.35482096313253 \\ 0.4899999834233 & 0.7899999732743 \\ 0.189999935723 & -0.309999995127 \\ -0.8099999725977 & -0.8099999725977 \\ -0.3099999895127 & -0.3099999895127 \\ -0.7099999759807 & -1.0099999658317 \end{bmatrix} + \begin{bmatrix} 1.81 & 1.91 \end{bmatrix} = \begin{bmatrix} 2.49 & 2.39 \\ 0.5 & 0.7 \\ 2.19 & 2.89 \\ 1.89 & 2.19 \\ 3.08 & 2.99 \\ 2.30 & 2.7 \\ 2.01 & 1.59 \\ 1.01 & 1.11 \\ 1.5 & 1.6 \\ 1.1 & 0.9 \end{bmatrix}$$

So for the eigenvectors of first eigenvalue, data can be reconstructed similar to the original dataset. Thus we can say that the Principal Component of the dataset is λ_1 is 1.28403 followed by λ_2 **that is 0.0490834**

3.4.2. Linear Discriminant Analysis (LDA)

When working with high-dimensional datasets it is important to apply dimensionality reduction techniques to make data exploration and modeling more efficient. Linear Discriminant Analysis (LDA) also known as Normal Discriminant Analysis is supervised classification problem that helps separate two or more classes by converting higher-dimensional data space into a lower-dimensional space. It is used to identify a linear combination of features that best separates classes within a dataset.



Key Assumptions of LDA

For LDA to perform effectively, certain assumptions are made:

- **Gaussian Distribution**: The data in each class should follow a normal bell-shaped distribution.
- **Equal Covariance Matrices**: All classes should have the same covariance structure.
- **Linear Separability**: The data should be separable using a straight line or plane.

If these assumptions are met LDA can produce very good results. For example, when data points belonging to two classes are plotted if they are not linearly separable LDA will attempt to find a projection that maximizes class separability.

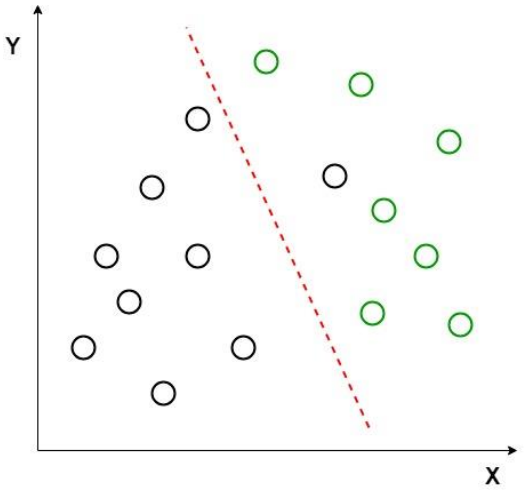
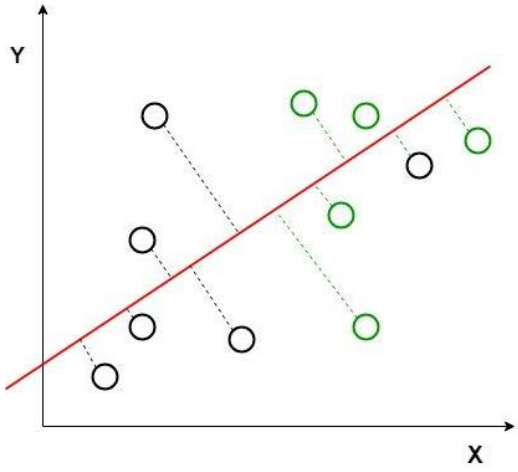


Image shows an example where the classes (black and green circles) are not linearly separable. LDA attempts to separate them using red dashed line. **It uses both axes (X and Y) to generate a new axis in such a way that it maximizes the distance between the means of the two classes while minimizing the variation within each class.** This transforms the dataset into a space where the classes are better separated. After transforming the data points along a new axis LDA maximizes the class separation. This new axis allows for clearer classification by projecting the data along a line that enhance the distance between the means of the two classes.



Perpendicular distance between the decision boundary and the data points helps us to visualize how LDA works by reducing class variation and increasing separability. After generating this

new axis using the above-mentioned criteria all the data points of the classes are plotted on this new axis and are shown in the figure given below.

It shows how LDA creates a new axis to project the data and separate the two classes effectively along a linear path. But it fails when the mean of the distributions is shared as it becomes impossible for LDA to find a new axis that makes both classes linearly separable. In such cases we use non-linear discriminant analysis.

How does LDA work

LDA works by finding directions in the feature space that best separate the classes. It does this by maximizing the difference between the class means while minimizing the spread within each class.

Let's assume we have two classes with d-dimensional samples such as

$x_1, x_2, \ldots x_n$

where:

- $n1$ samples belong to class c1.
- $n2$ samples belong to class c2.

If xi represents a data point its projection onto the line represented by the unit vector v is $v^T x_i$. Let the means of class $c1$ and class $c2$ before projection be μ1 and μ2 respectively. After projection the new means are

$$\hat{\mu}_1 = v^T \mu_1 \text{and } \hat{\mu}_2 = v^T \mu_2.$$

Our aim to normalize the difference $|\hat{\mu}_1 - \hat{\mu}_2|$ to maximize the class separation. The scatter for samples of class c1$c1$ is calculated as:

$$s_1^2 = \sum_{x_i \in c_1} (x_i - \mu_1)^2$$

Similarly for class $c2$:

$$s_2^2 = \sum_{x_i \in c_2} (x_i - \mu_2)^2$$

The goal is to maximize the ratio of the between-class scatter to the within-class scatter, which leads us to the following criteria:

$$J(v) = \frac{|\hat{\mu}_1 - \hat{\mu}_2|}{s_1^2 + s_2^2}$$

For the best separation we calculate the eigenvector corresponding to the highest eigenvalue of the scatter matrices $s_w^{-1} s_b$.

Extensions to LDA

1. **Quadratic Discriminant Analysis (QDA):** Each class uses its own estimate of variance (or covariance) allowing it to handle more complex relationships.
2. **Flexible Discriminant Analysis (FDA):** Uses non-linear combinations of inputs such as splines to handle non-linear separability.
3. **Regularized Discriminant Analysis (RDA):** Introduces regularization into the covariance estimate to prevent overfitting.

Advantages of LDA

- Simple and computationally efficient.
- Works well even when the number of features is much larger than the number of training samples.
- Can handle multicollinearity.

Disadvantages of LDA

- Assumes Gaussian distribution of data which may not always be the case.
- Assumes equal covariance matrices for different classes which may not hold in all datasets.
- Assumes linear separability which is not always true.
- May not always perform well in high-dimensional feature spaces.

Applications of LDA

1. **Face Recognition**: It is used to reduce the high-dimensional feature space of pixel values in face recognition applications helping to identify faces more efficiently.
2. **Medical Diagnosis**: It classifies disease severity in mild, moderate or severe based on patient parameters helping in decision-making for treatment.
3. **Customer Identification**: It can help identify customer segments most likely to purchase a specific product based on survey data.

Numerical Sample

Numerical Problem: Clustering Overview
A dataset with 6 points is to be clustered using a method that aims to group similar data points based on distance. The points are [1, 2], [1.5, 2.5], [5, 6], [5.5, 6.5], [2, 3], [2.5, 3.5].

1. Calculate the Euclidean distance between [1, 2] and [1.5, 2.5].
2. Suggest a potential number of clusters based on the distance pattern (intuitive guess).

Numerical Problem: DBSCAN Application
Revisit the dataset [1, 1], [1.2, 1.1], [5, 5] with epsilon = 1.0 and MinPts = 2.

1. Perform one iteration of DBSCAN and identify clusters.
2. Calculate the distance from a noise point to the nearest core point.

Numerical Problem: Dimensionality Reduction Overview
A dataset with 3 features has covariance matrix [[2, 1, 0], [1, 2, 0], [0, 0, 1]].

1. Identify the eigenvalues of the covariance matrix.
2. Determine how many principal components are needed to retain 90% of the variance.

Numerical Problem: PCA Calculation
For a dataset with points [1, 2], [2, 3], [4, 5], compute the mean and the first principal component direction (assuming standardized data).

1. Calculate the mean vector.
2. Estimate the first principal component direction (simplified covariance approach).

Numerical Problem: LDA Projection
Two classes have centroids C1 = [1, 2] and C2 = [3, 4], with within-class scatter matrix S_w = [[2, 0], [0, 2]].

1. Calculate the between-class scatter matrix S_b.
2. Determine the direction of the LDA projection (simplified).

Chapter 4

Artificial Neural Network

4.1. Introduction to Neural Network

Artificial Neural Networks contain artificial neurons, which are called **units**. These units are arranged in a series of layers that together constitute the whole **Artificial Neural Network** in a system.

A layer can have only a dozen units or millions of units**,** as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, **an** Artificial Neural Network has an input layer, an output layer, as well as hidden layers.

The input layer receives data from the outside world**,** which the neural network needs to analyze or learn about. Then, this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer.

Finally, the output layer provides an output in the form of a response of the **Artificial Neural Networks** to the input data provided.

In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit.

As the data transfers from one unit to another, the neural network learns more and more about the data, which eventually results in an output from the output layer.

- The structures and operations of human neurons serve as the basis for artificial neural networks. It is also known as neural networks or neural nets.
- The **input layer** of an artificial neural network is the **first layer,** and it receives input from external sources and releases it to the **hidden layer, which is the second layer**.
- In the hidden layer, each neuron receives input from the previous layer neurons, computes the weighted sum, and sends it to the neurons in the next layer.
- These connections are weighted means effects of the inputs from the previous layer are optimized more or less by assigning different-different weights to each input and it is adjusted during the training process by optimizing these weights for improved model performance.

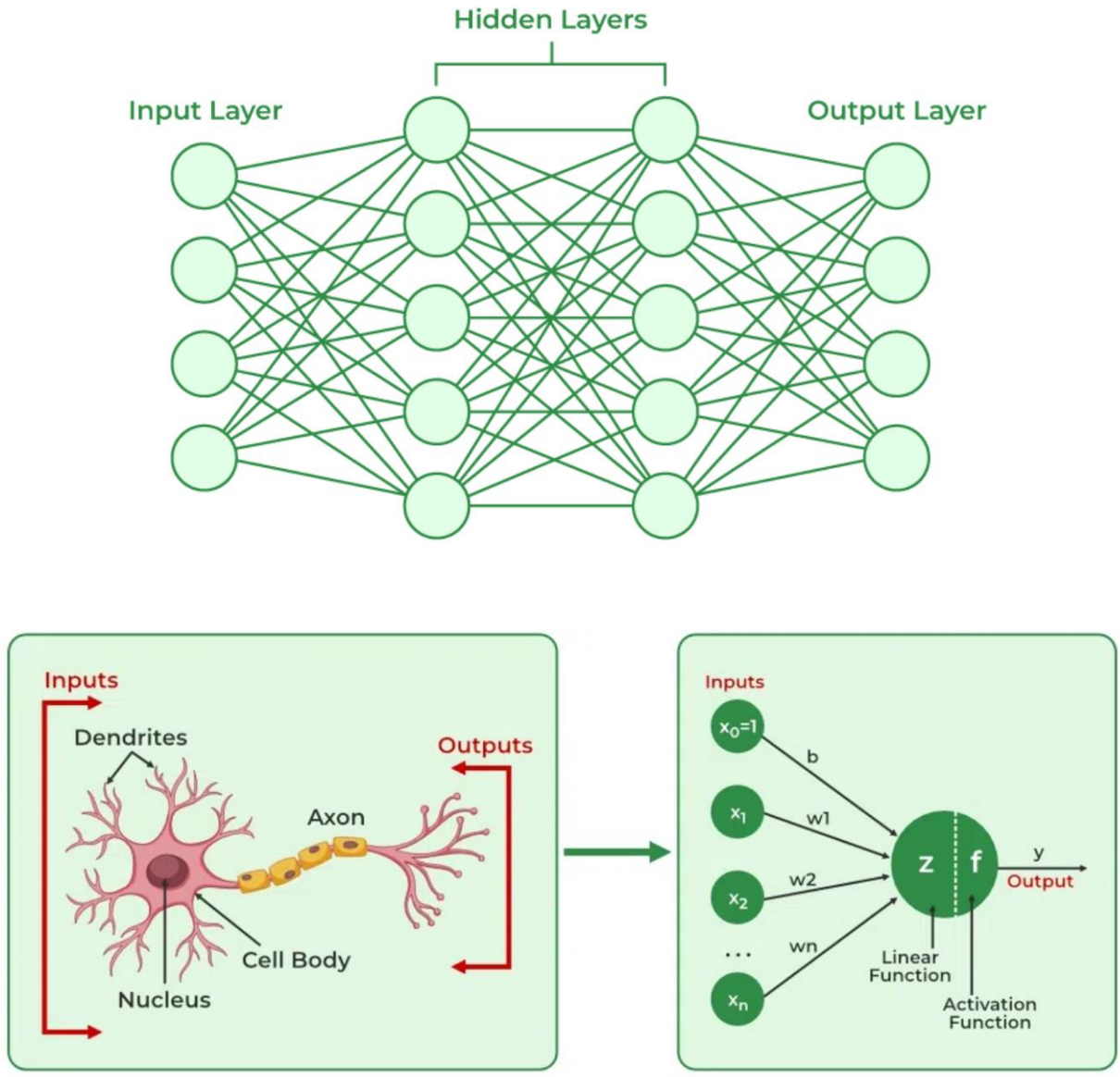


Fig. Biological neurons to Artificial neurons

Key Differences Between Biological and Artificial Neurons

Feature	Biological Neurons	Artificial Neurons
Components	Dendrites, Axons, Synapses	Input Layer, Hidden Layers, Output Layer
Function	Transmit and process electrical impulses	Take weighted inputs, apply activation function and generate output

Feature	Biological Neurons	Artificial Neurons
Learning Process	Neuron connectivity changes over time	Weights are adjusted using optimization algorithms like Gradient Descent
Processing Speed	Slow (milliseconds per computation)	Fast (nanoseconds per computation)

Characteristics of Artificial Neural Network

1. It is neuraly implemented mathematical model
2. It contains huge number of interconnected processing elements called neurons to do all operations
3. Information stored in the neurons are basically the weighted linkage of neurons
4. The input signals arrive at the processing elements through connections and connecting weights.
5. It has the ability to learn, recall and generalize from the given data by suitable assignment and adjustment of weights.
6. The collective behavior of the neurons describes its computational power, and no single neuron carries specific information.

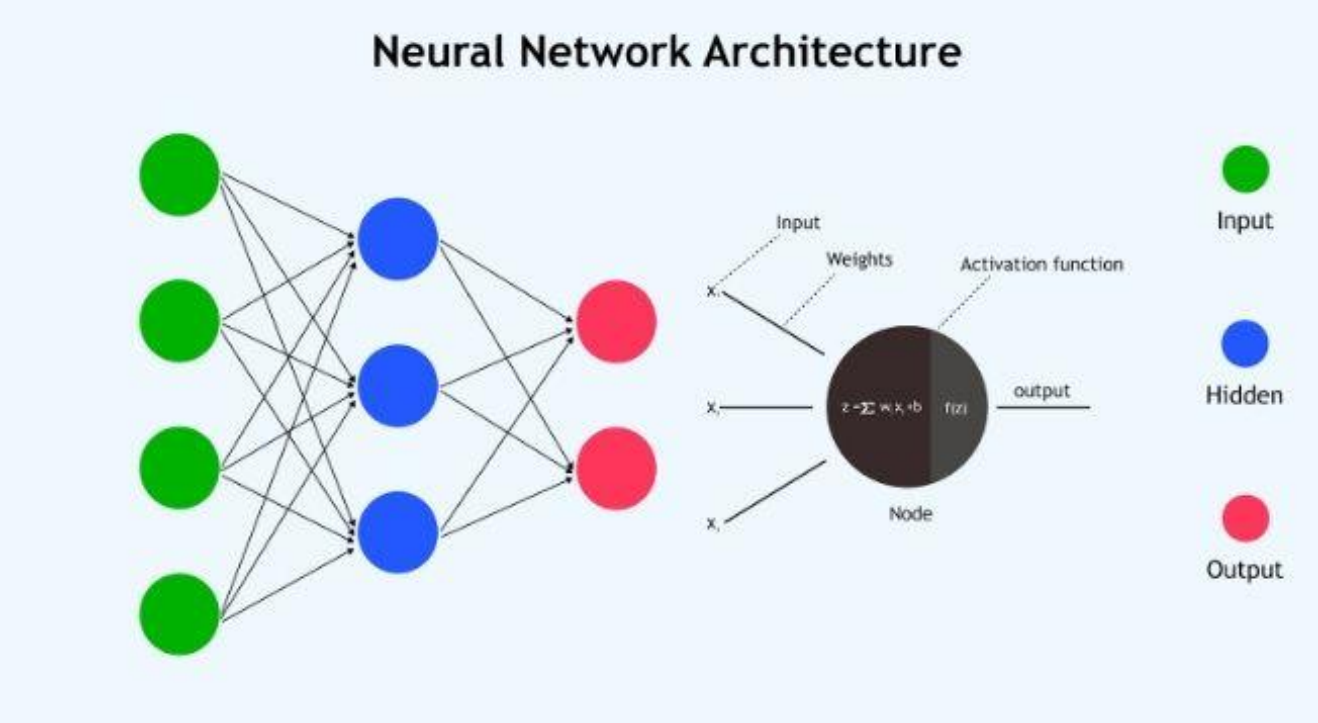
Applications of Artificial Neural Networks

- **Social Media:** Artificial Neural Networks are used heavily in Social Media. For example, let's take the **'People you may know'** feature on Facebook that suggests people that you might know in real life so that you can send them friend requests. Well, this magical effect is achieved by using Artificial Neural Networks that analyze your profile, your interests, your current friends, and also their friends and various other factors to calculate the people you might potentially know.
- **Marketing and Sales:** When you log onto E-commerce sites like Amazon and Flipkart, they will recommend you products to buy based on your previous browsing history. Similarly, suppose you love Pasta, then Zomato, Swiggy, etc. will show you restaurant recommendations based on your tastes and previous order history. This is true across all new-age marketing segments like Book sites, Movie services, Hospitality sites, etc. and it is done by implementing **personalized marketing.**

- **Healthcare**: Artificial Neural Networks are used in Oncology to train algorithms that can identify cancerous tissue at the microscopic level at the same accuracy as trained physicians. Various rare diseases may manifest in physical characteristics and can be identified in their premature stages by using **Facial Analysis** on the patient photos.

- **Personal Assistants: P**ersonal assistants like Alexa, Siri uses **Natural Language Processing** to interact with the users and formulate a response accordingly. Natural Language Processing uses artificial neural networks that are made to handle many tasks of these personal assistants such as managing the language syntax, semantics, correct speech, the conversation that is going on, etc.

4.1.1. Neural Network Architectures



- **Layers:** Neural networks consist of layers of neurons, which include the input layers, hidden layers, and output layers.

- **Connections:** Neurons are interconnected, with each connection having an associated weight that determines the strength of the connection.

- **Neurons (Nodes):** Neurons are the basic computational units that perform a weighted sum of their inputs, apply a bias, and pass the result through an activation function.

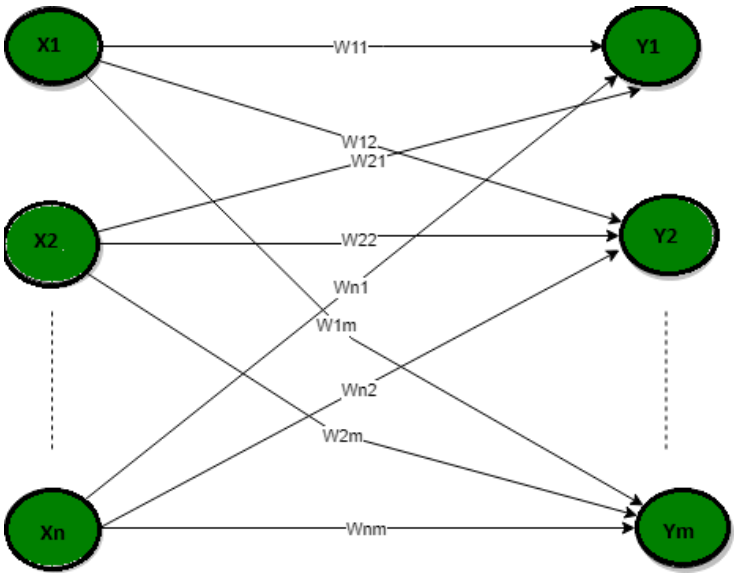
- **Weights and biases:** Weights represent the strength of the connections between neurons, and biases allow neurons to make predictions even when all inputs are zero.

- **Activation function:** Non-linear functions (like ***ReLU*** and ***Sigmoid***) are used to introduce non-linearity into the network, enabling it to model complex relationships.

There exist five basic types of neuron connection architecture:

- Single-layer feed-forward network
- Multilayer feed-forward network
- Single node with its own feedback
- Single-layer recurrent network
- Multilayer recurrent network

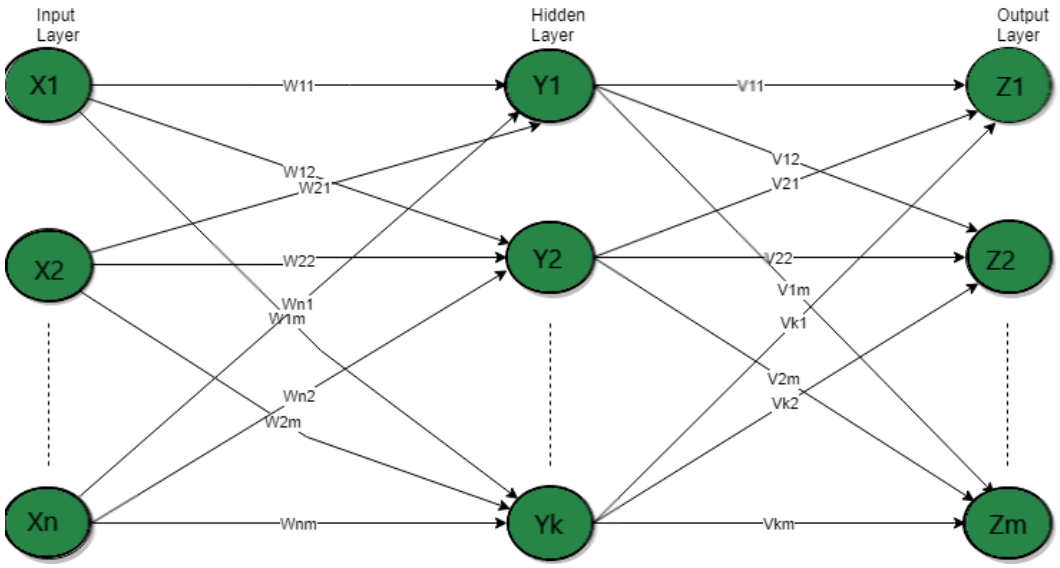
1. Single-layer feed-forward network



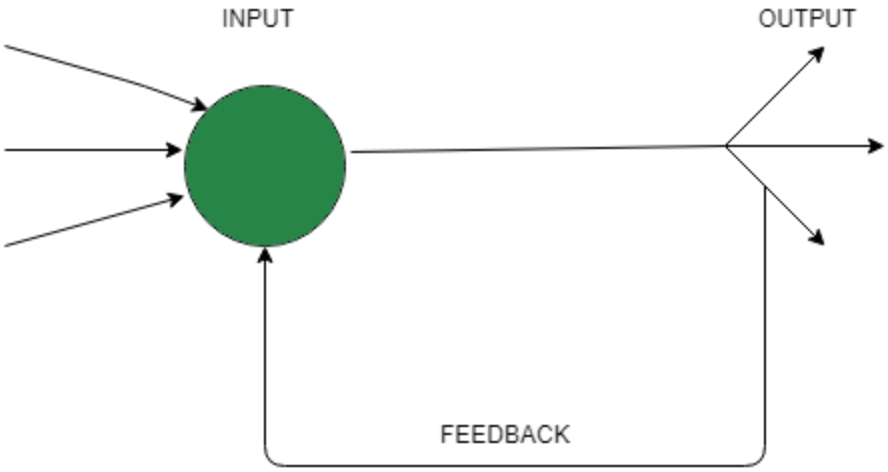
In this type of network, we have only two layers input layer and the output layer but the input layer does not count because no computation is performed in this layer. The output layer is formed when different weights are applied to input nodes and the cumulative effect per node is taken. After this, the neurons collectively give the output layer to compute the output signals.

2. Multilayer feed-forward network

This layer also has a hidden layer that is internal to the network and has no direct contact with the external layer. The existence of one or more hidden layers enables the network to be computationally stronger, a feed-forward network because of information flow through the input function, and the intermediate computations used to determine the output Z. There are no feedback connections in which outputs of the model are fed back into itself.



3. Single node with its own feedback



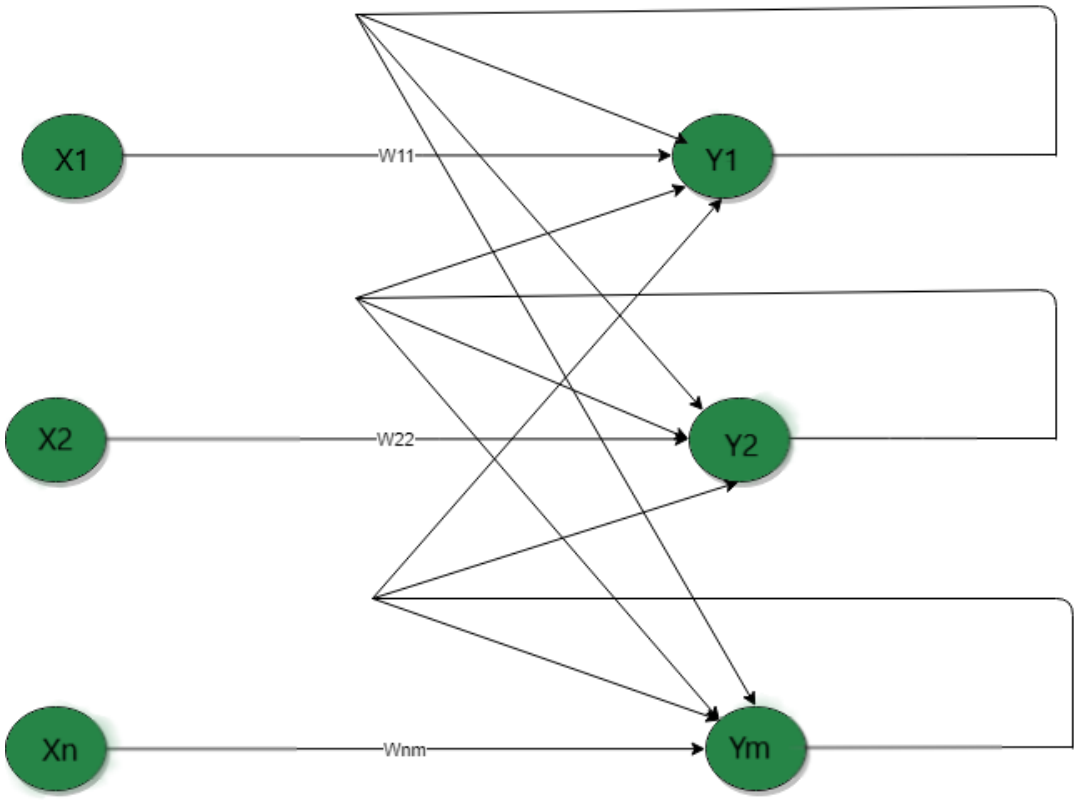
Single Node with own Feedback

When outputs can be directed back as inputs to the same layer or preceding layer nodes, then it results in feedback networks. Recurrent networks are feedback networks with closed loops. The above figure shows a single recurrent network having a single neuron with feedback to itself.

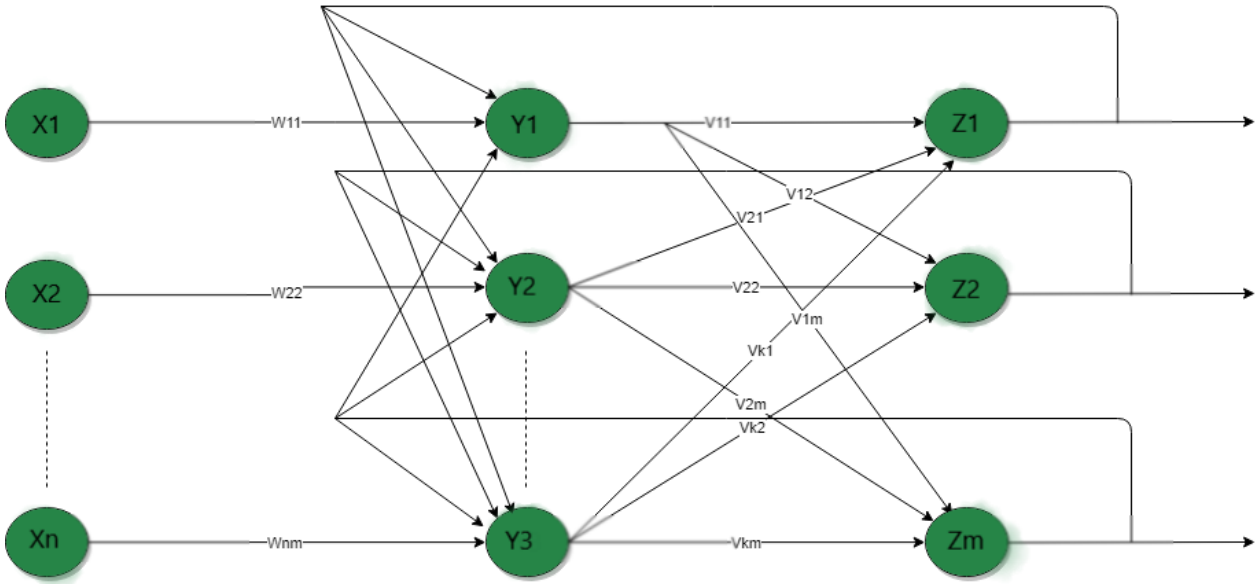
4. Single-layer recurrent network

The above network is a single-layer network with a feedback connection in which the processing element's output can be directed back to itself or to another processing element or both. A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for

a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.



5. Multilayer recurrent network



In this type of network, processing element output can be directed to the processing element in the same layer and in the preceding layer forming a multilayer recurrent network. They perform the same task for every element of a sequence, with the output being dependent on the previous computations. Inputs are not needed at each time step. The main feature of a Recurrent Neural Network is its hidden state, which captures some information about a sequence.

Types of Neural Network Architectures:

Feedforward Neural Networks:

The simplest type, where information flows in one direction from input to output, without feedback loops.

Convolutional Neural Networks (CNNs):

Designed for processing data with grid-like topology, like images. CNNs use convolutional layers to extract features from the input.

Recurrent Neural Networks (RNNs):

Designed for sequential data, where the output depends on previous inputs in the sequence. RNNs have feedback loops that allow them to maintain a "memory" of past inputs.

4.1.1.1. Feedforward

Feedforward Neural Network (FNN) is a type of artificial ***neural network*** in which information flows in a single direction—from the input layer through hidden layers to the output layer—without loops or feedback. It is mainly used for pattern recognition tasks like image and speech classification.

A feedforward neural network (FNN) is a type of artificial neural network where information flows in one direction, from input to output, without any feedback loops or cycles. It's a foundational architecture used in various machine learning tasks like classification and regression.

Key Characteristics:

- **Unidirectional Information Flow:** Data enters the network through the input layer, passes through one or more hidden layers (optional), and finally reaches the output layer.
- **No Feedback Loops:** Unlike recurrent neural networks, FNNs do not have connections that loop back, meaning information only moves forward through the network.

- **Supervised Learning:** FNNs are commonly used for supervised learning tasks, where the network learns from labeled data to map inputs to outputs.

Structure of a Feedforward Neural Network

Feedforward Neural Networks have a structured layered design where data flows sequentially through each layer.

1. **Input Layer**: The input layer consists of neurons that receive the input data. Each neuron in the input layer represents a feature of the input data.

2. **Hidden Layers**: One or more hidden layers are placed between the input and output layers. These layers are responsible for learning the complex patterns in the data. Each neuron in a hidden layer applies a weighted sum of inputs followed by a non-linear activation function.

3. **Output Layer**: The output layer provides the final output of the network. The number of neurons in this layer corresponds to the number of classes in a classification problem or the number of outputs in a regression problem.

Each connection between neurons in these layers has an associated weight that is adjusted during the training process to minimize the error in predictions.

Components:

- **Input Layer:** Receives the initial data.
- **Hidden Layers:** Perform intermediate computations and feature extraction.
- **Output Layer:** Produces the final prediction or result.
- **Neurons:** The basic processing units within each layer, performing calculations based on inputs and weights.
- **Weights:** Represent the strength of connections between neurons, learned during training.
- **Activation Functions:** Introduce non-linearity, allowing the network to learn complex patterns.

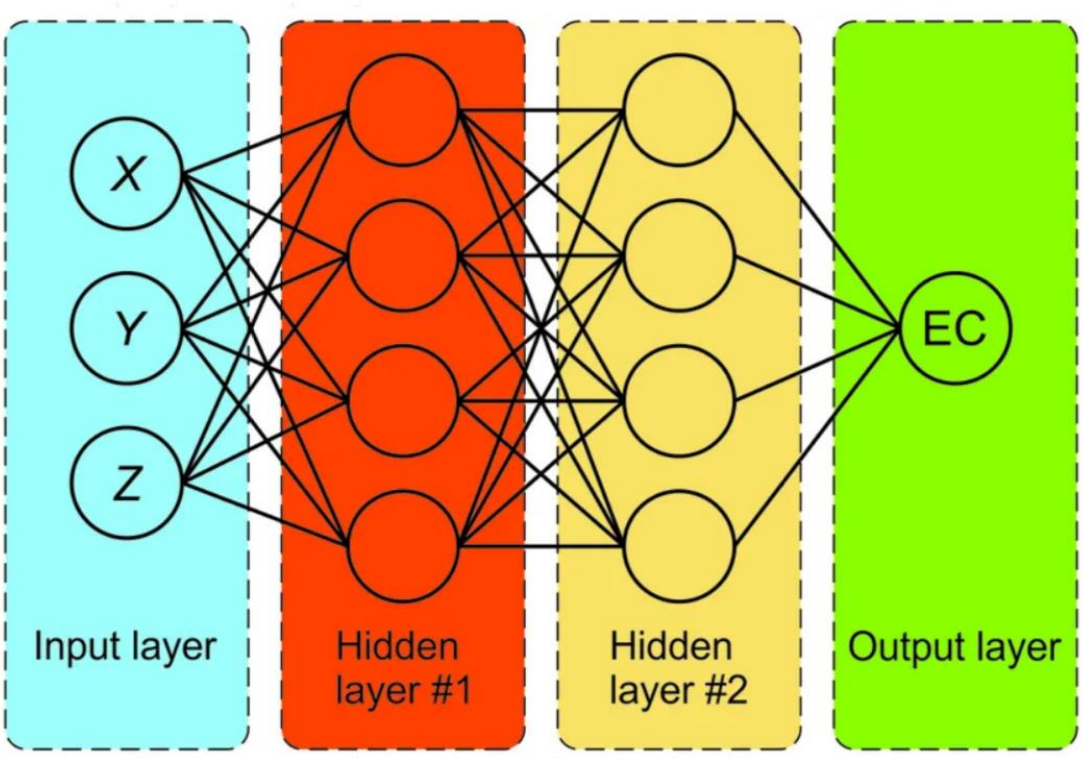
How it Works:

I. **Input:** Data is fed into the input layer.

II. **Forward Propagation:** Information flows through the network, with each neuron performing a weighted sum of its inputs, adding a bias, and applying an activation function.

III. **Output:** The output layer produces the network's prediction.

IV. **Backpropagation (for training):** The error between the predicted output and the actual output is calculated, and the weights are adjusted to minimize this error.



Activation Functions

Activation functions introduce non-linearity into the network enabling it to learn and model complex data patterns.

Common activation functions include:

- **Sigmoid**: $\sigma(x) = \frac{1}{1+e^{-x}}$
- **Tanh**: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- **ReLU:** $\mathrm{ReLU}(x) = \max(0, x)$

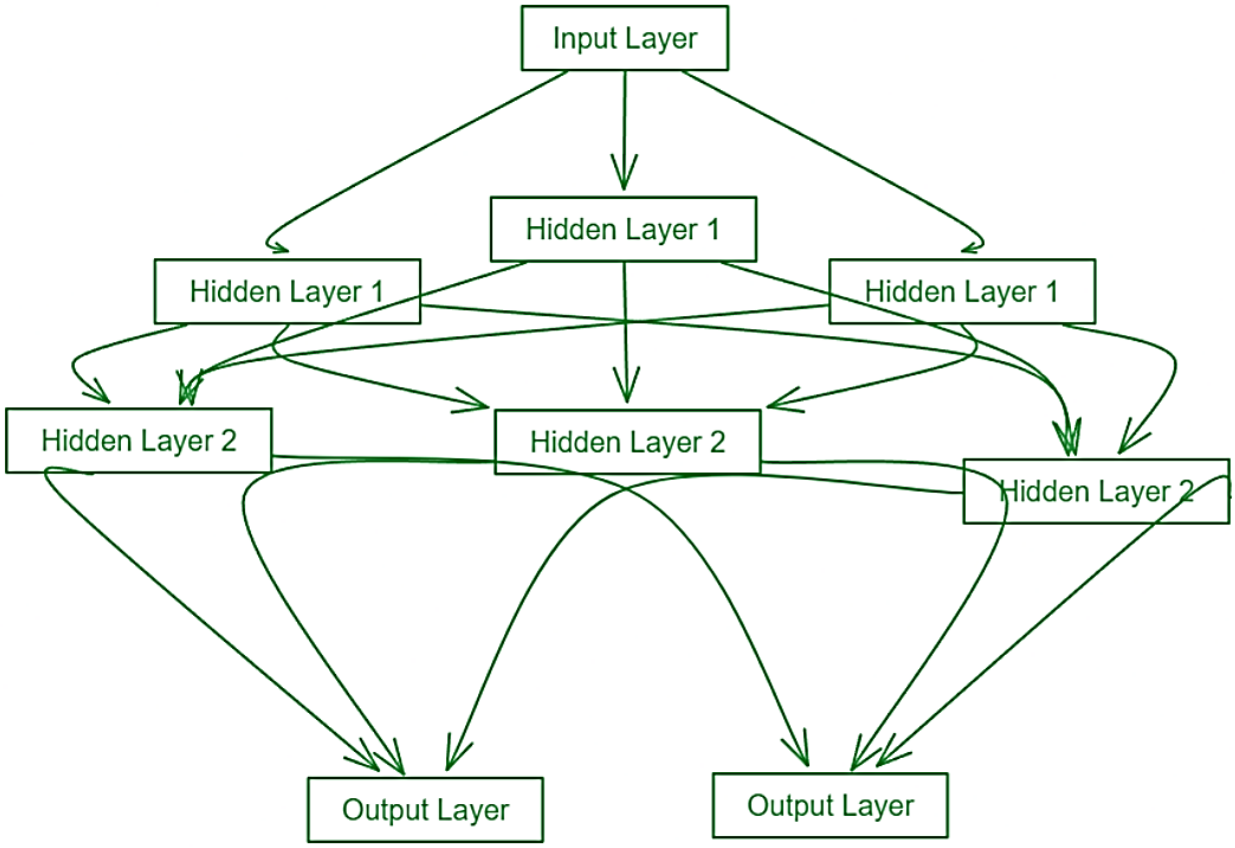
Applications:

- **Image Classification:** Identifying objects or patterns in images.
- **Regression:** Predicting continuous values, like stock prices or temperatures.
- **Pattern Recognition:** Identifying relationships and trends in data.

Training a Feedforward Neural Network

Training a Feedforward Neural Network involves adjusting the weights of the neurons to minimize the error between the predicted output and the actual output. This process is typically performed using backpropagation and gradient descent.

1. Forward Propagation: During forward propagation the input data passes through the network and the output is calculated.
2. Loss Calculation: The loss (or error) is calculated using a loss function such as Mean Squared Error (MSE) for regression tasks or Cross-Entropy Loss for classification tasks.
3. Backpropagation: In backpropagation the error is propagated back through the network to update the weights. The gradient of the loss function with respect to each weight is calculated and the weights are adjusted using gradient descent.



Evaluation of Feedforward neural network

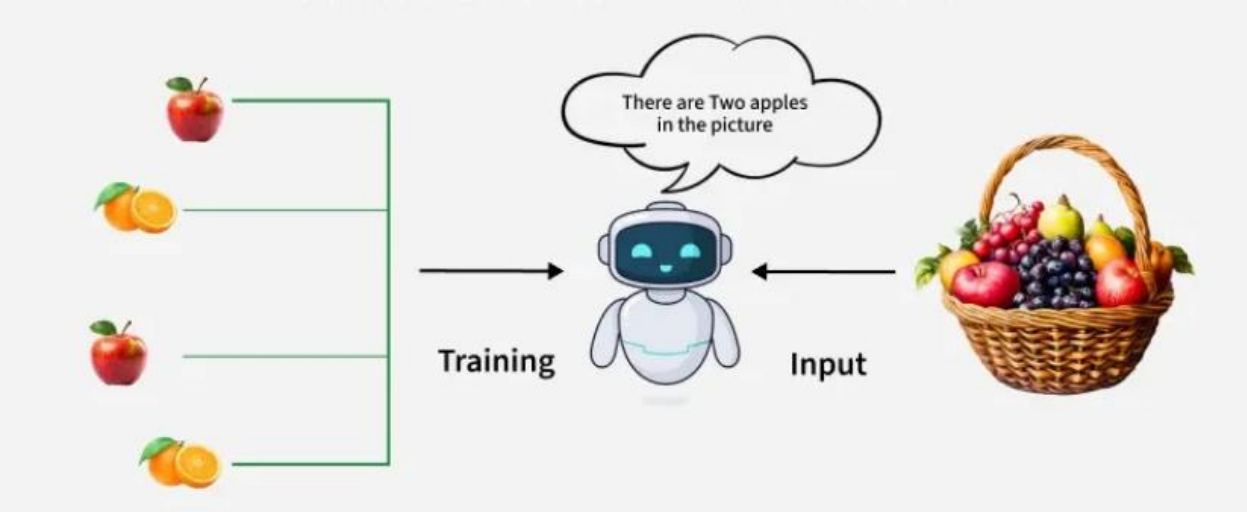
Evaluating the performance of the trained model involves several metrics:

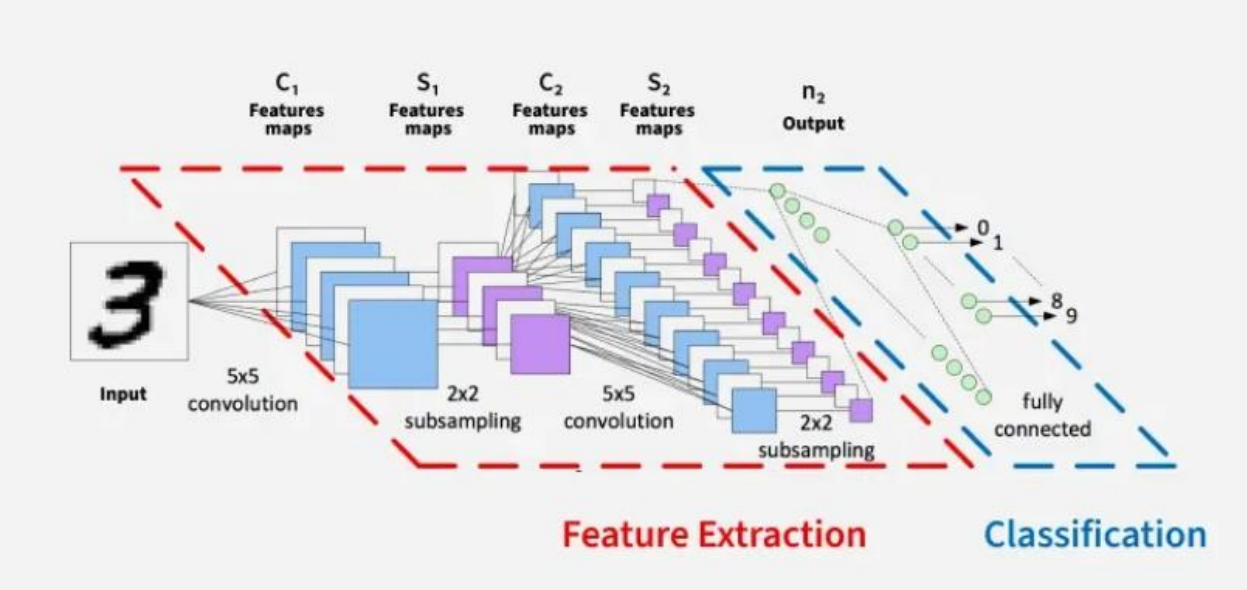
- Accuracy: The proportion of correctly classified instances out of the total instances.
- Precision: The ratio of true positive predictions to the total predicted positives.
- Recall: The ratio of true positive predictions to the actual positives.

- F1 Score: The harmonic means of precision and recall, providing a balance between the two.
- Confusion Matrix: A table used to describe the performance of a classification model, showing the true positives, true negatives, false positives, and false negatives.

4.1.1.2. Convolution

Convolutional Neural Networks (CNNs) are deep learning models designed to process data with a grid-like topology such as images. They are the foundation for most modern computer vision applications to detect features within visual data.





Key Components of a Convolutional Neural Network

1. **Convolutional Layers:** These layers apply convolutional operations to input images using filters or kernels to detect features such as edges, textures and more complex patterns. Convolutional operations help preserve the spatial relationships between pixels.

2. **Pooling Layers:** They down sample the spatial dimensions of the input, reducing the computational complexity and the number of parameters in the network. Max pooling is a common pooling operation where we select a maximum value from a group of neighboring pixels.

3. **Activation Functions:** They introduce non-linearity to the model by allowing it to learn more complex relationships in the data.

4. **Fully Connected Layers:** These layers are responsible for making predictions based on the high-level features learned by the previous layers. They connect every neuron in one layer to every neuron in the next layer.

How CNNs Work?

1. **Input Image**: CNN receives an input image which is preprocessed to ensure uniformity in size and format.

2. **Convolutional Layers**: Filters are applied to the input image to extract features like edges, textures and shapes.

3. **Pooling Layers**: The feature maps generated by the convolutional layers are down sampled to reduce dimensionality.

4. **Fully Connected Layers**: The down sampled feature maps are passed through fully connected layers to produce the final output, such as a classification label.

5. **Output**: The CNN outputs a prediction, such as the class of the image.

How to Train a Convolutional Neural Network?

CNNs are trained using a supervised learning approach. This means that the CNN is given a set of labeled training images. The CNN learns to map the input images to their correct labels.

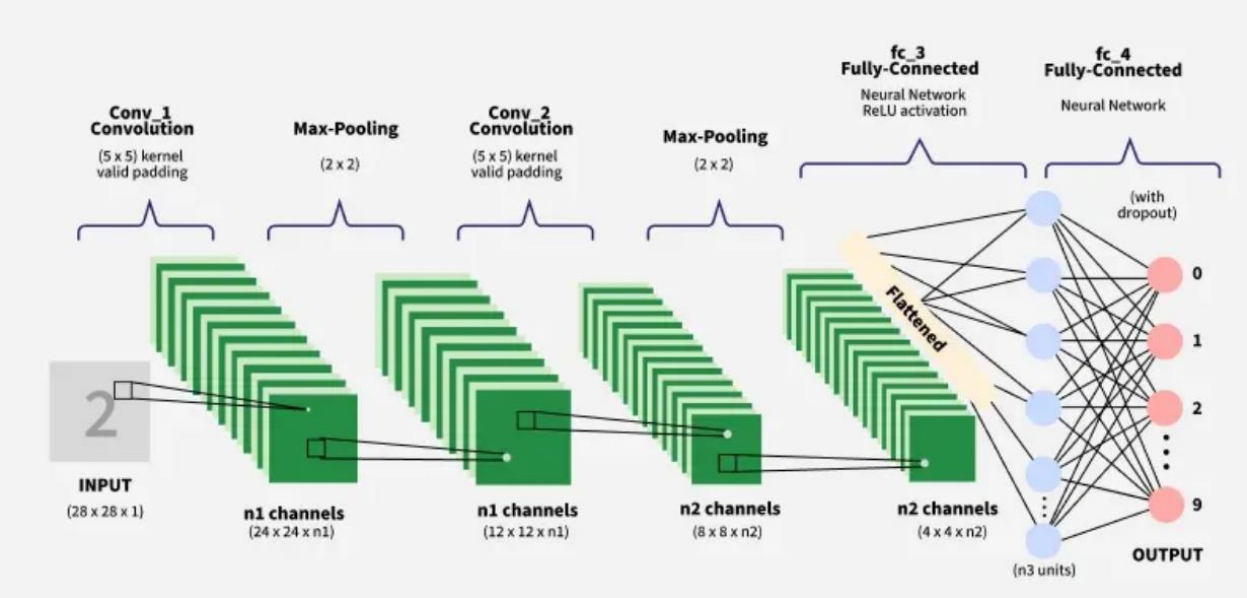
The training process for a CNN involves the following steps:

1. **Data Preparation:** The training images are preprocessed to ensure that they are all in the same format and size.

2. **Loss Function:** A loss function is used to measure how well the CNN is performing on the training data. The loss function is typically calculated by taking the difference between the predicted labels and the actual labels of the training images.

3. **Optimizer:** An optimizer is used to update the weights of the CNN in order to minimize the loss function.

4. **Backpropagation:** Backpropagation is a technique used to calculate the gradients of the loss function with respect to the weights of the CNN. The gradients are then used to update the weights of the CNN using the optimizer.



Different Types of CNN Models

1. LeNet

LeNet developed by Yann LeCun and his colleagues in the late 1990s was one of the first successful CNNs designed for handwritten digit recognition. It laid the foundation for modern CNNs and achieved high accuracy on the MNIST dataset which contains 70,000 images of handwritten digits (0-9).

2. AlexNet

AlexNet is a CNN architecture that was developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton in 2012. It was the first CNN to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) a major image recognition competition. It consists of several layers of convolutional and pooling layers followed by fully connected layers. The architecture includes five convolutional layers, three pooling layers and three fully connected layers.

3. Resnet

ResNets (Residual Networks) are designed for image recognition and processing tasks. They are renowned for their ability to train very deep networks without overfitting making them highly effective for complex tasks. It introduces skip connections that allow the network to learn residual functions making it easier to train deep architecture.

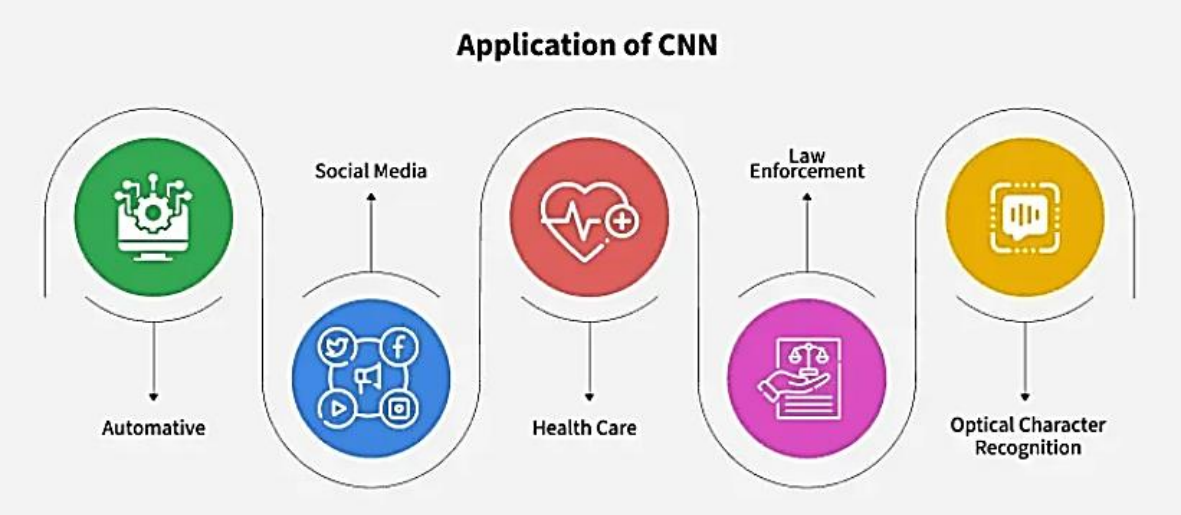
4. GoogleNet

GoogleNet also known as InceptionNet is renowned for achieving high accuracy in image classification while using fewer parameters and computational resources compared to other state-of-the-art CNNs. The core component of GoogleNet allows the network to learn features at different scales simultaneously to enhance performance.

5. VGG

VGGs are developed by the Visual Geometry Group at Oxford, it uses small 3x3 convolutional filters stacked in multiple layers, creating a deep and uniform structure. Popular variants like VGG-16 and VGG-19 achieved state-of-the-art performance on the ImageNet dataset demonstrating the power of depth in CNNs.

Applications of CNN



- **Image classification:** CNNs are the state-of-the-art models for image classification. They can be used to classify images into different categories such as cats and dogs.
- **Object detection:** It can be used to detect objects in images such as people, cars and buildings. They can also be used to localize objects in images which means that they can identify the location of an object in an image.

- **Image segmentation:** It can be used to segment images which means that they can identify and label different objects in an image. This is useful for applications such as medical imaging and robotics.
- **Video analysis:** It can be used to analyze videos such as tracking objects in a video or detecting events in a video. This is useful for applications such as video surveillance and traffic monitoring.

Advantages of CNN

- **High Accuracy**: They can achieve high accuracy in various image recognition tasks.
- **Efficiency**: They are efficient, especially when implemented on GPUs.
- **Robustness**: They are robust to noise and variations in input data.
- **Adaptability**: It can be adapted to different tasks by modifying their architecture.

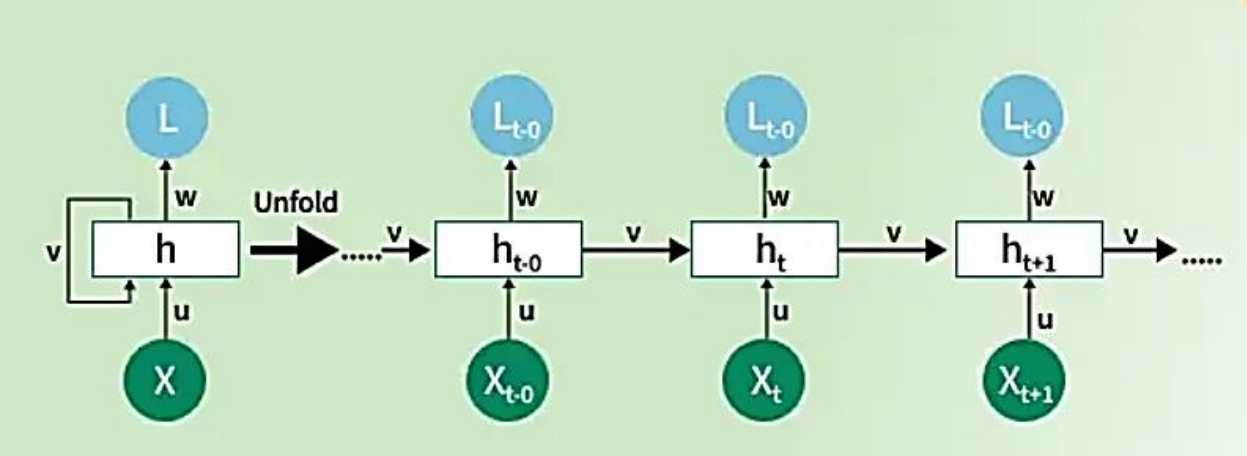
Disadvantages of CNN

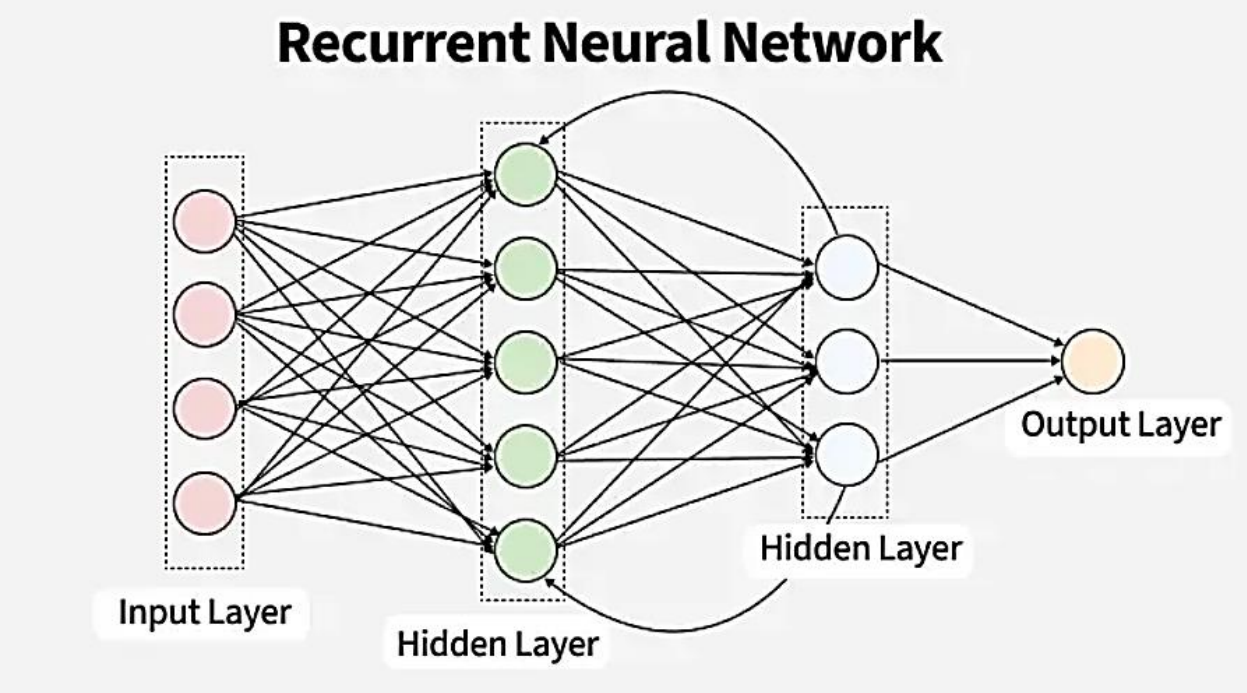
- **Complexity**: It can be complex and difficult to train, especially for large datasets.
- **Resource-Intensive**: It require significant computational resources for training and deployment.
- **Data Requirements**: They need large amounts of labeled data for training.
- **Interpretability**: They can be difficult to interpret making it challenging to understand their predictions.

4.1.1.3. Recurrent

Recurrent Neural Networks (RNNs) differ from regular neural networks in how they process information. While standard neural networks pass information in one direction i.e from input to output, RNNs feed information back into the network at each step.

Imagine reading a sentence and you try to predict the next word; you don't rely only on the current word but also remember the words that came before. RNNs work similarly by "remembering" past information and passing the output from one step as input to the next i.e. it considers all the earlier words to choose the most likely next word. This memory of previous steps helps the network understand context and make better predictions.

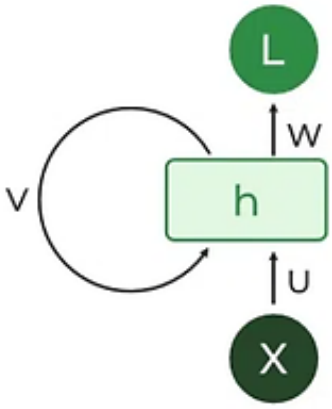

L
w
Unfold
v
h
u
X
L_{t-0}
h_{t-0}
X_{t-0}
h_t
X_t
h_{t+1}
X_{t+1}
.....


Recurrent Neural Network
Input Layer
Hidden Layer
Hidden Layer
Output Layer

Key Components of RNNs

There are mainly two components of RNNs that we will discuss.

1. Recurrent Neurons

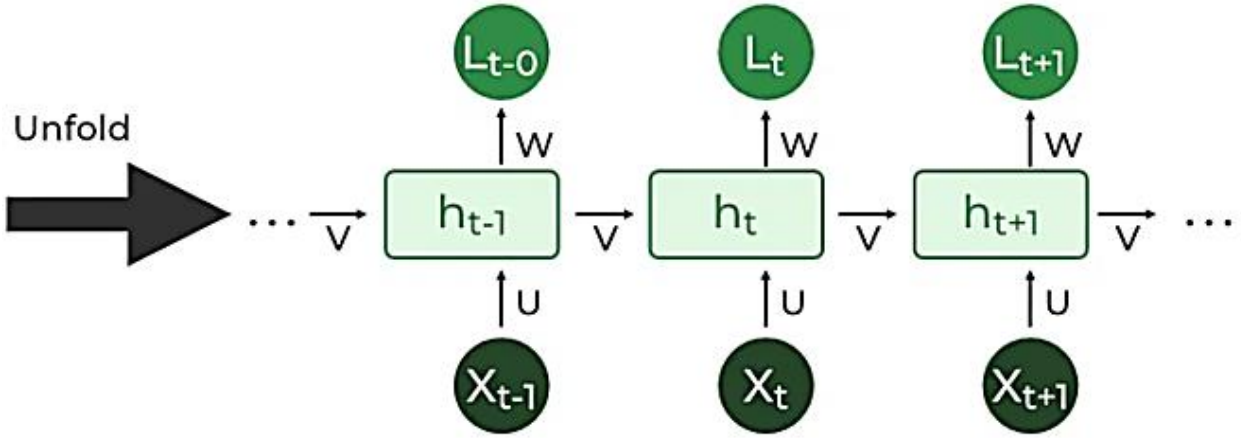


The fundamental processing unit in RNN is a **Recurrent Unit.** They hold a hidden state that maintains information about previous inputs in a sequence. Recurrent units can "remember" information from prior steps by feeding back their hidden state, allowing them to capture dependencies across time.

2. RNN Unfolding

RNN unfolding or unrolling is the process of expanding the recurrent structure over time steps. During unfolding each step of the sequence is represented as a separate layer in a series illustrating how information flows across each time step.

This unrolling enables **backpropagation through time (BPTT)** a learning process where errors are propagated across time steps to adjust the network's weights enhancing the RNN's ability to learn dependencies within sequential data.



Recurrent Neural Network Architecture

RNNs share similarities in input and output structures with other deep learning architectures but differ significantly in how information flows from input to output. Unlike traditional deep neural networks where each dense layer has distinct weight matrices. RNNs use shared weights across time steps, allowing them to remember information over sequences.

In RNNs the hidden state H_i is calculated for every input X_i to retain sequential dependencies. The computations follow these core formulas:

1. Hidden State Calculation:

$$h = \sigma(U \cdot X + W \cdot h_{t-1} + B)$$

Here:

- ***h*** represents the current hidden state.
- ***U*** and ***W*** are weight matrices.
- ***B*** is the bias.
-

2. Output Calculation:

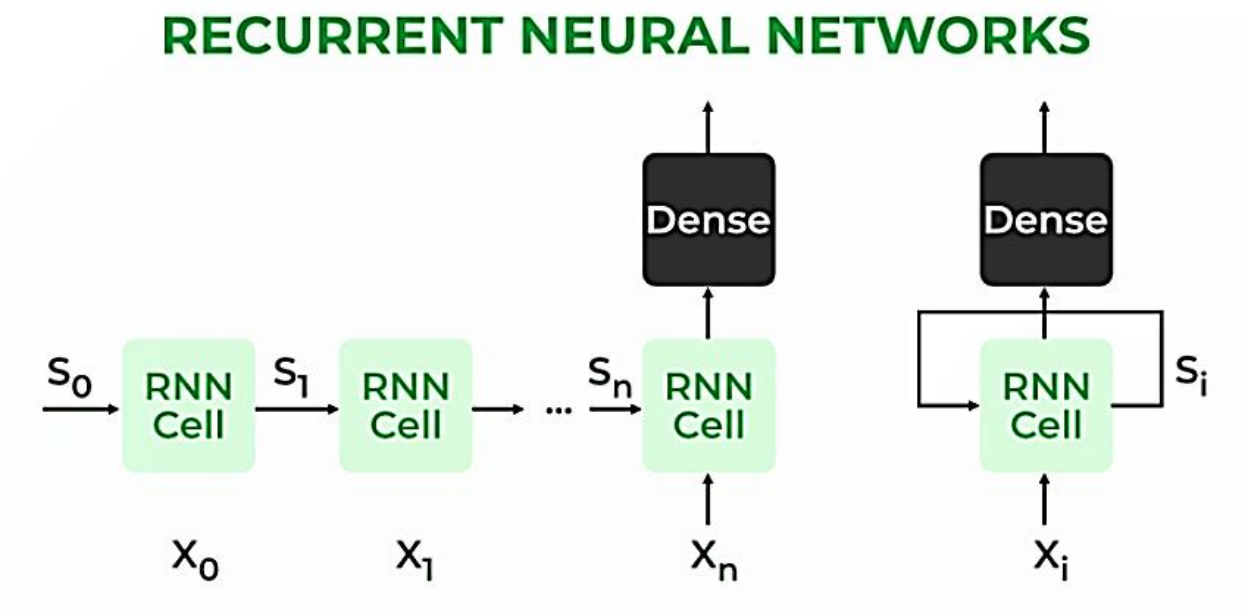
$$Y = O(V \cdot h + C)$$

The output ***Y*** is calculated by applying ***O*** an activation function to the weighted hidden state where ***V*** and ***C*** represent weights and bias.

3. Overall Function:

$$Y = f(X, h, W, U, V, B, C)$$

This function defines the entire RNN operation where the state matrix ***S*** holds each element s_i representing the network's state at each time step ***i***.



How does RNN work?

At each time step RNNs process units with a fixed activation function. These units have an internal hidden state that acts as memory that retains information from previous time steps. This memory allows the network to store past knowledge and adapt based on new inputs.

Updating the Hidden State in RNNs

The current hidden state h_t depends on the previous state h_{t-1} and the current input x_t and is calculated using the following relations:

1. State Update:

$$h_t = f(h_{t-1}, x_t)$$

where:

- h_t is the current state
- h_{t-1} is the previous state
- x_t is the input at the current time step

2. Activation Function Application:

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$$

Here, W_{hh} is the weight matrix for the recurrent neuron and W_{xh} is the weight matrix for the input neuron.

3. Output Calculation:

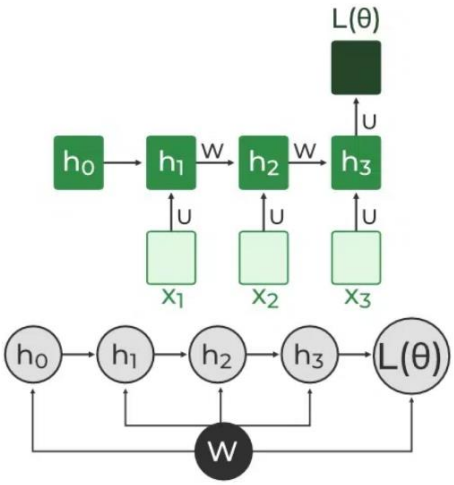
$$y_t = W_{hy} \cdot h_t$$

where y_t is the output and W_{hy} is the weight at the output layer.

These parameters are updated using backpropagation. However, since RNN works on sequential data here we use an updated backpropagation which is known as **backpropagation through time**.

Backpropagation Through Time (BPTT) in RNNs

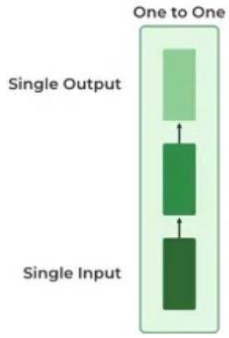
Since RNNs process sequential data **Backpropagation Through Time (BPTT)** is used to update the network's parameters. The loss function L(θ) depends on the final hidden state $h3$ and each hidden state relies on preceding ones forming a sequential dependency chain:
$h3$ depends on depends on $h2$ depends on $h1,\ldots,h1$ depends on $h0$.



In BPTT, gradients are backpropagated through each time step. This is essential for updating network parameters based on temporal dependencies.

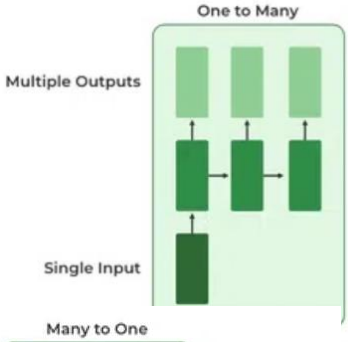
1. One-to-One RNN

This is the simplest type of neural network architecture where there is a single input and a single output. It is used for straightforward classification tasks such as binary classification where no sequential data is involved.



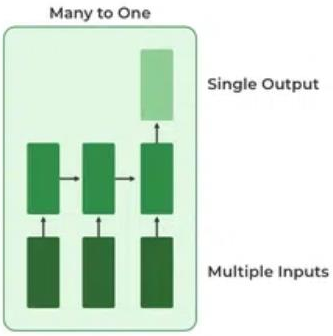
2. One-to-Many RNN

In a One-to-Many RNN the network processes a single input to produce multiple outputs over time. This is useful in tasks where one input triggers a sequence of predictions (outputs). For example, in image captioning a single image can be used as input to generate a sequence of words as a caption.

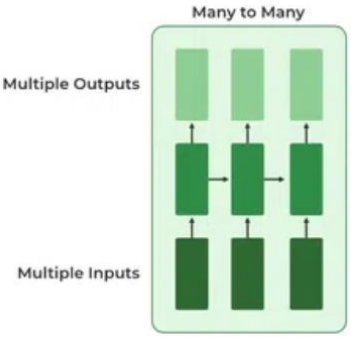


3. Many-to-One RNN

The **Many-to-One RNN** receives a sequence of inputs and generates a single output. This type is useful when the overall context of the input sequence is needed to make one prediction. In sentiment analysis the model receives a sequence of words (like a sentence) and produces a single output like positive, negative or neutral.



4. Many-to-Many RNN

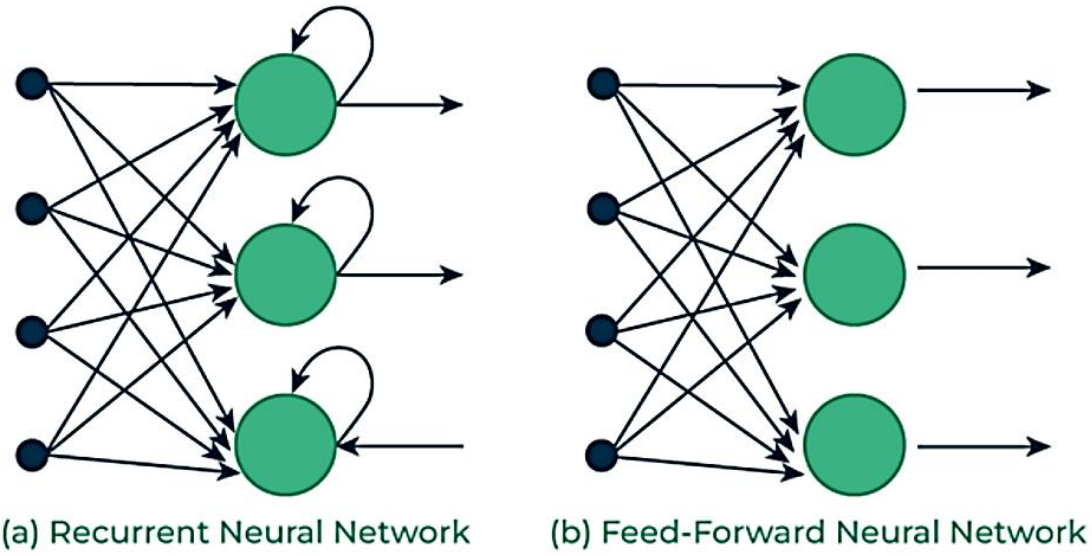


The **Many-to-Many RNN** type processes a sequence of inputs and generates a sequence of outputs. In language translation task a sequence of words in one language is given as input and a corresponding sequence in another language is generated as output.

How RNN Differs from Feedforward Neural Networks?

Feedforward Neural Networks (FNNs) process data in one direction from input to output without retaining information from previous inputs. This makes them suitable for tasks with independent inputs like image classification. However, FNNs struggle with sequential data since they lack memory.

Recurrent Neural Networks (RNNs) solve this **by incorporating loops that allow information from previous steps to be fed back into the network**. This feedback enables RNNs to remember prior inputs making them ideal for tasks where context is important.



Advantages of Recurrent Neural Networks

- **Sequential Memory**: RNNs retain information from previous inputs making them ideal for time-series predictions where past data is crucial.
- **Enhanced Pixel Neighborhoods**: RNNs can be combined with convolutional layers to capture extended pixel neighborhoods improving performance in image and video data processing.

Limitations of Recurrent Neural Networks (RNNs)

While RNNs excel at handling sequential data they face two main training challenges i.e., vanishing gradient and exploding gradient problem:

- **Vanishing Gradient**: During backpropagation gradients diminish as they pass through each time step leading to minimal weight updates. This limits the RNN's ability to learn long-term dependencies which is crucial for tasks like language translation.
- **Exploding Gradient**: Sometimes gradients grow uncontrollably causing excessively large weight updates that de-stabilize training.

These challenges can hinder the performance of standard RNNs on complex, long-sequence tasks.

Applications of Recurrent Neural Networks

RNNs are used in various applications where data is sequential or time-based:

- **Time-Series Prediction**: RNNs excel in forecasting tasks, such as stock market predictions and weather forecasting.
- **Natural Language Processing (NLP)**: RNNs are fundamental in NLP tasks like language modeling, sentiment analysis and machine translation.
- **Speech Recognition**: RNNs capture temporal patterns in speech data, aiding in speech-to-text and other audio-related applications.
- **Image and Video Processing**: When combined with convolutional layers, RNNs help analyze video sequences, facial expressions and gesture recognition.

4.1.2. Perceptions

Perceptron is a type of neural network that performs binary classification that maps input features to an output decision, usually classifying data into one of two categories, such as 0 or 1.

Perceptron consists of a single layer of input nodes that are fully connected to a layer of output nodes. It is particularly good at learning **linearly separable patterns**. It utilizes a variation of artificial neurons called **Threshold Logic Units (TLU)**, which were first introduced by McCulloch and Walter Pitts in the 1940s. This foundational model has played a crucial role in the development of more advanced neural networks and machine learning algorithms.

Types of Perceptron

1. **Single-Layer Perceptron** is a type of perceptron is limited to learning linearly separable patterns. It is effective for tasks where the data can be divided into distinct categories through a straight line. While powerful in its simplicity, it struggles with more complex problems where the relationship between inputs and outputs is non-linear.

2. **Multi-Layer Perceptron** possess enhanced processing capabilities as they consist of two or more layers, adept at handling more complex patterns and relationships within the data.

Basic Components of Perceptron

A Perceptron is composed of key components that work together to process information and make predictions.

- **Input Features:** The perceptron takes multiple input features, each representing a characteristic of the input data.
- **Weights:** Each input feature is assigned a weight that determines its influence on the output. These weights are adjusted during training to find the optimal values.
- **Summation Function:** The perceptron calculates the weighted sum of its inputs, combining them with their respective weights.
- **Activation Function:** The weighted sum is passed through the **Heaviside step function**, comparing it to a threshold to produce a binary output (0 or 1).
- **Output:** The final output is determined by the activation function, often used for **binary classification** tasks.
- **Bias:** The bias term helps the perceptron make adjustments independent of the input, improving its flexibility in learning.
- **Learning Algorithm:** The perceptron adjusts its weights and bias using a learning algorithm, such as the Perceptron Learning Rule, to minimize prediction errors.

These components enable the perceptron to learn from data and make predictions. While a single perceptron can handle simple binary classification, complex tasks require multiple perceptron's organized into layers, forming a neural network.

How does Perceptron work?

A weight is assigned to each input node of a perceptron, indicating the importance of that input in determining the output. The Perceptron's output is calculated as a weighted sum of the inputs, which is then passed through an activation function to decide whether the Perceptron will fire.

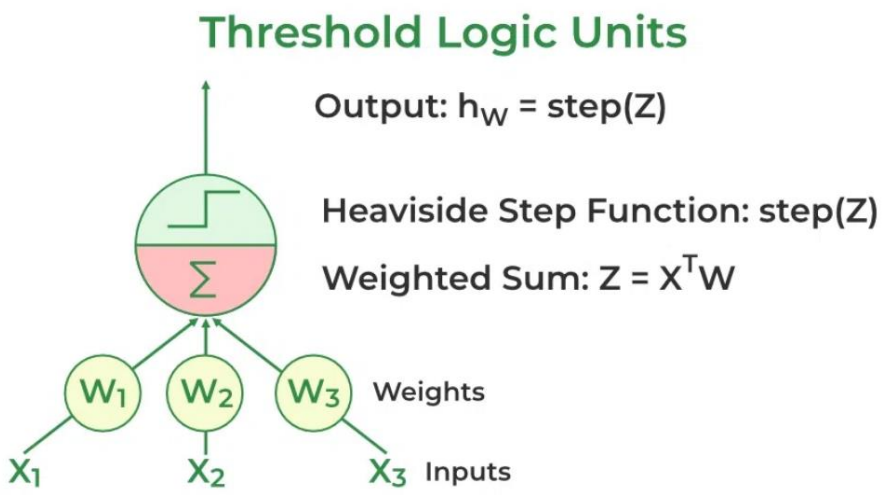
The weighted sum is computed as:

$$z = w_1x_1 + w_2x_2 + \ldots + w_nx_n = X^TW$$

The step function compares this weighted sum to a threshold. If the input is larger than the threshold value, the output is 1; otherwise, it's 0. This is the most common activation function used in Perceptron's are represented by the Heaviside step function:

$$h(z) = \begin{cases} 0 & \text{if } z < \text{Threshold} \\ 1 & \text{if } z \geq \text{Threshold} \end{cases}$$

A perceptron consists of a single layer of Threshold Logic Units (TLU), with each TLU fully connected to all input nodes.



In a fully connected layer, also known as a dense layer, all neurons in one layer are connected to every neuron in the previous layer.

The output of the fully connected layer is computed as:

$$f_{W,b}(X) = h(XW + b)$$

where *X* is the input *W* is the weight for each inputs neurons and *b* is the bias and *h* is the step function.

During training, the Perceptron's weights are adjusted to minimize the difference between the predicted output and the actual output. This is achieved using supervised learning algorithms like the delta rule or the Perceptron learning rule.

The weight update formula is:

$$w_{i,j} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$$

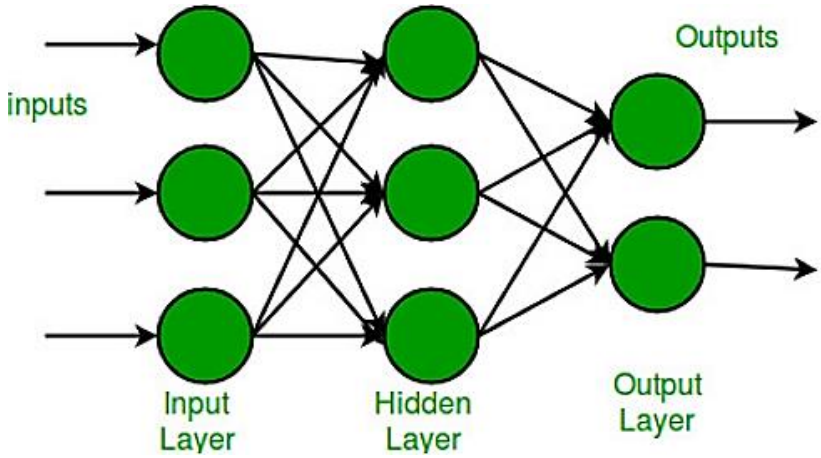
Where:

- $w_{i,j}$ is the weight between the i^{th} input and j^{th} output neuron,
- x_i is the i^{th} input value,
- y_j is the actual value, and $\hat{y}_j$ is the predicted value,
- η is the **learning rate**, controlling how much the weights are adjusted.

4.1.2.1. Single layer perceptron

Single Layer Perceptron is inspired by biological neurons and their ability to process information. To understand the SLP we first need to break down the workings of a single artificial neuron which is the fundamental building block of neural networks. An **artificial neuron** is a simplified computational model that mimics the behavior of a biological neuron. It takes inputs, processes them and produces an output. Here's how it works step by step:

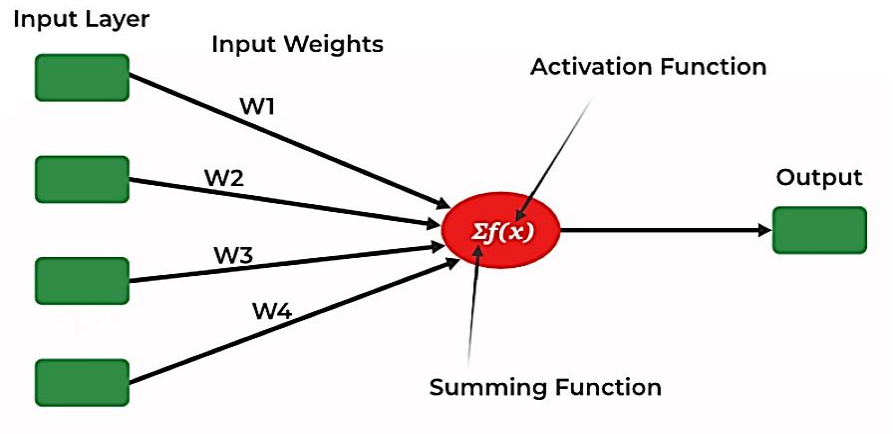
- Receive signal from outside.
- Process the signal and decide whether we need to send information or not.
- Communicate the signal to the target cell, which can be another neuron or gland.



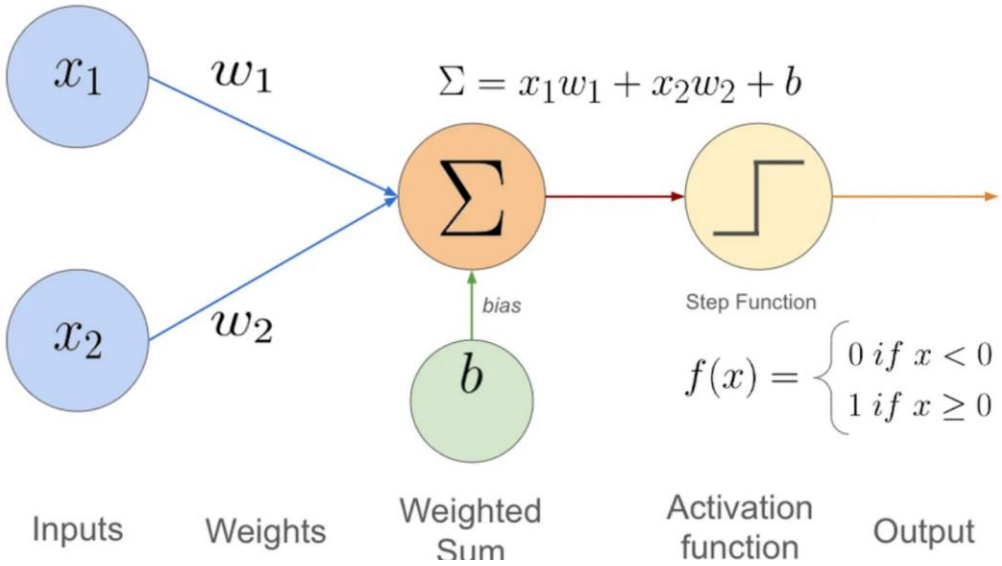
Perceptron is mainly used to compute the logical gate like **AND, OR and NOR** which has binary input and binary output.

The main functionality of the perceptron is: -

- Takes input from the input layer
- Weight them up and sum it up.
- Pass the sum to the nonlinear function to produce the output.



Here activation functions can be anything like **sigmoid, tanh, relu** based on the requirement we will be choosing the most appropriate nonlinear activation function to produce the better result. Now let us implement a single-layer perceptron.



Numerical:

Draw and train a single layer perceptron to learn the 2-input logical AND function. Assume learning rate =0.1. Initialize weights and bias with 0.5. Calculate for 2 passes/epochs.

- **Inputs:** $x_1, x_2 \in \{0, 1\}$
- **Output:** AND function
  - $y = 1$ only if $x_1 = 1$ and $x_2 = 1$; otherwise, $y = 0$
- **Learning Rate:** $\eta = 0.1$
- **Initial Weights:**
  - $w_1 = 0.5$, $w_2 = 0.5$, bias $b = 0.5$
- **Activation Function:**

$$f(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **Epochs:** 2 (i.e., go over the dataset 2 times)

Training Data: AND truth table

Input (x_1)	Input (x_2)	Desired Output (d)
0	0	0
0	1	0
1	0	0
1	1	1

Perceptron Update Rule

For each training sample:

1. Compute: $y = f(x_1 \cdot w_1 + x_2 \cdot w_2 + b)$
2. Update if $y \neq t$:

$$w_i \leftarrow w_i + \eta \cdot (t - y) \cdot x_i$$
$$b \leftarrow b + \eta \cdot (t - y)$$

Epoch 1:

Initial:

- $w_1 = 0.5, w_2 = 0.5, b = 0.5$

Sample 1: (0, 0), t = 0

$$z = 0 * 0.5 + 0 * 0.5 + 0.5 = 0.5 \Rightarrow y = 1 \quad \text{(wrong)}$$

Update:

- $\Delta = 0.1 \cdot (0 - 1) = -0.1$
- $w_1 = 0.5 + (-0.1 \cdot 0) = 0.5$
- $w_2 = 0.5 + (-0.1 \cdot 0) = 0.5$
- $b = 0.5 + (-0.1) = 0.4$

Sample 2: (0, 1), t = 0

$$z = 0 * 0.5 + 1 * 0.5 + 0.4 = 0.9 \Rightarrow y = 1 \quad \text{(wrong)}$$

Update:

- $\Delta = -0.1$
- $w_1 = 0.5$
- $w_2 = 0.5 + (-0.1 \cdot 1) = 0.4$
- $b = 0.4 + (-0.1) = 0.3$

Sample 3: (1, 0), t = 0

$$z = 1 * 0.5 + 0 * 0.4 + 0.3 = 0.8 \Rightarrow y = 1 \quad \text{(wrong)}$$

Update:

- $\Delta = -0.1$
- $w_1 = 0.5 + (-0.1 \cdot 1) = 0.4$
- $w_2 = 0.4$
- $b = 0.3 + (-0.1) = 0.2$

Result: No update

After Epoch 1:

- $w_1 = 0.4, w_2 = 0.4, b = 0.2$

Epoch 2:

Sample 1: (0, 0), t = 0

$$z = 0 * 0.4 + 0 * 0.4 + 0.2 = 0.2 \Rightarrow y = 1 \quad \text{(wrong)}$$

Update:

- $\Delta = -0.1$
- $b = 0.2 - 0.1 = 0.1$

Sample 3: (1, 0), t = 0

$$z = 1 * 0.4 + 0 * 0.3 + 0 = 0.4 \Rightarrow y = 1 \quad \text{(wrong)}$$

Update:

- $w_1 = 0.4 - 0.1 = 0.3$
- $b = 0 - 0.1 = -0.1$

Sample 4: (1, 1), t = 1

$$z = 1 * 0.3 + 1 * 0.3 - 0.1 = 0.5 \Rightarrow y = 1 \quad \text{(correct)}$$

Result: No update

Final Weights After 2 Epochs:

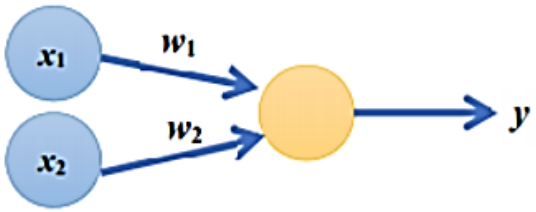
- $w_1 = 0.3$
- $w_2 = 0.3$
- $b = -0.1$

Final Perceptron Representation:

$$y = f(x_1 \cdot 0.3 + x_2 \cdot 0.3 - 0.1)$$

Numerical Problem:

Consider the single layer perceptron network shown in the figure below with inputs, $x_i \in \{0, 1\}$ and activation function $y = f(\sum_i x_i \cdot w_i + b)$ where $f(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{otherwise} \end{cases}$.



(a) Provide values for w_1, w_2, and b to use the perceptron for logical AND.
(b) Provide values for w_1, w_2, and b to use the perceptron for logical OR.
(c) Can this perceptron be used for logical XOR? If so, please provide the corresponding values of w_1, w_2, and b; otherwise, please build a two-layer perceptron network for XOR.

Hint: Truth tables

$y = x_1$ AND x_2

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$y = x_1$ OR x_2

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

$y = x_1$ XOR x_2

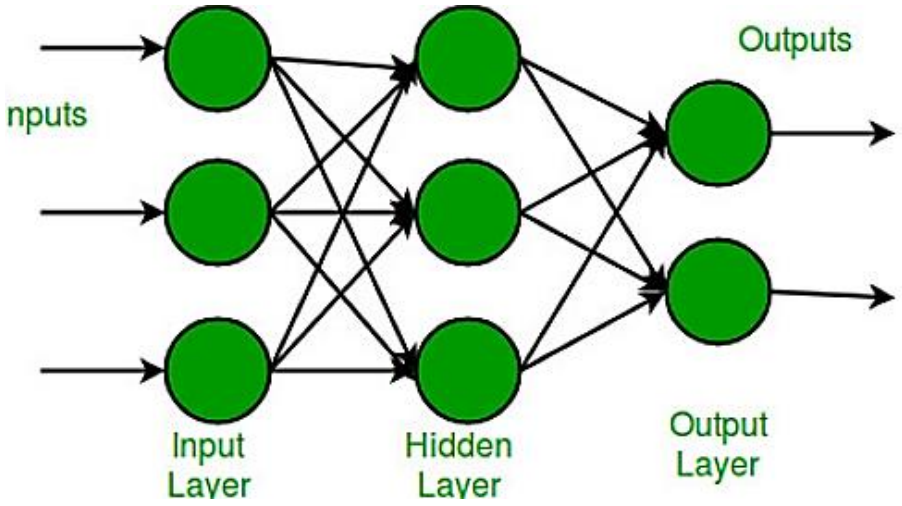
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

4.1.2.2. Multilayer Perceptron

Multi-Layer Perceptron (MLP) consists of fully connected dense layers that transform input data from one dimension to another. It is called **multi-layer** because it contains an input layer, one or more hidden layers and an output layer. The purpose of an MLP is to model complex relationships between inputs and outputs.

Components of Multi-Layer Perceptron (MLP)

- **Input Layer**: Each neuron or node in this layer corresponds to an input feature. For instance, if you have three input features the input layer will have three neurons.
- **Hidden Layers**: MLP can have any number of hidden layers with each layer containing any number of nodes. These layers process the information received from the input layer.
- **Output Layer**: The output layer generates the final prediction or result. If there are multiple outputs, the output layer will have a corresponding number of neurons.



Every connection in the diagram is a representation of the fully connected nature of an MLP. This means that every node in one layer connects to every node in the next layer. As the data moves through the network each layer transforms it until the final output is generated in the output layer.

Working of Multi-Layer Perceptron

Let's see working of the multi-layer perceptron. The key mechanisms such as forward propagation, loss function, backpropagation and optimization.

1. Forward Propagation

In **forward propagation** the data flows from the input layer to the output layer, passing through any hidden layers. Each neuron in the hidden layers processes the input as follows:

a. Weighted Sum: The neuron computes the weighted sum of the inputs:

$$z = \sum_i w_i x_i + b$$

Where:

- x_i is the input feature.
- w_i is the corresponding weight.
- b is the bias term.

b. Activation Function: The weighted sum z is passed through an activation function to introduce non-linearity. Common activation functions include:

- **Sigmoid**: $\sigma(z) = \frac{1}{1+e^{-z}}$
- **ReLU (Rectified Linear Unit)**: $f(z) = \max(0, z)$
- **Tanh (Hyperbolic Tangent)**: $\tanh(z) = \frac{2}{1+e^{-2z}} - 1$

2. Loss Function

Once the network generates an output the next step is to calculate the loss using a loss function. In supervised learning this compares the predicted output to the actual label.

For a classification problem the commonly used binary cross-entropy loss function is:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

Where:

- y_i is the actual label.
- $\hat{y}_i$ is the predicted label.
- N is the number of samples.

For regression problems the **mean squared error (MSE)** is often used:

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

3. Backpropagation

The goal of training an MLP is to minimize the loss function by adjusting the network's weights and biases. This is achieved through **backpropagation**:

1. **Gradient Calculation**: The gradients of the loss function with respect to each weight and bias are calculated using the chain rule of calculus.
2. **Error Propagation**: The error is propagated back through the network, layer by layer.
3. **Gradient Descent**: The network updates the weights and biases by moving in the opposite direction of the gradient to reduce the loss:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

Where:

- w is the weight.
- η is the learning rate.
- $\frac{\partial L}{\partial w}$ is the gradient of the loss function with respect to the weight.

4.1.2.3. Backpropagation

Back Propagation is also known as "Backward Propagation of Errors" is a method used to train neural network. Its goal is to reduce the difference between the model's predicted output and the actual output by adjusting the weights and biases in the network.

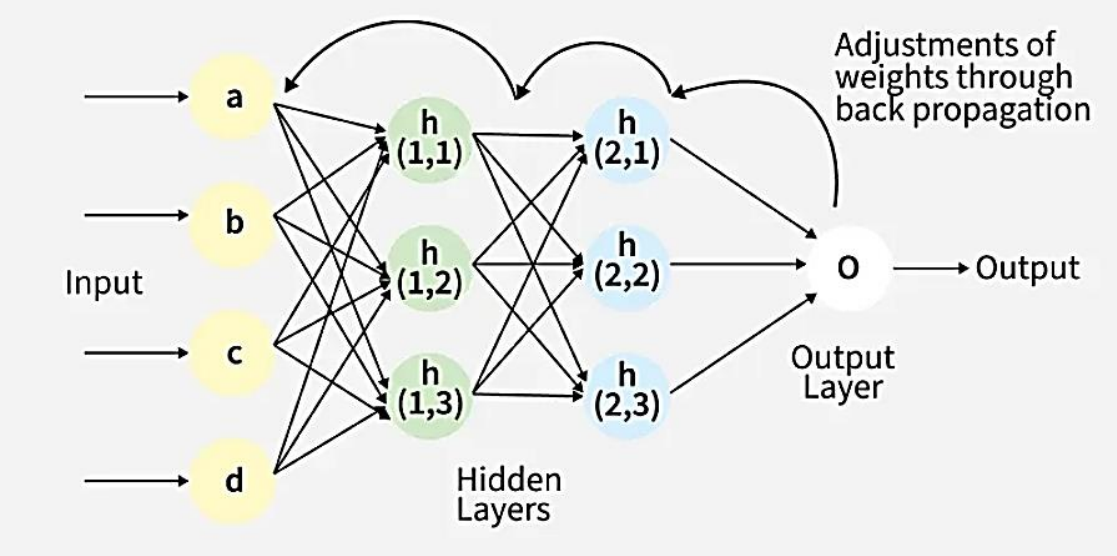
It works iteratively to adjust weights and bias to minimize the cost function. In each epoch the model adapts these parameters by reducing loss by following the error gradient. It often uses optimization algorithms like **gradient descent** or **stochastic gradient descent**. The algorithm computes the gradient using the chain rule from calculus allowing it to effectively navigate complex layers in the neural network to minimize the cost function.

Back Propagation plays a critical role in how neural networks improve over time. Here's why:

1. **Efficient Weight Update**: It computes the gradient of the loss function with respect to each weight using the chain rule making it possible to update weights efficiently.

2. **Scalability**: The Back Propagation algorithm scales well to networks with multiple layers and complex architectures making deep learning feasible.

3. **Automated Learning**: With Back Propagation the learning process becomes automated and the model can adjust itself to optimize its performance.



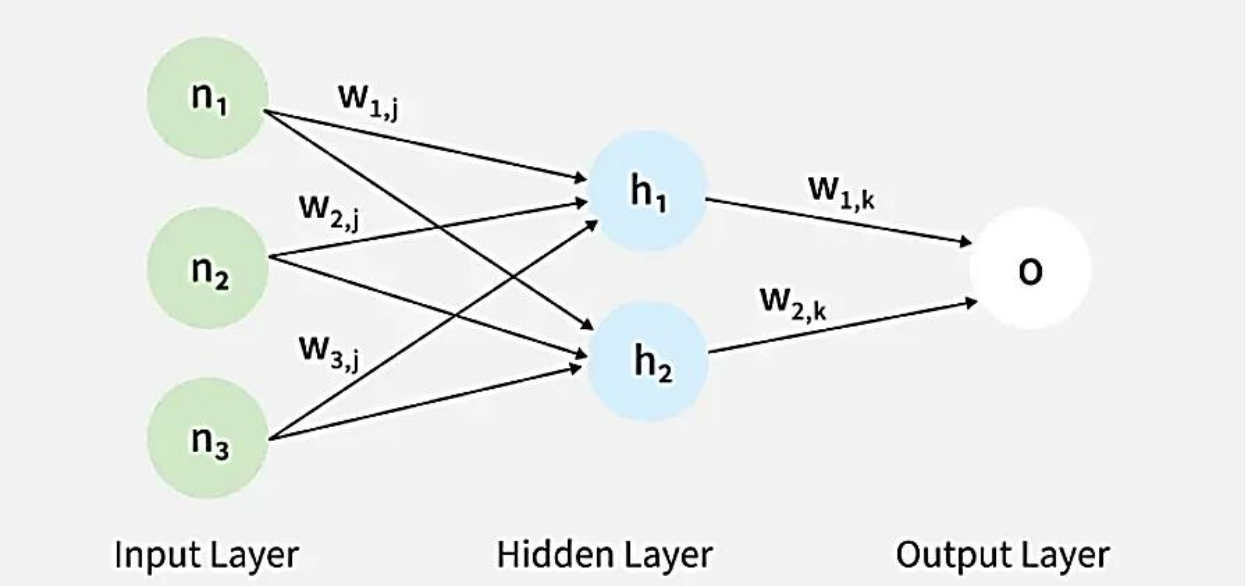
Working of Back Propagation Algorithm

The Back Propagation algorithm involves two main steps: the **Forward Pass** and the **Backward Pass**.

1. Forward Pass Work

In **forward pass** the input data is fed into the input layer. These inputs combined with their respective weights are passed to hidden layers. For example, in a network with two hidden layers (h1 and h2) the output from h1 serves as the input to h2. Before applying an activation function, a bias is added to the weighted inputs.

Each hidden layer computes the weighted sum (`a`) of the inputs then applies an activation function like **ReLU (Rectified Linear Unit)** to obtain the output (`o`). The output is passed to the next layer where an activation function such as **softmax** converts the weighted outputs into probabilities for classification.



2. Backward Pass

In the backward pass the error (the difference between the predicted and actual output) is propagated back through the network to adjust the weights and biases. One common method for error calculation is the **Mean Squared Error (MSE)** given by:

$$\text{MSE} = (\text{Predicted Output} - \text{Actual Output})^2$$

Once the error is calculated the network adjusts weights using **gradients** which are computed with the chain rule. These gradients indicate how much each weight and bias should be adjusted to minimize the error in the next iteration. The backward pass continues layer by layer ensuring that the network learns and improves its performance. The activation function through its derivative plays a crucial role in computing these gradients during Back Propagation.

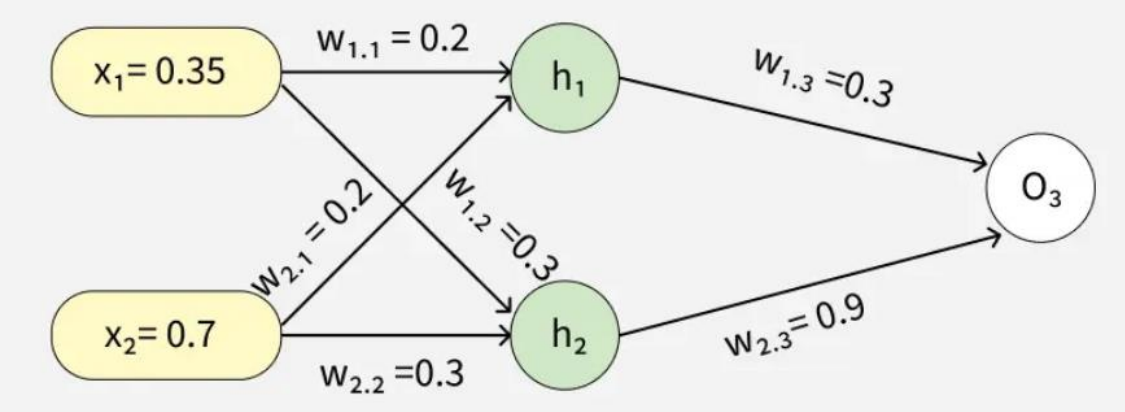
Difference between Multilayer Perceptron and Linear Regression

These are the differences between multi-layer perceptron's and linear regression-

Multi-layer Perceptron's	Linear Regression
Multi-layer perceptions are a network of neurons that can be used in binary/multiple class classification as well as regression problems.	A linear regression model determines a linear relationship between a dependent and independent variable.
The output function can be a linear or a continuous function. It need not be a straight line.	The output function is linear and can be represented in a straight line.
An MLP has multiple layers of neurons with an activation function and a threshold value.	A linear regression model has no activation function or threshold value.
An MLP usually has multiple inputs through its 1 or more input neurons.	Simple Linear regression requires only a single input- the value of the independent variable- to predict the value of the dependent variable.
MLPs are mostly used for supervised learning but has been used in unsupervised learning for clustering in rare examples like Kohonen Self Organizing Map.	Regression is a supervised learning technique.
Areas of Application of MLP include- Pattern Recognition, Autonomous Vehicle driving, Social Media Recommendations, etc.	Areas of application of Linear Regression include - prediction of housing prices, prediction of sales for a business, prediction of crop yield w.r.t. rainfall, etc.

Example of Back Propagation in Machine Learning

Let's walk through an example of Back Propagation in machine learning. Assume the neurons use the sigmoid activation function for the forward and backward pass. The target output is 0.5 and the learning rate is 1.



Forward Propagation

1. Initial Calculation

The weighted sum at each node is calculated using:

$$aj = \sum(wi, j * xi)$$

Where,

- a_j is the weighted sum of all the inputs and weights at each node
- $w_{i,j}$ represents the weights between the i^{th} input and the j^{th} neuron
- x_i represents the value of the i^{th} input

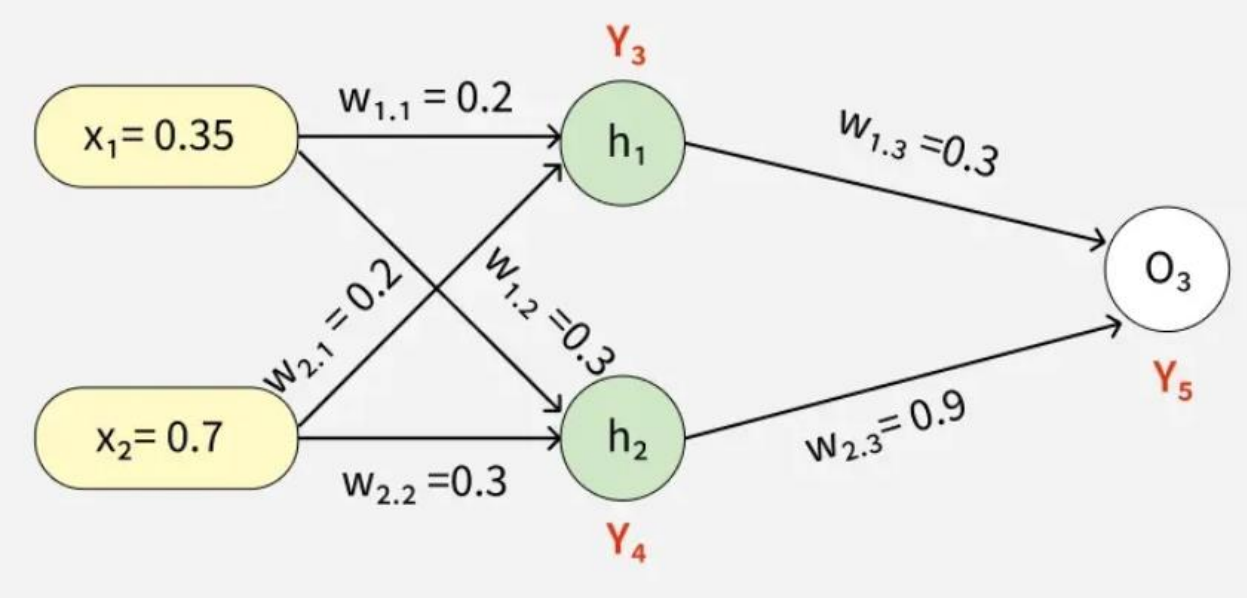
O (output): After applying the activation function to a, we get the output of the neuron:

$$O_j = activation\ function(a_j)$$

2. Sigmoid Function

The sigmoid function returns a value between 0 and 1, introducing non-linearity into the model.

$$y_j = \frac{1}{1+e^{-a_j}}$$



3. Computing Outputs

At h1 node,

$$\begin{aligned} a_1 &= (w_{1,1}x_1) + (w_{2,1}x_2) \\ &= (0.2 * 0.35) + (0.2 * 0.7) \\ &= 0.21 \end{aligned}$$

Once we calculated the a_1 value, we can now proceed to find the y_3 value:

$$y_j = F(a_j) = \frac{1}{1+e^{-a_1}}$$

$$y_3 = F(0.21) = \frac{1}{1+e^{-0.21}}$$

$$y_3 = 0.56$$

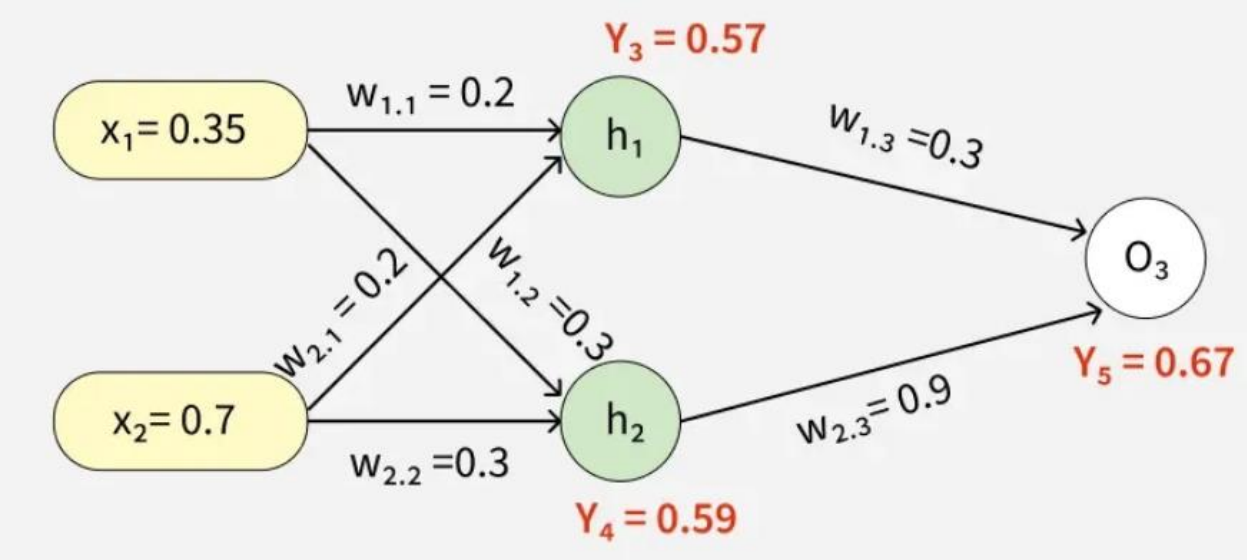
Similarly find the values of y_4 at $\mathbf{h_2}$ and y_5 at O_3

$$a_2 = (w_{1,2} * x_1) + (w_{2,2} * x_2) = (0.3 * 0.35) + (0.3 * 0.7) = 0.315$$

$$y_4 = F(0.315) = \frac{1}{1+e^{-0.315}}$$

$$a3 = (w_{1,3} * y_3) + (w_{2,3} * y_4) = (0.3 * 0.57) + (0.9 * 0.59) = 0.702$$

$$y_5 = F(0.702) = \frac{1}{1+e^{-0.702}} = 0.67$$



4. Error Calculation

Our actual output is 0.5 but we obtained 0.67. To calculate the error, we can use the below formula:

$$Error_j = y_{target} - y_5$$

$$\Rightarrow 0.5 - 0.67 = -0.17$$

Using this error value, we will be backpropagating.

Back Propagation

1. Calculating Gradients

The change in each weight is calculated as:

$$\Delta w_{ij} = \eta \times \delta_j \times O_j$$

Where:

- δ_j is the error term for each unit,
- η is the learning rate.

2. Output Unit Error

For O3:

$$\delta_5 = y_5(1 - y_5)(y_{target} - y_5)$$
$$= 0.67(1 - 0.67)(-0.17) = -0.0376$$

3. Hidden Unit Error

For h1:

$$\delta_3 = y_3(1 - y_3)(w_{1,3} \times \delta_5)$$
$$= 0.56(1 - 0.56)(0.3 \times -0.0376) = -0.0027$$

For h2:

$$\delta_4 = y_4(1 - y_4)(w_{2,3} \times \delta_5)$$
$$= 0.59(1 - 0.59)(0.9 \times -0.0376) = -0.0819$$

4. Weight Updates

For the weights from hidden to output layer:

$$\Delta w_{2,3} = 1 \times (-0.0376) \times 0.59 = -0.022184$$

New weight:

$$w_{2,3}(\text{new}) = -0.022184 + 0.9 = 0.877816$$

For weights from input to hidden layer:

$$\Delta w_{1,1} = 1 \times (-0.0027) \times 0.35 = 0.000945$$

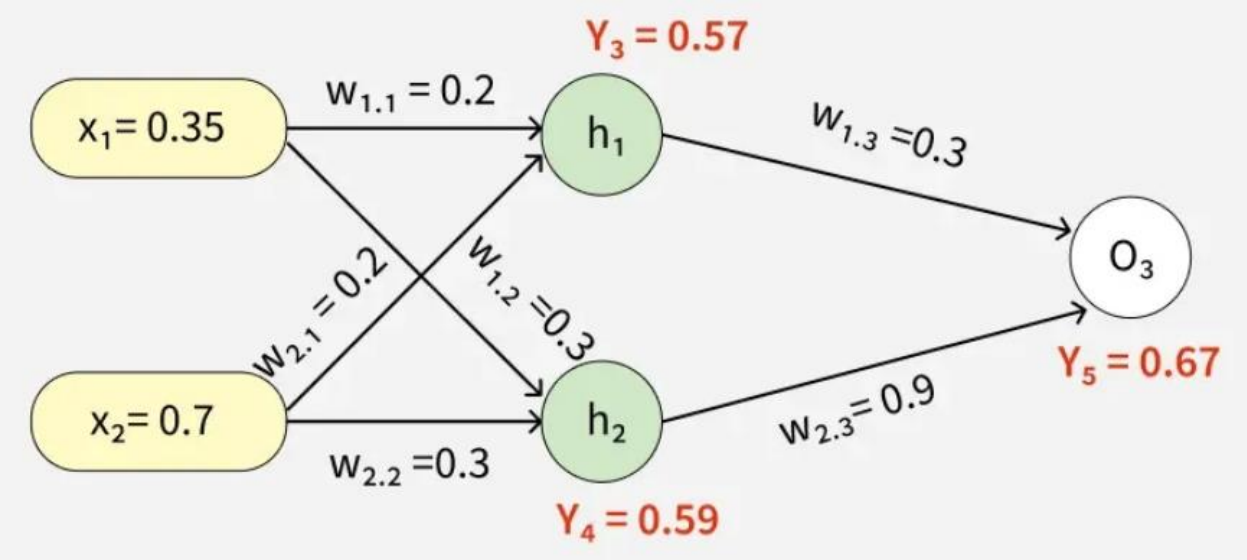
New weight:

$$w_{1,1}(\text{new}) = 0.000945 + 0.2 = 0.200945$$

Similarly other weights are updated:

- $w_{1,2}(\text{new}) = 0.273225$
- $w_{1,3}(\text{new}) = 0.086615$
- $w_{2,1}(\text{new}) = 0.269445$
- $w_{2,2}(\text{new}) = 0.18534$

The updated weights are illustrated below



After updating the weights, the forward pass is repeated yielding:

- y3=0.57
- y4=0.56
- y5=0.61

Since y5=0.61 is still not the target output the process of calculating the error and backpropagating continues until the desired output is reached.

This process demonstrates how Back Propagation iteratively updates weights by minimizing errors until the network accurately predicts the output.

$$Error = y_{target} - y_5$$
$$= 0.5 - 0.61 = -0.11$$

This process is said to be continued until the actual output is gained by the neural network.

Advantages of Back Propagation for Neural Network Training

The key benefits of using the Back Propagation algorithm are:

1. **Ease of Implementation:** Back Propagation is beginner-friendly requiring no prior neural network knowledge and simplifies programming by adjusting weights with error derivatives.

2. **Simplicity and Flexibility:** Its straightforward design suits a range of tasks from basic feedforward to complex convolutional or recurrent networks.

3. **Efficiency**: Back Propagation accelerates learning by directly updating weights based on error especially in deep networks.

4. **Generalization:** It helps models generalize well to new data improving prediction accuracy on unseen examples.

5. **Scalability:** The algorithm scales efficiently with larger datasets and more complex networks making it ideal for large-scale tasks.

Challenges with Back Propagation

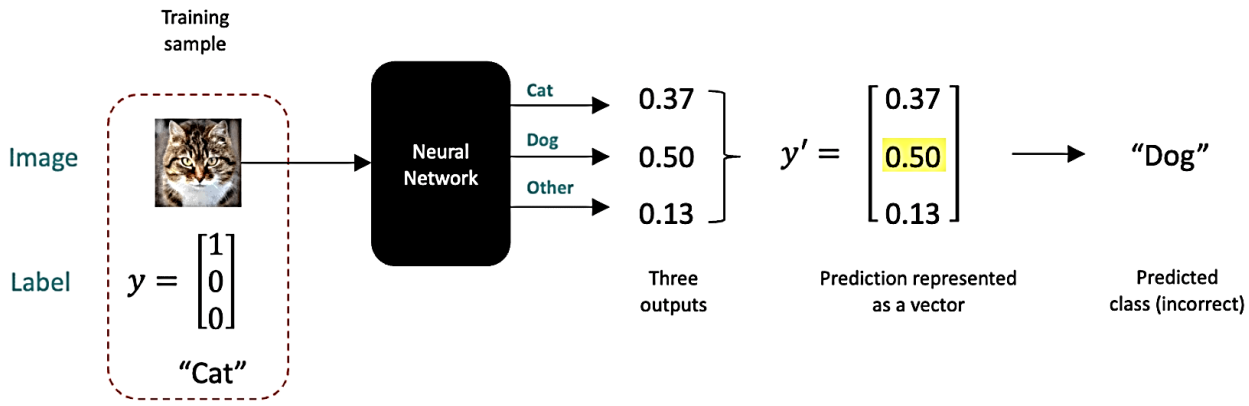
While Back Propagation is useful it does face some challenges:

1. **Vanishing Gradient Problem**: In deep networks the gradients can become very small during Back Propagation making it difficult for the network to learn. This is common when using activation functions like sigmoid or tanh.

2. **Exploding Gradients**: The gradients can also become excessively large causing the network to diverge during training.

3. **Overfitting:** If the network is too complex it might memorize the training data instead of learning general patterns.

4.2. Training Neural Network



Training a neural network involves adjusting the network's internal parameters (weights and biases) to minimize the difference between its predictions and the correct answers (ground truth) for a given set of training data. This process iteratively refines the network's ability to map inputs to outputs accurately.

Key Aspects of Neural Network Training:

- **Data:**

  Neural networks learn from data. Training data is used to adjust the network's parameters.

- **Loss Function:**

  A loss function quantifies the error between the network's predictions and the actual values.

- **Optimization:**

  Optimization algorithms (like gradient descent) are used to adjust the network's weights and biases to minimize the loss function.

- **Backpropagation:**

  A common algorithm for calculating the gradient of the loss function with respect to the network's parameters, enabling efficient weight updates.

- **Hyperparameter Tuning:**

  Adjusting parameters that control the training process, such as the learning rate or the number of hidden layers, to optimize performance.

- **Validation and Testing:**
  Validation data is used to monitor performance during training and prevent overfitting. Test data is used to evaluate the final model's performance on unseen data.

Training Process:

a) **Prepare Data:** Split the data into training, validation, and test sets.

b) **Build a Model:** Define the network architecture (number of layers, neurons, etc.).

c) **Choose a Loss Function:** Select a suitable loss function for the task.

d) **Initialize Weights:** Randomly initialize the network's weights.

e) **Iterate:** For each training example:

   a. Feed the input into the network.

   b. Calculate the output and the loss.

   c. Use backpropagation to compute the gradients of the loss with respect to the weights.

   d. Update the weights using an optimization algorithm (e.g., gradient descent).

f) **Monitor Performance:** Evaluate the model on the validation set during training.

g) **Fine-tune:** Adjust hyperparameters and potentially retrain the model.

h) **Evaluate on Test Data:** Assess the final model's performance on the unseen test data.

4.2.1. Forward and Backward Propagation

4.2.1.1. Forward Propagation

Forward propagation is the fundamental process in a neural network where input data passes through multiple layers to generate an output. It is the process by which input data passes through each layer of neural network to generate output. In this article, we'll more about forward propagation and see how it's implemented in practice.

Understanding Forward Propagation

In **Forward propagation** input data moves through each layer of neural network where each neuron applies weighted sum, adds bias, passes the result through an activation function and making predictions. This process is crucial before backpropagation updates the weights. It determines the output of neural network with a given set of inputs and current state of model parameters (weights and biases). Understanding this process helps in optimizing neural networks for various tasks like classification, regression and more. Below is the step by step working of forward propagation:

1. **Input Layer**

- The input data is fed into the network through the input layer.
- Each feature in the input dataset represents a neuron in this layer.
- The input is usually normalized or standardized to improve model performance.

2. **Hidden Layers**

- The input moves through one or more hidden layers where transformations occur.
- Each neuron in hidden layer computes a weighted sum of inputs and applies activation function to introduce non-linearity.
- Each neuron receives inputs, computes:

  $\boldsymbol{Z=WX+b}$,

  where:

  W is the weight matrix

  X is the input vector

  b is the bias term

- The activation function such as **ReLU** or sigmoid is applied.

3. **Output Layer**

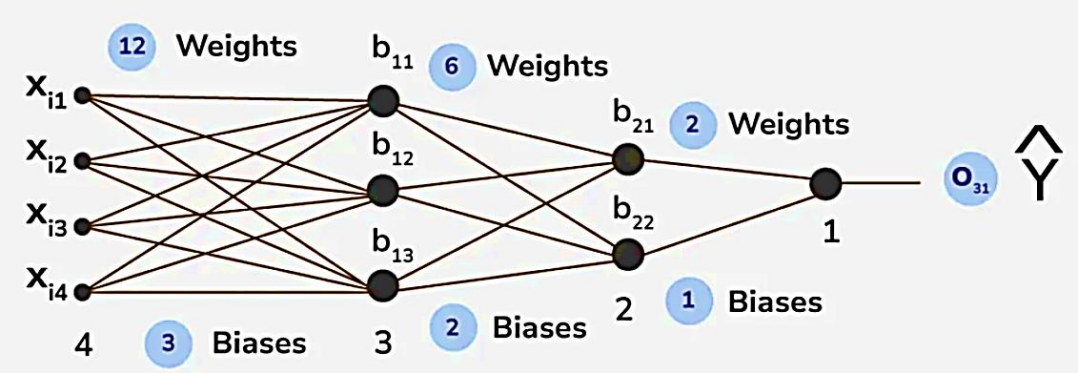
- The last layer in the network generates the final prediction.
- The activation function of this layer depends on the type of problem:
    - **Softmax** (for multi-class classification)
    - **Sigmoid** (for binary classification)
    - **Linear** (for regression tasks)

4. **Prediction**

- The network produces an output based on current weights and biases.
- The loss function evaluates the error by comparing predicted output with actual values.

Mathematical Explanation of Forward Propagation

Consider a neural network with one input layer, two hidden layers and one output layer.



1. Layer 1 (First Hidden Layer)

The transformation is: $A^{[1]} = \sigma(W^{[1]}X + b^{[1]})$ where:

- $W^{[1]}$ is the weight matrix,
- X is the input vector,
- $b^{[1]}$is the bias vector,
- σ is the activation function.

2. Layer 2 (Second Hidden Layer)

$$A^{[2]} = \sigma(W^{[2]}A^{[1]} + b^{[2]})$$

3. Output Layer

$Y = \sigma(W^{[3]}A^{[2]} + b^{[3]})$ where Y is the final output. Thus the complete equation for forward propagation is:

$$A^{[3]} = \sigma(\sigma(\sigma(XW^{[1]} + b^{[1]})W^{[2]} + b^{[2]})W^{[3]} + b^{[3]})$$

This equation illustrates how data flows through the network:

- Weights (W) determine the importance of each input
- Biases (b) adjust activation thresholds
- Activation functions (σ) introduce non-linearity to enable complex decision boundaries.

4.2.1.2. Gradient Descent

Gradient descent is the backbone of the learning process for various algorithms, including linear regression, logistic regression, support vector machines, and neural networks which serves as a fundamental optimization technique to minimize the cost function of a model by **iteratively adjusting the model parameters to reduce the difference between predicted and actual values, improving the model's performance**.

Let's see its role in machine learning:

1. Training Machine Learning Models

Neural networks are trained using Gradient Descent (or its variants) in combination with backpropagation. Backpropagation computes the gradients of the **loss function with respect to each parameter (weights and biases) in the network by applying the chain rule.** The process involves:

- **Forward Propagation**: Computes the output for a given input by passing data through the layers.
- **Backward Propagation**: Uses the chain rule to calculate gradients of the loss with respect to each parameter (weights and biases) across all layers.

Gradients are then used by Gradient Descent to update the parameters layer-by-layer, moving toward minimizing the loss function.

2. Minimizing the Cost Function

The algorithm minimizes a cost function, which quantifies the error or loss of the model's predictions compared to the true labels for:

1. Linear Regression

Gradient descent minimizes the Mean Squared Error (MSE) which serves as the loss function to find the best-fit line. Gradient Descent is used to iteratively update the weights (coefficients) and bias by computing the gradient of the MSE with respect to these parameters.

Since MSE is a convex function **gradient descent guarantees convergence to the global minimum if the learning rate is appropriately chosen.** For each iteration:

The algorithm computes the gradient of the MSE with respect to the weights and biases.

It updates the weights (w) and bias (b) using the formula:

- Calculating the gradient of the log-loss with respect to the weights.
- Updating weights and biases iteratively to maximize the likelihood of the correct classification:

$$w = w - \alpha \cdot \frac{\partial J(w,b)}{\partial w}, \quad b = b - \alpha \cdot \frac{\partial J(w,b)}{\partial b}$$

The formula is the **parameter update rule for gradient descent**, which adjusts the weights w and biases b to minimize a cost function. This process iteratively adjusts the line's slope and intercept to minimize the error.

2. Logistic Regression

In logistic regression, gradient descent minimizes the **Log Loss (Cross-Entropy Loss)** to optimize the decision boundary for binary classification. Since the output is probabilistic (between 0 and 1), the sigmoid function is applied. The process involves:

- Calculating the gradient of the log-loss with respect to the weights.
- Updating weights and biases iteratively to maximize the likelihood of the correct classification:

$$w = w - \alpha \cdot \frac{\partial J(w)}{\partial w}$$

This adjustment shifts the decision boundary to separate classes more effectively.

3. Support Vector Machines (SVMs)

For SVMs, gradient descent optimizes the **hinge loss**, which ensures a maximum-margin hyperplane. The algorithm:

- Calculates gradients for the hinge loss and the regularization term (if used, such as L2 regularization).
- Updates the weights to maximize the margin between classes while minimizing misclassification penalties with same formula provided above.

Gradient descent ensures the **optimal placement of the hyperplane to separate classes with the largest possible margin.**

Gradient Descent Learning Rate

The learning rate is a critical hyperparameter in the context of gradient descent, influencing the size of steps taken during the optimization process to update the model parameters. Choosing an appropriate learning rate is crucial for efficient and effective model training.

When the learning rate is **too small**, the optimization process progresses very slowly. The model makes tiny updates to its parameters in each iteration, leading to sluggish convergence and potentially getting stuck in local minima.

On the other hand, an **excessively large learning rate** can cause the optimization algorithm to overshoot the optimal parameter values, leading to divergence or oscillations that hinder convergence.

Achieving the right balance is essential. A small learning rate might result in vanishing gradients and slow convergence, while a large learning rate may lead to overshooting and instability.

Advantages & Disadvantages of gradient descent

Advantages of Gradient Descent

1. **Widely used:** Gradient descent and its variants are widely used in machine learning and optimization problems because they are effective and easy to implement.
2. **Convergence**: Gradient descent and its variants can converge to a global minimum or a good local minimum of the cost function, depending on the problem and the variant used.

3. **Scalability**: Many variants of gradient descent can be parallelized and are scalable to large datasets and high-dimensional models.

4. **Flexibility**: Different variants of gradient descent offer a range of trade-offs between accuracy and speed, and can be adjusted to optimize the performance of a specific problem.

Disadvantages of gradient descent:

1. **Choice of learning rate:** The choice of learning rate is crucial for the convergence of gradient descent and its variants. Choosing a learning rate that is too large can lead to oscillations or overshooting while choosing a learning rate that is too small can lead to slow convergence or getting stuck in local minima.

2. **Sensitivity to initialization:** Gradient descent and its variants can be sensitive to the initialization of the model's parameters, which can affect the convergence and the quality of the solution.

3. **Time-consuming:** Gradient descent and its variants can be time-consuming, especially when dealing with large datasets and high-dimensional models. The convergence speed can also vary depending on the variant used and the specific problem.

4. **Local optima:** Gradient descent and its variants can converge to a local minimum instead of the global minimum of the cost function, especially in non-convex problems. This can affect the quality of the solution, and techniques like random initialization and multiple restarts may be used to mitigate this issue.

4.3. Advanced Neural Network Architecture

An Artificial Neural Network (***ANN***) is an information processing paradigm that is inspired by the brain. ANNs, like people, learn by examples. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning largely involves adjustments to the synaptic connections that exist between the neurons.

Artificial Neural Networks (***ANNs***) are a type of machine learning model that are inspired by the structure and function of the human brain. They consist of layers of interconnected "neurons" that process and transmit information.

There are several different architectures for ANNs, each with their own strengths and weaknesses. Some of the most common architectures include:

Feedforward Neural Networks: This is the simplest type of ANN architecture, where the information flows in one direction from input to output. The layers are fully connected, meaning each neuron in a layer is connected to all the neurons in the next layer.

Recurrent Neural Networks (***RNNs***): These networks have a "memory" component, where information can flow in cycles through the network. This allows the network to process sequences of data, such as time series or speech.

Convolutional Neural Networks (***CNNs***): These networks are designed to process data with a grid-like topology, such as images. The layers consist of convolutional layers, which learn to detect specific features in the data, and pooling layers, which reduce the spatial dimensions of the data.

Autoencoders: These are neural networks that are used for unsupervised learning. They consist of an encoder that maps the input data to a lower-dimensional representation and a decoder that maps the representation back to the original data.

Generative Adversarial Networks (***GANs***): These are neural networks that are used for generative modeling. They consist of two parts: a generator that learns to generate new data samples, and a discriminator that learns to distinguish between real and generated data.

The model of an artificial neural network can be specified by three entities:

- **Interconnections**
- **Activation functions**
- **Learning rules**

Interconnections:

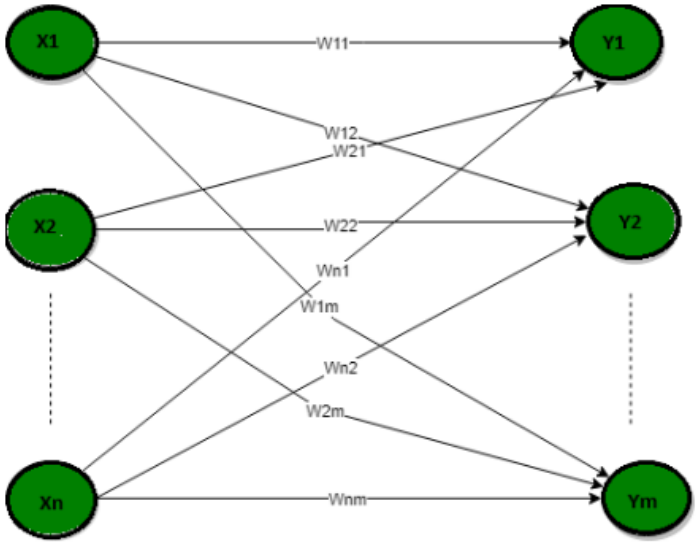
Interconnection can be defined as the way processing elements (Neuron) in ANN are connected to each other. Hence, the arrangements of these processing elements and geometry of interconnections are very essential in ANN.

These arrangements always have two layers that are common to all network architectures, the Input layer and output layer where the input layer buffers the input signal, and the output layer generates the output of the network. The third layer is the Hidden layer, in which neurons are neither kept in the input layer nor in the output layer. These neurons are hidden from the people who are interfacing with the system and act as a black box to them. By increasing the hidden layers with neurons, the system's computational and processing power can be increased but the training phenomena of the system get more complex at the same time.

There exist five basic types of neuron connection architecture:

1. Single-layer feed-forward network
2. Multilayer feed-forward network
3. Single node with its own feedback
4. Single-layer recurrent network
5. Multilayer recurrent network

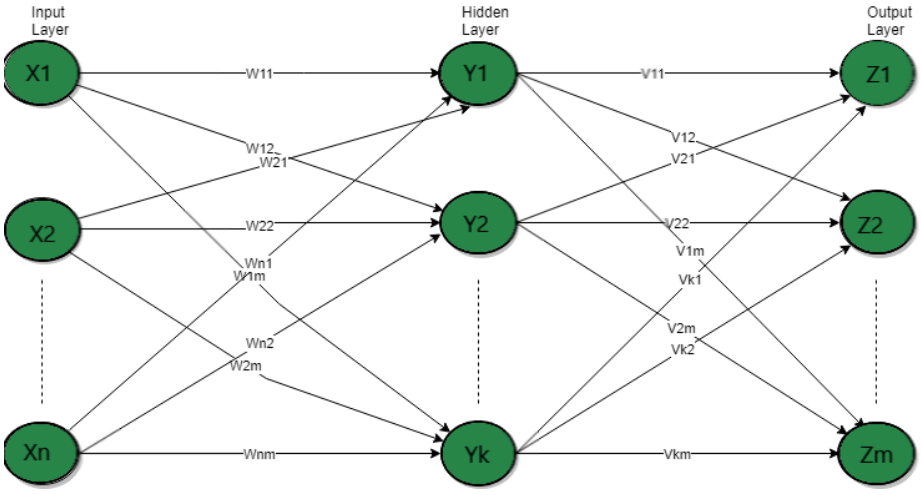
1. Single-layer feed-forward network



In this type of network, we have only two layers input layer and the output layer but the input layer does not count because no computation is performed in this layer. The output layer is formed when different weights are applied to input nodes and the cumulative effect per node is taken. After this, the neurons collectively give the output layer to compute the output signals.

2. Multilayer feed-forward network

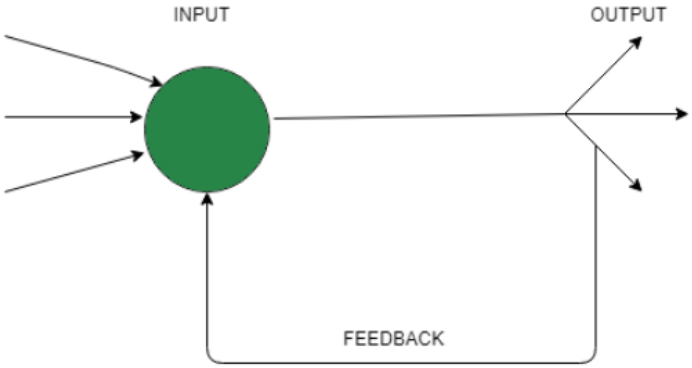
This layer also has a hidden layer that is internal to the network and has no direct contact with the external layer.



The existence of one or more hidden layers enables the network to be computationally stronger, a feed-forward network because of information flow through the input function, and the intermediate computations used to determine the output Z.

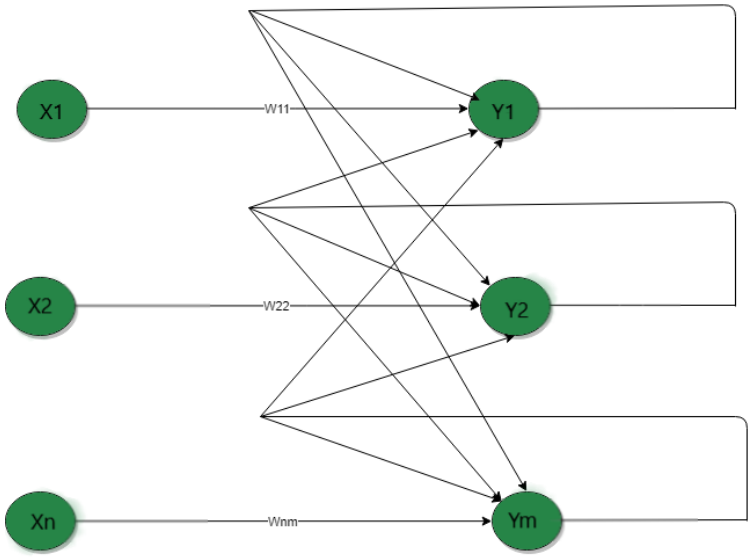
There are no feedback connections in which outputs of the model are fed back into itself.

3. Single node with its own feedback



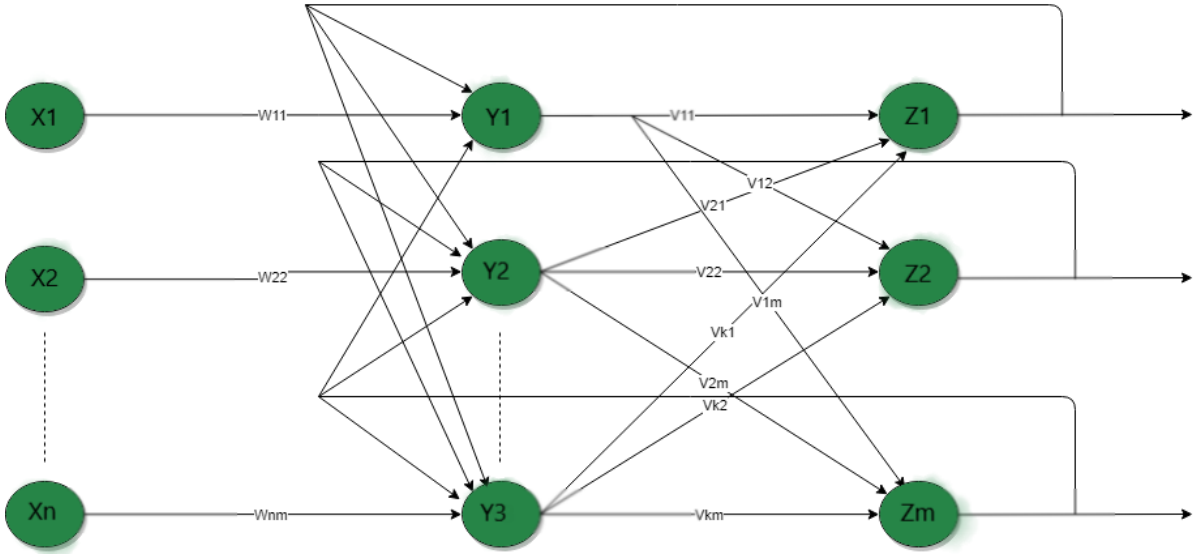
When outputs can be directed back as inputs to the same layer or preceding layer nodes, then it results in feedback networks. Recurrent networks are feedback networks with closed loops. The above figure shows a single recurrent network having a single neuron with feedback to itself.

4. Single-layer recurrent network



The above network is a single-layer network with a feedback connection in which the processing element's output can be directed back to itself or to another processing element or both. A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.

5. Multilayer recurrent network



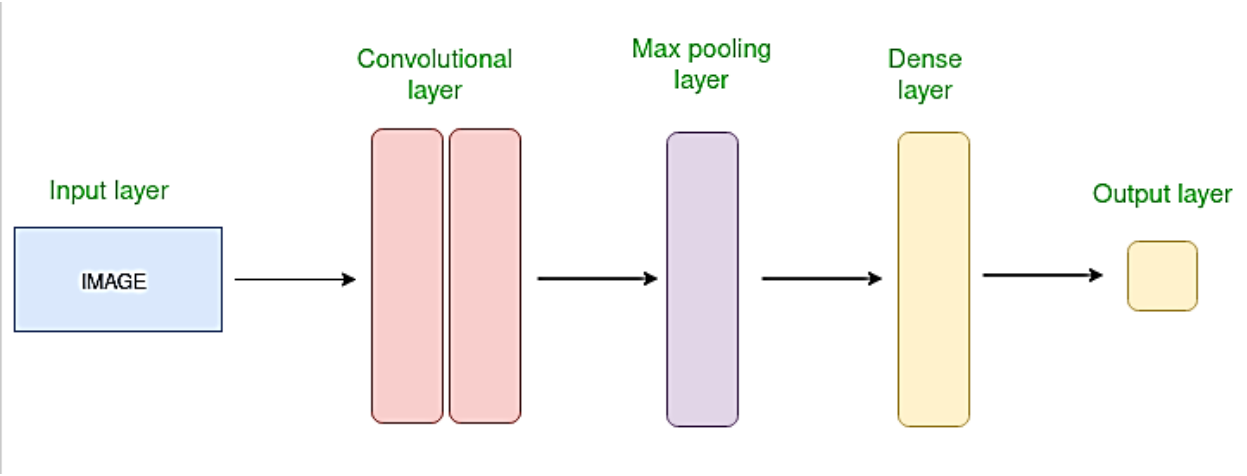
In this type of network, processing element output can be directed to the processing element in the same layer and in the preceding layer forming a multilayer recurrent network. They perform

the same task for every element of a sequence, with the output being dependent on the previous computations. Inputs are not needed at each time step. The main feature of a Recurrent Neural Network is its hidden state, which captures some information about a sequence.

4.3.1. Convolution Neural Network (CNNs)

Convolutional Neural Network (CNN) is an advanced version of **artificial neural networks (ANNs)**, primarily designed to extract features from grid-like matrix datasets. This is particularly useful for visual datasets such as images or videos, where data patterns play a crucial role. CNNs are widely used in **computer vision** applications due to their effectiveness in processing visual data.

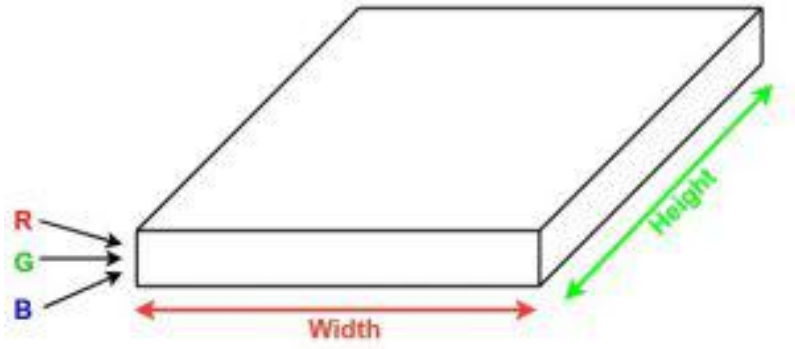
CNNs consist of multiple layers like the input layer, Convolutional layer, pooling layer, and fully connected layers. Let's learn more about CNNs in detail.



Simple CNN architecture

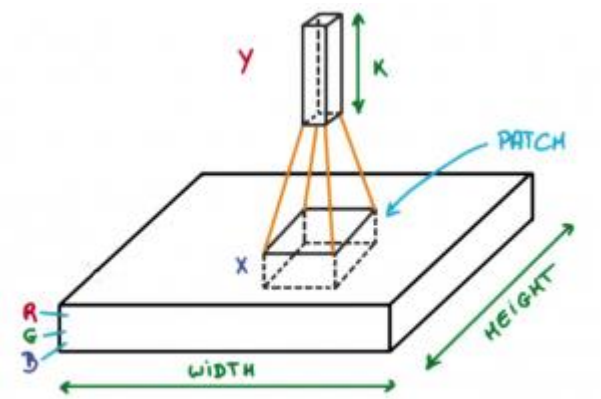
How Convolutional Layers Works?

Convolution Neural Networks are neural networks that share their parameters.



Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e. the channel as images generally has red, green, and blue channels).

Now imagine taking a small patch of this image and running a small neural network, called a filter or kernel on it, with say, K outputs and representing them vertically.



Now slide that neural network across the whole image, as a result, we will get another image with different widths, heights, and depths. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called **Convolution**. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

Mathematical Overview of Convolution

Now let's talk about a bit of mathematics that is involved in the whole convolution process.

- Convolution layers consist of a set of learnable filters (or kernels) having small widths and heights and the same depth as that of input volume (3 if the input layer is image input).
- For example, if we have to run convolution on an image with dimensions 34x34x3. The possible size of filters can be axax3, where 'a' can be anything like 3, 5, or 7 but smaller as compared to the image dimension.
- During the forward pass, we slide each filter across the whole input volume step by step where each step is called **stride** (which can have a value of 2, 3, or even 4 for high-dimensional images) and compute the dot product between the kernel weights and patch from input volume.
- As we slide our filters, we'll get a 2-D output for each filter and we'll stack them together as a result, we'll get output volume having a depth equal to the number of filters. The network will learn all the filters.

Layers Used to Build *ConvNets*

A complete Convolution Neural Networks architecture is also known as covnets. A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

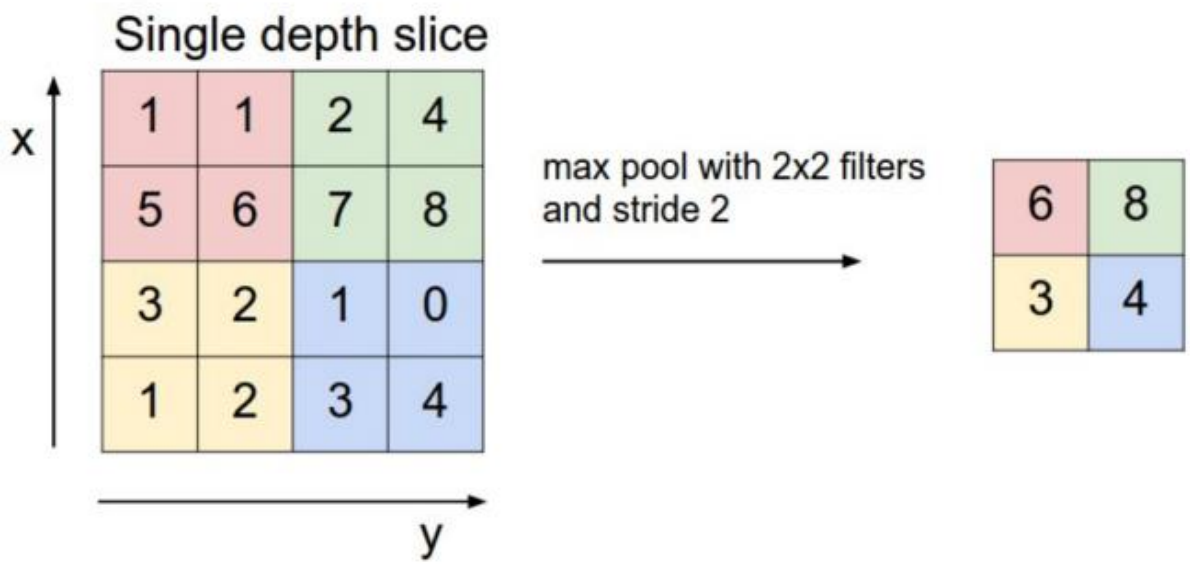
Let's take an example by running a *covnets* on of image of dimension 32 x 32 x 3.

- **Input Layers:** It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.

- **Convolutional Layers:** This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2x2, 3x3, or 5x5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12.

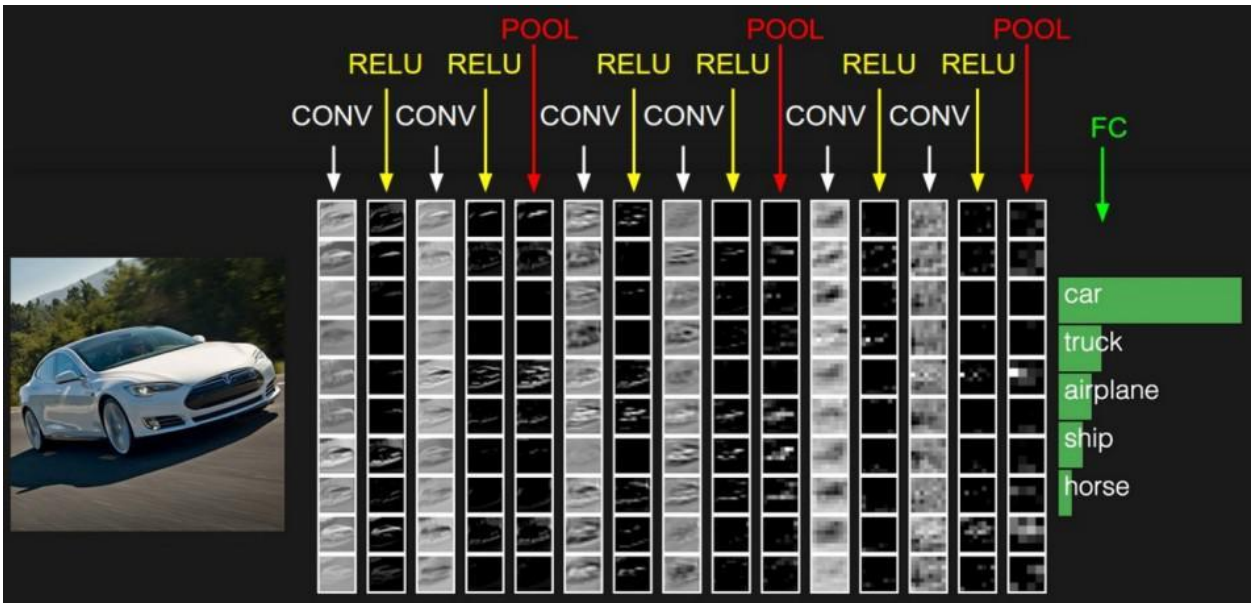
- **Activation Layer**: By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are **RELU**: max (0, x), **Tanh**, **Leaky RELU**, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.

- **Pooling layer**: This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.



- **Flattening:** The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.

- **Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.



Output Layer: The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or SoftMax which converts the output of each class into the probability score of each class.

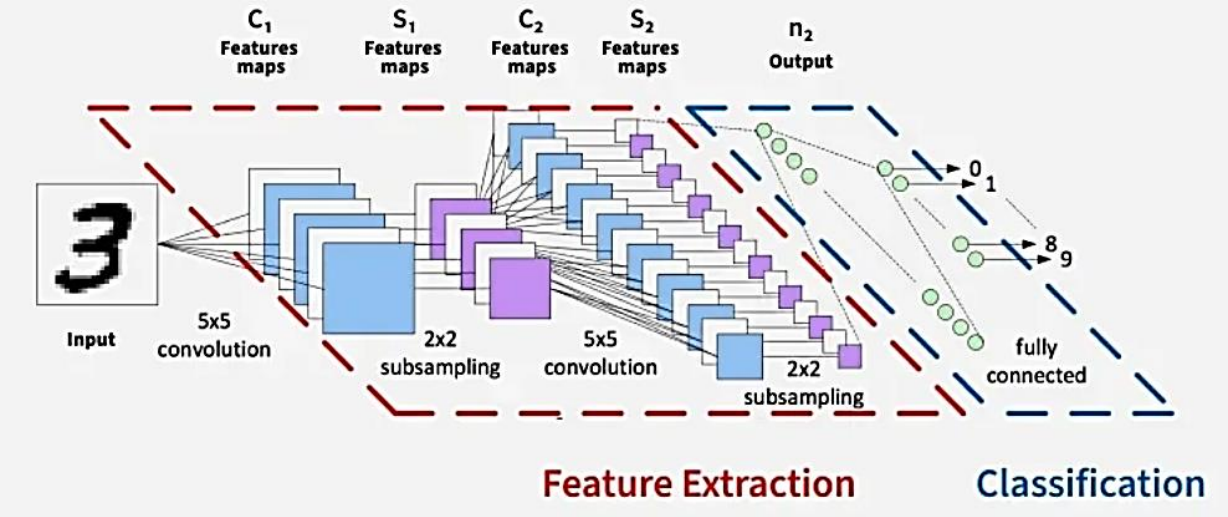
Advantages of CNNs

1. Good at detecting patterns and features in images, videos, and audio signals.
2. Robust to translation, rotation, and scaling invariance.
3. End-to-end training, no need for manual feature extraction.
4. Can handle large amounts of data and achieve high accuracy.

Disadvantages of CNNs

1. Computationally expensive to train and require a lot of memory.
2. Can be prone to overfitting if not enough data or proper regularization is used.
3. Requires large amounts of labeled data.
4. Interpretability is limited, it's hard to understand what the network has learned.

4.3.1.1. CNNs and their components



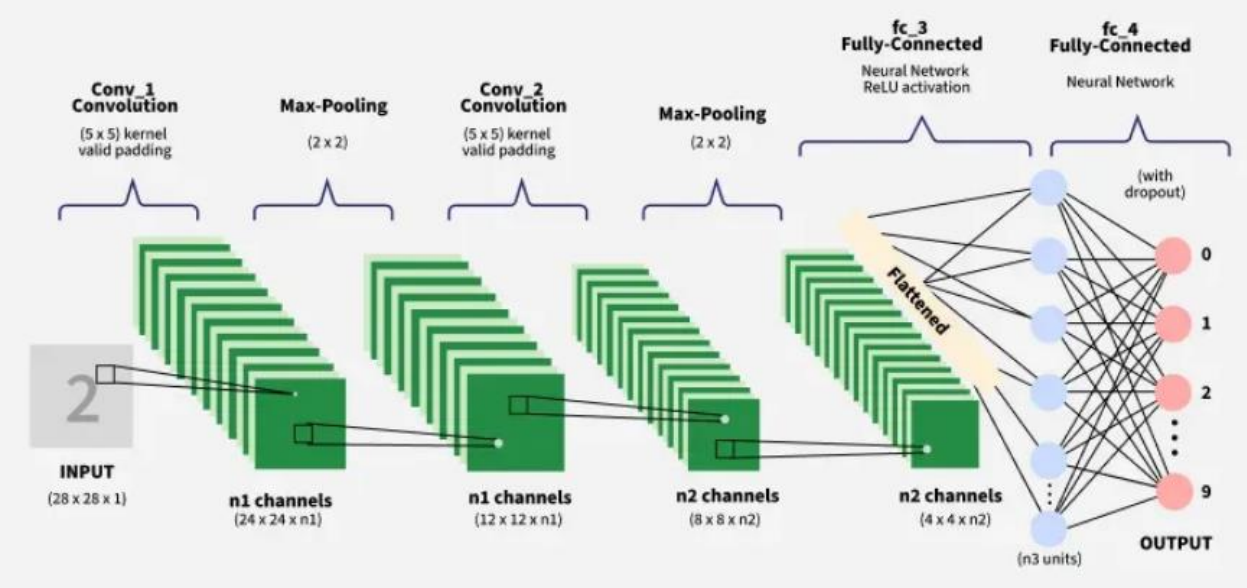
1. **Convolutional Layers:** These layers apply convolutional operations to input images using filters or kernels to detect features such as edges, textures and more complex patterns. Convolutional operations help preserve the spatial relationships between pixels.
2. **Pooling Layers:** They down sample the spatial dimensions of the input, reducing the computational complexity and the number of parameters in the network. Max pooling is a common pooling operation where we select a maximum value from a group of neighboring pixels.
3. **Activation Functions:** They introduce non-linearity to the model by allowing it to learn more complex relationships in the data.
4. **Fully Connected Layers:** These layers are responsible for making predictions based on the high-level features learned by the previous layers. They connect every neuron in one layer to every neuron in the next layer.

How CNNs Work?

1. **Input Image**: CNN receives an input image which is preprocessed to ensure uniformity in size and format.
2. **Convolutional Layers**: Filters are applied to the input image to extract features like edges, textures and shapes.
3. **Pooling Layers**: The feature maps generated by the convolutional layers are down sampled to reduce dimensionality.

4. **Fully Connected Layers**: The down sampled feature maps are passed through fully connected layers to produce the final output, such as a classification label.

5. **Output**: The CNN outputs a prediction, such as the class of the image.



4.3.1.2. Convolution, Pooling and fully connected layers

Convolution Layers

Convolution layers are fundamental components of convolutional neural networks (CNNs), which have revolutionized the field of computer vision and image processing. These layers are designed to automatically and adaptively learn spatial hierarchies of features from input images, enabling tasks such as image classification, object detection, and segmentation. This article will provide a comprehensive introduction to convolution layers, exploring their structure, functionality, and significance in deep learning.

Key Components of a Convolution Layer

1. **Filters (Kernels)**: Filters are small, learnable matrices that extract specific features from the input data. For example, a filter might detect horizontal edges, while another might detect vertical edges. During training, the values of these filters are adjusted to optimize the feature extraction process.

2. **Stride**: The stride determines how much the filter moves during the convolution operation. A stride of 1 means the filter moves one pixel at a time, while a stride of 2 means it moves two pixels at a time. Larger strides result in smaller output feature maps and faster computations.

3. **Padding**: Padding involves adding extra pixels around the input data to control the spatial dimensions of the output feature map. There are two common types of padding: 'valid' padding, which adds no extra pixels, and 'same' padding, which adds pixels to ensure the output feature map has the same dimensions as the input.

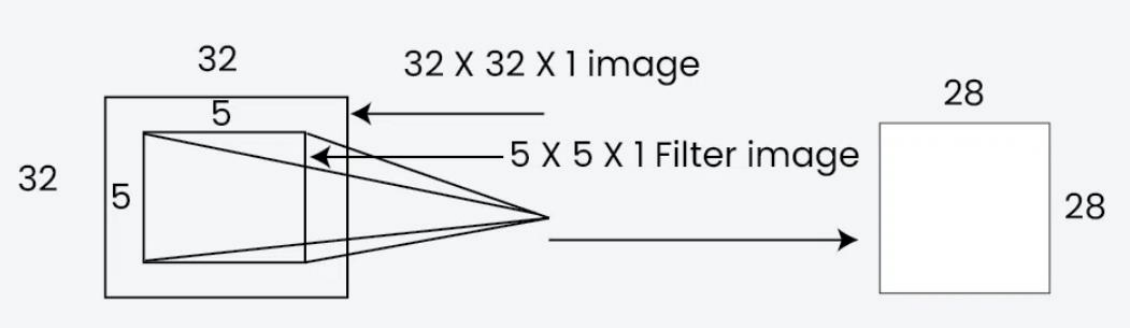
4. **Activation Function**: After the convolution operation, an activation function, typically the Rectified Linear Unit (ReLU), is applied to introduce non-linearity into the model. This helps the network learn complex patterns and relationships in the data.

Steps in a Convolution Layer

1. **Initialize Filters:**
    - Randomly initialize a set of filters with learnable parameters.
2. **Convolve Filters with Input:**
    - Slide the filters across the width and height of the input data, computing the dot product between the filter and the input sub-region.
3. **Apply Activation Function:**
    - Apply a non-linear activation function to the convolved output to introduce non-linearity.
4. **Pooling (Optional):**
    - Often followed by a pooling layer (like max pooling) to reduce the spatial dimensions of the feature map and retain the most important information.

Example Of Convolution Layer

Consider an input image of size 32x32x3 (32x32 pixels with 3 color channels). A convolution layer with ten 5x5 filters, a stride of 1, and 'same' padding will produce an output feature map of size 32x32x10. Each of the 10 filters detects different features in the input image.



Benefits of Convolution Layers

- **Parameter Sharing:** The same filter is used across different parts of the input, reducing the number of parameters and computational cost.
- **Local Connectivity:** Each filter focuses on a small local region, capturing local patterns and features.
- **Hierarchical Feature Learning:** Multiple convolution layers can learn increasingly complex features, from edges and textures in early layers to object parts and whole objects in deeper layers.

Pooling layer is used in CNNs to reduce the spatial dimensions (width and height) of the input feature maps while retaining the most important information. It involves sliding a two-dimensional filter over each channel of a feature map and summarizing the features within the region covered by the filter.

Pooling layer is used in CNNs to reduce the spatial dimensions (width and height) of the input feature maps while retaining the most important information. It involves sliding a two-dimensional filter over each channel of a feature map and summarizing the features within the region covered by the filter.

For a feature map with dimensions $\mathbf{n_h \times n_w \times nc}$, the dimensions of the output after a pooling layer are:

$$\left(\frac{n_h - f + 1}{s}\right) \times \left(\frac{n_w - f + 1}{s}\right) \times n_c$$

where:

- $n_h \rightarrow$ height of the feature map
- $n_w \rightarrow$ width of the feature map
- $n_c \rightarrow$ number of channels in the feature map
- $f \rightarrow$ size of the pooling filter
- $s \rightarrow$ stride length

A typical CNN model architecture consists of multiple convolution and pooling layers stacked together.

Why are Pooling Layers Important?

1. **Dimensionality Reduction**: Pooling layers reduce the spatial size of the feature maps, which decreases the number of parameters and computations in the network. This makes the model faster and more efficient.

2. **Translation Invariance**: Pooling helps the network become invariant to small translations or distortions in the input image. For example, even if an object in an image is slightly shifted, the pooled output will remain relatively unchanged.

3. **Overfitting Prevention**: By reducing the spatial dimensions, pooling layers help prevent overfitting by providing a form of regularization.

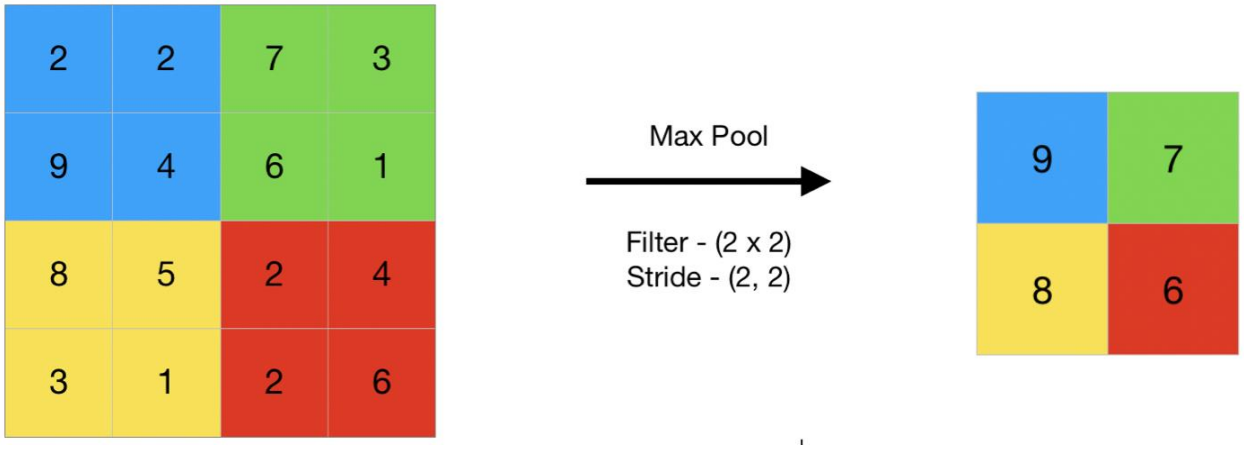
4. **Feature Hierarchy**: Pooling layers help build a hierarchical representation of features, where lower layers capture fine details and higher layers capture more abstract and global features.

Types of Pooling Layers

1. Max Pooling

Max pooling selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

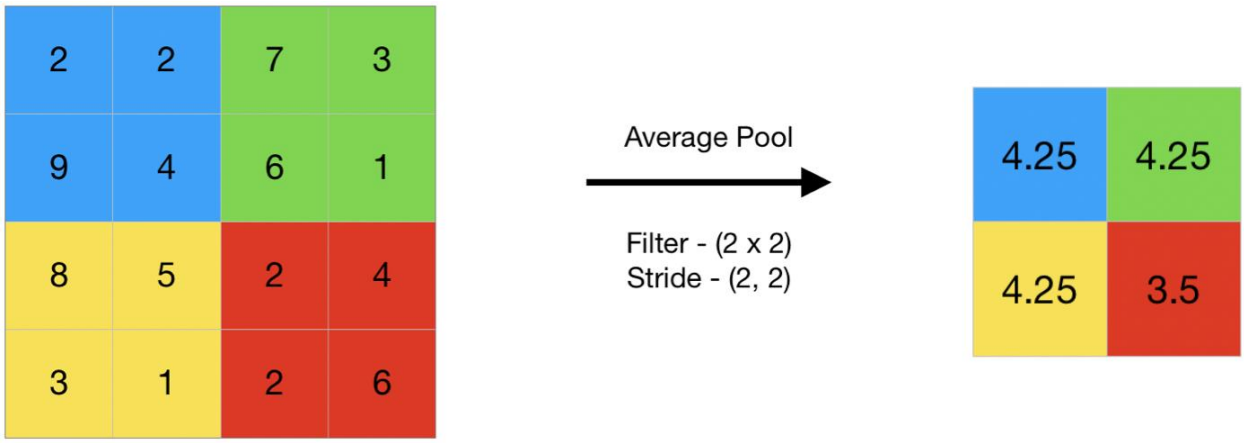
Max pooling layer preserves the most important features (edges, textures, etc.) and provides better performance in most cases.



2. Average Pooling

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

Average pooling provides a more generalized representation of the input. It is useful in the cases where preserving the overall context is important.



3. Global Pooling

Global pooling reduces each channel in the feature map to a single value, producing a $\mathbf{1\times1\times}n_c$ output. This is equivalent to applying a filter of size $n_h \times n_w$.

There are two types of global pooling:

- **Global Max Pooling**: Takes the maximum value across the entire feature map.
- **Global Average Pooling**: Computes the average of all values in the feature map.

How Pooling Layers Work?

1. **Define a Pooling Window (Filter)**: The size of the pooling window (e.g., 2x2) is chosen, along with a stride (the step size by which the window moves). A common choice is a 2x2 window with a stride of 2, which reduces the feature map size by half.
2. **Slide the Window Over the Input**: The pooling operation is applied to each region of the input feature map covered by the window.
3. **Apply the Pooling Operation**: Depending on the type of pooling (max, average, etc.), the operation extracts the required value from each window.

4. **Output the Down sampled Feature Map**: The result is a smaller feature map that retains the most important information.

Key Factors to Consider for Optimizing Pooling Layer

- **Pooling Window Size**: The size of the pooling window affects the degree of down sampling. A larger window results in more aggressive down sampling but may lose important details.
- **Stride**: The stride determines how much the pooling window moves at each step. A larger stride results in greater dimensionality reduction.
- **Padding**: In some cases, padding is used to ensure that the pooling operation covers the entire input feature map.

Advantages of Pooling Layer

1. **Dimensionality reduction:** Pooling layer helps in reducing the spatial dimensions of the feature maps. This reduces the computational cost and also helps in avoiding overfitting by reducing the number of parameters in the model.
2. **Translation invariance:** Pooling layers are useful in achieving translation invariance in the feature maps. This means that the position of an object in the image does not affect the classification result, as the same features are detected regardless of the position of the object.
3. **Feature selection:** Pooling layers help in selecting the most important features from the input, as max pooling selects the most salient features and average pooling provides a balanced representation.

Disadvantages of Pooling Layers

1. **Information Loss**: Pooling reduces spatial resolution, which can lead to a loss of important fine details.
2. **Over-smoothing**: Excessive pooling may blur out crucial features.
3. **Hyperparameter Tuning**: The choice of pooling size and stride affects performance and requires careful tuning.

4.3.1.3. Applications in image processing and computer vision

- **Image classification:** CNNs are the state-of-the-art models for image classification. They can be used to classify images into different categories such as cats and dogs.
- **Object detection:** It can be used to detect objects in images such as people, cars and buildings. They can also be used to localize objects in images which means that they can identify the location of an object in an image.
- **Image segmentation:** It can be used to segment images which means that they can identify and label different objects in an image. This is useful for applications such as medical imaging and robotics.
- **Video analysis:** It can be used to analyze videos such as tracking objects in a video or detecting events in a video. This is useful for applications such as video surveillance and traffic monitoring.

Fully Connected Layer vs Convolutional Layer

Features	Fully Connected Layer	Convolutional Layer
Definition	Every neuron is connected to every neuron in the previous layer.	Neurons are connected only to a local region of the previous layer.
Connectivity	Dense connections; each neuron connects to all neurons in the previous layer.	Sparse connections; each neuron connects only to a local patch of the input.
Parameters	Large number of parameters due to full connectivity.	Fewer parameters due to shared weights and local connectivity.
Weight Sharing	No weight sharing; each connection has its own weight.	Weights are shared across spatial positions, reducing the number of parameters.

Features	Fully Connected Layer	Convolutional Layer
Typical Use Cases	Final classification layers in neural networks.	Feature extraction, especially in image and video processing.
Computation Cost	Higher computational cost due to large number of connections.	Lower computational cost per neuron due to local connections.
Overfitting	Higher risk of overfitting due to large number of parameters.	Lower risk of overfitting due to fewer parameters and regularization effects of local connections.
Dimensionality Reduction	Does not inherently reduce dimensionality.	Can reduce dimensionality through pooling layers.
Examples	Multilayer Perceptron (MLP), Dense layers in CNNs.	Convolutional Neural Networks (CNNs), such as layers in AlexNet, VGGNet.

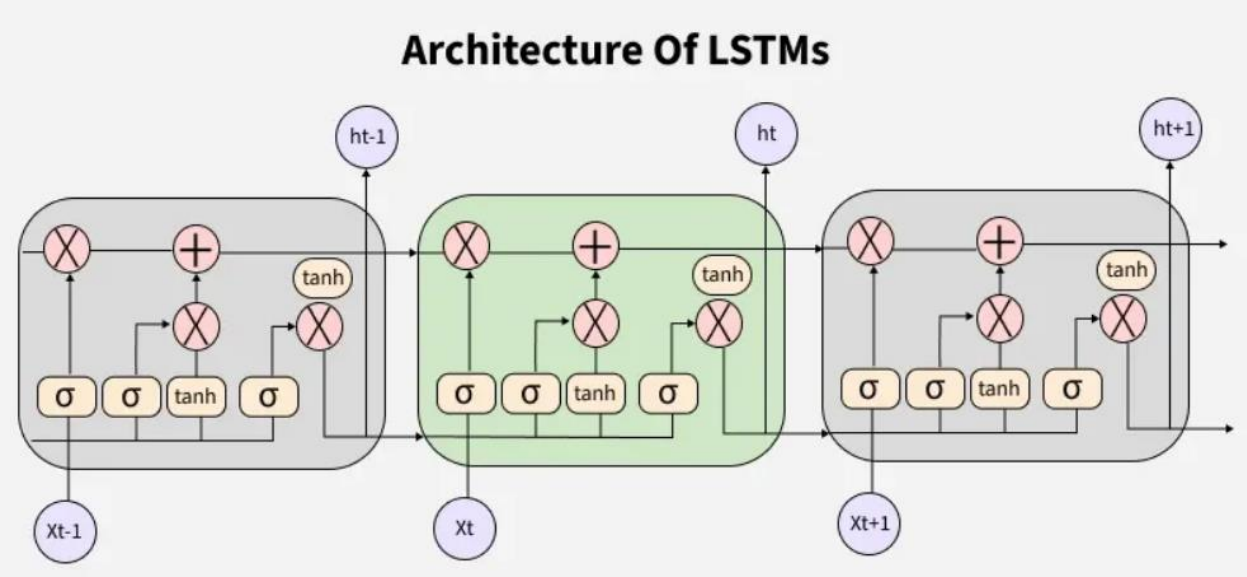
4.3.2. Recurrent Neural Networks (RNNs)

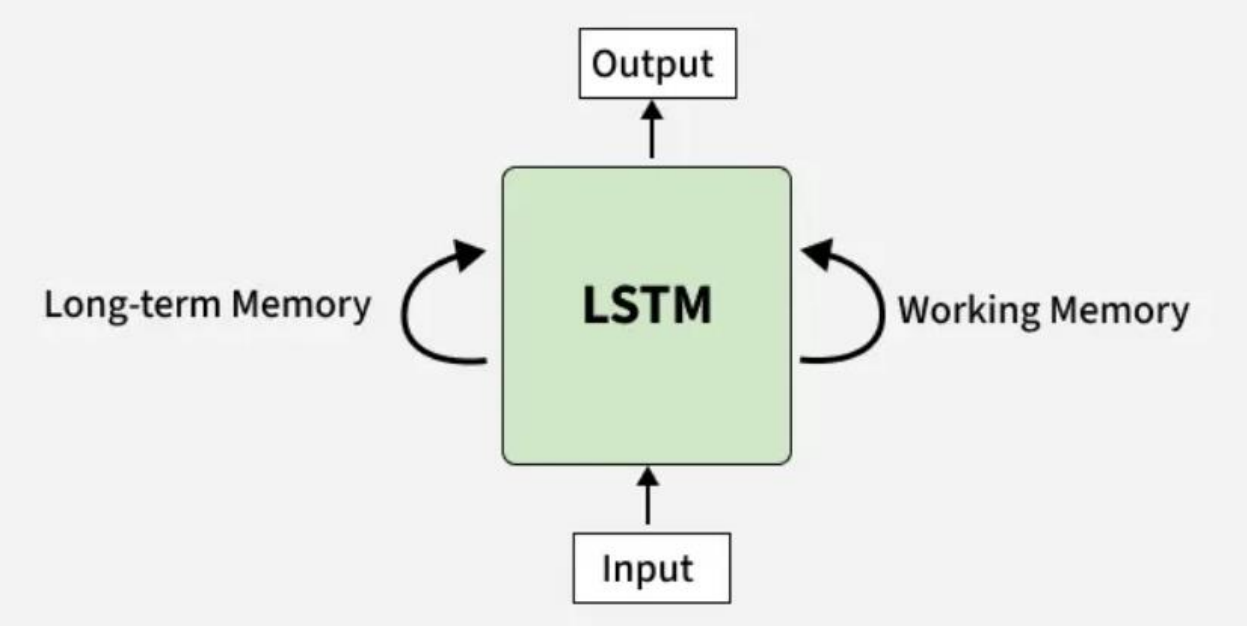
Recurrent Neural Networks (RNNs) differ from regular neural networks in how they process information. While standard neural networks pass information in one direction i.e from input to output, RNNs feed information back into the network at each step.

4.3.2.2. Long Short-Term Memory (LSTM)

Long Short-Term Memory (**LSTM**) is an enhanced version of the Recurrent Neural Network (RNN). LSTMs can capture long-term dependencies in sequential data making them ideal for tasks like language translation, speech recognition and time series forecasting.

Unlike traditional RNNs which use a single hidden state passed through time LSTMs introduce a memory cell that holds information over extended periods addressing the challenge of learning long-term dependencies.





Problem with Long-Term Dependencies in RNN

Recurrent Neural Networks (RNNs) are designed to handle sequential data by maintaining a hidden state that captures information from previous time steps. However, they often face challenges in learning long-term dependencies where information from distant time steps becomes crucial for making accurate predictions for current state. This problem is known as the vanishing gradient or exploding gradient problem.

- **Vanishing Gradient**: When training a model over time, the gradients which help the model learn can shrink as they pass through many steps. This makes it hard for the

model to learn long-term patterns since earlier information becomes almost irrelevant.

- **Exploding Gradient**: Sometimes gradients can grow too large causing instability. This makes it difficult for the model to learn properly as the updates to the model become erratic and unpredictable.

Both of these issues make it challenging for standard RNNs to effectively capture long-term dependencies in sequential data.

LSTM Architecture

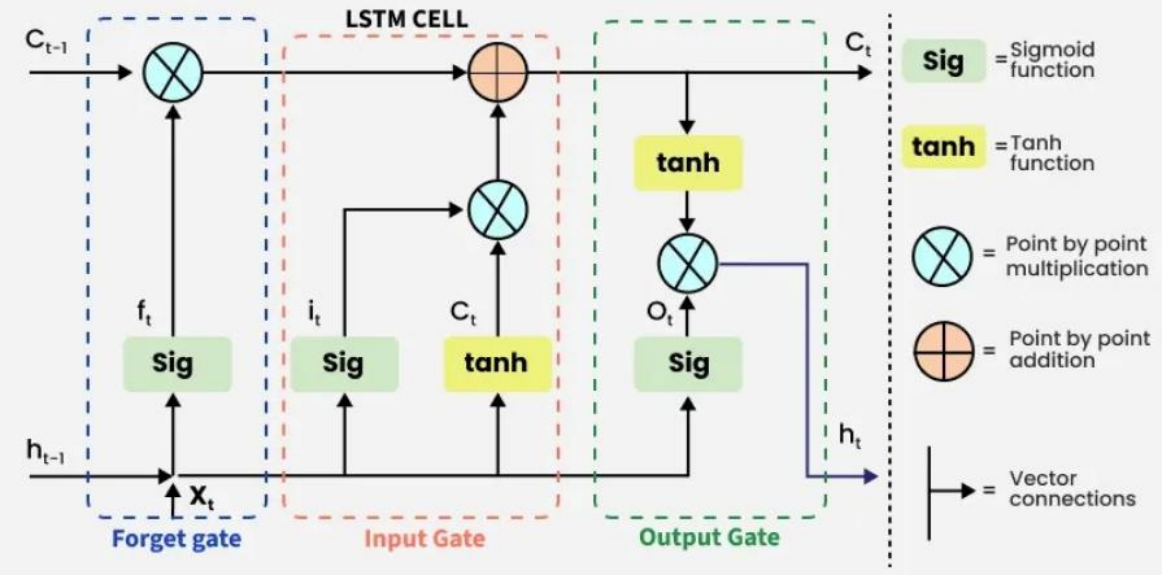
LSTM architectures involves the memory cell which is controlled by three gates:

1. **Input gate**: Controls what information is added to the memory cell.
2. **Forget gate**: Determines what information is removed from the memory cell.
3. **Output gate**: Controls what information is output from the memory cell.

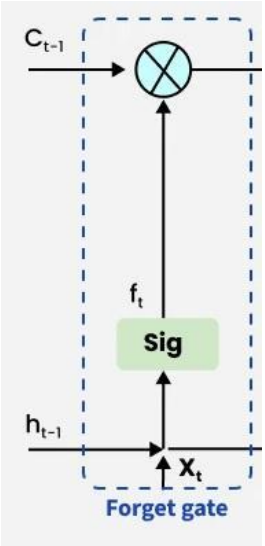
This allows ***LSTM*** networks to selectively retain or discard information as it flows through the network which allows them to learn long-term dependencies. The network has a hidden state which is like its short-term memory. This memory is updated using the current input, the previous hidden state and the current state of the memory cell.

Working of LSTM

LSTM architecture has a chain structure that contains four neural networks and different memory blocks called cells.



LSTM Model



Information is retained by the cells and the memory manipulations are done by the gates**.** There are three gates -

1. Forget Gate

The information that is no longer useful in the cell state is removed with the forget gate. Two inputs x_t (input at the particular time) and h_{t-1} (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use.

The equation for the forget gate is:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Where:

- W_f represents the weight matrix associated with the forget gate.
- $[h_t - 1, x_t]$ denotes the concatenation of the current input and the previous hidden state.
- b_f is the bias with the forget gate.
- σ is the sigmoid activation function.

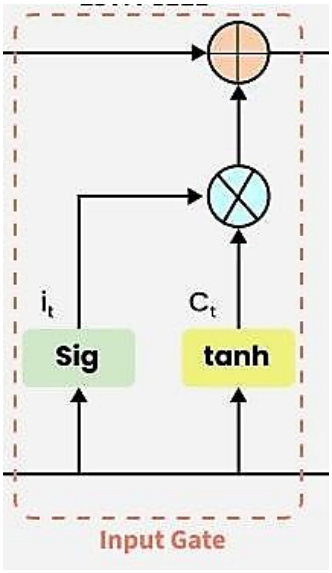
2. Input gate

The addition of useful information to the cell state is done by the input gate. First the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs h_{t-1} and x_t. Then, a vector is created using *tanh* function that gives an output from -1 to +1 which contains all the possible values from h_{t-1} and x_t. At last, the values of the vector and the regulated values are multiplied to obtain the useful information. The equation for the input gate is:

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\hat{C}_t = \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$$

We multiply the previous state by f_t effectively filtering out the information we had decided to ignore earlier. Then we add $i_t \odot C_t$ which represents the new candidate values scaled by how much we decided to update each state value.

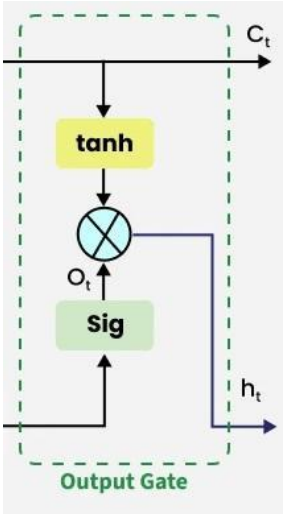


$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

where

- $\odot$ denotes element-wise multiplication
- tanh is activation function

3. Output gate



The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using input h_{t-1} and x_t. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell. The equation for the output gate is:

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$

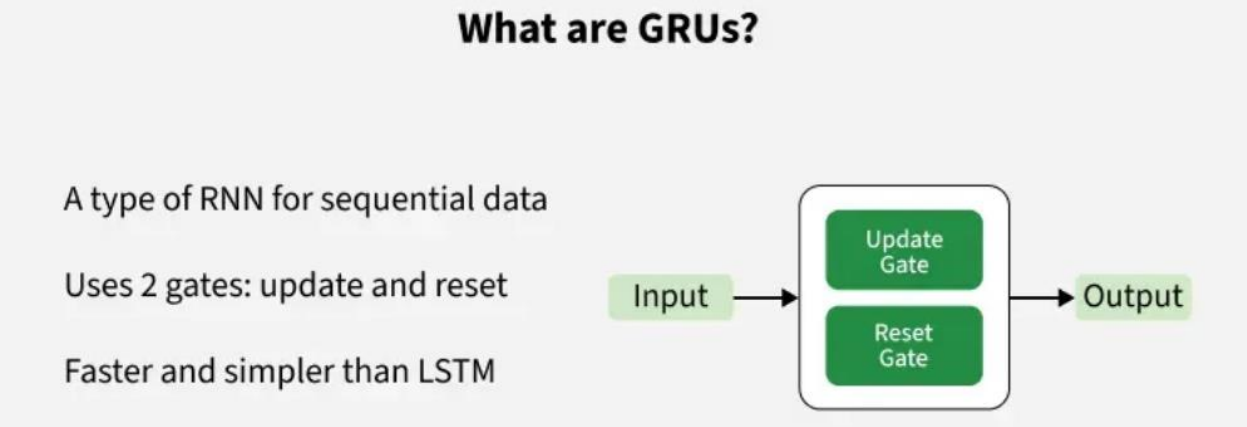
Applications of LSTM

Some of the famous applications of LSTM includes:

- **Language Modeling**: Used in tasks like language modeling, machine translation and text summarization. These networks learn the dependencies between words in a sentence to generate coherent and grammatically correct sentences.
- **Speech Recognition**: Used in transcribing speech to text and recognizing spoken commands. By learning speech patterns, they can match spoken words to corresponding text.
- **Time Series Forecasting**: Used for predicting stock prices, weather and energy consumption. They learn patterns in time series data to predict future events.

- **Anomaly Detection**: Used for detecting fraud or network intrusions. These networks can identify patterns in data that deviate drastically and flag them as potential anomalies.
- **Recommender Systems**: In recommendation tasks like suggesting movies, music and books. They learn user behavior patterns to provide personalized suggestions.
- **Video Analysis**: Applied in tasks such as object detection, activity recognition and action classification. When combined with Convolutional Neural Networks (CNNs) they help analyze video data and extract useful information.

4.3.2.3. Gradient Recurrent Units (GRU)

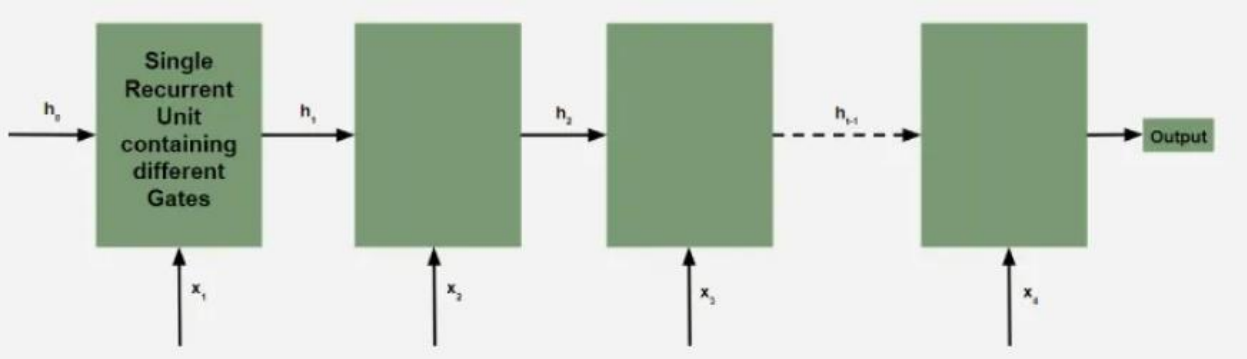


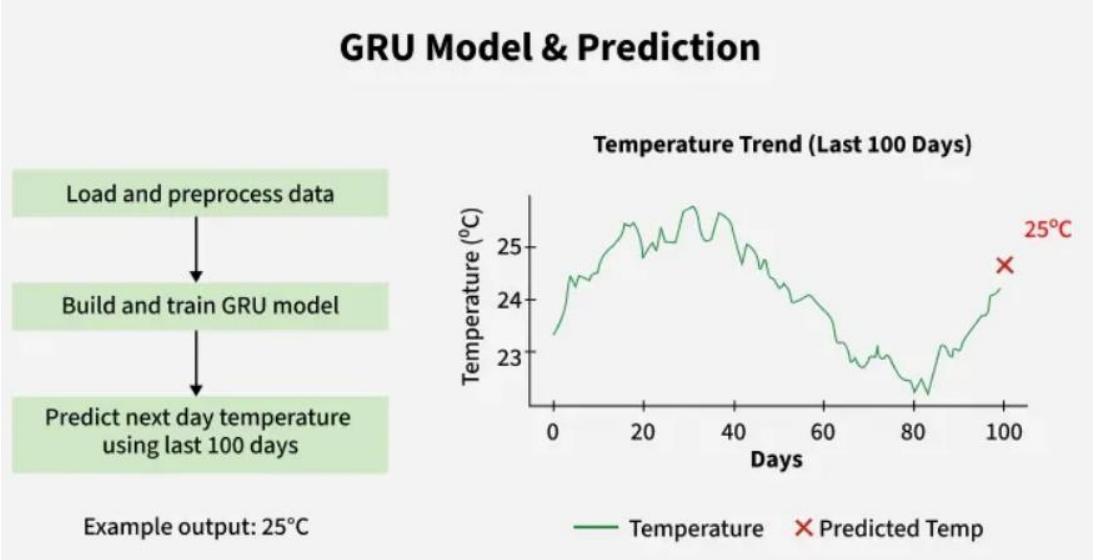
In machine learning Recurrent Neural Networks (RNNs) are essential for tasks involving sequential data such as text, speech and time-series analysis. While traditional RNNs struggle with capturing long-term dependencies due to the **vanishing gradient problem** architectures like Long Short-Term Memory (LSTM) networks were developed to overcome this limitation.

However, LSTMs are very complex structure with higher computational cost. To overcome this **Gated Recurrent Unit (GRU)** where introduced which uses LSTM architecture by merging its gating mechanisms offering a more efficient solution for many sequential tasks without sacrificing performance. In this article we'll learn more about them.

What are Gated Recurrent Units (GRU)?

Gated Recurrent Units (GRUs) are a type of RNN introduced by Cho et al. in 2014. The core idea behind GRUs is to use **gating mechanisms** to selectively update the hidden state at each time step allowing them to remember important information while discarding irrelevant details. GRUs aim to simplify the LSTM architecture by merging some of its components and focusing on just two main gates: the **update gate** and the **reset gate**.





The GRU consists of **two main gates**:

1. **Update Gate** (z_t): This gate decides how much information from previous hidden state should be retained for the next time step.

2. **Reset Gate** (r_t): This gate determines how much of the past hidden state should be forgotten.

These gates allow GRU to control the flow of information in a more efficient manner compared to traditional RNNs which solely rely on hidden state.

Equations for GRU Operations

The internal workings of a GRU can be described using following equations:

1. Reset gate:

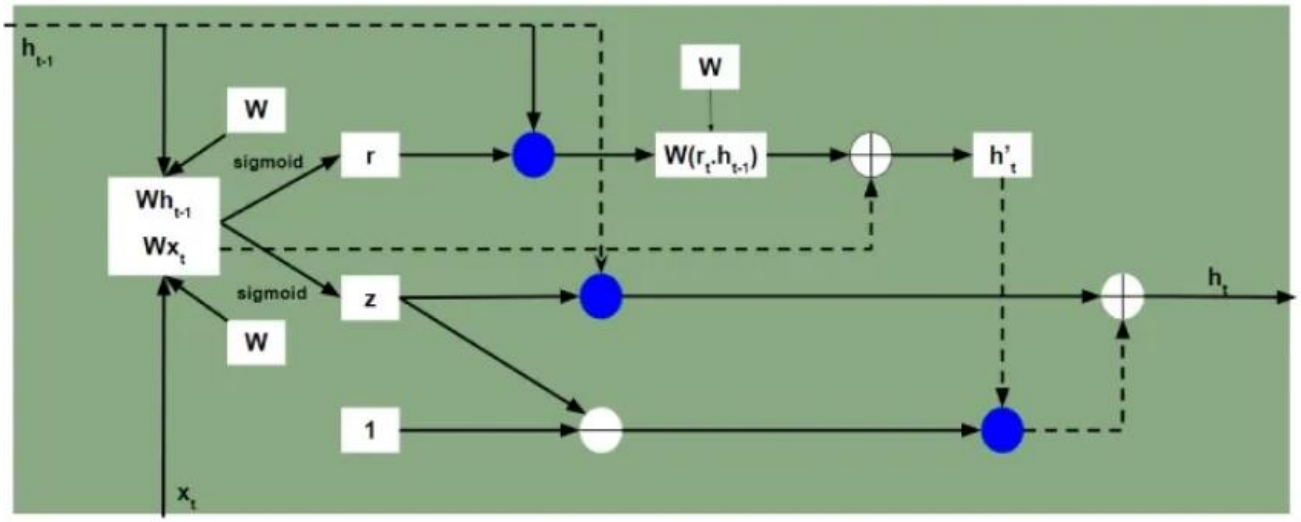
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

The reset gate determines how much of the previous hidden state h_{t-1} should be forgotten.

2. Update gate:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

The update gate controls how much of the new information x_t should be used to update the hidden state.



3. Candidate hidden state:

$$h'_t = \tanh(W_h \cdot [r_t \cdot h_{t-1}, x_t])$$

This is the potential new hidden state calculated based on the current input and the previous hidden state.

4. Hidden state:

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot h'_t$$

The final hidden state is a weighted average of the previous hidden state h_{t-1} and the candidate hidden state h_t' based on the update gate z_t.

GRU vs LSTM

GRUs are more computationally efficient because they combine the forget and input gates into a single update gate. GRUs do not maintain an internal cell state as LSTMs do, instead they store information directly in the hidden state making them simpler and faster.

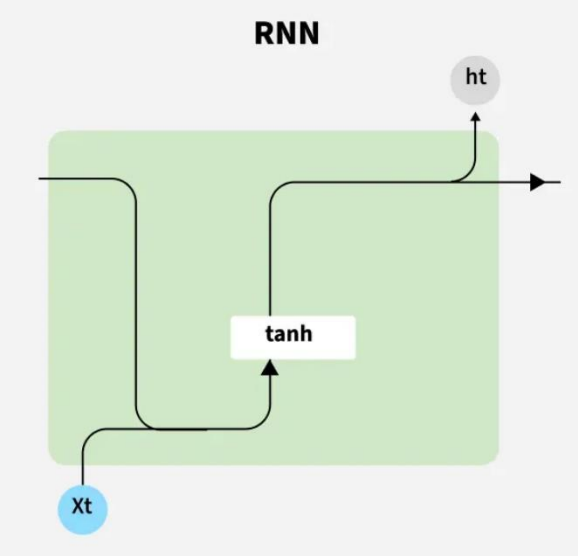
Feature	LSTM (Long Short-Term Memory)	GRU (Gated Recurrent Unit)
Gates	3 (Input, Forget, Output)	2 (Update, Reset)
Cell State	Yes, it has cell state	No (Hidden state only)
Training Speed	Slower due to complexity	Faster due to simpler architecture
Computational Load	Higher due to more gates and parameters	Lower due to fewer gates and parameters
Performance	Often better in tasks requiring long-term memory	Performs similarly in many tasks with less complexity

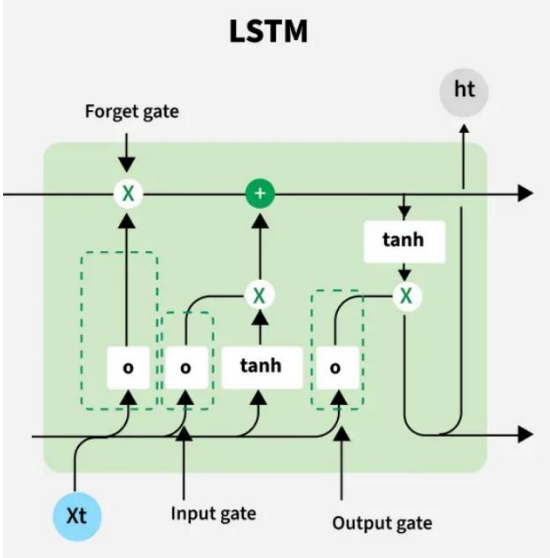
RNN vs LSTM vs GRU vs Transformers

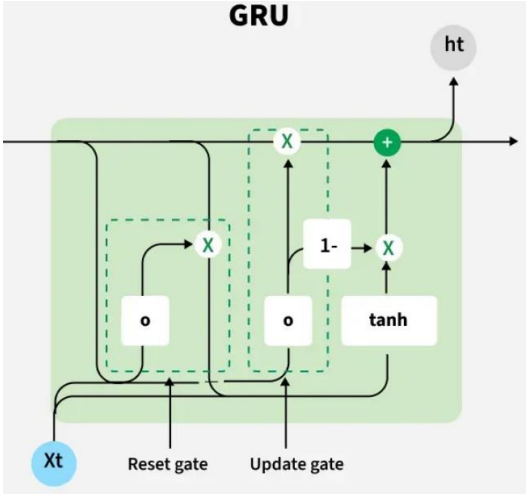
Parameter	RNN (Recurrent Neural Network)	LSTM (Long Short-Term Memory)	GRU (Gated Recurrent Unit)	Transformers
Architecture	Simple structure with loops.	Memory cells with input, forget and output gates.	Combines input and forget gates into update gate; fewer parameters	Utilizes an attention-based mechanism without recurrence.

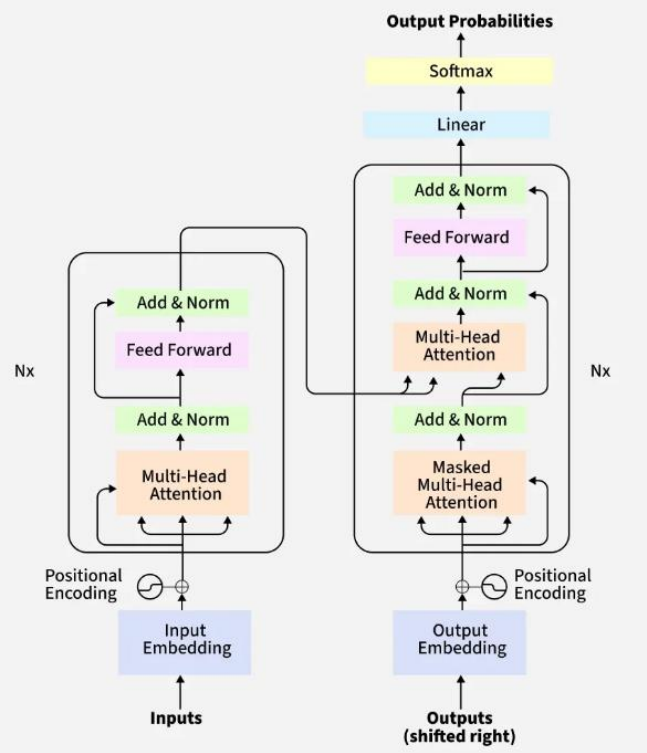
Parameter	RNN (Recurrent Neural Network)	LSTM (Long Short-Term Memory)	GRU (Gated Recurrent Unit)	Transformers
Handling Long Sequences	Struggles to maintain long-term dependencies due to vanishing gradients.	Excels in capturing long-term dependencies.	Better than RNNs but slightly less effective than LSTMs for long-term dependencies.	Effectively manages long sequences using self-attention mechanisms.
Training Time	Fast but less accurate on complex data	Slower due to multiple gates and memory operations	Faster than LSTMs but slower than RNNs.	Requires heavy computation but allows parallel training
Memory Usage	Low memory requirements.	Higher memory consumption due to its complex architecture.	Lower memory usage compared to LSTMs but higher than RNNs.	High memory usage due to multi-head attention and feed-forward layers.
Parameter Count	Fewer parameters overall.	More parameters than RNNs because of gates and memory cells.	Fewer parameters than LSTMs due to its simplified gating structure.	A large number of parameters in multi-head attention layers.
Ease of Training	Prone to vanishing gradients making it	Easier to train for long sequences due to effective	Simpler than LSTMs and easier to train than RNNs.	Requires significant computational

Parameter	RNN (Recurrent Neural Network)	LSTM (Long Short-Term Memory)	GRU (Gated Recurrent Unit)	Transformers
	harder to train on long sequences.	gradient management.		power and require GPUs.
Use Cases	Suitable for simple sequence modeling like stock prices.	Ideal for time-series forecasting, text generation and tasks requiring long-term dependencies.	Similar applications as LSTM but preferred when computational efficiency is important.	Best suited for NLP tasks like translation, summarization, computer vision and speech processing.
Parallelism	Limited due to sequential computation constraints.	Same limitations as RNNs; sequential processing restricts parallelism.	Same limitations as RNNs; sequential processing restricts parallelism.	High parallelism enabled by the attention mechanism and non-sequential design.
Performance on Long Sequences	Poor performance on long sequences.	Good performance on long sequences.	Moderate to good performance on long sequences.	Excellent performance on long sequences.


RNN
ht
tanh
Xt


LSTM
Forget gate
ht
X
+
tanh
X
X
o
o
tanh
o
Xt
Input gate
Output gate


GRU
ht
X
+
X
1-
X
o
o
tanh
Xt
Reset gate
Update gate


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

4.3.2.4. Applications in time-series prediction

Time series forecasting, which analyzes data collected over time to predict future values, has numerous applications across various fields. These include forecasting stock prices, weather patterns, energy consumption, and disease outbreaks, as well as optimizing inventory management and resource allocation.

Applications:

1. Finance:

- **Stock Price Prediction:**

  Time series analysis helps predict future stock prices based on historical data, assisting investors in making informed decisions.

- **Financial Market Forecasting:**

  Predicting trends in stock markets, exchange rates, and bond yields to guide investment strategies.

- **Risk Management:**
  Analyzing financial time series data to assess and mitigate risks associated with market fluctuations.

2. Business & Economics:

- **Sales and Demand Forecasting:**

  Predicting future sales and demand to optimize inventory, supply chain, and pricing strategies.

- **Economic Forecasting:**

  Analyzing economic indicators like GDP, inflation, and unemployment rates to predict future economic conditions.

- **Resource Allocation:**
  Businesses can use time series analysis to optimize resource allocation, such as in production planning and workforce management.

3. Other Applications:

- **Weather Forecasting:**

  Predicting temperature, rainfall, and other weather parameters to prepare for and mitigate weather-related risks.

- **Energy Consumption Forecasting:**

  Analyzing energy usage patterns to optimize energy production and distribution, ensuring a balance between supply and demand.

- **Healthcare:**

  Predicting disease outbreaks, patient volumes, and resource needs to improve healthcare delivery and resource management.

- **Environmental Monitoring:**

  Studying environmental factors like air quality, water levels, and climate patterns to understand trends and inform environmental management.

- **Anomaly Detection:**

  Identifying unusual patterns or outliers in time series data, which can be crucial for fraud detection, network monitoring, and other applications.

- **Signal Processing:**

  Analyzing signals like audio or sensor data to extract meaningful information and predict future signal behavior.

- **Engineering:**

  Analyzing sensor data in various engineering systems to predict potential failures or optimize performance.

- **Social Sciences:**
  Analyzing social trends and patterns to understand social phenomena and inform social policies.

Chapter 5

Model Evaluation and Validation

5.1. Need of Model Evaluation in ML

Model evaluation is essential in machine learning to assess how well a model performs on unseen data, identify potential issues like overfitting or underfitting, and ultimately choose the best model for a specific task. It helps ensure that the model generalizes well and provides reliable predictions in real-world scenarios.

Model evaluation is a process that uses some metrics which help us to analyze the performance of the model.

Model development is a multi-step process and we need to keep a check on how well the model do future predictions and analyze a model's weaknesses. There are many metrics for that. Cross Validation is one technique that is followed during the training phase and it is a model evaluation technique.

5.2. Model Evaluation Metrics

Evaluation is always good in any field, right? In the case of machine learning, it is best practice. In this post, we will cover almost all the popular as well as common metrics used for machine learning.

Model evaluation in machine learning is the process of determining a model's performance via a metrics-driven analysis. It can be performed in two ways:

- **Offline**: The model is evaluated after training during experimentation or continuous retraining.
- **Online**: The model is evaluated in production as part of model monitoring.

The metrics selection for the analysis varies depending on the data, algorithm, and use case.

Performance metrics for model evaluation shed light on:

1. Model efficiency
2. Production readiness of the model
3. Potential performance enhancement with more training data
4. Overfitting or underfitting tendencies of the model

When applied for categorical predictions, ML models can produce four outcomes:

1. True Positives
2. True Negatives
3. False Positives (Type 2 error)
4. False Negatives (Type 1 error)

Term	Description	Meaning in Test/Train Evaluation
TP (True Positive)	Model correctly predicted positive	Example: Test data had a "positive" class (e.g., spam email, cancer detected), and the model predicted "positive".
TN (True Negative)	Model correctly predicted negative	Example: Test data had a "negative" class (e.g., not spam), and the model predicted "negative".
FP (False Positive)	Model predicted positive but it was actually negative	Model incorrectly flagged a **non-spam** email as **spam** — **Type I Error**.
FN (False Negative)	Model predicted negative but it was actually positive	Model missed a **spam** email and predicted **not spam** — **Type II Error**.

From these results, an assortment of performance metrics is available for model assessment.

5.2.1. Classification Metrics

In a classification task, our main task is to predict the target variable, which is in the form of discrete values. To evaluate the performance of such a model, following are the commonly used evaluation metrics:

- Accuracy
- Precision
- Recall
- F1 Score
- Confusion Matrix

5.2.1.1. Accuracy

Accuracy is defined as the ratio of number of correct predictions to the total number of predictions. This is the most fundamental metric used to evaluate the model. The formula is given by:

$$\text{Accuracy} = \frac{\text{No. of correct predictions}}{\text{Total number of input samples}}$$

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

However, Accuracy has a drawback. It cannot perform well on an imbalanced dataset. Suppose a model classifies that the majority of the data belongs to the major class label. It gives higher accuracy, but in general model cannot classify on minor class labels and has poor performance.
Where:

- TP (True Positive): Model correctly predicted the positive class.
- TN (True Negative): Model correctly predicted the negative class.
- FP (False Positive): Model incorrectly predicted positive (actually negative).
- FN (False Negative): Model incorrectly predicted negative (actually positive).

5.2.1.2. Precision, Recall and Fβ score

Precision is the ratio of true positives to the summation of true positives and false positives. It basically analyses the positive predictions.

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

The drawback of Precision is that it does not consider the True Negatives and False Negatives.

Recall is the ratio of true positives to the summation of true positives and false negatives. It basically analyses the number of correct positive samples.

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

The drawback of Recall is that often it leads to a higher false positive rate.

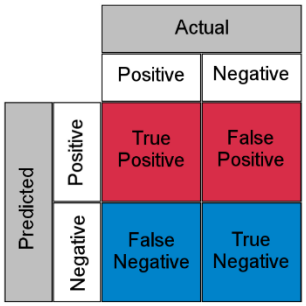
F1 score is the harmonic mean of precision and recall. It is seen that during the precision-recall trade-off if we increase the precision, recall decreases and vice versa. The goal of the F1 score is to combine precision and recall.

Lower recall and higher precision give you great accuracy but then it misses a large number of instances. The more the F1 score better will be performance. It can be expressed mathematically in this way:

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

5.2.1.3. Confusion Matrix

Confusion matrix creates a *N X N* matrix, where N is the number of classes or categories that are to be predicted. Here we have *N = 2*, so we get a *2 X 2* matrix. Suppose there is a problem with our practice which is a binary classification. Samples of that classification belong to either *Yes* or *No*. So, we build our classifier which will predict the class for the new input sample. After that, we tested our model with *165* samples, and we get the following result.



n = 165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

There are 4 terms you should keep in mind:

1. **True Positives:** It is the case where we predicted Yes and the real output was also Yes.
2. **True Negatives:** It is the case where we predicted No and the real output was also No.
3. **False Positives:** It is the case where we predicted Yes but it was actually No.
4. **False Negatives:** It is the case where we predicted No but it was actually Yes.

The accuracy of the matrix is always calculated by taking average values present in the ***main diagonal i.e.***

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total Samples}}$$
$$\text{Accuracy} = \frac{100+50}{165}$$
$$\text{Accuracy} = 0.91$$

Numerical

A binary classifier was tested on **100 samples** and produced the following results:

- **True Positives (TP):** 40
- **False Positives (FP):** 10
- **True Negatives (TN):** 45
- **False Negatives (FN):** 5

Calculate Accuracy, Precision, Recall, F1 Score and Confusion Matrix.

5.2.1.4. ROC and PR-Curve

A Receiver Operating Characteristic **(ROC)** curve is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. It's a visual tool used to assess how well a model distinguishes between two classes (e.g., true/false, diseased/healthy).

Key Components of a ROC Curve:

True Positive Rate:

Also called or termed sensitivity. True Positive Rate is considered as a portion of positive data points that are correctly considered as positive, with respect to all data points that are positive.

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

True Negative Rate

Also called or termed specificity. True Negative Rate is considered as a portion of negative data points that are correctly considered as negative, with respect to all data points that are negatives.

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

False Positive Rate

False Negatives rate is actually the proportion of actual positives that are incorrectly identified as negatives

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

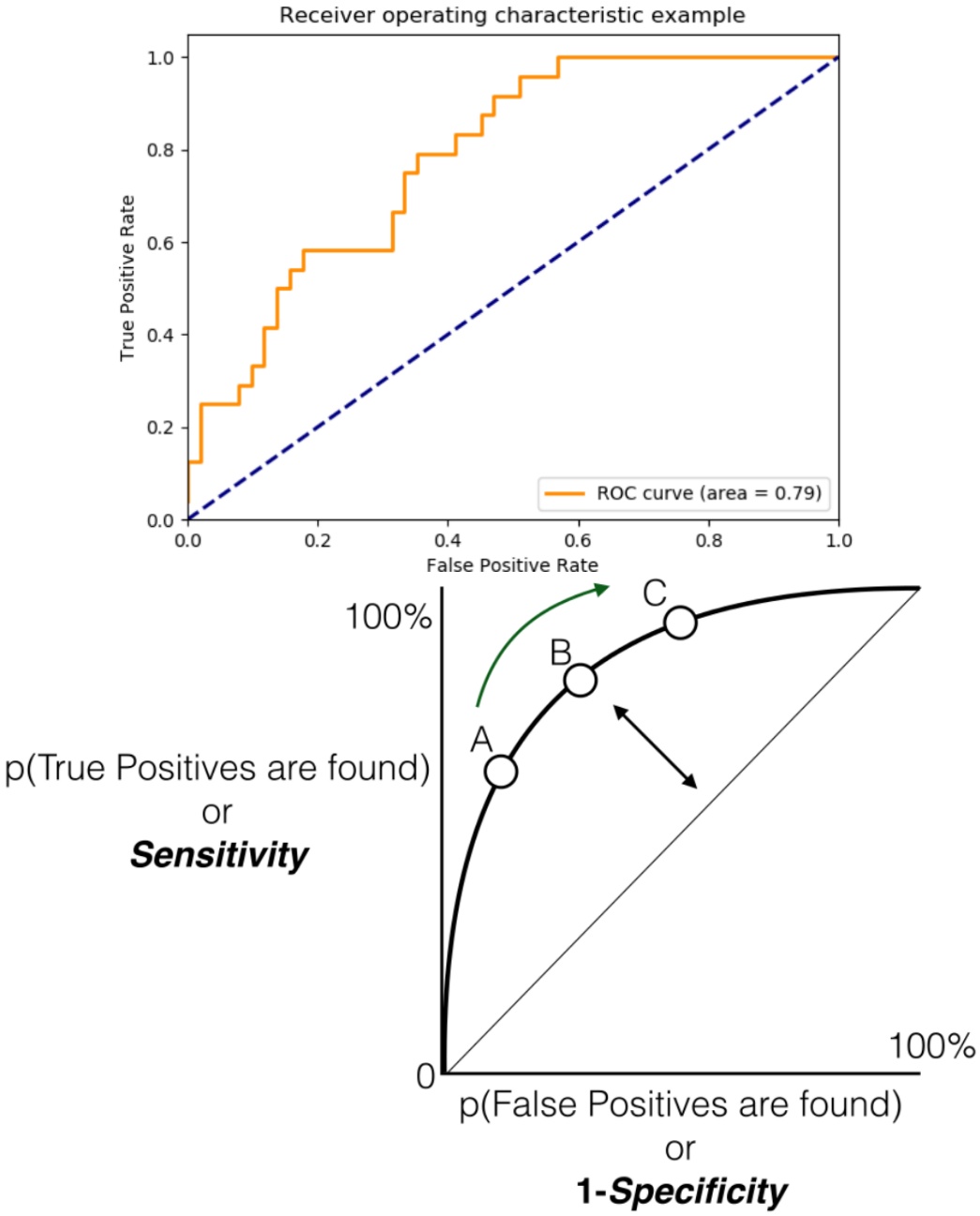
Threshold:
A value used to classify data points. The ROC curve is created by plotting the TPR against the FPR for all possible threshold values.

ROC Curve

It is a graphical representation of the True Positive Rate (TPR) vs the False Positive Rate (FPR) at different classification thresholds. The curve helps us visualize the trade-offs between sensitivity (TPR) and specificity (1 - FPR) across various thresholds. Area Under Curve (AUC) quantifies the overall ability of the model to distinguish between positive and negative classes.

- AUC = 1: Perfect model (always correctly classifies positives and negatives).
- AUC = 0.5: Model performs no better than random guessing.

- AUC < 0.5: Model performs worse than random guessing (showing that the model is inverted).



5.2.2. Regression Metrics

In the regression task, we are supposed to predict the target variable which is in the form of continuous values. To evaluate the performance of such a model below metrics are used:

5.2.2.1. Mean Absolute Error (MAE)

MAE calculates the average of the absolute differences between the predicted and actual values. It gives a clear view of the model's prediction accuracy but it doesn't show whether the errors are due to over- or under-prediction. It is simple to calculate and interpret helps in making it a good starting point for model evaluation.

$$\mathrm{MAE} = \frac{1}{\mathrm{N}} \sum_{\mathrm{j}=1}^{\mathrm{N}} |\mathrm{y_j} - \hat{\mathrm{y}}_\mathrm{j}|$$

Where:

- y_j = Actual value
- $\hat{y}_j$ = Predicted value

5.2.2.2. Mean-Squared Error (MSE)

MSE calculates the average of the squared differences between the predicted and actual values. Squaring the differences ensures that larger errors are penalized more heavily helps in making it sensitive to outliers. This is useful when large errors are undesirable but it can be problematic when outliers are not relevant to the model's purpose.
Formula:

$$\mathrm{MSE} = \frac{1}{\mathrm{N}} \sum_{\mathrm{j}=1}^{\mathrm{N}} (\mathrm{y_j} - \hat{\mathrm{y}}_\mathrm{j})^2$$

Where:

- y_j = Actual value
- $\hat{y}_j$ = Predicted value

5.2.2.3. Root Mean-Squared Error (RMSE)

RMSE is the square root of MSE, bringing the metric back to the original scale of the data. Like MSE, it heavily penalizes larger errors but is easier to interpret as it's in the same units as the target variable. It's useful when we want to know how much our predictions deviate from the actual values in terms of the same scale.
Formula:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{\mathrm{j}=1}^{\mathrm{N}} (\mathrm{y_j} - \hat{\mathrm{y}}_\mathrm{j})^2}{\mathrm{N}}}$$

Where:

- y_j = Actual value
- $\hat{y}_j$ = Predicted value

5.2.2.4. R-Squared (R^2)

R2 score represents the proportion of the variance in the dependent variable that is predictable from the independent variables. An R^2 value close to 1 shows a model that explains most of the variance while a value close to 0 shows that the model does not explain much of the variability in the data. R^2 is used to assess the goodness-of-fit of regression models.

Formula:

$$R^2 = 1 - \frac{\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}{\sum_{j=1}^{n}(y_j - \bar{y})^2}$$

Where:

- y_j = Actual value
- $\hat{y}_j$ = Predicted value

5.2.2.5. Root Mean Squared Logarithmic Error (RMSLE)

RMSLE is useful when the target variable spans a wide range of values. Unlike RMSE, it penalizes underestimations more than overestimations helps in making it ideal for situations where the model is predicting quantities that vary greatly in scale like predicting prices or population.

Formula:

$$\text{RMSLE} = \sqrt{\frac{\sum_{j=1}^{N}(\log(y_j+1) - \log(\hat{y}_j+1))^2}{N}}$$

Where:

- y_j = Actual value
- $\hat{y}_j$ = Predicted value

Numerical Problem Example

Suppose we have the following true values y and predicted values $\hat{y}$:

Instance	y_i (Actual)	$\hat{y}_i$ (Predicted)
1	3	2.5
2	-0.5	0.0
3	2	2
4	7	8

Calculate

Mean Absolute Error (MAE), Mean Squared Error (MSE),

Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE)

5.3. Model Validation Techniques

The process that helps us evaluate the performance of a trained model is called **Model Validation.** It helps us in validating the machine learning model performance on new or unseen data. It also helps us confirm that the model achieves its intended purpose.

Key Components of Model Validation

1. Data Validation

- **Quality:** Dropping missing values, detecting outliers, and errors in the data. This prevents the model from learning from incorrect data or misinformation.
- **Relevance:** Ensuring that the data is a true representation of the underlying problem that the model is designed to solve. Use of irrelevant information may end up leading to wrong conclusions.
- **Bias:** Ensuring that the data has appropriate representation for the model to avoid reproducing biased or inaccurate results. Using methods such as analyzing data demographics and employing unbiased sampling can help.

2. Conceptual Review

- **Logic:** Criticizing the logic of the model and examining whether it is useful for the problem under consideration. This includes finding out if the selected algorithms and techniques are suitable.
- **Assumptions:** Understanding and critically evaluating the assumptions embedded in model building. Expectations that are not based on assumptions can result in inaccurate forecasts.
- **Variables:** Relevance and informativeness of the selected variables about the purpose of the model. Extraneous variables can also lead to poor model predictions.

3. Testing

- **Train/Test Split:** Splitting the data into two – the training set to develop the model and the testing set to assess the model's prediction accuracy on new observations. This helps determine the capability of the model to make correct predictions with new data.

- **Cross-validation:** The basic principle of cross-validation is that the data is divided into a user defined number of folds and each fold is considered as validation set while training on the remaining ones. This gives a better insight to model's performance than the train/test split approach.

Benefits of Model Validation

1. Increased Confidence in Model Predictions

- **Reduced Risk of Errors:** Validation enables the model to avoid making wrong predictions by pointing out issues with the data or the model itself. This ensures more reliable and trustworthy results that you can rely upon to make decisions based on.

- **Transparency and Explainability:** Explanations explain why a model produces a particular outcome. This transparency enables the users to understand how the model arrives at the results which aids in the acceptance of the outputs of the model.

2. Improved Model Performance and Generalizability

- **Prevents Overfitting and Underfitting:** When a model is overly adjusted to fit the training data and fails to predict the new data it is called Overfitting. Underfitting occurs when the model is too weak and cannot capture the true relationships in the data. Validation methods assist in the identification of these issues and suggest corrections to increase the performance of the created model on new data.

- **Optimization for Specific Needs:** Validation allows you to test different model architectures and training hyperparameters to choose the optimal configuration on a particular task. This fine-tuning guarantees that the model is customized to suit your specific requirements.

3. Identification and Mitigation of Potential Biases and Errors

- **Fair and Unbiased Results:** Data can be inherently biased because of the bias in the real world. Validation helps you identify these biases and enables you to address them. This implies that the model will produce outcomes that are not discriminatory or unequal.

- **Early Detection and Correction:** Validation assists in identifying defects in the model's developmental process. This is advantageous because it makes it easier to address problems and address them before they are released into the market.

Model Validation Techniques

1. **Train/Test Split:** Train/Test Split is a basic model validation technique where the dataset is divided into training and testing sets. The model is trained on the training set and then evaluated on the separate, unseen testing set. This helps assess the model's generalization performance on new, unseen data. Common split ratios include 70-30 or 80-20, where the larger portion is used for training.

2. **k-Fold Cross-Validation:** In k-Fold Cross-Validation, the dataset is divided into k subsets (folds). The model is trained and evaluated k times, each time using a different fold as the test set and the remaining as the training set. The results are averaged, providing a more robust evaluation and reducing the impact of dataset partitioning.

3. **Leave-One-Out Cross-Validation:** Leave-One-Out Cross-Validation (LOOCV) is an extreme case of k-Fold Cross-Validation where k equals the number of data points. The model is trained on all data points except one, and the process is repeated for each data point. It provides a comprehensive assessment but can be computationally expensive.

4. **Leave-One-Group-Out Cross-Validation:** This variation considers leaving out entire groups of related samples during each iteration. It is beneficial when the dataset has distinct groups, ensuring that the model is evaluated on diverse subsets.

5. **Nested Cross-Validation:** Nested Cross-Validation combines an outer loop for model evaluation with an inner loop for hyperparameter tuning. It helps assess how well the model generalizes to new data while optimizing hyperparameters.

6. **Time-Series Cross-Validation:** In Time-Series Cross-Validation, temporal dependencies are considered. The dataset is split into training and testing sets in a way that respects the temporal order of the data, ensuring that the model is evaluated on future unseen observations.

5.3.1. Train-Test Split

Divide data into features (X) and labels (y).

The data frame gets divided into X_train,X_test , y_train and y_test. X_train and y_train sets are used for training and fitting the model. The X_test and y_test sets are used for testing the model if it's predicting the right outputs/labels. we can explicitly test the size of the train and test sets.

It is suggested to keep our train sets larger than the test sets.

Train set: The training dataset is a set of data that was utilized to fit the model. The dataset on which the model is trained. This data is seen and learned by the model.

Test set: The test dataset is a subset of the training dataset that is utilized to give an accurate evaluation of a final model fit.

Validation set: A validation dataset is a sample of data from your model's training set that is used to estimate model performance while tuning the model's hyperparameters.

By default, 25% of our data is test set and 75% data goes into training tests.

5.3.2. Cross-Validation

Cross-validation is a technique used to check how well a machine learning model performs on unseen data. It splits the data into several parts, trains the model on some parts and tests it on the remaining part repeating this process multiple times. Finally, the results from each validation step are averaged to produce a more accurate estimate of the model's performance.

The main purpose of cross validation is to prevent overfitting. If you want to make sure your machine learning model is not just memorizing the training data but is capable of adapting to real-world data cross-validation is a commonly used technique.

Types of Cross-Validation

There are several types of cross validation techniques which are as follows:

i. Holdout Validation

In Holdout Validation we perform training on the 50% of the given dataset and rest 50% is used for the testing purpose. It's a simple and quick way to evaluate a model. The major drawback of this method is that we perform training on the 50% of the dataset, it may possible that the remaining 50% of the data contains some important information which we are leaving while training our model that can lead to higher bias.

ii. LOOCV (Leave One Out Cross Validation)

In this method we perform training on the whole dataset but leaves only one data-point of the available dataset and then iterates for each data-point. In LOOCV the model is trained on n−1n−1 samples and tested on the one omitted sample repeating this process for each data point in the dataset. It has some advantages as well as disadvantages also.

- An advantage of using this method is that we make use of all data points and hence it is low bias.
- The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation.
- Another drawback is it takes a lot of execution time as it iterates over the number of data points we have.

iii. Stratified Cross-Validation

It is a technique used in machine learning to ensure that each fold of the cross-validation process maintains the same class distribution as the entire dataset. This is particularly important when dealing with imbalanced datasets where certain classes may be under represented. In this method:

- The dataset is divided into k folds while maintaining the proportion of classes in each fold.
- During each iteration, one-fold is used for testing and the remaining folds are used for training.
- The process is repeated k times with each fold serving as the test set exactly once.

Stratified Cross-Validation is essential when dealing with classification problems where maintaining the balance of class distribution is crucial for the model to generalize well to unseen data.

iv. K-Fold Cross Validation

In K-Fold Cross Validation we split the dataset into k number of subsets known as folds then we perform training on the all the subsets but leave one (k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

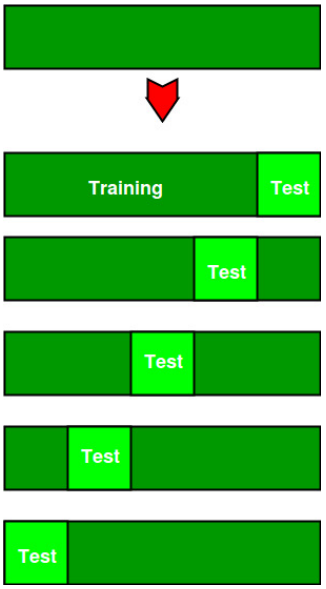
5.3.2.1. K-Fold Cross Validation

In K-Fold Cross Validation we split the dataset into k number of subsets known as folds then we perform training on the all the subsets but leave one (k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

Note: *It is always suggested that the value of k should be 10 as the lower value of k takes towards validation and higher value of k leads to LOOCV method.*

Example of K Fold Cross Validation

The diagram below shows an example of the training subsets and evaluation subsets generated in k-fold cross-validation. Here we have total 25 instances. In first iteration we use the first 20 percent of data for evaluation and the remaining 80 percent for training like [1-5] testing and [5-25] training while in the second iteration we use the second subset of 20 percent for evaluation and the remaining three subsets of the data for training like [5-10] testing and [1-5] and 10-25] training and so on.



Iteration	Training Set Observations	Testing Set Observations
1	[5-24]	[0-4]
2	[0-4, 10-24]	[5-9]
3	[0-9, 15-24]	[10-14]
4	[0-14, 20-24]	[15-19]
5	[0-19]	[20-24]

Each iteration uses different subsets for testing and training, ensuring that all data points are used for both training and testing.

Comparison between K-Fold Cross-Validation and Hold Out Method

Feature	K-Fold Cross-Validation	Hold-Out Method
Definition	The dataset is divided into 'k' subsets (folds). Each fold gets a turn to be the test set while the others are used for training.	The dataset is split into two sets: one for training and one for testing.
Training Sets	The model is trained 'k' times, each time on a different training subset.	The model is trained once on the training set.
Testing Sets	The model is tested 'k' times, each time on a different test subset.	The model is tested once on the test set.
Bias	Less biased due to multiple splits and testing.	Can have higher bias due to a single split.
Variance	Lower variance, as it tests on multiple splits.	Higher variance, as results depend on the single split.

Feature	K-Fold Cross-Validation	Hold-Out Method
Computation Cost	High, as the model is trained and tested 'k' times.	Low, as the model is trained and tested only once.
Use in Model Selection	Better for tuning and evaluating model performance due to reduced bias.	Less reliable for model selection, as it might give inconsistent results.
Data Utilization	The entire dataset is used for both training and testing.	Only a portion of the data is used for testing, so some data is not used for validation.
Suitability for Small Datasets	Preferred for small datasets, as it maximizes data usage.	Less ideal for small datasets, as a significant portion is held out for testing.
Risk of Overfitting	Less prone to overfitting due to multiple training and testing cycles.	Higher risk of overfitting as the model is trained on one set.

Advantages:

1. **Overcoming Overfitting:** Cross validation helps to prevent overfitting by providing a more robust estimate of the model's performance on unseen data.
2. **Model Selection:** Cross validation is used to compare different models and select the one that performs the best on average.
3. **Hyperparameter tuning:** This is used to optimize the hyperparameters of a model such as the regularization parameter by selecting the values that result in the best performance on the validation set.
4. **Data Efficient:** It allow the use of all the available data for both training and validation making it more data-efficient method compared to traditional validation techniques.

Disadvantages:

1. **Computationally Expensive:** It can be computationally expensive especially when the number of folds is large or when the model is complex and requires a long time to train.

2. **Time-Consuming:** It can be time-consuming especially when there are many hyperparameters to tune or when multiple models need to be compared.

3. **Bias-Variance Tradeoff:** The choice of the number of folds in cross validation can impact the bias-variance tradeoff i.e too few folds may result in high bias while too many folds may result in high variance.

5.4. Hyperparameter Tuning

Hyperparameter tuning is the process of selecting the optimal values for a machine learning model's hyperparameters. These are typically set before the actual training process begins and control aspects of the learning process itself. They influence the model's performance its complexity and how fast it learns.

These settings can affect both the speed and quality of the model's performance.

- A high learning rate can cause the model to converge too quickly possibly skipping over the optimal solution.

- A low learning rate might lead to slower convergence and require more time and computational resources.

Techniques for Hyperparameter Tuning

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. The two best strategies for Hyperparameter tuning are:

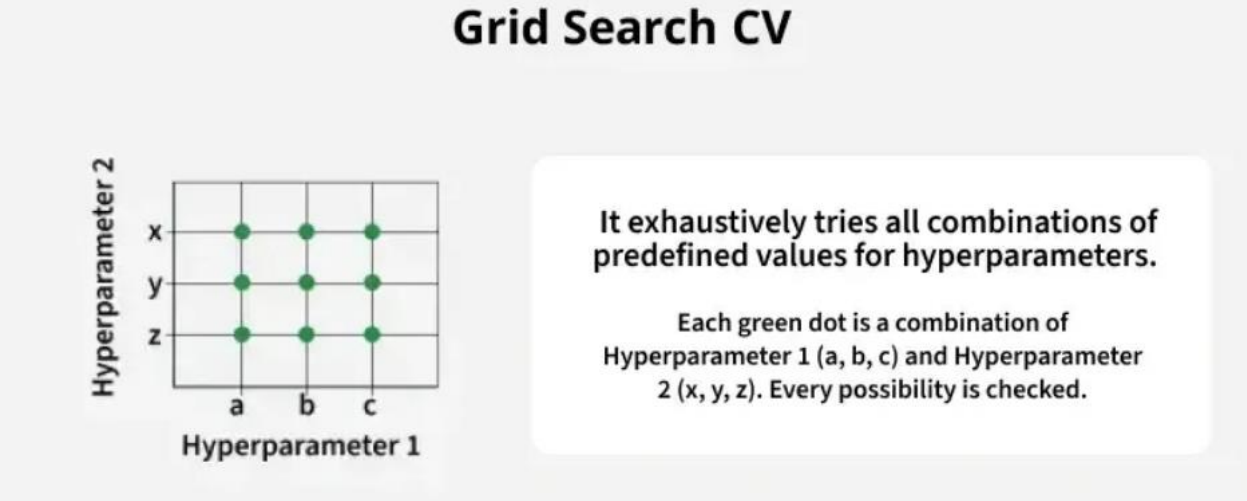
- Grid Search
- Random Search

5.4.1. Grid Search

Grid Search is a brute-force technique for hyperparameter tuning. It trains the model using **all possible combinations** of specified hyperparameter values to find the best-performing setup. It

is **slow and uses a lot of computer power** which makes it hard to use with **big datasets or many settings**. It works using below steps:

- Create a **grid** of potential values for each hyperparameter.
- Train the model for **every combination** in the grid.
- Evaluate each model using **cross-validation**.
- Select the combination that gives the **highest score**.



For example, if we want to tune two hyperparameters C and Alpha for a Logistic Regression Classifier model with the following sets of values:

C = [0.1, 0.2, 0.3, 0.4, 0.5]
Alpha = [0.01, 0.1, 0.5, 1.0]

The grid search technique will construct multiple versions of the model with all possible combinations of C and Alpha, resulting in a total of 5 * 4 = 20 different models. The best-performing combination is then chosen.

C	0.1	0.2	0.3	0.4
0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
0.1	0.698	0.692	0.688	0.675

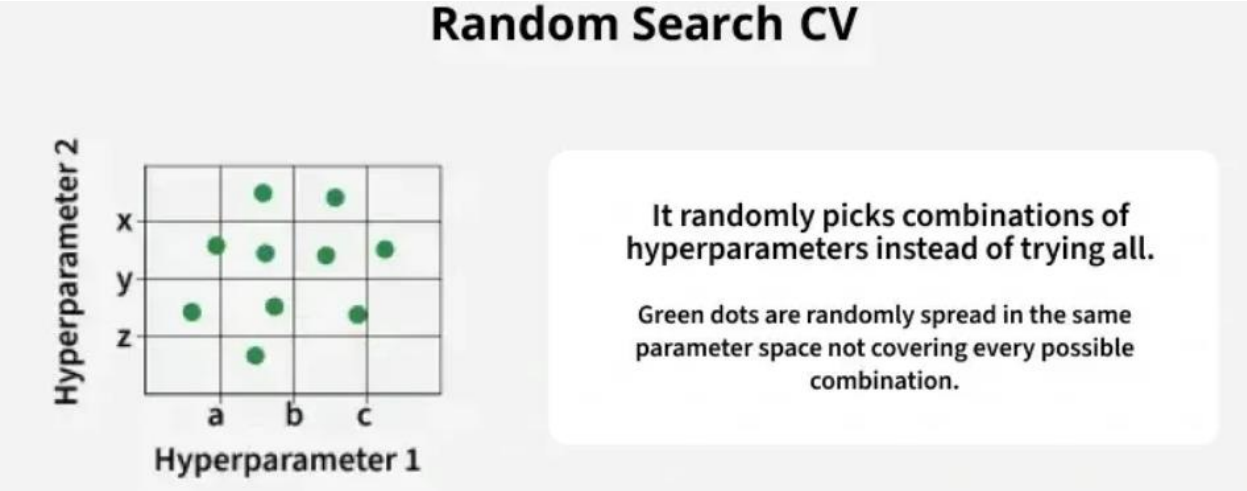
Alpha

5.4.2. Random Search

As the name suggests Randomized Search picks random combinations of hyperparameters from the given ranges instead of checking every single combination like Grid Search.

- In each iteration it **tries a new random combination** of hyperparameter values.

- It **records the model's performance** for each combination.
- After several attempts it **selects the best-performing set**.



Why Randomized Search is better than Grid Search?

One advantage of Randomized Search over Grid Search is that Randomized Search can be more efficient if the search space is large since it only samples a subset of the possible combinations rather than evaluating them all. This can be especially useful if the model is computationally expensive to fit, or if the hyperparameters have continuous values rather than discrete ones. In these cases, it may not be feasible to explore the entire search space using Grid Search.

Another advantage of Randomized Search is that it can be more robust to the risk of overfitting since it does not exhaustively search the entire search space. If the hyperparameter search space is very large and the model is relatively simple, it is possible that Grid Search could overfit to the training data by finding a set of hyperparameters that works well on the training set but not as well on unseen data. Randomized Search can help mitigate this risk by sampling randomly from the search space rather than evaluating every combination.

It is worth noting that both Randomized Search and Grid Search can be computationally expensive, especially if the model is complex and the search space is large. In these cases, it may be necessary to use techniques such as parallelization or early stopping to speed up the search process.

Numerical Problems:

1. Given a dataset with one feature *x* and target *y* you perform simple linear regression and obtain the model y=2x+3. The dataset has 5 points: (1,5), (2,7), (3,9), (4,11), (5,13).
    - Calculate the Mean Squared Error (MSE) for the model on this dataset.

- If ridge regression is applied with a regularization parameter λ=0.1, compute the ridge regression cost function value for the current model parameters (w=2, b=3).

2. You are applying K-Means clustering with k = 2 to a dataset with 4 points in 2D: (1, 1), (2, 2), (5, 5), (6, 6). Initial centroids are C1 = (1, 1) and C2 = (5, 5). Perform one iteration of K-Means clustering and compute the new centroids.
3. A binary classification model predicts outcomes for 100 samples: 60 true positives (TP), 10 false positives (FP), 20 true negatives (TN), and 10 false negatives (FN).
   - Calculate the accuracy, precision, recall, and F1 score for the model.
   - If 5-fold cross-validation is applied, how many samples will be in the training and validation sets for each fold?

4. A K-Nearest Neighbors (KNN) classifier with (k = 3) is applied to a dataset with 3 points in 2D: (1, 1, 1), (2, 2, 0), (3, 3, 1), where the third coordinate is the class label (0 or 1). A decision tree is trained on the same dataset, splitting on feature (x1) at threshold 1.5.

   - For a test point (2.5, 2.5). calculate the Euclidean distances to each training point and predict the class using KNN.
   - Calculate the information gain for the decision tree split on (x1 = 1.5), using entropy.

5. A dataset has 4 points in 2D: (1, 1), (1.5, 1.5), (5, 5), (5.5, 5.5). Apply DBSCAN with (epsilon = 1) and MinPts = 2 (use Euclidean distance).

   - Determine the clusters formed by DBSCAN.
   - If Linear Discriminant Analysis (LDA) is applied to project the points onto a 1D line defined by the vector (1, -1) , compute the projected coordinate of the point (5, 5).

6. A single neuron in a CNN layer applies a convolution operation with a 1D filter (w = [0.5, 0.3]), bias (b = 0.2), and input (x = [1, 2, 3]). The activation function is sigmoid $\sigma = \frac{1}{1+e^{-z}}$. The true output for the first convolution is 0.9.

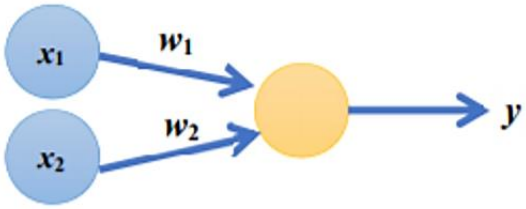
   - Compute the output of the convolution and the sigmoid activation for the first position.
   - Calculate the Mean Squared Error (MSE) loss for the first convolution output.

7. A binary classification model outputs probabilities for 5 samples: true labels ([1, 0, 1, 0, 1]), predicted probabilities ([0.8, 0.3, 0.6, 0.4, 0.9]). The dataset has 100 samples.

   - Calculate the True Positive Rate (TPR) and False Positive Rate (FPR) at a threshold of 0.5.
   - If 10-fold cross-validation is applied, how many samples are in the training and validation sets per fold?

8.

Consider the single layer perceptron network shown in the figure below with inputs, $x_i \in \{0, 1\}$ and activation function $y = f(\sum_i x_i \cdot w_i + b)$ where $f(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{otherwise} \end{cases}$.



(a) Provide values for w_1, w_2, and b to use the perceptron for logical AND.
(b) Provide values for w_1, w_2, and b to use the perceptron for logical OR.
(c) Can this perceptron be used for logical XOR? If so, please provide the corresponding values of w_1, w_2, and b; otherwise, please build a two-layer perceptron network for XOR.

Hint: Truth tables

$y = x_1$ AND x_2

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$y = x_1$ OR x_2

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

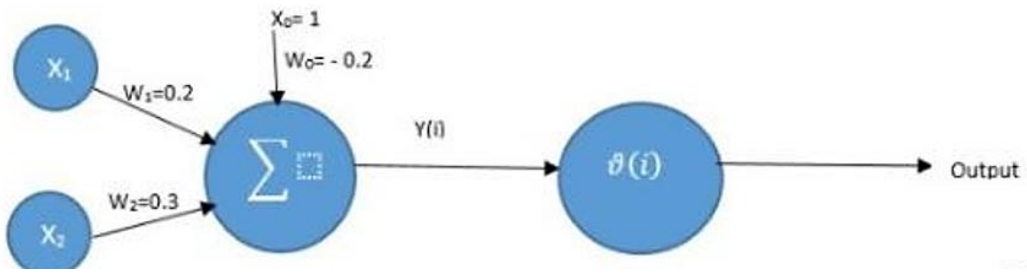
$y = x_1$ XOR x_2

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

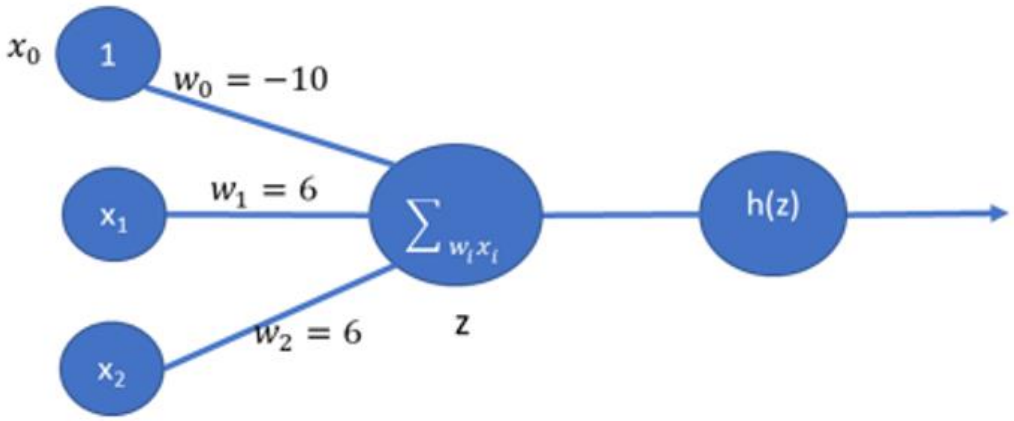
9.

A single layer perceptron neural network is used to classify the 2 input logical gate OR shown in figure Q4. Using as a learning rate of 0.1, train the neural network for the first 3 epochs. use a limiting function:

$$\vartheta(x_i) = \begin{cases} 1 \; if \; Y(i) > 0 \\ 0 \; other \; wise \end{cases}$$



10. Given the following perceptron model:



x_1 and x_2 are binary values input (have only two values 0 or 1). x_0 always equal to 1 (the top leftmost node) is called the bias term. According to the perceptron, $z = \sum w_i x_i$. Then the activation function h(z) is used to determine the output. For this problem, h(z) is defined below:

$$h(z) = 1 \; if \; z > 0; 0 \; otherwise$$

a. Determine if the given perceptron model is equivalent to any of the following logical function: AND, OR, XOR. Explain your answer.

b. Find a set of weight parameters (w0, w1, w2) to make the perceptron simulate the NOR gate on x1 and x2 (NOR = NOT OR).

11.

Neural network basic concepts: A single perceptron (neuron) can represent many Boolean functions (True or False decisions). Assume Boolean values of 1 (true) and -1 (false).

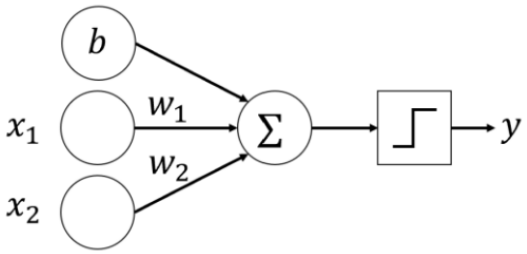


Figure 1. A perceptron.

Where the activation function returns 1 if $f_\Sigma > 0$, and -1 if $f_\Sigma < 0$

a) If $w_1 = 0.8$, $w_2 = 1.0$, $b = -0.5$, calculate the output y for the input vector [1, 0].
b) What could be a set of possible values of $[w_1, w_2, b]$, and the input $[x_1, x_2]$, where x_i can be 1 or 0 if the perceptron is to represent the **AND** function that returns $y = 1$ **if and only if** $[x_1, x_2] = [1,1]$?

12. How a single perceptron can be used to represent the Boolean functions such as AND, OR.
    Let, w0=-0.8 w1=0.5, w2=0.5 for Boolean functions AND
    w0=-0.3 w1=0.5, w2=0.5 for Boolean functions OR
13. Design a two-input perceptron that implements the Boolean function *A ∧ ¬B (A AND NOT B)*. Design at wo-layer network of perceptron's that implements A XOR B.
14. Give the following data, use **PCA** to reduce the dimension from 2 to 1

Feature	Example 1	Example 2	Example 3	Example 4
x	4	8	13	7
y	11	4	5	14

15. Given a perceptron with:

    Learning rate α=0.1

    Initial weights:
    w1=0.5
    w2=0.5
    bias b=0.5

    The activation function is: $$y = \begin{cases} 1 & \text{if } f(x) > 0 \\ 0 & \text{otherwise} \end{cases}$$

    The perceptron is trained on the following logical **AND** dataset:

Input X_1	**Input X_2**	**Target (T)**
0	0	0
0	1	0
1	0	0
1	1	1

Train the perceptron for **1 epoch** (1 pass over the full dataset).
Use the perceptron learning rule:

$$\Delta w_i = \alpha(T - y)x_i, \quad \Delta b = \alpha(T - y)$$

Show updated weights and bias after the full epoch.

16. The XOR truth table is as follows:

Input X_1	Input X_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

a) Explain why a **single-layer perceptron** cannot solve the XOR problem.
b) Design a **Multilayer Perceptron architecture** (number of layers, neurons, activation functions) that can solve the XOR problem. Briefly describe what each layer computes.
c) Assume the hidden layer contains two neurons. Give the expected weights and biases (can be approximate or logical) that would allow the MLP to learn the XOR function.

17. Give Data= {30,40,25,5,10,15,20,30,40,45,35,45,30,10,15,20,25,35,45,50} compute the principal component using **PCA** algorithm
18. Consider the following dataset for weather: Calculate the **information gain** when splitting by the attribute "***Outlook***".

Outlook	Play
Sunny	No
Sunny	No
Overcast	Yes
Rainy	Yes
Rainy	Yes
Rainy	No
Overcast	No
Sunny	Yes
Overcast	Yes
Rainy	Yes

19. Consider the following dataset

x1	**2.5**	**0.5**	**2.2**	**1.9**	**3.1**	**2.3**	**2**	**1**	**1.5**	**1.1**
x2	**2.4**	**0.7**	**2.9**	**2.2**	**3**	**2.7**	**1.6**	**1.1**	**1.6**	**0.9**

Compute the covariance matrix of the standardized dataset.

Calculate the eigenvalues and eigenvectors of the covariance matrix.

Identify the principal components and explain what they represent.

Project the original data onto the first principal component to reduce the data to one dimension.

20. The table below shows a dataset with two features collected from three samples:

Sample	X_1	X_2
A	2	0
B	0	2
C	3	1

a. Perform **mean centering** on the dataset (i.e., subtract the mean of each column from all data points).
b. Compute the **covariance matrix** for the mean-centered data.
c. Explain what the **principal components** represent in PCA and how you would use them to reduce the dimensionality of the dataset.

21. Consider the following dataset about whether to play tennis based on weather conditions:

Instance	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No

- Using the **ID3 algorithm**, calculate the **information gain** for each attribute.
- Draw the **first level** of the decision tree (just the root and its immediate branches) based on your calculations.

22. A bank uses a decision tree to decide whether to approve a loan application. The dataset is shown below:

ID	CreditScore	Income	Student	LoanApproved
1	High	High	No	No
2	High	High	Yes	No
3	Medium	High	No	Yes
4	Low	Medium	No	Yes
5	Low	Low	Yes	Yes
6	Low	Low	No	No
7	Medium	Low	No	Yes
8	High	Medium	No	No

- Using the **ID3 algorithm**, calculate the **information gain** for each attribute.

- Draw the **first level** of the decision tree (just the root and its immediate branches) based on your calculations.

23. The labeled training data is:

Customer	Age	Salary (in $k)	Buy
A	25	40	No
B	30	60	Yes
C	45	80	Yes
D	35	50	No
E	50	100	Yes

Using **K = 3**, classify a **new customer** with:

- Age = **40**
- Salary = **70**

a) Use **Euclidean distance** to calculate the distance between the new customer and each existing one.
b) Based on your results, determine the predicted class (**Buy: Yes or No**). Justify your answer.

24. Given the following 2D dataset with 4 data points:

Point	X	Y
A	2	10
B	2	5
C	8	4
D	5	8

- Explain the objective of the **K-Means clustering algorithm**.
- Using **K = 2**, and initial centroids:
  - Cluster 1 centroid at A (2, 10)
  - Cluster 2 centroid at C (8, 4)
- **Perform one iteration** of the K-Means algorithm:
  - Assign each point to the nearest centroid (using Euclidean distance).
  - Compute the new centroids for the two clusters.
- Explain when the K-Means algorithm **converges**.

25. A dataset contains the following 2D points:

- P1= (1,1)
- P2= (1.5,1.5)
- P3= (5,5)
- P4= (5.5,5.5)

Given:

- ***DBSCAN*** parameters: **epsilon** = 1, **MinPts** = 2
- A projection vector for LDA: **w = (1, -1)**

Calculate:

1. Using Euclidean distance, determine the **clusters formed by DBSCAN**. Clearly identify core points, border points, and noise points.
2. Apply **LDA-style projection**: Project the point (5,5) (5, 5) (5,5) onto the 1D line defined by vector w= (1, −1) w = (1, -1) w= (1, −1). Compute the resulting **1D coordinate**.