UNIT-1

Overview of supervised learning:

Supervised machine learning is a fundamental approach for machine learning and artificial intelligence. It involves training a model using labeled data, where each input comes with a corresponding correct output.

supervised learning is a type of machine learning where a model is trained on labeled data meaning each input is paired with the correct output. The goal of supervised learning is to make accurate predictions when given new, unseen data.

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

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

While training the model, data is usually split in the ratio of 80:20 i.e. 80% as training data and the rest as testing data. In training data, we feed input as well as output for 80% of data. The model learns from training data only.

Types of supervised machine learning:

Regression:

- Regression is used to predict a continuous value (like price, temperature, etc.) based on input features.
- It models the relationship between a dependent variable and one or more independent variables.
- Linear Regression is the simplest form, assuming a straight-line relationship.

Variants include:

- Multiple Linear Regression (multiple inputs)
- Polynomial Regression (curved relationships)
- Ridge/Lasso Regression (to avoid overfitting)

- Widely used in areas like real estate, finance, healthcare, and more.

Classification:

- Classification is used to predict categories or classes (e.g., spam/not spam, pass/fail, disease/no disease).
- The target variable is categorical, not continuous.
- It learns from labeled data to assign new inputs to the correct class.
  Common algorithms include:
  - Logistic Regression
  - Decision Trees
  - Random Forest
  - Support Vector Machines (SVM)
  - K-Nearest Neighbors (KNN)
- Used in applications like email filtering, medical diagnosis, image recognition, etc.

Training a Supervised Learning Model: Key Steps:

- **Defining the Problem:** Clearly identify the real-world problem to be solved.
- **Data Collection:** Gather necessary data from various sources. The data should be related to the problem you're trying to solve.
- **Data Cleaning and Preprocessing:** removing missing, duplicate, and incorrect values. Convert data into a usable format, and encode categorical values.
- **Feature Engineering:** select only the most relevant ones to improve model performance.
- **Splitting Data:** Split the dataset into training and testing sets (e.g., 80% training, 20% testing).
- **Model Selection:** Choose the most appropriate machine learning algorithm based on the type of problem.
- **Model Training:** Train the chosen model using the training data. he model learns patterns and relationships in the data.
- **Model Evaluation:** Test the trained model on the testing data to see how well it performs. Use metrics like accuracy, precision, recall, MAE depending on the task.
- **Model Deployment:** Deploy the final model into a real-world system or application where it can make predictions on new data.

Advantages:

- Gives accurate results when trained with good data.
- Easy to check if the model is working correctly since we already know the answers.
- Works well for real-world problems like spam detection or predicting prices.

Disadvantages:

- Needs a lot of labeled data, which can take time and effort to prepare.
- May not work well on new or unexpected situations.
- Slow with big data, depending on the algorithm.

Linear regression models and least squares

Linear regression is 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 assumes that there is a linear relationship between the input and output. This relationship is represented by a straight line.

For example: we want to predict a student's exam score based on how many hours they studied. We observe that as students study more hours, their scores go up. In the example of predicting exam scores based on hours studied. Here

- **Independent variable (input):** Hours studied because it's the factor we control or observe.
- **Dependent variable (output):** Exam score because it depends on how many hours were studied.

Why we use linear regression

- **Simplicity and Interpretability:** It's easy to understand and interpret, making it a starting point for learning about machine learning.
- **Predictive Ability**: Helps predict future outcomes based on past data, making it useful in various fields like finance, healthcare and marketing.
- **Basis for Other Models:** Many advanced algorithms, like logistic regression or neural networks, build on the concepts of linear regression.
- **Widely Used:** It's one of the most widely used techniques in both statistics and machine learning for regression tasks.

Types of Linear Regression Models:

Simple Linear Regression:

- Involves one independent variable and one dependent variable.
- Fits a straight line to predict the output.
- Example: Predicting salary based on years of experience.
- Equation for simple linear regression: $y=mx+b$
  Where:
  - x is the independent value
  - y is the dependent value
  - **m** is the slope of the line (how much y changes when x changes)
  - **b** is the intercept (the value of y when $x = 0$)

Multiple Linear Regression:

- Involves two or more independent variables.
- Still predicts a single output but considers multiple factors.
- Example: Predicting house price based on size, number of bedrooms, and location.
- Equation for multiple linear regression $y=b_0+b_1x_1+b_2x_2+...+b_nx_n$
  Where
  - y = predicted value
  - $x_1,x_2,...,x_n$ = input variables
  - b_0 = intercept
  - $b_1,b_2,...,b_n$ = coefficients (weights for each input variable)

Polynomial Regression:

- A type of linear regression where the relationship is modeled as a curved line using polynomial terms (e.g., x^2, x^3).
- Useful when data shows a non-linear trend.
- Example: Predicting growth that accelerates over time.

Least Squares:

The least squares method is a mathematical approach used in linear regression to find the best-fitting line through a set of data points. the best-fit line is the straight line that most accurately represents the relationship between the independent variable (input) and the dependent variable (output).

The goal of linear regression is to find a straight line that minimizes the error (the difference) between the observed data points and the predicted values. This line helps us predict the dependent variable for new, unseen data.

Formula: Minimize $\sum(yi-(mxi+b))^2$

Where:

- y_i is the actual value
- mx_i+b is the predicted value
- The goal is to find the best m (slope) and b(intercept)

Applications of Linear Regression:

1. **House Price Prediction:** Estimates house prices using features like size, location, and rooms by learning patterns from past sales data to guide smart buying or selling.
2. **Student Performance Prediction:** Helps predict student exam scores using study time, attendance, and past marks, allowing teachers to support students who may need help.
3. **Medical Cost Estimation:** Estimates treatment costs based on patient age, illness, and treatment type, helping hospitals and insurance companies with planning and budgeting.
4. **Weather Forecasting:** Uses past weather data to predict future conditions like temperature or rain, providing quick and simple weather forecasts for short-term use.

Multiple Regression:

simple linear regression is a statistical method used for predictive analysis. It models the relationship between a dependent variable and a single independent variable by fitting a linear equation to the data. Multiple Regression in Predictive Analytics is a powerful statistical technique used to predict the value of a dependent variable based on the values of two or more independent variables. It's commonly used when the outcome is **continuous** (e.g., income, sales, scores).

General formula: $y=b_0+b_1x_1+b_2x_2+...+b_nx_n$

Where

- y = predicted value
- $x_1,x_2,...,x_n$ = input variables
- b_0 = intercept
- $b_1,b_2,...,b_n$ = coefficients (weights for each input variable)

In multiple regression may encounter the categorical data such as statement (true/false), binary (0/1). Since regression model requires numerical data then the categorical data must be transformed into numerical data this process known as label encoder.

Multicollinearity:

Multicollinearity occurs when two or more independent variables (input) in a regression model are highly correlated with each other. This makes it difficult to determine the individual effect of each variable on the dependent variable (output), leading to unstable or misleading coefficients. It can reduce the reliability of the model's predictions.

Example:

In a real estate scenario, multiple regression can be used to predict house prices based on factors like size of the house, number of bedrooms, type of the house, and distance from the city center. These factors act as independent variables, while the price is the dependent variable. By analyzing these inputs, the model helps estimate accurate property prices, aiding in better pricing, buying, and selling decisions.

Multiple outputs:

We know in simple regression we predict a single output variable but where the goal is to predict multiple target variables instead of just one we use Multi-output regression. Multi-output regression is mostly used for addressing complex real-world problems. This approach is particularly useful when variables are interrelated or when it is more efficient to make joint predictions as the prediction of one variable can affect the prediction of other variables.

General formula: $y1,y2,...,ym=f(x1,x2,...,xn)$

Where

- y1,y2,…..ym are the dependent (output) variables
- x1,x2,…..xn are the independent (input) variables
- f is the regression function

Example:

Multiple output regression can be used to predict both a person's height and weight based on inputs like age, gender, nutrition score, and exercise hours per week. These inputs help the model learn patterns to estimate both outputs at the same time, making it useful in areas like health assessments or fitness tracking where multiple related outcomes need to be predicted together.

Subset selection:

- Subset selection is the way of selecting the subset of the most relevant features from the original features set by removing the redundant, irrelevant, noisy features.
- While developing the machine learning model only a few variables in the dataset are useful for building the model and the rest features are either irrelevant.
- If we use dataset with all these redundant and irrelevant features, it may negatively impact and reduce the overall performance and accuracy of the model.

- Hence it is very important to identify and select the most appropriate features from the data and remove the irrelevant or less important features, which is done with the help of feature selection in machine learning.
- Feature selection is one of the important concepts of machine learning which highly impact performance of the model
- As machine learning works on the concept of “Garbage in Garbage out”, so we always need to input the most appropriate and relevant dataset to the model in order to get a better result.

Subset selection is a technique used in statistics and machine learning to choose a subset of features or variables from larger set. The goal is to identify the most relevant and informative subset that optimizes a certain criterion, model performance.

Ther are various methods for subset selection, there are three types

Forward selection:

- Start with an empty set and iteratively add variables that most improve the model performance.
- Algorithmic steps:
- Evaluate models with the one variable and select the one with the best performance.
- Add one variable at a time, selecting the variable that contributes the most improvement.
- Stop when a certain criterion is met.

Backward elimination:

- Start with all variables and iteratively remove the least valuable variable.
- Algorithmic steps:
- Evaluate the model with all the variables and select the one with the least contribution.
- Remove one variable at a time, excluding the variable with the least impact.
- Stop when the certain criterion is met.

Stepwise selection:

- Combine elements of both forward and backward selection.
- Algorithmic steps:

- Evaluate the model with one variable and select the one with the best performance.
- At each step, add or remove one variable based on the impact.
- Continue until the addition or removal of variable no longer improves the model.

These methods are often used in the context of linear regression or other statistical model, where the goal is to improve model fit or reduce overfitting**.**

Example: predicting housing price

Features:

Square footage(X1)

Number of bedrooms(X2)

Distance to city center (X3)

Presence of nearby schools(X4)

Crime rate in the neighbourhood (X5)

Target variables:

Housing price (Y)

Example outcome:

After the subset selection process, the model identify that square footages(X1), number of bedrooms(X2), and distance to city center(X3) are the most significant features for predicting housing price.

Lasso regression:

Lasso Regression is a regression method is used in regression analysis for variable selection and regularization. It helps remove irrelevant data features. Lasso Regression is a regularization technique used to prevent overfitting. It improves linear regression by adding a penalty term to the standard regression equation. However in real-world datasets features have strong correlations with each other known as multicollinearity where Lasso Regression actually helps.

- Regularization is the technique used to reduce error by fitting the function appropriately on the given training set and avoid overfitting.
- 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).

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

Where

m – number of features (input variables)

n – number of training examples (data points)

y_i – actual target value

$\hat{y}_i$ – predicted target value

w_i – weight (coefficient)

λ - Regularization parameter (controls penalty strength)

Uses of Lasso regression:

1. **Feature Selection:** It automatically selects most important features by reducing the coefficients of less significant features to zero.
2. **Collinearity:** When there is multicollinearity it can help us by reducing the coefficients of correlated variables and selecting only one of them.

3. **Regularization:** It helps preventing overfitting by penalizing large coefficients which is useful when the number of predictors is large.
4. **Interpretability**: This creates another models helps in making them simpler to understand and explain which is important in fields like healthcare and finance.
5. **Handles Large Feature Spaces:** It is effective in handling high-dimensional data such as images and videos.

Ridge Regression:

Ridge Regression is a regression method used in regression analysis to handle multicollinearity and improve model performance. It is also a regularization technique used to prevent overfitting by adding a penalty term to the linear regression Lost function. Unlike Lasso, Ridge Regression doesn't eliminate any features by setting their coefficients to zero. Instead, it reduces the size of the coefficients, making them smaller, while still keeping all input variables in the model.

- Regularization is the technique used to reduce error by fitting the function appropriately on the given training set and avoid overfitting.
- L2 regularization 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

m – number of features (input variables)

n – number of training examples (data points)

y_i – actual target value

$\hat{y}_i$ – predicted target value

w_i – weight (coefficient)

λ - Regularization parameter (controls penalty strength)

Uses of Ridge regression:

Multicollinearity Handling: When features are highly correlated, Ridge reduces model variance without eliminating variables.

Regularization: Helps prevent overfitting, especially when the number of predictors is high compared to the number of observations.

Stability in High Dimensions: Ridge Regression performs well when the number of features is greater than the number of observations, making it suitable for high-dimensional datasets.

Improved Prediction Accuracy: It improves model performance on unseen data by applying a penalty to large coefficient values, leading to more balanced predictions.

All Features Retained: Unlike Lasso, Ridge keeps all features, which is useful when we believe every predictor has some contribution.

Differences

Characteristic	**Ridge Regression**	**Lasso Regression**
Regularization Type	L2 regularization – penalizes squared coefficients	L1 regularization – penalizes absolute coefficients
Feature Selection	Keeps all features (no feature elimination)	Eliminates less important features (coefficients become zero)
When to Use	When all predictors are relevant	When only a few predictors are important
Output Model	Includes all features with smaller weights	Simpler model with only key features
Impact on Prediction	Shrinks coefficients but keeps all	Shrinks some coefficients to zero
Computation	Faster, no feature elimination	Slightly slower due to selection process

Linear Discriminate analysis

Linear discriminant analysis (LDA) is an approach used in supervised machine learning to solve multi-class classification problems. LDA separates multiple classes with multiple features through data dimensionality reduction. This technique is important in data science as it helps optimize machine learning models.

- Linear discriminant analysis, also known as normal discriminant analysis (NDA) or discriminant function analysis (DFA), follows a generative model framework.
- LDA algorithms make predictions by using Bayes to calculate the probability of whether an input data set will belong to a particular output.
- LDA works by identifying a linear combination of features that separates or characterizes two or more classes of objects or events.
- LDA does this by projecting data with two or more dimensions into one dimension so that it can be more easily classified.
- Unlike logistic regression, which is limited to binary classification.
- LDA is thus often applied to enhance the operation of other learning classification algorithms such as decision tree, random forest or support vector machines (SVM).

The origin of linear discriminate analysis:

Linear discriminant analysis (LDA) is based on Fisher's linear discriminant. Fisher's method aims to identify a linear combination of features that discriminates between two or more classes of labelled objects or events. Fisher's method reduces dimensions by separating classes of projected data. Separation means maximizing the distance between the projected means and minimizing the projected variance within classes.

How LDA Works

1. Compute class-wise means and overall mean.
2. Compute within-class scatter matrix (Sw) – measures spread within each class.
3. Compute between-class scatter matrix (Sb) – measures spread between classes.
4. Solve the generalized eigenvalue problem to find directions (linear discriminants) that maximize separation.
5. Project data onto these new axes.
6. Use a decision rule (e.g., Euclidean distance, probability) to classify new points.

Advantages:

- Simple and computationally efficient.
- Works well with normally distributed features and linearly separable classes.
- Helps both in dimensionality reduction and classification.

Disadvantages:

- Assumes normal distribution of features.
- Assumes equal covariance among the classes.
- Less effective when classes are not linearly separable.

Logistic Regression:

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

How It Works:

Logistic regression uses an equation similar to linear regression, but instead of outputting a straight line, it uses a special function called the sigmoid (or logistic) function. This function squeezes the output into a range between 0 and 1, which can then be interpreted as a probability.

The sigmoid function:

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

Formula for sigmoid function :

$$\text{Probability} = \frac{1}{1+e^{-z}}, \quad \text{where } z = b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n$$

Types of Logistic Regression:

The Logistic Regression can be classified into three types:

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

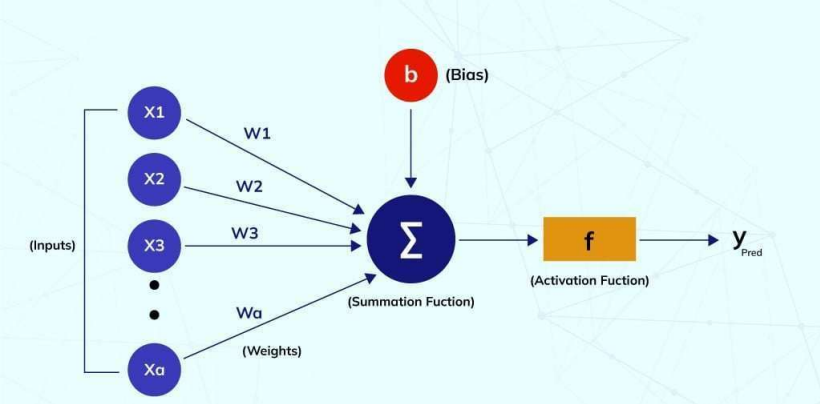
Applications of Logistic Regression:

1. **Email Spam Detection:** Classifies emails as spam or not by analyzing keywords, sender information, and formatting. Helps email systems filter unwanted messages and improve inbox organization for users.
2. **Job Application Screening:** HR systems use logistic regression to predict if a candidate is likely to succeed in a role, based on resume data, experience, and skill-matching scores.
3. **Loan Approval:** Banks use logistic regression to decide loan eligibility by analyzing income, credit score, and repayment history. Helps reduce risk and automate approval processes.
4. **Medical Diagnosis:** Used to predict whether a patient has a disease (Yes/No) based on symptoms, test results, and patient history. For example, predicting diabetes or heart disease.

Perceptron linear algorithm:

The perceptron is one of the simplest artificial neural network architectures Despite being one of the simplest forms of artificial neural networks, the Perceptron model proved to be highly effective in solving specific classification problems.

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).



Components of perceptron:

Input features: The perceptron takes multiple input features, each representing a characteristic of the input data.

Represented as x=[x1,x2,...,xn]

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.

Represented as w=[w1,w2,...,wn]

Bias(b): Bias is a constant value added to the weighted sum of inputs in a perceptron to help shift the decision boundary left or right.

Summation function (Σ):

This part calculates the total weighted input plus bias

$$z=w1x1+w2x2+\cdots+wnxn+b$$

Activation function: An activation function transforms the input signal into an output signal to decide the neuron's action.

Output: The final output is determined by the activation function, often used for binary classification tasks.

The weight update formula is:

$wi,j=wi,j+\eta(yj-y\hat{}j)xi$

Where:

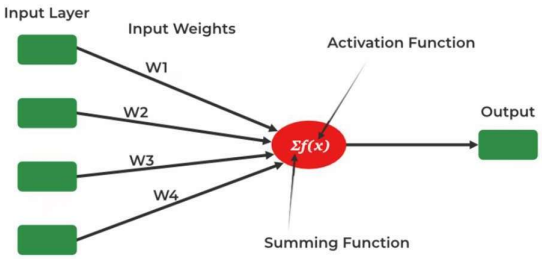
- wi,j is the weight between the ith input and jth output neuron,
- xi is the ith input value,
- yj is the actual value, and y^j is the predicted value,
- η is the learning rate, controlling how much the weights are adjusted.

Types of perceptron

Single layer perceptron:

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.

Perceptron is also known as an artificial neural network. Perceptron is mainly used to compute the logical gate like AND, OR and NOR which has binary input and binary output.

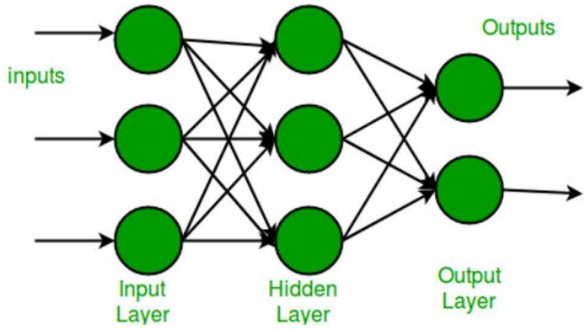


Multi-Layer Perceptron:

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.



In the above diagram 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.

UNIT-2

Bias and variance:

Bias

Bias means the error that happens when a model makes wrong predictions because of incorrect assumptions. It is the difference between the model's predicted value and the actual value. In simple terms, bias is a consistent or systematic error in a model's learning process.

Let Y be the true value of a parameter, and let $\hat{Y}$ be an estimator of Y based on a sample of data. Then, the bias of the estimator $\hat{Y}$ is given by:

$$\text{Bias}(\hat{Y}) = E(\hat{Y}) - Y$$

- **Low Bias:** Low bias value means fewer assumptions are taken to build the target function. In this case, the model will closely match the training dataset.
- **High Bias:** High bias value means more assumptions are taken to build the target function. In this case, the model will not match the training dataset closely.

The high-bias model will not be able to capture the dataset trend. It is considered as the underfitting model which has a high error rate. It is due to a very simplified algorithm.

Variance:

Variance measures how much a model's predictions change when it is trained on different parts of the data. In simple terms, it shows how sensitive a model is to changes in the training data — that is, how much its performance varies with different datasets.

Let Y be the actual values of the target variable and $\hat{Y}$ be the predicted values of the target variable. Then the variance of a model can be measured as the expected value of the square of the difference between predicted values and the expected value of the predicted values.

$$\text{Variance} = E[(\hat{Y} - E[\hat{Y}])^2]$$

where $E[\bar{Y}]$ is the expected value of the predicted values. Here expected value is averaged over all the training data.

Variance errors can be **low** or **high**:

- **Low Variance:** The model is not much affected by changes in the training data and gives consistent results. However, if it's too simple, it may underfit, performing poorly on both training and testing data.
- **High Variance:** The model changes a lot with different training data. It may perform well on training data but poorly on new data, leading to overfitting the model learns the training data too closely and fails to generalize.

Model complexity:

Model complexity refers is a measure of how well a model can capture the underlying patterns in the data. In the context of machine learning, model complexity is often associated with the number of parameters in a model and its ability to fit both the training data and generalize to new, unseen data.

There are two main aspects of model complexity:

- **Simple Models:** Simple models have few parameters, making them less flexible therefore they struggle to capture the complexity of the underlying patterns in the data leading to underfitting, where the model performs poorly on the training data as well as on unseen data.
- **Complex Models:** Complex models have a larger number of parameters, allowing them to represent more intricate relationships in the data. While complex models may perform well on the training data, model tends to overfitting.

Modelling complexity can be influenced by several factors:

1. **Number of Features:** The more attributes or features your model scrutinizes, the higher its complexity is likely to be. Too many features can potentially magnify noise and result in overfitting.
2. **Model Algorithm:** The nature of the algorithm used influences the complexity of the model. For instance, decision trees are considerably simpler than neural networks.
3. **Hyperparameters:** Settings such as the learning rate, number of hidden layers, and regularization parameters can influence the complexity of a machine learning model.

How to Avoid Model Complexity

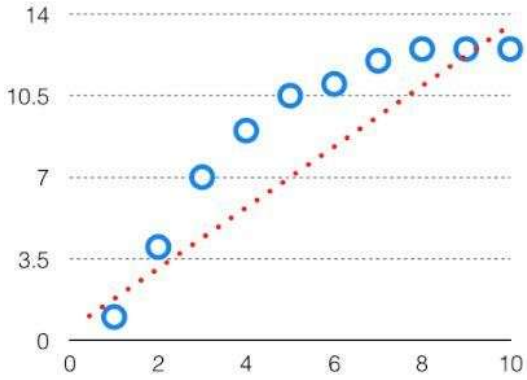
- **Regularization:** Regularization techniques introduce penalties for complexity in the loss function of the model which discourages learning overly complex model parameters, discouraging overfitting. L1 and L2 regularization are common methods to control the magnitude of coefficients, preventing the model from becoming overly complex.
- **Cross-validation:** Cross-Validation is a technique that Assess model generalization and provides a realistic measure of how well the model is likely to perform on unseen data, helping to assess its level of complexity and overfitting.
- **Reducing Features:** By minimizing the number of input features, we could lower the complexity, and thus, prevent overfitting.
- **Split the dataset into training and testing Data**: Splitting your dataset is crucial because it ensures the model doesn't simply memorize the training data and can generalize to unseen examples.

Bias-variance trade off:

Understanding prediction errors like bias and variance is important for achieving accuracy in machine learning. There is a balance, called the bias-variance trade-off, between reducing bias and variance. Knowing these errors helps prevent overfitting and underfitting while training a model.

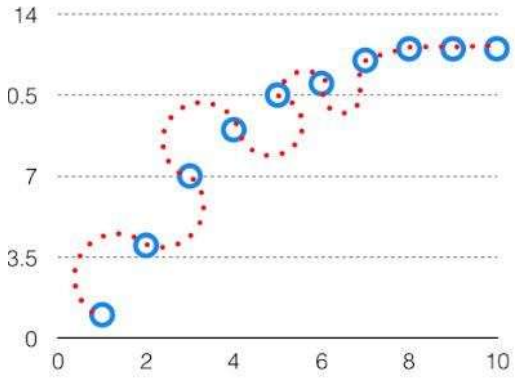
Bias:

Bias means the error that happens when a model makes wrong predictions because of incorrect assumptions. It is the difference between the model's predicted value and the actual value. In simple terms, bias is a consistent or systematic error in a model's learning process.



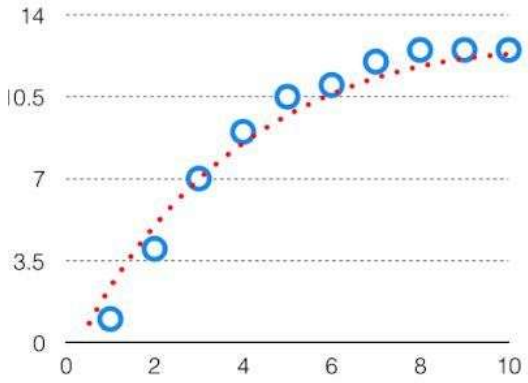
Variance:

Variance measures how much a model's predictions change when it is trained on different parts of the data. In simple terms, it shows how sensitive a model is to changes in the training data — that is, how much its performance varies with different datasets.



Bias-Variance Tradeoff:

If a model is too simple, it has high bias and low variance, leading to more errors. If a model is too complex, it has low bias but high variance, performing well on training data but poorly on new data. The bias-variance tradeoff is about finding the right balance between simplicity and complexity to get the best performance.



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 Bias-Variance Tradeoff highlights the need to balance a model's simplicity and complexity. A good model should not be too simple (causing high bias) or too complex (causing high variance). The goal is to find the optimal point where both bias and variance are minimized, giving the best accuracy on both training and test data.

Handling missing data and outliers

Missing values are common in machine learning and data analysis. They appear as blank cells, null values, or symbols like "NA", "NaN", or "unknown." If not handled well, they can reduce data size, add bias, and lower model accuracy. Since many methods need complete data, dealing with missing values is important to build reliable and accurate models. In this article, we'll explore strategies to handle missing data effectively.

	School ID	Name	Address	City	Subject	Marks	Rank	Grade
0	101.0	Alice	123 Main St	Los Angeles	Math	85.0	2	B
1	102.0	Bob	456 Oak Ave	New York	English	92.0	1	A
2	103.0	Charlie	789 Pine Ln	Houston	Science	78.0	4	C
3	NaN	David	101 Elm St	Los Angeles	Math	89.0	3	B
4	105.0	Eva	NaN	Miami	History	NaN	8	D
5	106.0	Frank	222 Maple Rd	NaN	Math	95.0	1	A
6	107.0	Grace	444 Cedar Blvd	Houston	Science	80.0	5	C
7	108.0	Henry	555 Birch Dr	New York	English	88.0	3	B

Types of missing values

1. **Missing Completely at Random (MCAR):** Data is missing by chance, with no pattern. Example: a random system error.
2. **Missing at Random (MAR):** Missingness depends on other known information, not on the missing value itself. Example: younger people skipping a question.
3. **Missing Not at Random (MNAR):** Missingness depends on the missing value itself. Example: rich people not reporting their income.

Methods for Identifying Missing Data

Detecting and managing missing data is important for data analysis

- **.isnull()** → Checks and identifies missing values.
- **.info()** → Gives a quick summary of data, including missing value counts.
- **dropna()** → Removes rows or columns with missing data.
- **fillna()** → Fills missing values with mean, median, or other methods.

Importance of Handling Missing Values

Handling missing values is important for ensuring the accuracy and reliability of data analysis and machine learning models. Key reasons include:

- **Improved Model Accuracy:** Addressing missing values helps avoid incorrect predictions and boosts model performance.
- **Increased Statistical Power:** Imputation or removal of missing data allows the use of more analysis techniques, maintaining the sample size.
- **Bias Prevention:** Proper handling ensures that missing data doesn't introduce systematic bias, leading to more reliable results.
- **Better Decision-Making:** A clean dataset leads to more informed, trustworthy decisions based on accurate insights.

Outliers

Outliers are data points that are very different from the rest of the dataset. They appear as unusually high or low values that can affect the overall data pattern. For example, if one month's sales are much higher than all other months, that value is an outlier.

Removing Outliers is Necessary

- **Impact on Analysis:** Outliers can distort averages and results, leading to wrong conclusions. Removing them makes the analysis more accurate.
- **Statistical Significance:** Outliers can reduce the reliability of statistical results. Removing them helps keep the analysis valid and meaningful.

Feature engineering and feature scaling:

Feature engineering is the broader process of creating new features or transforming existing ones from raw data, while feature scaling is a specific preprocessing step within feature engineering that brings all numerical features to a common scale.

Feature engineering aims to improve model performance by creating more informative features, while feature scaling ensures that no single feature with a larger magnitude unfairly dominates the learning process of algorithms sensitive to feature scale, such as those using gradient descent or distance metrics.

Feature Engineering

The process of using domain knowledge to create new features, select the most relevant ones, or transform existing features from raw data to improve machine learning model performance.

Purpose: To enhance the information content of features to help the model learn patterns more effectively, reduce noise, and avoid overfitting.

Techniques:

- **Creation**: Creating new variables (e.g., deriving a "price per square foot" feature from "price" and "square footage").
- **Transformation**: Applying mathematical functions or other operations (e.g., taking the logarithm of a skewed feature).
- **Handling Missing Data**: Imputing missing values.
- **Encoding**: Converting categorical variables into a numerical format.
- **Feature Scaling**: A specific technique that is part of feature engineering.

Feature Scaling

A preprocessing technique that rescales numerical features to a common range or scale without changing the distribution of the data.

Purpose: To prevent features with larger values from dominating the model and to help optimization algorithms converge more efficiently.

Techniques:

- **Normalization (Min-Max Scaling)**: Rescales features to a specific range typically 0 to 1, using the formula:
$$X_{scaled} = (X - X_{min})/(X_{max} - X_{min})$$
- **Standardization (Z-score normalization)**: Transforms features to have a mean of 0 and a standard deviation of 1.

Key Relationship and Workflow

- Feature engineering is a broad process that includes feature scaling as a final step.
- A common workflow is: data cleaning → feature engineering (including creating new features and handling missing data) → feature scaling. After scaling, you may also perform feature selection.

- **Why scale after feature engineering?** By scaling after creating new features, you ensure that all final, transformed features are on a similar scale before being fed into the model. This ensures that the scaling process accounts for the new feature values created during feature engineering.

Confusion Matrix:

Confusion matrix is a simple table used to measure how well a classification model is performing. It compares the predictions made by the model with the actual results and shows where the model was right or wrong. This helps you understand where the model is making mistakes so you can improve it. It breaks down the predictions into four categories:

- **True Positive (TP):** The model correctly predicted a positive outcome i.e the actual outcome was positive.
- **True Negative (TN):** The model correctly predicted a negative outcome i.e the actual outcome was negative.
- **False Positive (FP):** The model incorrectly predicted a positive outcome i.e the actual outcome was negative. It is also known as a Type I error.
- **False Negative (FN):** The model incorrectly predicted a negative outcome i.e the actual outcome was positive. It is also known as a Type II error.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

It also helps calculate key measures like accuracy, precision and recall which give a better idea of performance especially when the data is imbalanced.

Metrics based on Confusion Matrix Data

➢ **Accuracy**

Accuracy measures the proportion of correctly predicted instances (both positive and negative) out of all predictions. It gives an overall sense of how often the model is right.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

- **Precision**

  Precision measures how many of the instances predicted as positive are actually positive. It tells you how "precise" or trustworthy your positive predictions are.

$$\text{Precision} = \frac{TP}{TP+FP}$$

- **Recall**

  Recall measures how many of the actual positive instances were correctly identified by the model. It reflects the model's ability to capture all positive cases.

$$\text{Recall} = \frac{TP}{TP+FN}$$

- **F1-Score**

  The F1-score is the harmonic mean of precision and recall. It balances the trade-off between precision and recall, especially useful when classes are imbalanced.

$$\text{F1-Score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Cross validation:

Cross-validation is a resampling technique in machine learning used to assess how accurately a predictive model will perform on an independent dataset. The main idea is to divide the available data into several parts, systematically train the model on some parts, and test it on the others. By repeating this process, cross-validation provides a more stable and unbiased estimate of a model's performance compared to a single train-test split. It is widely applied in tasks such as hyperparameter tuning, model selection, and preventing overfitting, making it one of the most reliable methods for evaluating machine learning models.

- A technique for evaluating model performance using repeated train–test splits.
- Resamples data multiple times to make efficient use of limited datasets.
- Ensures every data point gets a chance to be in both training and testing sets.
- Reduces the risk of overfitting and selection bias.
- Commonly applied for model comparison and hyperparameter optimization.
- Produces a more robust estimate of generalization error.

Methods of cross validation

Hold-Out Method:

The hold-out method is the simplest form of cross-validation. The dataset is split into two parts: one for training the model and the other for testing it. A common split is 70% for training and 30% for testing. The model is trained once and evaluated on the test set to estimate its performance. While this method is fast and easy, the results may vary depending on how the split is done.

- Data is divided into training set and test set.
- Common splits: 70–30 or 80–20.
- Simple and quick.
- Performance may be biased if the split is not representative.

K-Fold Cross-Validation

In k-fold cross-validation, the dataset is divided into *k* equal-sized parts, called folds. The model is trained on *k–1* folds and tested on the remaining fold. This process is repeated *k* times, each time with a different fold as the test set. The final performance is the average of all runs, which gives a more reliable result than a single split. Typical values for *k* are 5 or 10.

- Data is split into k folds (equal parts).
- Each fold gets to be the test set once.
- Final score = average of k runs.
- Reduces variance due to random splitting.
- Commonly used values: k = 5 or 10.

Stratified K-Fold Cross-Validation

Stratified k-fold cross-validation is a variation of k-fold designed for classification problems with imbalanced classes. It ensures that each fold has approximately the same proportion of classes as the overall dataset. This prevents the model from being trained or tested on folds where some classes are missing or underrepresented, leading to a fairer evaluation.

- Special case of k-fold for classification tasks.
- Maintains class proportion in each fold.
- Useful for imbalanced datasets.
- Ensures fair representation of all classes.

Leave-One-Out Cross-Validation (LOOCV)

Leave-one-out cross-validation is an extreme form of k-fold where the number of folds equals the number of samples in the dataset. In this method, the model is trained on all data points except one, which is used as the test set. This process is repeated until every data point has been used for testing. LOOCV makes the most use of the data but is very computationally expensive for large datasets.

- A special case of k-fold where k = n (number of samples).
- Train on n–1 samples, test on 1 sample.
- Repeat for every data point.
- Very thorough, maximizes data usage.
- Very slow for large datasets.

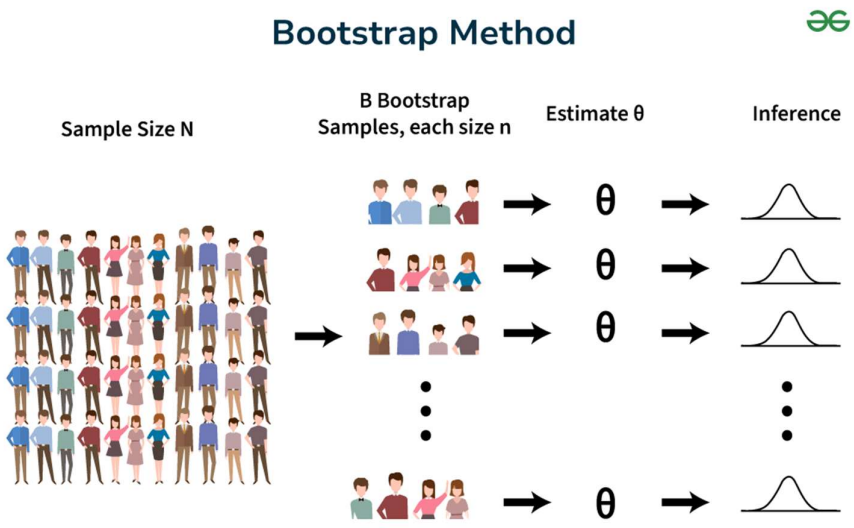
Bootstrap Methods:

Bootstrap Method (Bootstrapping) is a statistical technique that creates many new samples from a single dataset by sampling with replacement. This helps calculate things like standard errors, confidence intervals, and test hypotheses. In machine learning, bootstrapping is used to estimate how well a model will perform on new, unseen data.

The bootstrap approach is an easy and useful alternative to traditional hypothesis testing. Traditional methods rely on the sampling distribution and standard error of a statistic, using a single sample to estimate the population and draw conclusions. In reality, we only observe the sample, not the entire population.

How Bootstrapping Works:

1. Take a sample of size n from the population and call it S.
2. Instead of using theory, create a sampling distribution by resampling from S with replacement m times. Each resample also has n observations.
3. If S represents the population well, resampling m times simulates drawing m samples from the population.
4. The estimates from these resamples approximate the theoretical distribution used in traditional methods.
5. Increasing the number of resamples m does not add new information; the true information depends on the sample size n.
6. More resamples mainly help to get a more accurate estimate of the sampling distribution.



Bootstrap Sampling Process:

1. Choose a sample size from the dataset.
2. If the sample is smaller than needed, randomly pick observations from the dataset.
3. Add each selected observation with replacement (so the same observation can appear multiple times).
4. Repeat this process many times to create multiple small samples.
5. Calculate statistics for each sample and average them to estimate population values.

Advantages of Bootstrap

- **Non-parametric:** Bootstrap doesn't assume a specific data distribution, making it useful for complex or unknown data.
- **Versatile:** It works for many statistics—means, medians, variances, regression coefficients—and for all data types (continuous, discrete, categorical).
- **Better with small samples:** It improves estimates when sample sizes are small by resampling the data.
- **Easy to use:** Bootstrap is simple to implement with modern software by repeatedly resampling the data.
- **Internal validation:** It lets you check the stability and reliability of models without extra data.
- **Handles complex data:** It can work with complicated data, like time-series or hierarchical data, making it widely applicable.

Limitations of Bootstrap Methods

Various limitations of Bootstrap Methods are:

- **Time-Consuming:** Accurate bootstrap requires thousands of simulated samples.
- **Computationally Intensive:** Because bootstrap requires thousands of samples and is time-consuming, it also requires more computing power.
- **Sometimes Incompatible:** Bootstrapping is not always the best solution for your situation, especially when dealing with spatial data or time series.
- **Prone to Bias:** Bootstrapping does not always take into account the variability of the distribution, which introduces errors and bias into your calculations.

Conditional or expected test error:

Conditional test error is the average error of a model on new data from the same distribution, representing its generalization performance on a specific dataset. Expected test error is the average error across all possible training sets, which is a more theoretical and less commonly used metric as it averages over the model-building process itself. In predictive analytics, the more practical goal is usually to estimate conditional test error using techniques like cross-validation, which assesses how well a single, trained model is likely to perform on future, unseen data.

Conditional test error

The expected prediction error of a model on an independent test set. It is a measure of how well a model generalizes to new data from the same underlying distribution.

Purpose: To assess the performance of a specific trained model, as opposed to the entire model-building process.

Estimation: Directly estimated using methods like cross-validation or a validation set, where the model is evaluated on a holdout sample of data that was not used for training.

Expected test error

The average of the conditional test errors over all possible training datasets. It evaluates the entire process of building the model, including both parameter estimation and model selection.

Purpose: To understand the average performance of the model-building strategy itself. A low expected test error indicates that the entire process, on average, yields good predictions for future observations.

Usage: Less often of direct interest than conditional test error because the goal is typically to evaluate one specific model, not the average performance across all possible models that could have been built.

Feature	**Conditional Test Error**	**Expected Test Error**
Focus	Evaluates a *specific* model trained on a *specific* dataset	Evaluates the *learning algorithm* as a whole
Source of randomness	Randomness from the *test data* only	Randomness from both *training* and *test data*
Interpretation	What you care about *after* training a model	What you care about *before* choosing a model or algorithm
Dependency	Depends on the particular training set used	Averages over all possible training sets (hypothetical)
Use in practice	Useful for *model selection* and *model evaluation*	Useful in *theoretical analysis* of algorithms

UNIT-3

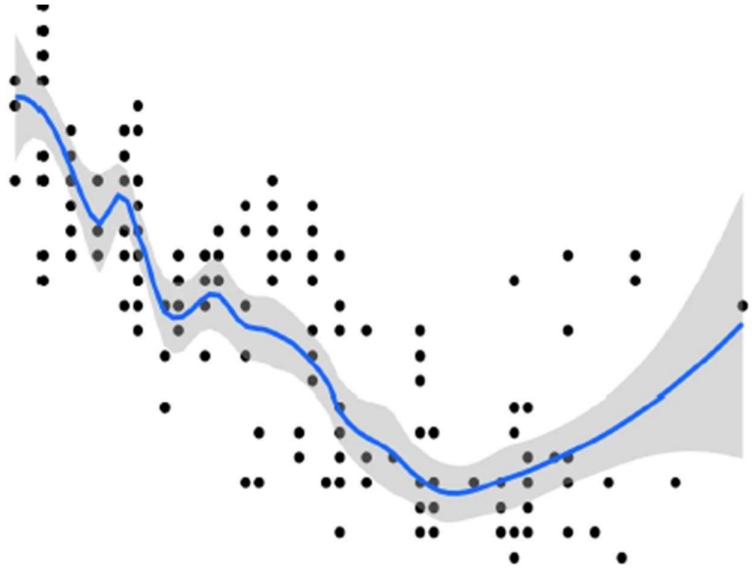
Generalized additive models:

Generalized Additive Models (GAMs) are a versatile statistical modelling technique used in predictive analytics to understand and predict relationships between variables. These additive models are more flexible than traditional linear models because they can handle non-linear patterns in the data. linear models, only capture linear relationship between the input features and target variables.

The generative additive model combining multiple **smooth functions** of predictor variables. This means Generalized Additive Models can model real-world problems more accurately where the effect of an input variable on the output is not just a straight line.

In a regular linear model, we assume that the relationship between the input factors (like study hours, number of practice tests, or sleep time) and the output (like exam score or performance level) is a straight line. But in real life, this relationship is not always straight or simple.

- Generalized Additive Models (GAMs) solve this problem by allowing each input variable to have its own **smooth curve** instead of a straight line.
- Instead of using a straight-line formula like:
  Y = b0+b1x1+b2x2+...+bnxn
- GAMs use:
  Y = b0 + f1(x1) + f2(x2) + ...+fnxn
- Here, f1(x1), f2(x2) are smooth, flexible functions that better match real-world data patterns.
- This makes GAMs more accurate and realistic for predicting outcomes.



Uses of GAM:

Handle Complex Relationships: GAMs are very helpful when the effect of a variable is not linear or straightforward. For example, customer spending may increase with income up to a point and then slow down a curve, not a line.

Interpretability: Even though GAMs use smooth curves instead of straight lines, they are still easy to explain. GAMs are still easy to interpret because they show how each variable individually affects the prediction. This is useful for business users and domain experts.

Better Accuracy: Since they can model real-world patterns more naturally, GAMs often provide better predictions than simple models.

Applications of GAMs

- **Healthcare:** To predict patient risk based on age, blood pressure, etc.
- **Finance:** To forecast stock trends or credit risk based on historical data.
- **Marketing:** To understand how different factors like price, discount, or season affect sales.
- **Environment:** To model pollution levels or temperature patterns over time.

GAMs are a powerful tool in predictive analytics. They offer a good balance between flexibility and clarity. They are especially useful when data shows curves or unusual patterns, and we still want to understand the role of each input variable. Because of this, they are widely used in practical, real-world data science problems.

Classification tree:

A Classification Tree is a type of decision tree used when the target variable is categorical. It splits the dataset into subsets based on input features to predict a class label (e.g., "Yes" or "No", "Spam" or "Not Spam"). The tree grows by maximizing the purity of classes in each node using measures like Gini Index or Entropy.

weekend	weather	parents	money	decision
w1	sunny	yes	rich	cinema
w2	sunny	no	rich	tennis
w3	windy	yes	rich	cinema
w4	rainy	yes	poor	cinema
w5	rainy	no	rich	stay in
w6	sunny	no	rich	tennis

In the above dataset available features are weekend, weather, parents, money, decision

- The input features are considered as weather, parents, money.
- The target variable are decision.

Compute Gini index for the overall collection of training examples

There are three possible output variables cinema, tennis, stay in.

The data has 3 instances of cinema, 2 instances of tennis, 1 instance of stay in.

$$Gini=1-\sum pi2$$

Gini index for decision

$$Gini=1-[(3/6)2+(2/6)2+(1/6)2]=1-[0.25+0.111+0.027]$$

$$=1-0.388$$

$$= 0.612$$

Gini index for weather

Weather has 3 instances sunny, rainy, windy

Weather = sunny

Sunny → w1, w2, w6 → [cinema, tennis, tennis]

$$Gini = 1 - [1/3^2 + 2/3^2]$$

$$=1 - [0.111 + 0.444] = 0.445$$

Weather = rainy

Rainy → w4, w5 → [cinema, stay in]
Gini = $1 - [0.5^2 + 0.5^2] = 0.5$

Weather = windy

Windy → w3 → [cinema]
Gini = 0

$$\text{Gini split} = (3/6)(0.445)+(1/6)(0)+(2/6)(0.5)$$

$$=0.222+0+0.167$$

$$=0.389$$

Gini index for parent

Parent has two instances yes, no

Parent = yes

Yes → w1, w3, w4 → [cinema, cinema, cinema] → Gini = 0

Parent = no

No → w2, w5, w6 → [tennis, stay in, tennis]

Gini = $1 - [4/9 + 1/9] = 4/9 = 0.444$

$$\text{Gini split} = (3/6)(0)+(3/6)(0.444)$$

$$= 0.222$$

Gini index for money

Money has two instances poor, rich

Money = poor

Poor → w4 → [cinema] → Gini = 0

money = rich

Rich → w1, w2, w3, w5, w6 → [cinema, tennis, cinema, stay in, tennis]

Gini = $1 - [0.16 + 0.16 + 0.04] = 0.64$

$$\text{Gini Split} = (5/6)(0.64) + (1/6)(0)$$

$$= 0.533$$

Gini split for weather = 0.38

Gini split for parent = 0.22

Gini split for money = 0.53

Where comparing to three features parents has less Gini value then the parent feature will be considered as root node

Parents

/ \

Yes No

[cinema] ?

Left (Yes): w1, w3, w4 → All cinema

Right (No): w2, w5, w6 → [tennis, stay in, tennis]

When compare to weather and money the weather has less Gini value then the weather feature can be consider as child node

Then the classification tree will be look like

Parents

/ \

Yes No

[cinema] Weather

/ \

Sunny Rainy

[tennis] [stay in]

Regression tree:

A regression tree is a supervised machine learning model that uses a tree-like structure to predict continuous numerical values by splitting the data into subsets based on feature values, minimizing the variance (usually using Standard Deviation Reduction) at each step.

day	outlook	temp	wind	golf players
d1	sunny	hot	weak	25
d2	sunny	mild	weak	35
d3	overcast	hot	weak	30
d4	rain	mild	strong	20
d5	overcast	cool	strong	30
d6	rain	cool	weak	25

In the above dataset available features are day, outlook, temp, wind, golf player.

- The input features are considered as outlook, temp, wind.
- The target variable are considered as golf players.

Compute standard deviation for the overall collection of training examples.

Target: golf players = [25, 35, 30, 20, 30, 25]

Average = (25 + 35 + 30 + 20 + 30 + 25) / 6 = **27.5**

Standard Deviation (SD):

$$SD = \sqrt{\frac{1}{n}\sum(x_i - \bar{x})^2}$$

Standard deviation for golf players:

$$SD = \sqrt{\frac{(25-27.5)^2 + (35-27.5)^2 + (30-27.5)^2 + (20-27.5)^2 + (30-27.5)^2 + (25-27.5)^2}{6}}$$
$$= \sqrt{\frac{6.25 + 56.25 + 6.25 + 56.25 + 6.25 + 6.25}{6}}$$
$$= \sqrt{\frac{137.5}{6}} \approx \sqrt{22.92} \approx 4.79$$

$SD_{parent} = 4.79$

Standard deviation for outlook:

Outlook has 3 instances sunny, overcast, rain

Outlook = Sunny → d1 (25), d2 (35)

Average =25+35/2 = 30

$$SD = \sqrt{[(25-30)^2 + (35-30)^2]/2}$$

$$= \sqrt{[25 + 25]/2}$$

$$= \sqrt{25}$$

$$= 5$$

Outlook = overcast → d4 (20), d6 (25)

Average = 20+25/2 = 22.5

$$SD = \sqrt{[(20-22.5)^2 + (25-22.5)^2]/2}$$

$$= \sqrt{[6.25 + 6.25]/2}$$

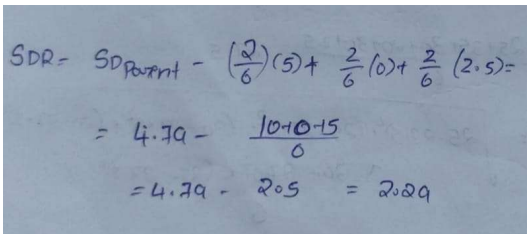
$$= 2.5$$

Outlook = rain → d3 (30), d5 (30)

Average = 30+30/2 = 30

$$SD = 0$$

SDR (Standard Deviation Reduction):



Standard deviation for temp

Temp has 3 instances hot, cold, mild

Temp = hot → d1 (25), d3 (30)

Average = 25+30/2 = 27.5

$$SD = \sqrt{[(25-27.5)^2+(30-27.5)^2]/2}$$

$$= \sqrt{[6.25+6.25]/2}$$

$$= 2.5$$

Temp = cold → d5 (30), d6 (25)

Average = 30+25/2 = 27.5

$$SD = \sqrt{[(30-27.5)^2+(25-27.5)^2]/2}$$

$$= \sqrt{[6.25+6.25]/2}$$

$$= 2.5$$

Temp = Mild → d2 (35), d4 (20)

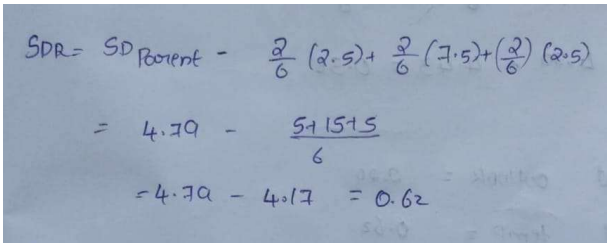
Average = 35+20/2 = 27.5

$$SD = \sqrt{[(35-27.5)^2+(20-27.5)^2]/2}$$

$$= \sqrt{[7.25+7.25]/2}$$

$$= 7.5$$

SDR (Standard Deviation Reduction):



Standard deviation for wind:

Wind has 2 instances weak, strong

Wind = weak → d1 (25), d2 (35), d3 (30), d6 (25)

Average = 25+35+30+25/4 = 28.75

$$SD = \sqrt{[(25-28.75)^2+(35-28.75)^2+(30-28.75)^2+(25-28.75)^2]/24}$$

$$= \sqrt{[14.06 + 39.06 + 1.56 + 14.06]/4}$$

$$=\sqrt{68.75/4}$$

$$=\sqrt{17.19} \approx 4.14$$

Wind = strong → d4 (20), d5 (30)

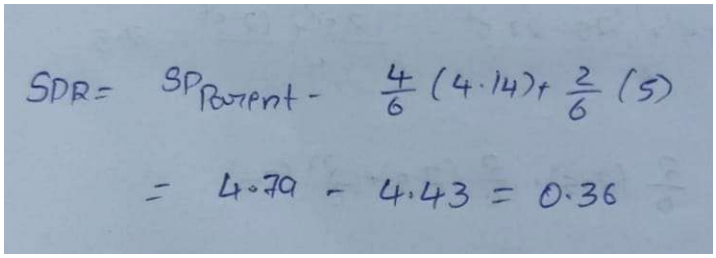
Average = 20+30/2 = 25

$$SD = \sqrt{[(25-27.5)^2+(30-27.5)^2]/2}$$

$$= \sqrt{[(5)^2+ (5)^2/2}$$

$$= 5$$

SDR (Standard Deviation Reduction):

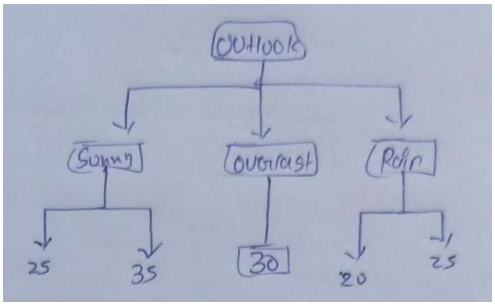


SDR for outlook = 2.29

SDR for temp = 0.62

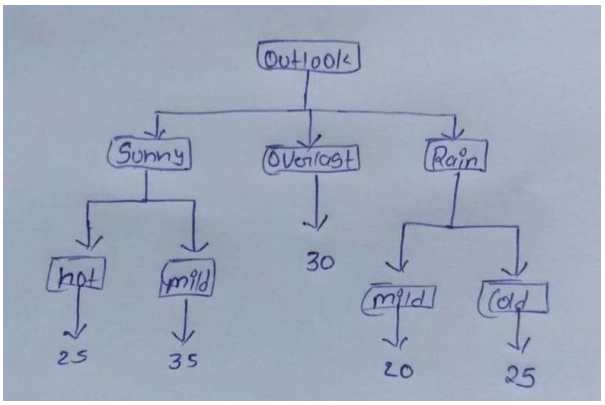
SDR for wind = 0.36

Where comparing to three features outlook has high SD value then the outlook feature will be considered as root node.



Now the sunny and rain has multiple outcomes so now compare the remaining feature of wind, temp. When comparing these two SD values the temp have high SD value then the temp has been considered has child node.

Then the regression tree looks like



Boosting methods-exponential loss and AdaBoost:

Boosting is an ensemble learning technique that sequentially combines multiple weak classifiers to create a strong classifier. It is done by training a model using training data and is then evaluated. Next model is built on that which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly.

Boosting is like a sports coach:

In a football camp, the coach watches who made mistakes and gives them more attention in the next practice. This continues every day the coach keeps focusing on weaker players until the whole team improves.

Exponential Loss – Formal Definition

Exponential loss is a commonly used loss function in boosting algorithms, especially AdaBoost. It measures the error between the true class label and the predicted score by penalizing misclassifications exponentially.

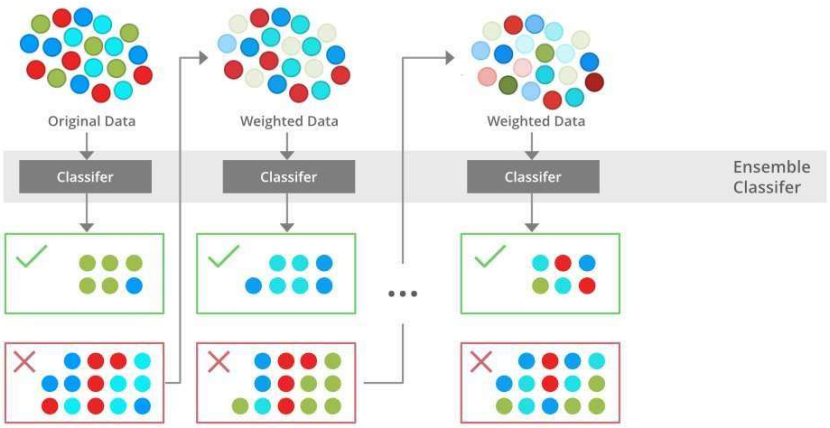
For a binary classification problem where the class labels are yi $\in$ {−1,+1} and the predicted function is f(xi), the exponential loss is defined as:

$$L(y_i, f(x_i)) = e^{-y_i f(x_i)}$$

This loss function assigns larger penalties to incorrect predictions, thereby encouraging the model to focus more on misclassified or difficult examples. It forms the theoretical basis for the AdaBoost algorithm.

Adaboost and its working:

AdaBoost (Adaptive Boosting) is a boosting technique that assigns equal weights to all training samples initially adjusts these weights by focusing more on misclassified datapoints for next model. It effectively reduces bias and variance making it useful for classification tasks but it can be sensitive to noisy data and outliers.



Step 1: Initial Model (B1)

- The dataset consists of multiple data points (red, blue and green circles).
- Equal weight is assigned to each data point.
- The first weak classifier attempts to create a decision boundary.
- Some data points are wrongly classified.

Step 2: Adjusting Weights (B2)

- The misclassified points from B1 are assigned higher weights (shown as darker points in the next step).
- A new classifier is trained with a refined decision boundary focusing more on the previously misclassified points.
- Some previously misclassified points are now correctly classified.
- few data points are wrongly classified.

Step 3: Further Adjustment (B3)

- The newly misclassified points from B2 receive higher weights to ensure better classification.
- The classifier adjusts again using an improved decision boundary and some data points remain misclassified.

Step 4: Final Strong Model (B4 - Ensemble Model)

- The final ensemble classifier combines B1, B2 and B3 to get strengths of all weak classifiers.
- By aggregating multiple models the ensemble model achieves higher accuracy than any individual weak model.

Types of boosting algorithms:

Gradient boosting: Gradient Boosting constructs models in a sequential manner where each weak learner minimizes the residual error of the previous one using gradient descent. Instead of adjusting sample weights like AdaBoost Gradient Boosting reduces error directly by optimizing a loss function.

XGBoost: XGBoost is an optimized version of Gradient Boosting that uses regularization to prevent overfitting. It is faster, efficient and supports handling both numerical and categorical variables.

CatBoost: CatBoost is particularly effective for datasets with categorical features. It employs symmetric decision trees and a unique encoding method that considers target values, making it superior in handling categorical data without preprocessing.

Advantages of Boosting

- **Improved Accuracy**: By combining multiple weak learners it enhances predictive accuracy for both classification and regression tasks.
- **Robustness to Overfitting**: Unlike traditional models it dynamically adjusts weights to prevent overfitting.
- **Handles Imbalanced Data Well**: It prioritizes misclassified points making it effective for imbalanced datasets.

Numerical optimization via gradient boosting:

Gradient Boosting is a ensemble learning method used for classification and regression tasks. It is a powerful boosting algorithm that combine multiple weak learner to create a strong predictive model. It works by sequentially training models where each new model tries to correct the errors made by its previous model. In gradient boosting each new model is trained to minimize the loss function such as mean squared error or cross-entropy of the previous model using gradient descent.

In each iteration the algorithm computes the gradient of the loss function with respect to predictions and then trains a new weak model to minimize this gradient Predictions of the new model are then added to the ensemble (all models prediction) and the process is repeated until a stopping criterion is met.

Gradient descent: Gradient descent is a method used to improve machine learning models like linear regression, logistic regression, SVMs, and neural networks. It works by slowly changing the model's values (parameters) to reduce errors between predicted and actual results, helping the model learn better.

Numerical Optimization Concept

Numerical optimization is about minimizing (or maximizing) a mathematical function when there's no closed-form solution. In Gradient Boosting, we numerically minimize the loss function:

- Common loss functions:
    - Mean Squared Error (for regression)
    - Binary Cross-Entropy (for classification)

Working of gradient boosting:

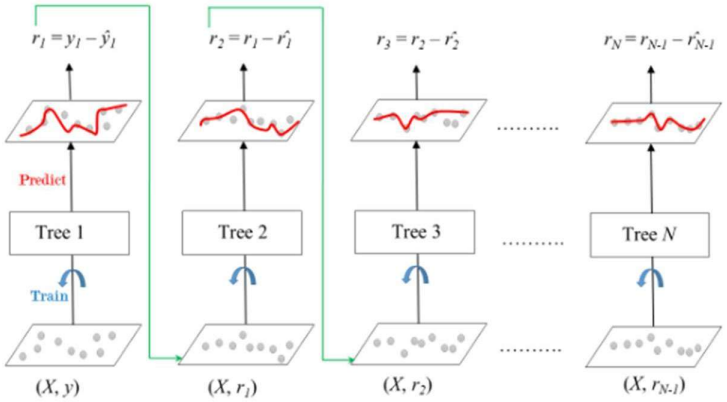
1. Sequential Learning Process

- Gradient Boosting builds models one after another, each trying to correct the errors made by the previous one.
- Initially, the first tree (Tree 1) is trained using the input features X and actual labels y.
- It makes predictions y^1, and the error/residual is calculated:

$$r1=y-y^1$$

2. Residuals Calculation

In the second iteration **Tree 2** is trained using the feature matrix x*x* and the errors from Tree 1 as labels. This means Tree 2 is trained to predict the errors of Tree 1. This process continues for all the trees in the ensemble. Each subsequent tree is trained to predict the errors of the previous tree.



3. Shrinkage (Learning Rate η)

- After each tree predicts errors, its output is scaled by a learning rate η ($0 < \eta \leq 1$).
- This helps control overfitting by reducing how much each tree influences the final prediction.
- The final model prediction is the sum of all the trees' scaled outputs:

$$y_{pred} = y1 + \eta \cdot r1 + \eta \cdot r2 + \cdots + \eta \cdot rN$$

Where r1,r2,…,rN are the errors predicted by each tree.

In short summary

- Train tree 1 on (X,Y) and get initial predictions
- Compute residuals r1=y−y^1
- Train Tree 2 on (X,r1), get r2, and so on
- Multiply each prediction by learning rate η
- Add all trees' predictions to get final output

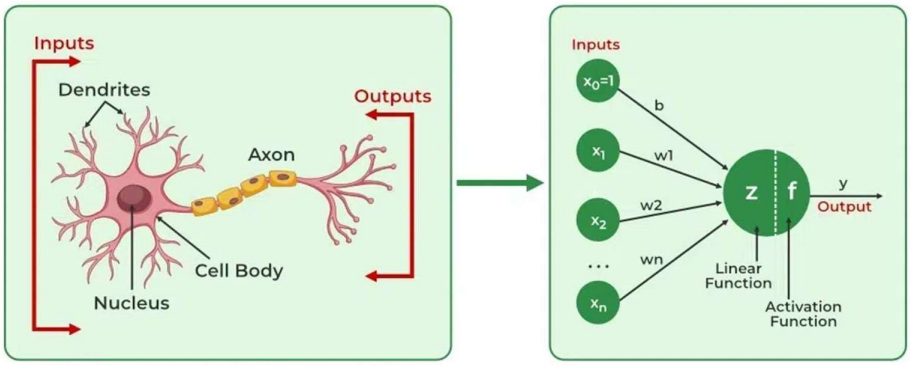
UNIT-4

Neural Networks:

Neural networks are computer programs that work a bit like the human brain. They have many small connected parts called “neurons” that pass information to each other. By looking at lots of examples, they learn patterns in the data and can help with things like recognizing images, understanding speech, or making decisions.

- They can automatically adapt to new patterns in data without needing the rules to be rewritten.
- They can combine information from different sources like text, numbers, and images in a single model.

Neural networks are important in identifying complex patterns, solving intricate challenges and adapting to dynamic environments. Their ability to learn from vast amounts of data is transformative, impacting technologies like natural language processing, self-driving vehicles and automated decision-making.



Biological neuron **Artificial neuron**

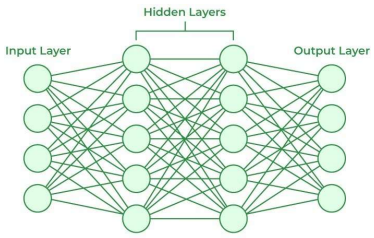
Understanding Neural Networks in Machine Learning

Neural networks in machine learning are models that learn patterns from data to make predictions or classifications, without needing explicit, hand-written rules. They operate using a collection of interconnected units and processes:

- **Neurons**: Small processing units that receive inputs, perform calculations, and send outputs.
- **Connections**: Links between neurons through which data flows, influenced by adjustable parameters.
- **Weights and Biases**: Numbers that control how strongly one neuron affects another.

- **Activation Functions**: Rules that decide whether a neuron should pass its signal forward.
- **Training Algorithm**: The method that updates weights and biases so the network learns from mistakes.

Neural Network Architecture



1. **Input Layer:** This is where the network receives its input data. Each input neuron in the layer corresponds to a feature in the input data.
2. **Hidden Layers:** These layers perform most of the computational heavy lifting. A neural network can have one or multiple hidden layers. Each layer consists of units (neurons) that transform the inputs into something that the output layer can use.
3. **Output Layer:** The final layer produces the output of the model. The format of these outputs varies depending on the specific task like classification, regression.

Working Approaches

Forward Propagation:

In forward propagation, the input data is passed through the network starting from the input layer, moving through hidden layers, and reaching the output layer. Each neuron processes the data by applying weights, adding a bias, and using an activation function to introduce non-linearity. This step produces the network's prediction or output.

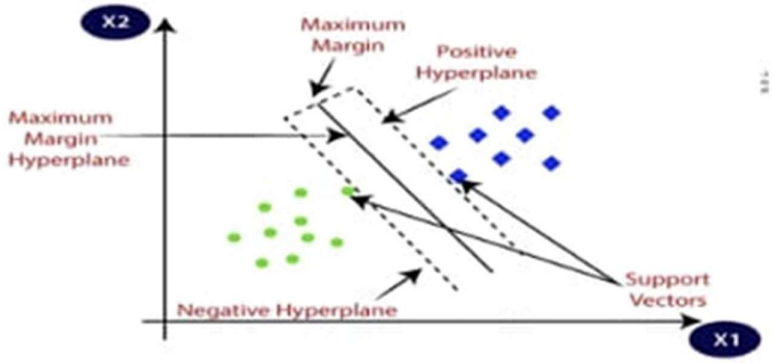
Backpropagation:

In backpropagation, the network checks how far its prediction is from the correct answer using a loss function. The error is then traced back through the network to identify which weights and biases contributed to it. Using optimization methods like gradient descent, the weights and biases are updated. This process repeats until the network improves and makes more accurate predictions**.**

Support vector machine (SVM):

Support vector machine or SVM is one of the most popular supervised Learning algorithms, which is used for classification as well as regression problems. Primarily, it is used for classification problem in machine learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.

This decision boundary is called as hyperplane. SVM chooses the extreme points / vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as support vector machine.



Working of SVM:

1. Input Data

SVM starts with training data that has features and class labels. The goal is to separate the data into different groups by finding a clear boundary between them.

2. Find a Decision Boundary

SVM looks for a hyperplane (a line in 2D, a plane in 3D) that divides the data into classes. This boundary helps decide which side each point belongs to.

3. Maximize the Margin

SVM chooses the hyperplane that has the widest gap, or margin, between the two classes. A bigger margin makes the model stronger and less likely to make mistakes on new data.

4. Support Vectors

The closest points to the hyperplane are called support vectors. They are very important because they decide the exact position of the hyperplane.

5. Non-Linear Data

If data is not linearly separable, SVM uses a Kernel Trick. The kernel function transforms data into higher dimensions where it becomes linearly separable.

6. Classification / Prediction

After finding the best hyperplane, SVM classifies new points by checking which side of the line (or boundary) they fall on. If the point is on one side, it belongs to one class; if it's on the other side, it belongs to the other class. This makes SVM useful for both two-class and multi-class problems.

Types of SVM

Linear SVM:

Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM:

A Non-Linear SVM is used for non-linearly separable data, which means is a dataset cannot be classified by using s straight line, then such data is termed as non- linear data and classifier is used called as non-linear SVM classifier.

Advantages:

- Works well for high- dimensional spaces.
- Good for both linear and non linear problems.
- SVM is effective for both binary classification and multiclass classification suitable for applications in text classification.

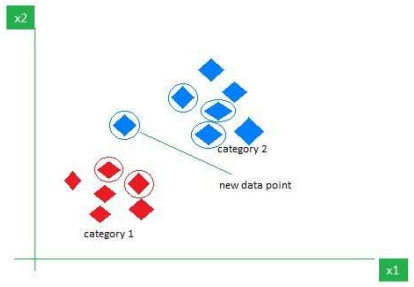
Disadvantages:

- Can be slow for lager datasets.
- Not ideal, not suitable for noisy data.
- Choosing the right kernel function can be difficult.

K-nearest neighbour:

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 makes a predictions based on the majority class (for classification) or the average value (for regression).

- K-Nearest Neighbors (KNN) is called a lazy learner algorithm.
- It does not learn from the training dataset immediately.
- Instead, it stores the entire training dataset.
- At the time of classification, it uses the stored data to make predictions.
- KNN works by finding the nearest neighbors to the query point and then classifying based on majority voting (for classification) or averaging (for regression).



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

Example: let us take an example dataset

BMI	Age	Sugar
33.6	50	1
26.6	30	0
23.4	40	0
35.3	23	1
36.7	45	1
25.7	46	0

Apply k nearest neighbor classifier to predict the diabetic patient with the given features BMI, Age.

Let us assume k=3,

Test example BMI = 43.6, Age =40, sugar =?

First calculate the distance between the test instances and training instances

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

BMI	Age	Sugar	Distance	
33.6	50	1	$\sqrt{(43.6 - 33.6)^2 + (40 - 50)^2}$	14.14
26.6	30	0	$\sqrt{(43.6 - 26.6)^2 + (40 - 30)^2}$	19.72
23.4	40	0	$\sqrt{(43.6 - 23.4)^2 + (40 - 40)^2}$	20.2
35.3	23	1	$\sqrt{(43.6 - 35.3)^2 + (40 - 23)^2}$	18.92
36.7	45	1	$\sqrt{(43.6 - 36.7)^2 + (40 - 45)^2}$	8.52
25.7	46	0	$\sqrt{(43.6 - 25.7)^2 + (40 - 46)^2}$	18.88

Now we assign ranking for the required distances in ascending order which means the lowest distance to highest distance then the dataset will look like:

BMI	Age	Sugar	Distance	Rank
33.6	50	1	14.14	2
26.6	30	0	19.72	
23.4	40	0	20.2	
35.3	23	1	18.92	
36.7	45	1	8.52	1
25.7	46	0	18.88	3

We have already assigned k=3 so when we keep the first three ranks and ignore the remaining rankings.

Now we compare the rank column with sugar column.

1st rank have sugar=1

2nd rank have sugar=1

3rd rank have sugar= 0

The majority of ranks holding the sugar =1

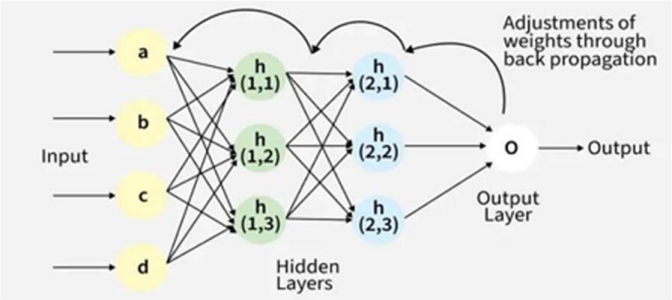
Then the test example of the sugar will be consider as 1

Then the final answer is BMI=43.6, Age=40, sugar= 1

Back Propagation:

Backpropagation, also called Backward Propagation of Errors, is a method used to train neural networks. Its main goal is to reduce the gap between predicted output and actual output by changing the weights and biases of the network.

The key idea is that backpropagation uses the chain rule from calculus to compute gradients. These gradients tell the network how much to change each weight and bias, even in complex multi-layer networks, so that the cost function keeps reducing.



Working of Backpropagation Algorithm

The Backpropagation algorithm has two main steps:

1. Forward Pass

2. Backward Pass

Forward Pass

- Input data is given to the input layer.
- These inputs are multiplied by their respective weights, and a bias is added.
- Each hidden layer calculates a weighted sum (a) of the inputs, then applies an activation function like ReLU (Rectified Linear Unit) to produce the output (o).
- Finally, at the output layer, an activation function like Softmax is used to convert the results into probabilities for classification.

Backward Pass

The error (difference between predicted output and actual output) is sent backward through the network.

- A common way to calculate this error is by using Mean Squared Error (MSE):
- MSE = (Predicted Output−Actual Output)2

- Once error is calculated, the network computes gradients (slopes) using the chain rule.
- These gradients tell how much each weight and bias should be changed to reduce the error.
- The weights are then updated layer by layer.

Types of Backpropagation Algorithms

There are two main types of backpropagation algorithms used in neural network training. These are based on how the error is propagated through the network:

Static Backpropagation

In this method, the input-output mapping remains fixed.

- Each input is processed independently, without depending on past inputs.
- Mostly used in Feedforward Neural Networks (FNNs).
- Applications: Image recognition, pattern classification, regression tasks.
- Works with fixed data inputs and outputs, Suitable when inputs are independent of each other.

Recurrent Backpropagation

- Used in Recurrent Neural Networks (RNNs) where the output depends on both the current input and previous inputs.
- Errors are propagated through time, adjusting weights based on sequential data.
- Applications: Time-series forecasting, speech recognition, natural language processing (NLP), machine translation.
- Works with sequential/temporal data, Suitable when the order of inputs matters.

Advantages:

- Simple and easy to use in neural networks.
- Works well for both simple and complex problems.
- Can be applied to different network types like FNN, CNN, and RNN.

Disadvantages:

- Needs careful tuning of learning rate, epochs, and weights.
- Requires high processing power and time for large networks.
- Can overfit when the model is complex and data is small.

Issues in training neural networks

Training neural networks has several challenges. Models can overfit and fail on new data, face unstable learning due to vanishing or exploding gradients, and require careful hyperparameter tuning, large high-quality data, and significant computational resources. They can also get stuck in local minima instead of finding the best solution.

- **Overfitting**: The model achieves high accuracy on training data but fails to generalize to unseen data because it memorizes patterns instead of learning underlying relationships.
- **Vanishing and Exploding Gradients**: In very deep networks, the values used to update weights can become too small or too large. This makes learning either very slow or unstable.
- **Hyperparameter Tuning**: Choosing the best settings (like learning rate, batch size, or number of layers) is important, but it often takes a lot of trial and error.
- **Data Quality and Quantity**: Neural networks need large, diverse, and clean datasets. Insufficient or noisy data can lead to poor performance and unreliable predictions.
- **Computational Resources**: Training deep models needs powerful computers (like GPUs/TPUs) and takes a lot of time, which can limit progress.
- **Imbalanced Data**: When one class dominates the dataset, the model tends to Favor the majority class, resulting in biased or inaccurate predictions for minority classes.

Addressing these issues

1. **Overfitting**: Use more data, apply regularization techniques (like L1/L2 or dropout), and use early stopping during training.
2. **Vanishing and Exploding Gradients**: Use activation functions like ReLU, apply proper weight initialization, or use normalization techniques such as batch normalization.
3. **Hyperparameter Tuning**: Perform systematic searches like grid search or random search, or use automated methods like Bayesian optimization.
4. **Data Quality and Quantity**: Collect more data, clean and preprocess it, and use data Expansion to artificially increase dataset size.
5. **Imbalanced Data**: Apply techniques like over sampling the minority class, under sampling the majority class, or use class-weighted loss functions.

K-Means Clustering Algorithm:

K-Means Clustering algorithm is a type of machine learning algorithm that groups data points into clusters based on how similar they are. Unlike supervised learning, it doesn't need labeled data. Instead, it helps find hidden patterns or structures in data that isn't labeled.

For example, a college can use K-Means to group students into different courses like BSc, BA, and BCom based on their academic interests and past performance.

Working

Suppose 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" represents the number of groups or clusters we want to classify our items into.

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.

- **Initialization:** We begin by randomly selecting k cluster centroids.
- **Assignment Step:** Each data point is assigned to the nearest centroid, forming clusters.
- **Update Step:** After the assignment, we recalculate the centroid of each cluster by averaging the points within it.
- **Repeat:** This process repeats until the centroids no longer change or the maximum number of iterations is reached.

The goal is to partition the dataset into k clusters such that data points within each cluster are more similar to each other than to those in other clusters

Use k means clustering to cluster the following data into two groups then the value of k = 2

Example data points: {2, 4, 10, 12, 3, 11}

The distance function is Euclidean distance

$$\text{Distance} = \sqrt{(x_2 - x_1)^2}$$

Let us assume the cluster centroids m1=4, m2=11

Initial centroids:

M1=4, M2=11

	Distance to		
Data points	m1	m2	Cluster
2			
4			
10			
12			
3			
11			

Find the distance to all data points with first initial centroid with m1=4

Distance = $\sqrt{(4-2)^2} = \sqrt{4} = 2$

Distance = $\sqrt{(4-4)^2} = 0$

Distance = $\sqrt{(4-10)^2} = \sqrt{36} = 6$

Distance = $\sqrt{(4-12)^2} = \sqrt{64} = 8$

Distance = $\sqrt{(4-3)^2} = \sqrt{1} = 1$

Distance = $\sqrt{(4-11)^2} = \sqrt{49} = 7$

Find the distance to all data points with first initial centroid with m2=11

Distance = $\sqrt{(11-2)^2} = \sqrt{81} = 9$

Distance = $\sqrt{(11-4)^2} = \sqrt{49} = 7$

Distance = $\sqrt{(11-10)^2} = \sqrt{1} = 1$

Distance = $\sqrt{(11-12)^2} = \sqrt{1} = 1$

Distance = $\sqrt{(11-3)^2} = \sqrt{64} = 8$

Distance = $\sqrt{(11-11)^2} = 0$

	Distance to		
Data points	m1	m2	Cluster
2	2	9	
4	0	7	
10	6	1	
12	8	1	
3	1	8	
11	7	0	

Assign each data point to the cluster whose centroid (m1 or m2) is **closest**. The point goes to the cluster with the **smallest distance**.

	Distance to		
Data points	**m1**	**m2**	**Cluster**
2	2	9	c1
4	0	7	c1
10	6	1	c2
12	8	1	c2
3	1	8	c1
11	7	0	c2

Therefore c1= {2,4,3}, c2= {10,12,11}

Now average the c1 and c2

c1 = 2+4+3/3 = 3

c2 = 10+12+11/3 = 11

Then the new centroids are m1=3, m2=11

Again we calculate the distance to all data points with new centroids m1=3, m2=11

	Distance to		
Data points	**m1**	**m2**	**Cluster**
2	1	9	c1
4	1	7	c1
10	7	1	c2
12	9	1	c2
3	0	8	c1
11	8	0	c2

Again Assign each data point to the cluster whose centroid (m1 or m2) is **closest**. The point goes to the cluster with the **smallest distance**.

	Distance to			
Data points	**m1**	**m2**	**Old Cluster**	**new cluster**
2	1	9	c1	c1
4	1	7	c1	c1
10	7	1	c2	c2
12	9	1	c2	c2
3	0	8	c1	c1
11	8	0	c2	c2

Next, we compare the old clusters with the new clusters to see if they are the same. If there are differences, we recalculate the centroids, find the distances again, and reassign the data points. This process continues until the old and new clusters match. Once they match, the centroids are considered final.

In the example above, the old and new clusters are the same, so the final clusters are:

C1 = {2, 4, 3}
C2 = {10, 12, 11}

SVM for regression:

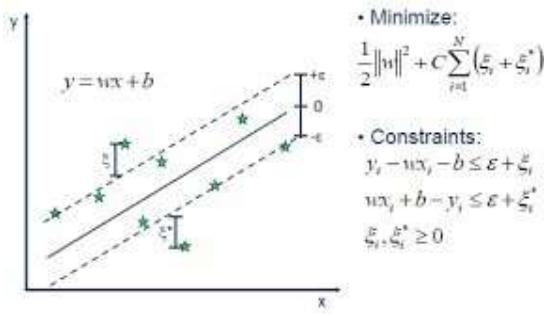
- Support vector machine (SVM) is mainly used for classification.
- It can also handle regression problems called support vector regression (SVR).
- Goal: predict continuous values with high accuracy.

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.

Basic concept:

Support Vector Regression (SVR) is a regression technique based on Support Vector Machines. Its goal is to find a function (line or curve) that best predicts continuous values.

- SVR tries to fit a function (line/curve) that is as flat as possible.
- It allows a tolerance margin (ε-tube) where errors are ignored.
- Only points outside are ε-tube influence the model these are called support vectors.



Key components:

- **Epsilon (ε):**
  Defines the width of the tolerance margin (ε-tube) around the regression line. Predictions within this tube are considered accurate enough, so small errors are ignored.
- **Support vectors:**
  Critical data points that lie outside the ε margin and directly influence the regression function. These points determine the slope and position of the line/curve, making them essential for the model.
- **Slack variables (ε, ε*):**
  Allow flexibility by permitting some points to fall outside the ε-tube. Useful when data is noisy, so the model doesn't overfit trying to capture every fluctuation.

- **C (regularization parameter):**
  Controls the trade-off between flatness of the function and tolerance for errors. A high C means the model tries to minimize error strictly, while a low C allows more flexibility.
- **Kernels:**
  Transform data into higher dimensions to handle non-linear regression problems. Common kernels include linear, polynomial, and RBF (radial basis function) for complex patterns.

Mathematical formula:

SVR predicts using f(x) = w.x+b

Optimization problem

$$\min_{w,b,\xi,\xi^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

Advantages:

- Works for linear and non-linear regression.
- Effective in high dimensional spaces.
- Less sensitive to outliers (due to ε margin)

Disadvantages:

- Requires careful tuning to c, ε, kernel.
- Computationally heavy for very large datasets.

Applications:

- **Stock Market Prediction:** Used to forecast stock prices or market trends by learning from historical financial data.
- **House Price Prediction:** Estimates real estate prices based on features like location, size, and amenities.
- **Weather Forecasting:** Predicts temperature, rainfall, or wind speed by analyzing past meteorological data.
- **Medical Field:** Used to predict disease progression, patient survival rates, or drug effectiveness.

K-nearest – Neighbour classifiers (Image Scene Classification):

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 makes a predictions based on the majority class (for classification) or the average value (for regression).

The k-Nearest Neighbor (k-NN) classifier for image scene classification is a supervised learning algorithm that classifies a new image by finding the k most similar labeled images in the feature space and assigning the scene label (e.g., Beach, Forest, City) that is most common among those neighbors.

Example:

Problem defining:

We want to classify a new test image into one of the scene categories (*Beach, Forest, City*). We will use the k-Nearest Neighbor (k-NN) algorithm with k=5. The aim is to identify the scene of the test image based on its nearest neighbors in feature space.

Step-1 dataset

- Gather a dataset of images that belong to different scene categories (e.g., *Beach, Forest, City*).
- Each image must be labeled with its correct class, since k-NN is a supervised learning algorithm.

Step-2 preprocess images

- Convert all images into a standard format (same size).
- Apply normalization so that pixel values/features fall in a similar range.
- This ensures fair comparison during distance calculation.

Step-3 extract features

- Images are transformed into feature vectors (numerical form).
- Examples:
  - Color Histogram →captures dominant colors (blue for beach, green for forest).
  - Texture Descriptors (HOG, LBP) → capture patterns like trees, waves, buildings.

- CNN Embeddings → deep features from pretrained models like ResNet.

Step-4 Compute Distances

Compute Euclidean distance between the test image and all training images.

Neighbor	Training Image Class	Distance (d)	Rank
N1	Forest	0.37	1
N2	Forest	0.40	2
N3	City	0.42	3
N4	Forest	0.45	4
N5	Forest	0.47	5

Step 5 – Find k Nearest Neighbors

- Choose a value of k (commonly 3, 5, or 7).
- Select the k closest training images based on distance.
- Example: For $k = 5$, nearest neighbors = {4 Forest, 1 City}.

Step 6 – Final Prediction

- Since majority neighbors belong to Forest, the test image is classified as:

Predicted Class = Forest Scene

UNIT-5

Association Rules:

Association rules are a fundamental concept used to find relationships, correlations or patterns within large sets of data items. They describe how often itemsets occur together in transactions and express implications of the form:

$$X \rightarrow Y$$

In association rules, X and Y are separate sets of items. The rule means that when items in X are bought, items in Y are also likely to be bought.

Key Components

- **Antecedent (X)**: The "if" part representing one or more items found in transactions.
- **Consequent (Y)**: The "then" part, representing the items likely to be purchased when antecedent items appear.

Rules are evaluated based on metrics that quantify their strength and usefulness:

Rule Evaluation Metrics

Support (Support = frequency.)

How often X and Y occur together in all transactions.

$$Support(X \rightarrow Y) = \frac{\text{Transactions with } (X \cup Y)}{\text{Total transactions}}$$

Confidence (Confidence = reliability.)

How often Y appears when X appears.

$$Confidence(X \rightarrow Y) = \frac{Support(X \cup Y)}{Support(X)}$$

Lift (Lift = strength of association.)

How much more likely X and Y occur together compared to being independent.

$$Lift(X \rightarrow Y) = \frac{Confidence(X \rightarrow Y)}{Support(Y)}$$

- Lift > 1 → Positive relation (items appear together more than expected).
- Lift = 1 → No relation (independent).

- Lift < 1 → Negative relation (items appear together less than expected).

Transaction ID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example Rule: {Milk} → {Bread}

Support: Count how many transactions have Milk and Bread together.

Transactions: 1, 4, 5 → **3 times**

Total transactions = 5

$$Support(Milk \rightarrow Bread) = \frac{3}{5} = 0.6$$

Confidence: Count how many transactions have Milk.

Transactions: 1, 3, 4, 5 → **4 times**

$$Confidence(Milk \rightarrow Bread) = \frac{Support(Milk \cup Bread)}{Support(Milk)} = \frac{3}{4} = 0.75$$

Lift: Count how many transactions have Bread.

Transactions: 1, 2, 4, 5 → **4 times**

$$Support(Bread) = \frac{4}{5} = 0.8$$

$$Lift(Milk \rightarrow Bread) = \frac{Confidence(Milk \rightarrow Bread)}{Support(Bread)} = \frac{0.75}{0.8} = 0.9375$$

- **Support = 0.6** → 60% of transactions contain both Milk and Bread.

- **Confidence = 0.75** → 75% of the time when Milk is bought, Bread is also bought.
- **Lift = 0.94 (<1)** → Milk and Bread actually occur together less than expected → weak/negative association.

Example 2: {Diaper} → {Beer}

Support: Count transactions that have Diaper and Beer together.

Transactions: 2, 3, 4 → **3 times**

$$Support(Diaper \rightarrow Beer) = \frac{3}{5} = 0.6$$

Confidence: Count transactions that have Diaper.

Transactions: 2, 3, 4, 5 → **4 times**

$$Confidence(Diaper \rightarrow Beer) = \frac{3}{4} = 0.75$$

Lift: Count transactions that have Beer.

Transactions: 2, 3, 4 → **3 times**

$$Support(Beer) = \frac{3}{5} = 0.6$$

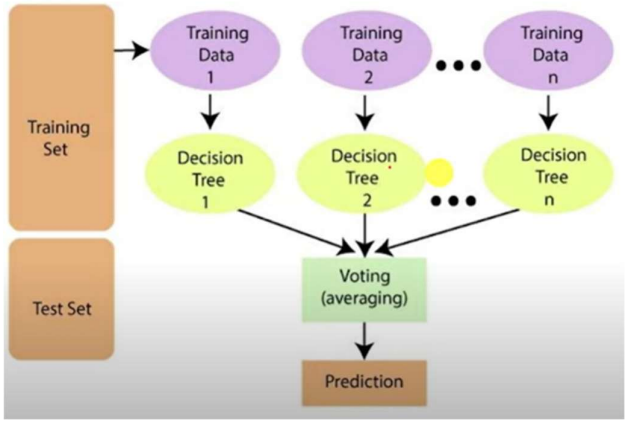
$$Lift(Diaper \rightarrow Beer) = \frac{0.75}{0.6} = 1.25$$

- Support = 0.6 → 60% of transactions have both Diaper and Beer.
- Confidence = 0.75 → 75% of the time when Diaper is bought, Beer is also bought.
- Lift = 1.25 (>1) → Positive association → Diaper buyers are more likely to buy Beer than by chance.

Random Forest and analysis:

Random Forest is a machine learning algorithm that builds many decision trees using random parts of the data. For classification, it combines the results of the trees by voting, and for regression, it averages their results. Since it is an ensemble learning technique, it improves accuracy and reduces errors compared to a single decision tree.

- Instead of building a single tree (like in Decision Tree models), it creates many trees and combines their outputs.
- Each tree is built using a random subset of features and data samples this reduces overfitting and improves generalization.
- Using random data and features for each tree helps avoid overfitting and makes the overall prediction more accurate and trustworthy.



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.
    - From the original dataset, multiple random samples are created (with replacement).
    - Each sample is used to train one decision tree.
- **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 and

for **regression** we predict a number as the final answer is the average of all the trees predictions.

Example:

Suppose you want to predict whether a person will buy a product.

- One decision tree might focus on **age.**
- Another decision tree might focus on **salary**.
- Another decision tree might focus on l**ocation.**
- Another decision tree might focus on **previous purchases**.
- Random Forest combines many such diverse trees → more reliable prediction.

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.

Advantages:

- Handles large datasets well
- Works for both classification and regression
- Reduces overfitting compared to decision trees
- Handles missing values and categorical data effectively
- When we combine multiple decision trees it reduces the risk of overfitting of the model.

Disadvantages:

- Slower to train and predict because it builds many trees
- Uses more memory and computational resources
- Harder to interpret than a single decision tree

Principal components:

Principal components come from Principal Component Analysis (PCA), a method that reduces the number of variables in a dataset. PCA combines the original, often correlated variables into a smaller set of new, uncorrelated variables called principal components. These new variables capture most of the important information (variance) from the original data.

Working:

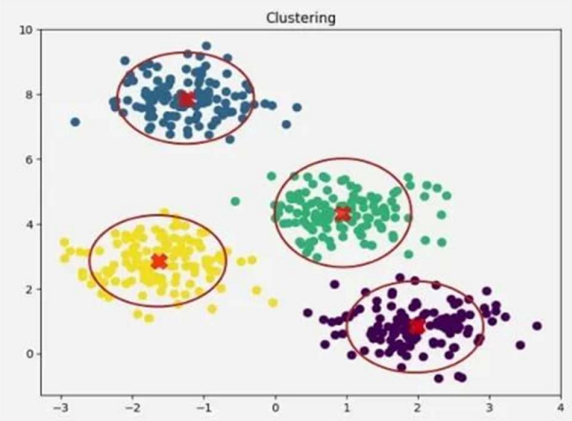
- **Dimensionality Reduction:** PCA reduces many original features into fewer new features called principal components.
- **Maximum Variance:** The first principal component (PC1) captures the most variation in the data.
- **Uncorrelated Components**: Each next component (PC2, PC3, …) captures the remaining variation and is uncorrelated with the previous ones.
- **Linear Combinations:** Principal components are weighted combinations of all original features, not just a subset. The weights show how much each original feature contributes.

Use in Predictive Analytics:

- **Feature Engineering:** Principal components can be used as new, stronger features, especially when original features are many or correlated.
- **Noise Reduction:** PCA keeps the most important information and filters out noise.
- **Better Model Performance:** Fewer, uncorrelated features make models faster, easier to train, and often more accurate.
- **Data Visualization:** PCA lets us plot high-dimensional data in 2D or 3D, helping to see patterns, clusters, and relationships.

Cluster analysis:

Clustering is an unsupervised machine learning technique that groups similar data points together into clusters based on their characteristics, without using any labelled data. Unlike classification, clustering does not use predefined labels; instead, it identifies hidden patterns and natural groupings within the data.



For example, if we have customer purchase data, clustering can group customers with similar shopping habits. These clusters can then be used for targeted marketing, personalized recommendations or customer segmentation.

Objectives of Clustering Analysis

- To identify similar groups of data points.
- To discover patterns, trends, and hidden structures in large datasets.
- To simplify complex data for better decision-making.
- To help in prediction, segmentation, and anomaly detection.

Steps in Clustering Analysis:

1. Collect and preprocess data (cleaning, normalization).
2. Select features that represent the data.
3. Choose similarity/distance measure (Euclidean distance , Manhattan distance)
4. Select clustering method (hierarchical, K-means, DBSCAN, etc.).
5. Apply clustering and form groups.
6. Evaluate results using measures like silhouette score, Davies-Bouldin index.

Types of Clustering Methods

1. **Partitioning Methods**
   - Example: K-Means, K-Medoids
   - Divides data into k clusters, each point belongs to the nearest cluster center.
2. **Hierarchical Methods**
   - Builds a tree of clusters (dendrogram).
   - Two types:
     - Agglomerative (bottom-up) – start with single points, merge step by step.
     - Divisive (top-down) – start with one big cluster, split into smaller ones.
3. **Density-Based Methods**
   - Example: DBSCAN
   - Groups dense regions of data; useful for clusters of irregular shapes.
4. **Grid-Based Methods**
   - Divides the data space into grids and then clusters them.

Advantages

- Works without predefined labels.
- Useful for exploratory data analysis.
- Handles large datasets effectively.

Limitations

- Choosing the right number of clusters is difficult.
- Sensitive to noise and outliers.
- Results depend on the algorithm and parameters used.