DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Machine Learning Notes

B.Tech III Year II Semester

Regulation: HM23

COURSE CODE : 23HPC3301

ANNAMACHARYA INSTITUTE OF TECHNOLOGY & SCIENCES (AUTONOMOUS)

Utukur(P), C.K.Dinne (V & M),Kadapa, YSR DIST.

Approved by A.I.C.T.E, New Delhi & Affiliated to JNTUA Ananthapuramu,

Accredited by NAAC with 'A' Grade, Bangalore & NBA (EEE,ECE & CSE)

PROGRAM OUTCOMES

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4**. Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8**. Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Vision of the INSTITUTE:

To emerge into excellence and premier institute, transforming individuals into highly enlightened professionals enriched with innovative skills entwined with intellectual, ethical and human values.

Mission of the INSTITUTE:

M1: Impart quality technical education to enhance knowledge and skills towards employability, higher education and research.

M2: Promote upgradation of teaching and research skills through quality infrastructure and resources.

M3: Enrich and elevate the rural education seekers, endow them with ethics, innovation thinking and leadership qualities enabling them to utilize their technical skills and competencies for the sustainable development of the Nation and mankind.

Vision of the Department

To become a Center of Excellence for highly competent and skilled Computer Science and Software Engineers with human and ethical values, to meet the challenges in the society.

Mission of the Department

M1: Infrastructure and Curriculum Development: To produce excellent infrastructure and innovative curriculum to prepare skilled and competent Software engineers.

M2: Skill Development: To Organize workshops, seminars and hands-on training sessions to equip students with the required skills and knowledge. Also emphasize the development of soft skills like communication, leadership, teamwork and ethics.

M3: Research, Innovation and Collaboration: To Encourage students and faculty to engage in ground-breaking research and innovative projects in association with Industry.

Program Educational Objectives (PEOs)

PEO1: Prospective Career: Exhibit knowledge and skills in the various domains of Computer Science and Engineering for prospective careers in National and Global Industries.

PEO2: Higher Education: Be strong in fundamentals of Computer Science and Engineering for successfully pursuing higher education and research in reputed institutions.

PEO3: Product Development: Apply their knowledge and innovative ideas to design and develop products for real time problems, as per the needs of the society, government and industries, and emerge as entrepreneurs.

PEO4: Teamwork & Life Long Learning: Be an effective team member, exhibit leadership abilities, professional ethics, communication skills, interpersonal skills and practice lifelong learning.

Program Specific Outcomes (PSO's)

PSO1: Ability to learn the applicability of various systems software elements for solving design problems.

PSO2: Ability to apply their skills in the field of Web Designing, Cloud Computing, Machine Learning, Artificial Intelligence and Internet of Things

III B.Tech II Semester

23HPC3301	MACHINE LEARNING	L	T	P	C
		3	0	0	3

Course Objectives: The objectives of the course are

- Define machine learning and its different types (supervised and unsupervised) and understand their applications.
- Apply supervised learning algorithms including decision trees and k-nearest neighbors (k-NN).
- Implement unsupervised learning techniques, such as K-means clustering.

,

Course Outcomes:

- CO1: Identify machine learning techniques suitable for a given problem. (L3)
- CO2: Solve real-world problems using various machine learning techniques. (L3)
- CO3: Apply Dimensionality reduction techniques for data preprocessing. (L3)
- CO4: Explain what is learning and why it is essential in the design of intelligent machines. (L2)
- CO5: Evaluate Advanced learning models for language, vision, speech, decision making etc. (L5)

UNIT-I: Introduction to Machine Learning: Evolution of Machine Learning, Paradigms for ML, Learning by Rote, Learning by Induction, Reinforcement Learning, Types of Data, Matching, Stages in Machine Learning, Data Acquisition, Feature Engineering, Data Representation, Model Selection, Model Learning, Model Evaluation, Model Prediction, Search and Learning, Data Sets.

UNIT-II: Nearest Neighbor-Based Models: Introduction to Proximity Measures, Distance Measures, Non-Metric Similarity Functions, Proximity Between Binary Patterns, Different Classification Algorithms Based on the Distance Measures ,K-Nearest Neighbor Classifier, Radius Distance Nearest Neighbor Algorithm, KNN Regression, Performance of Classifiers, Performance of Regression Algorithms.

UNIT-III: Models Based on Decision Trees: Decision Trees for Classification, Impurity Measures, Properties, Regression Based on Decision Trees, Bias–Variance Trade-off, Random Forests for Classification and Regression.
The Bayes Classifier: Introduction to the Bayes Classifier, Bayes‘ Rule and Inference, The Bayes Classifier and its Optimality, Multi-Class Classification | Class Conditional Independence and Naive Bayes Classifier (NBC)

UNIT-IV: Linear Discriminants for Machine Learning: Introduction to Linear Discriminants, Linear Discriminants for Classification, Perceptron Classifier, Perceptron Learning Algorithm, Support Vector Machines, Linearly Non-Separable Case, Non-linear SVM, Kernel Trick, Logistic Regression, Linear Regression, Multi-Layer Perceptrons (MLPs), Backpropagation for Training an MLP.

UNIT-V: Clustering : Introduction to Clustering, Partitioning of Data, Matrix Factorization | Clustering of Patterns, Divisive Clustering, Agglomerative Clustering, Partitional Clustering, K-Means Clustering, Soft Partitioning, Soft Clustering, Fuzzy C-Means Clustering, Rough

Clustering, Rough K-Means Clustering Algorithm, Expectation Maximization-Based Clustering, Spectral Clustering.

Textbooks:

1. ―Machine Learning Theory and Practice‖, M N Murthy, V S Ananthanarayana, Universities Press (India), 2024

Reference Books:

1. ―Machine Learning‖, Tom M. Mitchell, McGraw-Hill Publication, 2017
2. ―Machine Learning in Action‖,Peter Harrington, DreamTech
3. ―Introduction to Data Mining‖, Pang-Ning Tan, Michel Stenbach, Vipin Kumar, 7th Edition, 2019.

Unit 1: Introduction to Machine Learning - Detailed Study Notes

Evolution of Machine Learning

The "Father of Machine Learning" is often considered to be **Arthur Samuel**.

Machine Learning (ML) is a subfield of artificial intelligence (AI) that emphasizes algorithms capable of improving automatically from experience. Originating in the 1950s, ML sought to design machines that learn from data instead of relying on rigid programming rules. Initially, simple pattern recognition and rule-based systems dominated. However, developments in computing power, algorithmic strategies, and plentiful data have propelled modern sophisticated ML techniques. These are utilized across diverse fields such as natural language processing, computer vision, and robotics.

Detailed Evolution of Machine Learning

1940s – Foundations of Machine Learning and Artificial Intelligence

- The origin of machine learning is closely linked to the birth of artificial intelligence (AI). The foundational ideas began in the 1940s.
- **Alan Turing**, often called the “Father of Artificial Intelligence and Machine Learning,” introduced the concept of a machine that can simulate human intelligence. In his seminal 1950 paper, *"Computing Machinery and Intelligence,"* he proposed the famous “Turing Test” as a measure of machine intelligence.
- Before this, in 1943, **Warren McCulloch** and **Walter Pitts** proposed a simplified model of an artificial neuron (the McCulloch-Pitts neuron), establishing a theoretical basis for neural networks.
- These early works laid the theoretical groundwork for machines to process information and learn patterns.

1950s – Early Experiments and Learning Machines

- The term “machine learning” was not coined, but researchers began experimenting with machines learning from data rather than strictly following programmed rules.
- In 1957, **Frank Rosenblatt** developed the **Perceptron**, the first artificial neural network designed to classify patterns and learn from input data. It was capable of learning simple logical functions.
- Early optimism was high, with many believing that machines with human-level intelligence were imminent.
- Around this time, Arthur Samuel developed a checkers-playing program that improved by playing against itself, an early success in reinforcement learning.

1960s – Advances and Emerging Challenges

- Researchers explored multi-layer neural networks and pattern recognition methods.
- The limitations of the Perceptron surfaced in 1969 when **Marvin Minsky** and **Seymour Papert** demonstrated that single-layer perceptrons could not solve certain problems (e.g., the XOR problem), which dampened enthusiasm.

- Early efforts on symbolic AI and rule-based expert systems flourished but they lacked the capacity to learn from data effectively.

1970s – The First AI Winter: Decline in Enthusiasm

- The excitement faded due to computational and data limitations.
- Machine learning was constrained by slow hardware and the scarcity of large datasets.
- Funding for AI research declined, leading to the period called the **"AI Winter."**
- Despite this, some research continued on probabilistic models and early versions of algorithms like nearest neighbor and clustering.

1980s – Revival with Backpropagation and Statistical Approaches

- In 1986, the **backpropagation algorithm** was rediscovered and popularized by **Rumelhart, Hinton, and Williams**, enabling multi-layer neural networks to be trained effectively.
- This revived interest in neural networks.
- Statistical learning methods such as decision trees, k-nearest neighbor, and Bayesian inference gained popularity.
- The focus shifted towards algorithms that could learn from data statistically rather than through rule-based programming.

1990s – Expansion and Practical Applications

- Advances in Support Vector Machines (SVMs), ensemble methods like boosting, and kernel methods expanded the toolset of machine learning.
- The increased availability of data and computing power made ML methods practical.
- ML began to be used in commercial applications like speech recognition, image classification, and fraud detection.
- Companies started investing in data-driven products and services.

2000s – The Era of Big Data

- Explosion of digital data from the internet, social media, and sensors.
- The ability to store and process massive datasets allowed ML to flourish.
- **Ensemble learning techniques** like random forests and gradient boosting improved accuracy and robustness.
- Breakthroughs in natural language processing (NLP) and recommender systems powered many online services (Google, Amazon, Netflix).

2010s – Deep Learning and AI Revolution

- The rise of **deep learning,** a subset of ML involving deep neural networks with many layers.
- Use of graphical processing units (GPUs) for parallel computing enabled training of complex models on huge datasets.

- Deep learning models like **Convolutional Neural Networks (CNNs)** and **Recurrent Neural Networks (RNNs)** achieved state-of-the-art results in image recognition, machine translation, and speech recognition.
- Landmark achievements included:
    - **AlexNet** winning the ImageNet challenge in 2012 by a large margin.
    - **AlphaGo** defeating Go champions by combining deep learning and reinforcement learning.
    - Voice assistants like Siri, Alexa, and Google Assistant becoming mainstream.

Today and the Future

- **Explainable AI (XAI):** Addressing the "black box" problem of deep learning by developing models that are interpretable and transparent.
- **Federated Learning:** Training ML models across distributed devices while preserving privacy.
- **Few-shot and Zero-shot Learning:** Building models that learn from very little labeled data.
- **Ethics and Fairness:** Ensuring ML systems are unbiased, fair, and socially responsible.
- **Quantum Machine Learning:** Exploring quantum computing to accelerate algorithms.
- **General Artificial Intelligence:** Efforts continue towards machines capable of human-like reasoning beyond narrow tasks.

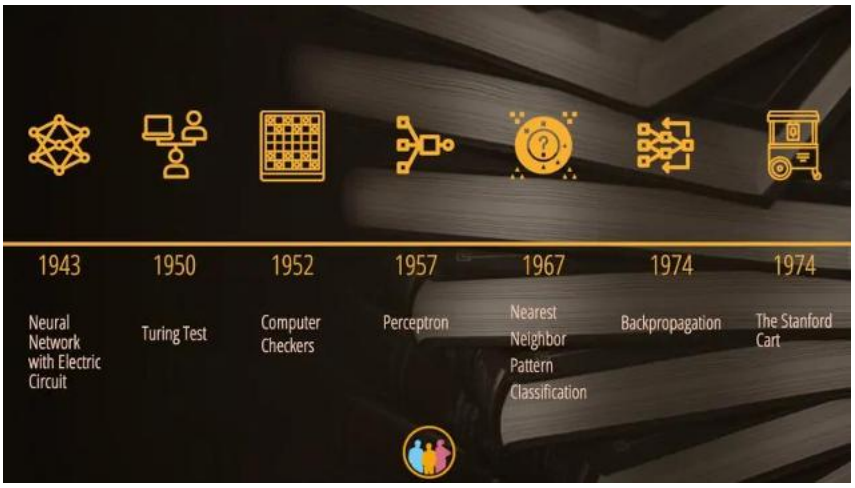


Fig: evolution of machine learning

Example: The evolution from early expert systems encoded with fixed rules to present-day deep learning architectures like convolutional neural networks illustrates this progression vividly.

Paradigms for Machine Learning

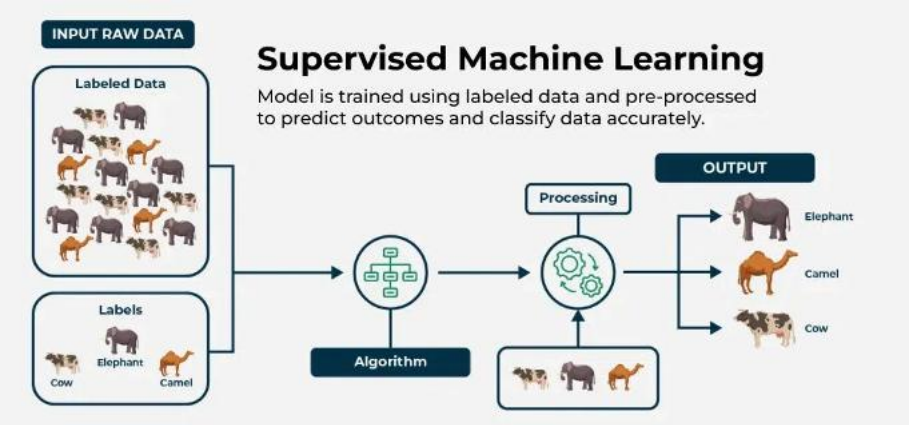
Machine learning paradigms define frameworks describing how learning is performed by models:

1. Supervised Learning

Description: In supervised learning, the algorithm is trained on a labeled dataset, meaning each input example comes with a correct output or labelled.

The goal is to learn a function that maps inputs to the correct outputs.

Often used for classification (categorical outputs) and regression (continuous outputs).



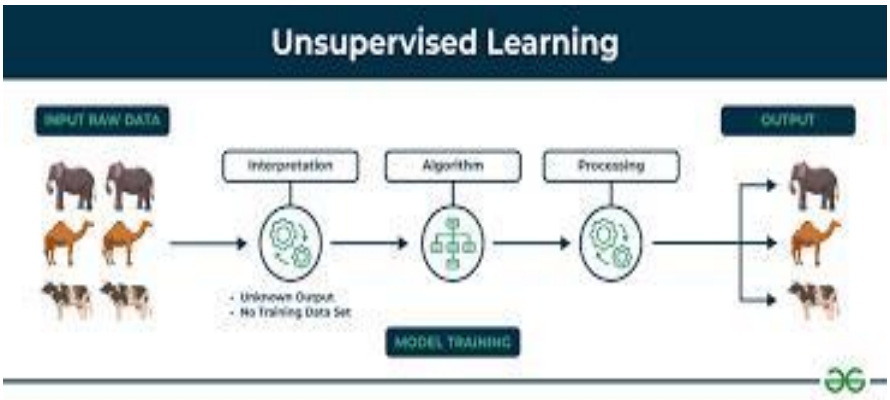
Example:

Email spam filtering: The model is trained on emails labeled as “spam” or “not spam.” It learns to classify new incoming emails based on this labeled data.

2. Unsupervised Learning

Description: Here, the data has no labels. The algorithm tries to find hidden patterns, structures, or groupings in the data.

Common tasks include clustering, association, and dimensionality reduction.



Example:

Customer segmentation: A retailer groups customers into clusters based on purchasing behavior to target marketing efforts, without pre-existing labels.

3. Reinforcement Learning

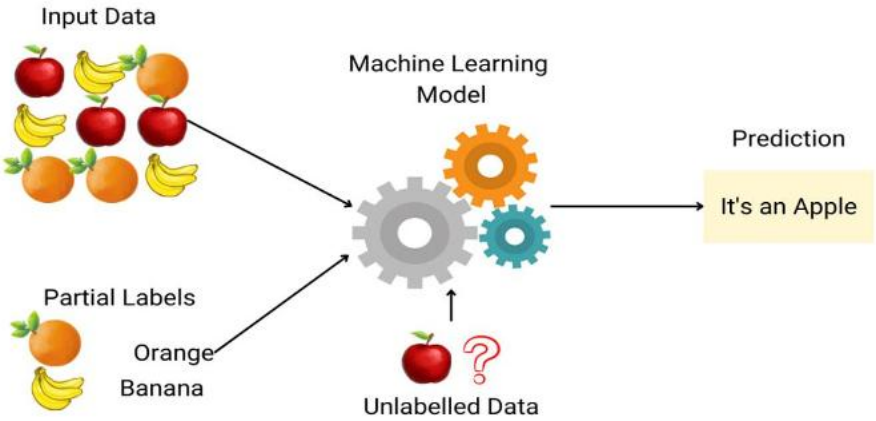
- **Description:** An agent learns to make decisions by interacting with an environment. It receives feedback in the form of rewards or penalties and aims to learn a policy that maximizes cumulative reward.
- Used when explicit input-output pairs are not available.

Example:

- Training a robot to navigate a maze by rewarding it when it moves closer to the goal and penalizing wrong moves.

4. Semi-supervised Learning

- **Description:** Combines a small amount of labeled data with a large amount of unlabeled data to improve learning accuracy.
- Useful when labeling data is expensive or time-consuming.



Example:

- In medical diagnosis, only a limited number of cases are labeled, but many unlabeled patient records exist. Semi-supervised learning helps leverage both.

Learning by Rote

Definition: Learning by rote refers to a type of learning where the system simply memorizes input-output pairs without understanding or generalizing any underlying pattern. The model stores the examples exactly as they are and recalls the stored output when it encounters the exact same input again.

Characteristics:

- No generalization beyond memorized data.
- Unable to handle new or unseen inputs.
- Simple storage and retrieval approach.

Examples:

Imagine a music player that only plays songs you have saved exactly as files on your device. It remembers the songs but cannot recognize or play any new song you add unless you save it first.

If you ask it to play a song it has never seen or saved before, it won't know what to do.

This shows rote learning—it remembers only exact examples without understanding or adapting to new ones.

Learning by Induction

Definition:
Learning by induction means the system learns general rules or patterns from specific examples, allowing it to make predictions or decisions about new, unseen data. Instead of just memorizing facts, it understands the underlying relationships.

How it works:

- The model observes many examples.
- It finds common features and patterns.
- It generalizes these patterns into rules.
- It applies these rules to new inputs.

Example 1: Animal Classification

Suppose you show a computer many pictures of animals labeled as "cats" and "dogs." It observes features like shape, size, ear type, and tail.

- The model learns the general characteristics of cats and dogs.
- When shown a new animal picture it has never seen, it can classify it correctly based on these learned patterns.

Example 2: Student Grades

If you provide a system with students' study hours and their exam results, it can learn the pattern that more study hours generally lead to better grades. Using this, it can predict how future students might perform based on their study time.

Reinforcement Learning (RL): Detailed Explanation

What is Reinforcement Learning?

Reinforcement Learning is a type of machine learning where an agent learns how to behave in an environment by performing actions and receiving feedback in the form of rewards or penalties. Unlike supervised learning, there are no labeled input-output pairs. Instead, the learning is driven by trial and error to maximize cumulative reward.

Key Components of Reinforcement Learning

1. **Agent:**
   The learner or decision-maker that interacts with the environment.
2. **Environment:**
   The external system with which the agent interacts.
3. **State (S):**
   A representation of the current situation or configuration of the environment.
4. **Action (A):**
   Choices available to the agent at a particular state.
5. **Reward (R):**
   Scalar feedback signal received after taking an action, indicating the immediate benefit or penalty**.**
6. **Policy (π):**
   The strategy or mapping from states to actions that the agent follows.
7. **Value Function (V):**
   Estimates rewards. how good a given state (or state-action pair) is in terms of expected future
8. **Model (optional):**
   The agent's understanding or prediction of how the environment will respond to actions.

How Reinforcement Learning Works: Step-by-Step

1. **Observation:**
   The agent observes the current state of the environment.
2. **Action Selection:**
   Based on its policy, the agent selects an action to perform.
3. **Transition:**
   The environment responds to the action and transitions into a new state.
4. **Reward:**
   The agent receives a reward signal based on the action taken.
5. **Learning:**
   Using the experience (state, action, reward, next state), the agent updates its policy and/or value function to improve future decisions.
6. **Iteration:**
   The cycle repeats as the agent continues to interact and learn.

Goals of Reinforcement Learning

- Maximize cumulative reward over time, rather than just immediate reward.
- Learn an optimal policy that defines the best action to take in each state.

Types of Reinforcement Learning Algorithms

1. **Model-Free Algorithms:**
   The agent learns solely from experiences without a model of the environment.
   - **Examples:** Q-Learning, SARSA.
2. **Model-Based Algorithms:**
   The agent tries to learn or use a model of the environment to plan ahead.
   - **Examples:** Dyna, Monte Carlo Tree Search.
3. **Policy Gradient Methods:**
   Directly optimize the policy without explicitly estimating value functions.
   - **Examples:** REINFORCE, Actor-Critic methods.

Example: Training a Robot to Navigate a Maze

- **State:** The robot's current location in the maze.
- **Action:** Move forward, backward, left, right.
- **Reward:** Getting closer to the exit is rewarded; hitting a wall is penalized.
- **Policy:** The robot learns which moves in each location maximize its chances of reaching the exit fastest.

By trial and error, the robot discovers the best path through the maze.

Applications of Reinforcement Learning

- **Robotics:** Teaching robots tasks like walking, grasping objects.
- **Game Playing:** Chess, Go (e.g., AlphaGo), video games.
- **Autonomous Vehicles:** Learning driving policies.
- **Resource Management:** Dynamic allocation in networks or cloud computing**.**
- **Finance: T**rading strategies optimizing returns.

Challenges in Reinforcement Learning

- **Exploration vs. Exploitation:** Balancing between trying new actions to gain information (exploration) and using known actions that yield high rewards (exploitation).
- **Sparse Rewards:** Receiving infrequent feedback makes learning difficult.
- **High-Dimensional Spaces:** Complex environments have many possible states and actions.

- **Sample Efficiency:** Learning can require a large number of interactions.

Summary

Example: Training an AI to play chess by rewarding victories and penalizing losses cultivates progressively better strategies.

Types of Data

Data is the foundation for machine learning models, and it comes in different forms. Understanding these types helps in choosing the right preprocessing and algorithms.

Data forms the foundation of ML and can be categorized as follows:

Data is the foundation for machine learning models, and it comes in different forms. Understanding these types helps in choosing the right preprocessing and algorithms.

Data Based on Structure:

Machine learning data can be categorized based on its structure and organization. Understanding these classifications helps in choosing appropriate storage, processing methods, and machine learning algorithms

1.Structured Data:

Definition:
Structured data is organized data that follows a predefined format or schema, usually arranged in rows and columns like in tables or spreadsheets. It is easy to enter, store, search, and analyze due to its fixed fields.

Key Points:

- Highly organized and easily searchable.
- Stored in relational databases or spreadsheets.
- Columns denote attributes/features; rows denote records/instances.

Examples of Structured Data:

- Customer database with columns like Name, Age, Address, and Purchase History.
- Employee records containing Employee ID, Department, Salary, and Hire Date.
- Sales data stored in tables with Date, Product, Quantity, and Price.

Usage in Machine Learning:

- Common input type for traditional machine learning algorithms like Decision Trees, Linear Regression, and Support Vector Machines.
- Can be easily processed for feature extraction and modeling.

2. Unstructured Data:

Definition:
Unstructured data is information that doesn't have a predefined format or organized schema. It's often in the form of raw text, images, audio, or video, and does not fit neatly into tables or rows.

Key Points:

- Lacks a fixed structure; difficult to store in traditional databases.
- Requires advanced processing techniques to extract meaningful information.
- Includes diverse formats like text documents, images, audio recordings, and videos.

Examples of Unstructured Data:

- Emails and chat messages.
- Photos and videos on social media.
- Audio files like voice recordings or podcasts.
- Articles, books, and written documents in raw text form.

Usage in Machine Learning:

- Requires preprocessing steps such as natural language processing (NLP) for text, computer vision techniques for images, or audio signal processing.
- Commonly used in tasks like sentiment analysis, image recognition, speech recognition, and recommendation systems.

3.Semi-structured Data:

Definition:
Semi-structured data is a form of data that does not conform to the formal structure of tables but still contains tags or markers to separate semantic elements and enforce hierarchies within the data. It is a mix between structured and unstructured data.

Key Points:

- Has organizational properties that make it easier to analyze than unstructured data.

- Does not reside in fixed fields but uses tags or labels to define elements.
- Can be stored in formats like XML, JSON, or NoSQL databases.

Examples of Semi-structured Data:

- JSON files containing user information with varying fields.
- XML documents that include both text and metadata tags.
- Email messages that have a standard header (From, To, Subject) but unstructured body content.
- Log files that record events with structured timestamps but variable message content.

Usage in Machine Learning:

- Often requires parsing or transforming into structured formats before analysis.
- Useful in web data extraction, log analysis, and handling data from APIs.

Based on Representation

Machine learning data can be categorized based on how it is represented. The two main types are **numerical (quantitative) data** and **categorical (qualitative) data**.

Numerical (Quantitative) Data:

Numerical data consists of measurable or countable values, making it suitable for statistical analysis and machine learning models. It is further divided into:

1.Discrete Data:

Definition:

Discrete data consists of distinct, separate values—usually counts—that can only take specific, separate values (often integers). There are no intermediate values between these points.

Key Points:

- *Countable in a finite amount of time.*
- *Values are separate and not continuous.*
- *Often represents quantities or occurrences.*

Examples of Discrete Data:

- *Number of students in a classroom: 20, 21, 22 (you can't have 20.5 students)*
- *Number of cars passing through a toll booth per hour: 5, 10, 15*
- *Number of emails received per day*

- *Number of goals scored in a football match*

Usage in Machine Learning:

- *Discrete data is often used as input features or target variables in classification or count-based regression problems.*
- *Can be treated as categorical if the counts represent categories or as numerical if the counts have quantitative meaning.*

2.Continuous Data:

Definition:
Continuous data consists of numeric values that can take any value within a range or interval. These values are measurable and can have infinite possible values, including decimals.

Key Points:

- Can take any value within a given range.
- Values are not restricted to separate distinct values.
- Often represents measurements like length, time, temperature.

Examples of Continuous Data:

- Height of a person: 170.5 cm, 171.2 cm, etc.
- Temperature: 36.6°C, 22.4°C, etc.
- Time taken to run a race: 12.34 seconds, 11.89 seconds
- Weight: 65.3 kg, 70.1 kg

Usage in Machine Learning:

- Often used in regression problems where the output is a continuous real number.
- Requires scaling or normalization as part of preprocessing for many algorithms.

Categorical (Qualitative) Data

Categorical data consists of labels or categories that classify objects or individuals. It is divided into:

1.Ordinal Data

Definition:
Ordinal data is a type of categorical data where the categories have a meaningful order or ranking, but the differences between the categories are not necessarily equal or known.

Key Points:

- The categories can be ranked or ordered.
- The actual difference or distance between categories is not measured.
- Useful for representing things like ratings, preferences, or levels.

Examples of Ordinal Data:

- **Education Level:** High School < Bachelor's < Master's < PhD
- **Customer Satisfaction:** Poor < Fair < Good < Excellent
- **Size rating:** Small < Medium < Large
- **Military Ranks:** Lieutenant < Captain < Major < Colonel

Usage in Machine Learning:

- Often encoded using label encoding or ordinal encoding to preserve the order.
- Important to treat ordinal data carefully because treating it as nominal may lose the order information**.**

Example:

- Education level: High School < Bachelor's < Master's < PhD.
- Customer satisfaction ratings: Poor < Fair < Good < Excellent.

2.Nominal Data:

Definition:
Nominal data is a type of categorical data where the values represent categories or names without any inherent order or ranking. The categories are simply different labels used to identify distinct groups.

Key Points:

- No natural order or hierarchy between categories.
- Categories are mutually exclusive (each data point belongs to one category).
- Cannot perform mathematical operations like addition or subtraction on nominal data.
- Often encoded as labels or symbols.

Examples of Nominal Data:

- Colors: Red, Blue, Green, Yellow
- Types of animals: Dog, Cat, Bird, Fish
- Gender: Male, Female, Other
- Marital Status: Single, Married, Divorced, Widowed
- Product brands: Nike, Adidas, Puma

Usage in Machine Learning:

- Encoded using techniques like one-hot encoding or label encoding to convert categories into numeric formats for algorithms.
- Common in classification problems where data points belong to distinct classes.

Based on time relation:

Based on time relation, data is generally classified into two types:

1.Static Data:

Definition:

Static data refers to information that does not change over time. It is constant and represents a fixed snapshot of a situation, object, or record.

Key Points:

- Does not vary with time.
- Collected once or updated rarely.
- Useful for describing stable attributes.

Examples of Static Data:

- Person's date of birth.
- Social Security number or national ID.
- Product specifications like model number or manufacturing date.
- Historical census data collected at one point in time.

Usage in Machine Learning:

- Static data is often used as input features that remain unchanged during analysis.
- Useful for classification or regression tasks involving stable attributes.

2.Dynamic Data

Definition:

Dynamic data is information that changes over time. It is updated regularly or continuously and reflects ongoing processes or events.

Key Points:

- Varies over time, showing trends or patterns.

- Collected periodically or in real-time.
- Important for monitoring, forecasting, and time-based analysis.

Examples of Dynamic Data:

- Daily stock market prices.
- Hourly weather temperature readings.
- Website user activity logs.
- Heart rate measured continuously during exercise.

Usage in Machine Learning:

- Dynamic data is often analyzed using time-series models or recurrent neural networks that capture temporal dependencies.
- Used in applications like prediction, anomaly detection, and behavior analysis.

Matching

Matching involves comparing data instances to find similarity or correspondence, essential in classification, pattern recognition, and information retrieval.

What is Data Matching?

Data matching is the process of comparing two or more datasets or records to find entries that refer to the same entity, even if they come from different sources or contain variations. It is also known as record linkage, data deduplication, or entity resolution.

Steps in Data Matching

1. Data Collection

- Gather data from various sources that need to be compared or combined.
- These sources can be databases, spreadsheets, external files, or logs.

2. Data Preprocessing

- Clean the data to fix errors like typos, inconsistent formats, or missing values.
- Standardize data formats (e.g., dates, phone numbers) so that they can be compared accurately.
- This step makes matching more precise and efficient.

3. Matching

- Compare records across different datasets to find similar or duplicate entries.
- Use exact matching (identical values) or fuzzy matching (allowing slight differences).

- Calculate similarity scores to decide how close records are.

4. Validation

- Review the matches found to check accuracy.
- Some matches may be uncertain and require manual inspection or additional rules to confirm.
- This reduces false positives (wrong matches) and false negatives (missed matches).

5. Consolidation

- Merge or link the matched records to create a unified, clean dataset.
- Remove duplicates and combine relevant information from different sources.
- This consolidated data is ready for further analysis or use.

Why is Data Matching Important?

- **Data Quality Improvement:** Eliminates duplicate records and inconsistencies.
- **Data Integration:** Combines information from multiple sources to create a unified dataset.
- **Accurate Analytics:** Ensures that analyses and reporting are based on correct and consistent data.
- **Improved Decision-Making:** Reliable data leads to better insights and decisions.

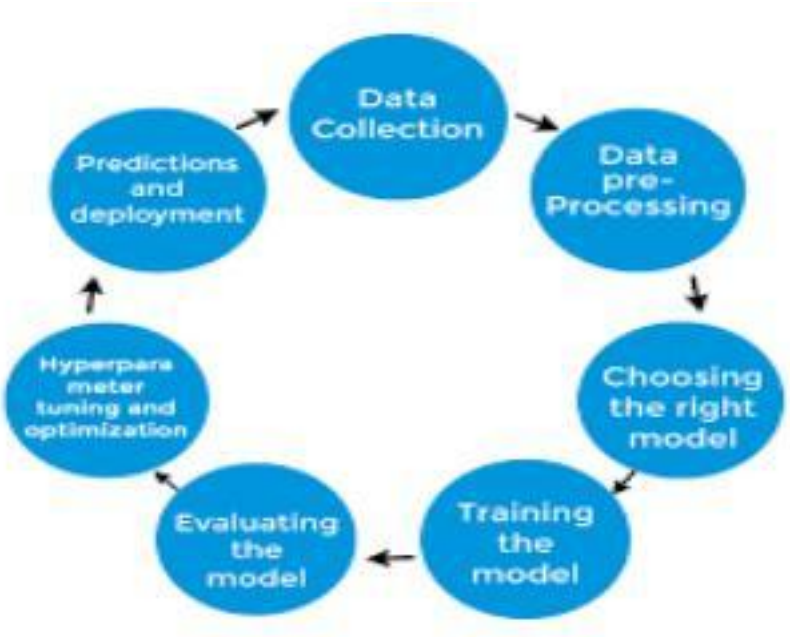
applications of Data Matching

- **Customer Data Integration:** Merging customer databases for a 360-degree view.
- **Healthcare:** Linking patient records across hospitals to maintain consistent medical histories.
- **Fraud Detection:** Identifying duplicate or suspicious transaction records.
- **Marketing:** Combining data from multiple campaigns to track customer engagement.
- **Government:** Consolidating records from different departments (tax, social services).

Example: Face recognition systems compare features in a new photo with stored images to identify individuals accurately.

Stages in Machine Learning

The ML workflow comprises several critical stages:



Detailed Stages of Machine Learning

1. Data Acquisition

- **Description:** This is the initial stage where relevant data is collected to address the specific problem or task.
- **Sources:** Data can be gathered from sensors, databases, user input, web scraping, APIs, or purchased datasets.
- **Importance:** The quality and quantity of data acquired directly influence the model's ability to learn and generalize well.
- **Challenges:** Ensuring data relevance, accuracy, completeness, and legality/privacy compliance.

2. Data Preprocessing

- **Description:** Raw data is often noisy and inconsistent. Preprocessing cleans and prepares the data for analysis.
- **Tasks:**
    - Handling missing data through imputation or removal.
    - Removing duplicates and outliers.
    - Correcting inconsistencies and typographical errors.
    - Converting data types and normalizing scales.
    - Encoding categorical variables (e.g., one-hot, label encoding).
- **Importance:** Improves data quality and ensures uniformity, leading to more reliable model training.

3. Feature Engineering

- **Description:** Creating, selecting, and transforming variables (features) that have predictive power.
- **Tasks:**
    - Feature selection: choosing the most relevant features.
    - Feature extraction: creating new features from raw data (e.g., extracting edges from images).
    - Dimensionality reduction: Techniques like PCA to reduce feature space.
    - Scaling and normalization.
- **Importance:** Well-crafted features significantly improve model accuracy and reduce computation.

4. Model Selection

- **Description:** Choosing an appropriate machine learning algorithm tailored for the task and data.
- **Considerations:**
    - Nature of the problem (classification, regression, clustering).
    - Data size and dimensionality.
    - Interpretability requirements.
    - Resource constraints.
- **Examples:** Decision Trees, Random Forests, Support Vector Machines, Neural Networks, k-Nearest Neighbors.

5. Model Training

- **Description:** Feeding the training data into the selected algorithm to learn patterns by optimizing model parameters.
- **Techniques:** Optimization algorithms like gradient descent adjust parameters to minimize loss functions.
- **Importance:** Training allows the model to generalize from examples to predict unseen data.
- **Challenges:** Overfitting (model too closely fits training data), underfitting (model too simple).

6. Model Evaluation

- **Description:** Assessing how well the trained model performs on unseen data using appropriate metrics.
- **Datasets:** Validation or test sets separate from training data.
- **Metrics for Classification:** Accuracy, precision, recall, F1 score, ROC-AUC.
- **Metrics for Regression:** Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-squared (R^2).

- **Importance:** Evaluates generalization and helps detect overfitting or underfitting.

7. Hyperparameter Tuning

- **Description:** Fine-tuning the model's hyperparameters, which are settings not learned from data but set before training.
- **Methods:**
    - Grid Search: Exhaustive search over specified parameter values.
    - Random Search: Randomly sampling hyperparameters.
    - Bayesian Optimization: Probabilistic modeling for efficient search.
- **Goal:** Improve model's predictive performance by finding optimal hyperparameter values.

8. Model Deployment

- **Description:** Integrating the trained model into a production environment for real-world use.
- **Tasks:**
    - Packaging the model.
    - Ensuring it can handle new inputs and provide predictions efficiently.
    - Setting up APIs or user interfaces.
- **Considerations:** Scalability, latency, security, and monitoring.

9. Monitoring and Maintenance

- **Description:** Continuously tracking model performance after deployment to detect degradation or changes.
- **Tasks:**
    - Detecting concept drift (when data patterns change over time).
    - Retraining or updating models with new data.
    - Logging predictions and user feedback.
- **Importance:** Maintains model relevance and accuracy over time.

Data Acquisition

This stage involves collecting data from diverse sources like sensors, databases, user inputs, or web scraping. Both data quantity and quality substantially influence ML effectiveness.

Definition:
Data acquisition is the first and foundational step in the machine learning process, involving the collection of relevant, sufficient, and high-quality data needed to solve a specific problem or build a predictive model.

What Happens During Data Acquisition?

- **Identifying Data Sources:**
  Determine where and how the required data can be obtained. Data might come from:
    - Internal databases and data warehouses
    - Sensors and IoT devices
    - Web scraping or APIs
    - Public datasets or purchased data
    - User-generated inputs or surveys
- **Gathering Data:**
  Acquire the data from these sources, often involving extraction, collection, or streaming.
- **Assessing Data Quantity and Quality:**
  Ensuring there is enough data for training the model and that data is reliable, accurate, and representative of the problem domain.

Why is Data Acquisition Important?

- The effectiveness of a machine learning model heavily depends on the quality and relevance of the data it learns from.
- Poor or insufficient data can lead to models that perform poorly or are biased.
- Data acquisition sets the stage for all subsequent stages like preprocessing, training, and evaluation.

Challenges in Data Acquisition

- **Data Availability:** Difficulty in obtaining sufficient or correct data.
- **Data Privacy and Compliance:** Ensuring legal and ethical guidelines are followed (e.g., GDPR).
- **Data Format and Storage:** Handling different data formats and storing large volumes efficiently.
- **Data Integration:** Combining data from heterogeneous sources can be complex.

Example

Suppose you are building a model to predict house prices. Data acquisition would involve collecting data on various houses including features like location, size, number of rooms, and their selling prices from real estate databases, public records, or online listings.

Tools and Techniques for Data Acquisition

- **APIs:** Interfaces for programmatically retrieving data from services like Twitter, Google Maps, or financial databases.
- **Web Scraping Tools:** Libraries such as BeautifulSoup or Scrapy to extract data from websites.
- **Database Queries:** Using SQL to extract data from structured databases.
- **Sensors and IoT Platforms:** For capturing real-time physical measurements (e.g., temperature, traffic).

- **Data Marketplaces:** Platforms offering datasets for purchase or free download.

Example: Capturing images with cameras to build datasets for training vision systems.

Feature Engineering

Feature engineering converts raw data into informative attributes that facilitate learning. It may involve selecting relevant variables, deriving new features, or encoding categorical variables numerically.

Feature engineering is the process of transforming raw data into meaningful features that better represent the underlying problem to the predictive models, improving their accuracy and efficiency. It involves selecting important variables, creating new variables, and transforming variables to make patterns more visible to machine learning algorithms.

Why is Feature Engineering Important?

- Raw data may have irrelevant, redundant, or noisy features that confuse models.
- Properly engineered features can highlight important relationships hidden in the data.
- Helps reduce model complexity and training time.
- Enhances the model's ability to generalize to new, unseen data.

Key Steps in Feature Engineering

1. Feature Selection

- **Goal:** Identify the most relevant features from the dataset and remove irrelevant or redundant ones.
- **Example:** In predicting house prices, the feature "Owner's Name" may have no predictive value and can be removed.

2. Feature Creation / Extraction

- **Goal:** Create new features from existing data that help models understand the problem better.
- **Example:**
    - From a "Date of Sale" field, create features like "Month," "Day of Week," or "Quarter" to capture seasonal trends.
    - In text data, extract the length of the text or count of specific keywords as new features.

3. Feature Transformation

- **Goal:** Modify features to correct scale differences, add non-linearity, or handle categorical data.
- **Examples:**

- **Scaling:** Normalize house sizes from 0-1 range so that large values don't dominate.
- **Logarithmic Transformation:** Apply log to skewed income data to make it more symmetric.
- **Encoding Categorical Variables:**
    - One-hot encoding for colors: "Red" → [1,0,0], "Blue" → [0,1,0].
    - Label encoding for education level: "High School"=1, "Bachelor"=2, "Master"=3.

4. Handling Missing Values

- **Goal:** Decide on a strategy for dealing with missing data to avoid biases or errors.
- **Examples:**
    - Imputing missing ages with the median age.
    - Creating a separate category "Unknown" for missing categorical values.

Example Scenario: Predicting House Prices

Suppose you have a dataset with the following raw features:

- **Address** (string)
- **Date Sold** (timestamp)
- **Number of Bedrooms** (integer)
- **Square Footage** (numeric)
- **Has Garage** (yes/no)
- **Price** (target variable)

Feature Engineering Steps:

1. **Feature Selection:**
   Remove "Address" because it's unlikely to help directly unless geocoding is applied.
2. **Feature Creation:**
   Extract "Month Sold" and "Year Sold" from "Date Sold" to capture seasonal market effects.
3. **Feature Transformation:**
   Convert "Has Garage" from yes/no to binary (1/0).
   Normalize "Square Footage" to have mean 0 and standard deviation 1.
4. **Handle Missing Values:**
   If "Number of Bedrooms" is missing for some records, fill those with the median number of bedrooms.

Now, the processed features provide clearer, more structured information for the model than the raw data.

Tools for Feature Engineering

- **Pandas & NumPy:** For data manipulation and transformations.
- **Scikit-learn:** Preprocessing modules for scaling, encoding, and imputation.
- **Featuretools:** Library for automated feature engineering.
- **Domain Knowledge:** Critical in designing meaningful features.

Example: Extracting image edges, computing moving averages of stock prices, or encoding colors as numerical codes.

Data Representation

Effective data representation adapts data into compatible formats such as vectors, matrices, graphs, or sequences for learning algorithms, crucial for efficient computation and model accuracy.

What is Data Representation?

Data representation is the process of transforming raw data into a format or structure that machine learning algorithms can effectively understand and work with. Good data representation captures essential patterns and relationships in the data while being computationally efficient.

Why is Data Representation Important?

- Machine learning models require input in numerical or structured formats.
- Proper representation simplifies learning and improves accuracy.
- Different types of data (text, images, audio) require different representation techniques.
- Poor representation may lead to information loss or noisy data, reducing model effectiveness.

Common Forms of Data Representation

1. Vector Representation

- Data points represented as vectors in a multi-dimensional space.
- Each dimension corresponds to a feature or attribute.

Example:
A house with features:

- Size = 1500 sq ft
- Bedrooms = 3
- Location Score = 7 (on a scale)

Represented as a vector:

$$\mathbf{x} = [1500,3,7]$$

2. Matrix Representation

- Collections of vectors forming matrices, especially useful for datasets with multiple samples and features.

- Each row is a data instance; each column is a feature.

Example:
Three houses with the above features:

$$X = \begin{bmatrix} 1500 & 3 & 7 \\ 1800 & 4 & 6 \\ 1200 & 2 & 8 \end{bmatrix}$$

3. Sparse Representation

- For data with many zero or missing values, sparse matrices efficiently store only non-zero elements.
- Common in text data (e.g., bag-of-words).

4. One-hot Encoding

- Converts categorical variables into binary vectors indicating the presence or absence of categories.

Example:
Color feature with categories [Red, Blue, Green]:

- Red → [1, 0, 0]
- Blue → [0, 1, 0]
- Green → [0, 0, 1]

5. Embeddings

- Dense, low-dimensional vectors that capture semantic relationships.
- Used extensively in natural language processing and image recognition.

Example:
Word embeddings like Word2Vec map words to vectors where similar words have close representations:

- "King" and "Queen" have vectors close in space.

6. Graph Representation

- Data represented as nodes and edges, capturing relationships.
- Used in social networks, recommendation systems.

Importance in Machine Learning

- The choice of data representation impacts which algorithms are suitable and how well they perform.
- Sometimes, domain knowledge guides the representation choice.
- Advanced representation learning (e.g., deep learning) automatically extracts features from raw data.

Example: Representing text documents as word frequency vectors (bag-of-words) or document embeddings in natural language processing.

Model Selection

Model choice depends on the specific problem type (classification, regression), data properties, and resource constraints. Examples include decision trees, support vector machines, and neural networks.

Definition:
Model selection is the process of choosing the most appropriate machine learning algorithm or model from a set of candidates to solve a specific problem effectively. The goal is to select a model that best captures the underlying patterns in the data and generalizes well to unseen data.

Why is Model Selection Important?

- Different algorithms have different strengths and weaknesses.
- The choice affects model accuracy, interpretability, training time, and scalability.
- A wrong model may lead to poor performance or overfitting/underfitting.

Factors to Consider in Model Selection

1. **Nature of the Problem:**
    - *Classification*: Predicting discrete categories (e.g., spam detection).
    - *Regression*: Predicting continuous values (e.g., house prices).
    - *Clustering*: Grouping similar items (e.g., customer segmentation).
2. **Size and Quality of Data:**
    - Some models require large datasets (e.g., neural networks).
    - Others perform well on small datasets (e.g., decision trees).
3. **Model Complexity:**
    - Complex models can capture intricate patterns but risk overfitting.
    - Simple models are easier to interpret and faster to train.
4. **Interpretability:**
    - Applications like healthcare may require explainable models.
    - Some black-box models (e.g., deep learning) are less interpretable.
5. **Computational Resources:**
    - Algorithm choice depends on available CPU/GPU and memory.
    - Some models require longer training times.
6. **Data Characteristics:**
    - Presence of noisy data, missing values, or categorical variables affects model suitability.

Model Selection Techniques

1. **Cross-Validation:**
    - Split data into training and validation sets multiple times to assess model stability.
2. **Grid Search / Random Search:**
    - Systematic exploration of model hyperparameters to find the best settings.
3. **Performance Metrics:**
    - Use appropriate metrics (accuracy, precision, recall, RMSE) to compare models.
4. **Ensemble Methods:**
    - Combine multiple models (e.g., bagging, boosting) to improve performance.

Example

For a spam email classification task:

- Start by trying simple models like logistic regression.
- If performance is inadequate, try decision trees or random forests.
- For even better accuracy, experiment with SVM or deep learning models.
- Use cross-validation and metrics like F1-score to compare models and choose the best one.

Example: Using linear regression to predict house prices or convolutional neural networks for classifying images.

Model Learning

During model learning, the chosen algorithm is trained by adjusting its parameters to minimize prediction error, often using optimization techniques like gradient descent.

What is Model Learning?

Model learning refers to the process by which a machine learning algorithm adjusts its parameters based on data to learn the underlying patterns or relationships. This learned model can then make predictions, classifications, or decisions on new, unseen data.

Learning is achieved by minimizing the discrepancy between the model's predictions and the actual outcomes through iterative updates.

Key Concepts in Model Learning

1. Parameters and Hypothesis

- **Parameters** are the internal variables that the learning algorithm adjusts (e.g., weights in neural networks, coefficients in regression).
- A **hypothesis** is the mathematical function or mapping from inputs to outputs defined by these parameters.

2. Training Data

- A set of examples with known outcomes used to teach the model.
- Usually split into training, validation, and test sets to gauge learning and generalization.

3. Loss Function (Objective Function)

- A mathematical function that quantifies the error between predicted outputs and actual target values.
- Common loss functions:
    - Mean Squared Error (MSE) for regression.
    - Cross-Entropy Loss for classification.

The Learning Process

1. **Initialization:**
   Start with initial values for model parameters, often random or zero.
2. **Forward Pass:**
   Input data is processed through the model to produce predictions based on current parameters.
3. **Loss Computation:**
   Calculate how much the predictions differ from real outputs using the loss function.
4. **Backward Pass (Optimization):**
   Use optimization algorithms (like gradient descent) to compute gradients—directions indicating how to change parameters to reduce loss.
5. **Parameter Update:**
   Adjust parameters slightly along the negative gradient direction scaled by a learning rate to reduce error.
6. **Iteration:**
   Repeat forward and backward passes across multiple iterations (epochs) until convergence.

Types of Learning Under Model Learning

- **Supervised Learning:** Model learns with labeled data by minimizing loss between predicted and true labels.
- **Unsupervised Learning:** Model identifies hidden patterns in unlabeled data without explicit loss based on target values.
- **Reinforcement Learning:** Agent learns to make sequential decisions based on rewards and penalties rather than direct input-output pairs.

- ## Key Components of Model Learning

1. Parameters

These are internal variables of the model (e.g., weights in a neural network, coefficients in linear regression) that define its behavior. The learning process focuses on finding the best set of parameters.

2. Training Data

This is the dataset that the model learns from. In supervised learning, each data point comes with an input and the corresponding expected output (label).

3. Loss Function (Objective Function)

A mathematical function that measures how well the model's predictions match the actual outputs. Examples include Mean Squared Error for regression, and Cross-Entropy Loss for classification. The goal is to minimize this loss.

4. Optimization Algorithm

A method to adjust the model's parameters to reduce the loss. Gradient Descent and its variants (Stochastic Gradient Descent, Adam optimizer) are widely used.

5. Hyperparameters

Settings that govern the learning process but are not learned from data (e.g., learning rate, number of epochs). Proper tuning of hyperparameters is crucial.

Challenges in Model Learning

- **Overfitting:** Model learns noise instead of general patterns.
- **Underfitting:** Model is too simple to capture real patterns.
- **Choosing Learning Rate:** Too large can overshoot minima; too small slows learning.
- **Computational Resources:** Complex models require significant computation.

Example: Training a neural network by iteratively updating weights to reduce classification mistakes.

Model Evaluation

Model performance is assessed using metrics suited to the domain and task, such as accuracy, precision, recall, and F1-score for classification, or mean squared error for regression. Evaluation uses validation or test datasets to ensure the model generalizes well.

Definition:
Model evaluation is the process of assessing how well a trained machine learning model performs on unseen data. It measures the model's ability to make accurate predictions and generalize beyond the training dataset.

Why is Model Evaluation Important?

- To understand the predictive accuracy and reliability of the model.
- To detect overfitting (model performs well on training data but poorly on new data) and underfitting (model performs poorly on both).
- To compare different models and select the best one.
- To ensure the model meets the required standards for deployment.

Key Concepts in Model Evaluation

1. Train, Validation, and Test Sets

- **Training set:** Data used to train the model.

- **Validation set:** Data used to tune hyperparameters and select the model.
- **Test set:** Final unseen data to evaluate the model's performance.

2. Evaluation Metrics

Depending on the task, different metrics are used:

For Classification:

- **Accuracy:** Percentage of correct predictions.
- **Precision:** Proportion of positive identifications that were correct.
- **Recall (Sensitivity):** Proportion of actual positives correctly identified.
- **F1 Score:** Harmonic mean of precision and recall.
- **ROC Curve and AUC:** Performance at various classification thresholds.

For Regression:

- **Mean Absolute Error (MAE):** Average absolute difference between predicted and actual values.
- **Mean Squared Error (MSE):** Average squared difference, penalizes larger errors.
- **Root Mean Squared Error (RMSE):** Square root of MSE, same units as target.
- **R-squared (R^2):** Proportion of variance explained by the model.

3. Cross-Validation

- A technique where data is split into multiple folds; models are trained and tested across folds to ensure robustness.

4. Confusion Matrix

- A table showing True Positives, False Positives, True Negatives, and False Negatives, giving a detailed view of classification performance.

Example of Model Evaluation for Classification

Suppose you build a spam email classifier. You test it on 100 emails and get:

- 90 correctly classified (both spam and not spam) → 90% accuracy
- 85 true positives (correctly identified spam)
- 5 false positives (non-spam marked as spam)
- 8 false negatives (spam not detected)

You would calculate precision, recall, and F1 score to better understand the model's strengths and weaknesses, beyond overall accuracy.

Example: Calculating the accuracy of a spam detection system on unseen email samples.

Model Prediction

After training and evaluation, the model is deployed to generate predictions on new inputs. Reliable performance on unseen data confirms successful learning.

Model Prediction

Definition:
Model prediction is the process of using a trained machine learning model to generate outputs (predictions) for new, unseen input data. This is the stage where the model applies what it has learned to make decisions or forecasts based on real-world or test inputs.

How Model Prediction Works

1. **Input Data:**
   Provide new data samples that the model has not seen during training.
2. **Feature Processing:**
   The input data is transformed or processed as required by the model, using the same steps applied during training (e.g., scaling, encoding).
3. **Applying the Model:**
   The processed data is fed into the trained model, which applies its learned parameters to compute outputs.
4. **Generating Outputs:**
   The model produces predictions, which can be:
   - A class label in classification tasks (e.g., spam or not spam).
   - A continuous value in regression tasks (e.g., house price estimate).
   - A probability score or ranking indicating confidence.

Example

Suppose you have trained a model to predict whether an email is spam based on features like word frequencies and sender's address. When a new email arrives:

- Extract features from the email text.
- Transform features (e.g., normalize word counts).
- Input features into the trained model.
- The model outputs a prediction—"spam" or "not spam."

Importance of Model Prediction

- It is the practical use of the model to solve real problems.
- Predictions guide decisions in applications like medical diagnosis, financial forecasting, or recommendation systems.

Considerations

- **Input Consistency:** New input data must be prepared in the same way as training data.

- **Prediction Confidence:** Many models provide confidence scores or probabilities which can help in decision-making.
- **Real-time vs Batch Prediction:** Models can predict instantly as data arrives or process large batches of data at once.

Example: Speech recognition systems transcribing spoken words into text in real-time applications.

Search and Learning

Search algorithms explore possible solutions or parameter configurations, while learning algorithms improve performance using data. These approaches can be combined, such as in reinforcement learning where search explores action spaces and learning optimizes policy evaluation.

What is Search?

Search is a fundamental problem-solving technique in artificial intelligence that involves exploring a set of possible states or configurations to find one or more that meet a specific goal or criteria.

Characteristics of Search:

- Operates in a problem space defined by possible states and actions.
- Uses algorithms to systematically explore states.
- Often used when the solution is not known in advance.

Examples of Search Problems:

- Finding the shortest path in a maze or map (e.g., GPS navigation).
- Puzzle solving (e.g., 8-puzzle, chess game moves).
- Scheduling and planning tasks.

Common Search Algorithms:

- **Uninformed Search:** Depth-first search, breadth-first search.
- **Informed Search:** A* algorithm, greedy best-first search.

What is Learning?

Learning in AI and machine learning is the process by which systems improve their performance or knowledge over time through experience or data.

Characteristics of Learning:

- Involves generalizing from data or interactions.
- Can be supervised, unsupervised, or reinforcement-based.
- Enables adaptation to new situations.

Types of Learning:

- **Supervised Learning:** Learning from labeled examples.
- **Unsupervised Learning:** Finding patterns in unlabeled data.

- **Reinforcement Learning:** Learning by trial and error based on feedback.

Relationship Between Search and Learning

- **Search** is often used in problem-solving where the environment or rules are known, and solutions are explored explicitly.
- **Learning** deals with situations where the system improves automatically from data or experience without explicit programming for each scenario.
- In some advanced AI systems, learning can guide search processes (e.g., learning heuristics to improve search efficiency).

Example: AlphaGo integrates game-tree search with reinforcement learning to master the game of Go.

Data Sets

Datasets are collections of data used for training and evaluating ML models. They typically comprise:

- **Training Set:** Data used to train the model.
- **Validation Set:** Data for tuning hyperparameters and monitoring overfitting.
- **Test Set:** Data for final assessment of the model's predictive ability.

Example: The MNIST handwritten digit dataset is commonly used as a benchmark in classification research.

What is a Dataset?

A dataset is a collection of data used to train, validate, and test machine learning models. It comprises multiple data points or examples, each containing input features and, often, corresponding labels or target values.

Types of Datasets in Machine Learning

1. Training Dataset

- The data used to train the machine learning model.
- The model learns patterns and relationships from this data by adjusting internal parameters.

2. Validation Dataset

- A separate portion of data used to tune hyperparameters and make decisions about model structure.
- Helps to prevent overfitting by evaluating the model's performance during training.

3. Test Dataset

- A final, unseen dataset used to assess the model's performance objectively.
- Provides an unbiased evaluation of the model's generalization ability.

Dataset Components

- **Features (Inputs):** Variables or attributes that describe each data point (e.g., age, height, pixel values).
- **Labels (Outputs/Targets):** The desired prediction outcomes corresponding to each input (e.g., cat or dog, house price).

Dataset Formats

- **Structured Data:** Tabular data with rows and columns (e.g., spreadsheets, CSV files).
- **Unstructured Data:** Data without a predefined structure (e.g., images, text, audio).
- **Semi-structured Data:** Data with some organization but not strictly tabular (e.g., JSON, XML).

Importance of Quality Datasets

- Quality and diversity in datasets determine the effectiveness of machine learning models.
- Datasets should be representative of the real-world problem to ensure good generalization.
- Poor data quality leads to inaccurate or biased models.

Examples of Popular Machine Learning Datasets

- **MNIST:** Handwritten digit images for classification.
- **CIFAR-10:** Images divided into 10 classes.
- **Iris:** Measurements of flower features for classification.
- **Titanic:** Passenger data used for survival prediction.

Dataset Preparation Steps

- Data collection from relevant sources.
- Data cleaning and preprocessing (handling missing values, encoding).
- Splitting into training, validation, and test sets.
- Feature engineering and normalization

Important Questions

1. **Define machine learning and explain its importance in modern technology.**
2. **Distinguish between supervised, unsupervised, and reinforcement learning with suitable examples.**
3. **What are the main stages in a machine learning workflow? Explain each briefly.**
4. **Explain the differences between training, validation, and test datasets. Why is each important?**
5. **What is overfitting? How can it be detected and prevented?**
6. **Describe the role and importance of feature engineering in machine learning.**

7. **Discuss various types of data commonly used in machine learning.**
8. **What is data preprocessing and why is it essential before model training?**
9. **Explain the concept of model evaluation and list common metrics used for classification and regression.**
10. **Describe real-world applications of machine learning that demonstrate its practical utility.**

UNIT – II

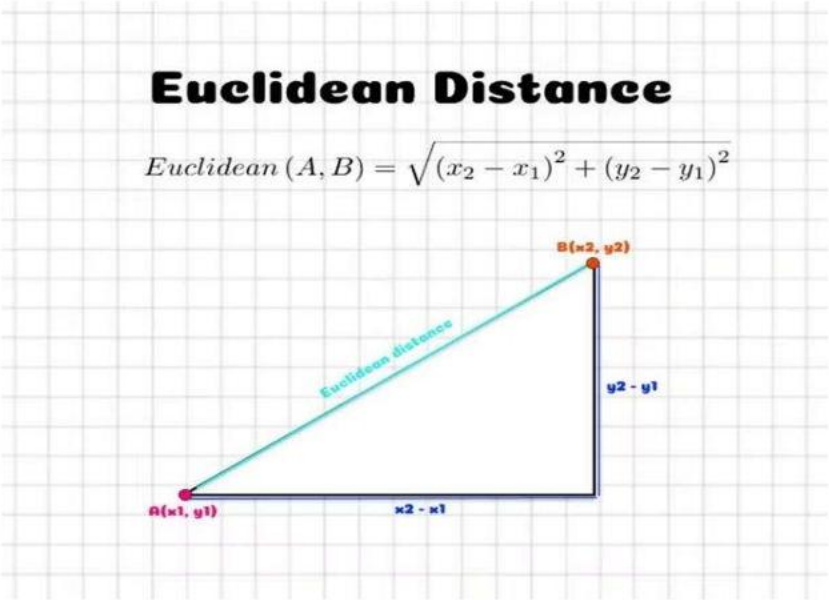
Nearest Neighbour-Based Models: Introduction to Proximity Measures, Distance Measures, Non-Metric Similarity Functions, Proximity Between Binary Patterns, Different Classification Algorithms Based on the Distance Measures, K-Nearest Neighbour Classifier, Radius Distance Nearest Neighbour Algorithm, KNN Regression, Performance of Classifiers, Performance of Regression Algorithms.

1. Introduction to Proximity Measures,

- Proximity measures are mainly mathematical techniques that calculate the similarity/dissimilarity of data points.
- Usually, proximity is measured in terms of similarity or dissimilarity i.e., how alike objects are to one another.

Euclidean distance

- Euclidean distance is a widely used distance metric. It works on the principle of the Pythagoras theorem and signifies the shortest distance between two points.
- Euclidean Distance represents the shortest distance between two vectors. It is the square root of the sum of squares of differences between corresponding elements.



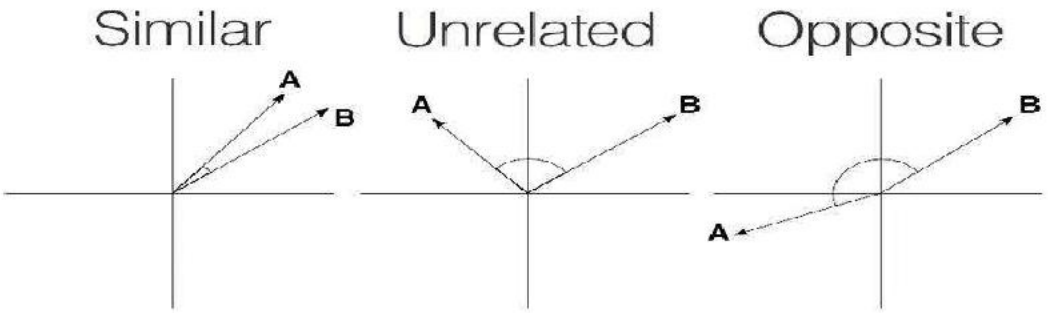
Cosine similarity

- Cosine similarity is described mathematically as the division between the dot product of vectors and the product of the Euclidean norms or magnitude of each vector.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

- The similarity measurement is a measure of the cosine of the angle between the two non-zero vectors A and B.

- The smaller the angle between the two vectors, the more similar they are to each other.
- Suppose the angle between the two vectors is 90 degrees, the cosine similarity will have a value of 0; this means that the two vectors are perpendicular to each other which means they have no correlation between them.
- As the cosine similarity measurement gets closer to 1, then the angle between the two vectors A and B becomes smaller. In this case, A and B are more similar to each other.



Jaccard Similarity

What is Jaccard Similarity?

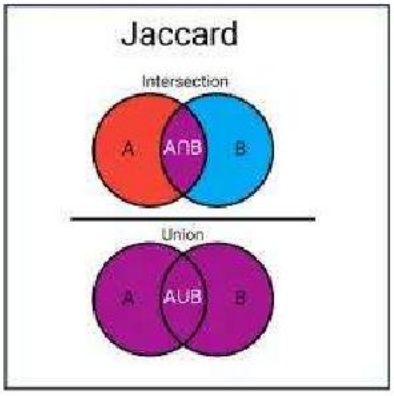
- Jaccard Similarity is a measure of similarity between two asymmetric binary vectors or we can say a way to find the similarity between two sets.
- It is a common proximity measurement used to compute the similarity of two items, such as two text documents.
- The index ranges from 0 to 1. Range closer to 1 means more similarity in two sets of data.

It is calculated by the formula:
Jaccard Similarity = (number of observations in both sets) / (number in either set)
or mathematically,
J(A, B) = $|A \cap B| / |A \cup B|$
If two datasets share exact same members then their Jaccard Similarity Index will be 1 and if there are no common members then Jaccard Similarity index will be 0.

Example:
As similar to Example 1, if we have text instead of numbers,
Set A = {'Lion' , 'Tiger' , 'Cheetah' , 'Leopard' , 'Rhino'}
Set B = {'Lion' , 'Monkey' , 'Cheetah' , 'Cat' , 'Dog'}

Number of observation in both: $(A \cap B)$ = {'Lion' , 'Cheetah'} = 2

Number of observation in either: (A∪B) = {'Lion' , 'Tiger' , 'Cheetah' , 'Leopard' , 'Rhino' , 'Monkey' , 'Cat' , 'Dog'} = 8
Hence, J(A, B) = |A∩B| / |A∪B| = |2| / |8| = 0.25
Since this number is fairly low and close to 0, it indicates that the two sets are quite dissimilar.



Example: Jaccard Similarity in Action
Let's imagine these sets:

- Set A: {Apple, Banana, Orange, Grape}
- Set B: {Banana, Pineapple, Kiwi, Apple}
- Intersection (A ∩ B): {Apple, Banana}
- Union (A ∪ B): {Apple, Banana, Kiwi, Orange, Grape, Pineapple}

Jaccard Similarity = 2 / 6 = 0.33
This indicates a moderate degree of similarity between the two sets

Hamming distance

- In machine learning, the hamming distance measures the similarity between two strings of the same length. It is the number of positions at which the corresponding characters are different.

- Let's understand the concept using an example. Let's say we have two strings: **"euclidean"** and **"manhattan"**
- Since the length of these strings is equal, we can calculate the Hamming Distance. We will match the strings character by character. The first character of both strings (e and m, respectively) differs.
- Similarly, the second character of both strings (u and a) is different, and so on.
- Look carefully – seven characters are different, whereas two characters (the last two characters) are similar:

- Hence, the hamming distance here will be 7. Note that the larger the hamming distance between two strings, the more dissimilar those strings will be (and vice versa).

2. Distance Measures

- Distance measure is used to find the dissimilarity between patterns represented as vectors.
- Patterns which are more similar should be closer.
- The distance function (d) could be a metric or a non-metric.
- The most popularly used family of distance metric is called the Minkowski metric.
- A metric is a type of measure that possesses three key features: positive reflexivity, symmetry and triangular inequality:
  Positive reflexivity: d(x,x)=0
  Symmetry: d(x,y)=d(y,x)
  Traingular inequality: d(x,y)<=d(x,z)+d(z,y), where x,y,z are three patterns.

  - Distance metrics play an important role in machine learning. They provide a strong foundation for several machine learning algorithms like k-nearest neighbors for supervised learning and k-means clustering for unsupervised learning. Different distance metrics are chosen depending upon the type of the data.

Why Distance Matters in Machine Learning

When we classify data, distance tells us how close an unknown data point is to known points, determining its label. In clustering, distance measures help us group data points based on their similarities.

Types of Distance Measures

A. Euclidean Distance (L2 Norm)

Euclidean distance is the straight-line distance between two points in Euclidean space. Think of it like measuring the shortest distance between two points on a map.
Formula:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Euclidean Distance Formula

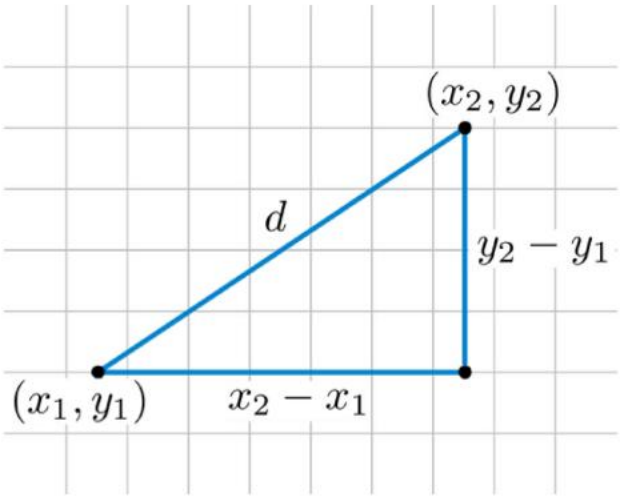
Example: If you are looking at two points A=(1,2) and B=(4,6), Euclidean Distance between them

$$d(A,B) = \sqrt{(4-1)^2 + (6-2)^2} = 5$$

d (A, B)

Diagram:

Imagine a 2D plane with two points. The Euclidean distance is the hypotenuse of the right-angled triangle formed by the points' horizontal and vertical distances.



This is the most commonly used distance metric in k-NN due to its simplicity and accuracy in many scenarios.

B. Manhattan Distance (L1 Norm)

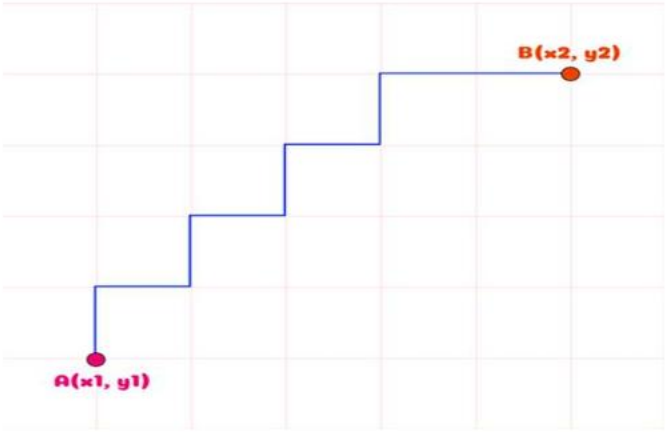
Manhattan Distance calculates the absolute differences along each axis. Imagine walking through city blocks to get from one point to another. formula:

$$d(p, q) = \sum_{i=1}^{n} |q_i - p_i|$$

Manhattan Distance

Example: Using points A=(1,2) and B=(4,6):
d(A,B)=|4−1|+|6−2|=7

Diagram:
Think of how you navigate a grid-like city such as Manhattan. You cannot travel directly, but move along grid lines.



This distance metric is often used in cases where movement is restricted to vertical and horizontal paths, or when outliers are expected in the data.

C. Minkowski Distance

A generalization of both Euclidean and Manhattan distances, Minkowski Distance has a variable p-parameter. If p=1, it's Manhattan Distance; if p=2, it's Euclidean Distance.

$$d(p,q) = \left(\sum_{i=1}^{n} |q_i - p_i|^p\right)^{1/p}$$

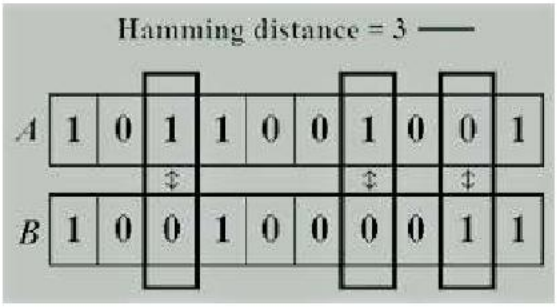
Example:
Using points A=(1,2) and B=(4,6) with p=3:
d(A,B)=(|4−1|^3+|6−2|^3)^1/3=4.64

D. Hamming Distance

Hamming Distance is used for categorical variables and counts how many positions differ between two strings. It's popular in genetics and telecommunications.

Example:
For strings "10101" and "10011," Hamming Distance is **2** (since the first and third positions differ).

Diagram:
Imagine comparing two binary strings like '101101001' and `1001000011`, the Hamming distance is 1 since only the second position is different.



Hamming distance is useful in categorical data or cases where features are binary (e.g., text processing).

E. Cosine Distance & Cosine Similarity

Cosine Similarity measures the angle between two vectors, making it great for high-dimensional spaces like text data. This measure is particularly popular in NLP applications, such as document similarity.

Formula:

$$\text{cosine similarity} = \frac{A \cdot B}{||A|| \times ||B||}$$

Cosine Similarity

Tip: If two vectors have a cosine similarity close to 1, they are more similar.
Cosine similarity is commonly used in text analysis, especially when comparing high-dimensional data like word embeddings, where magnitudes are less important than direction.

4. Choosing the Right Distance Measure

The best distance measure depends on your dataset and objective:

- **Euclidean** and **Manhattan** distances work well with numerical data.
- **Cosine similarity** is ideal for high-dimensional, sparse data like text.
- **Hamming** is suitable for categorical data or binary strings.

3. Non-Metric Similarity Functions

- This category includes similarity functions that do not obey the triangular inequality or symmetry.
- They are commonly used for image for string data and they are resistant to outliers or extremely noisy data.
- The squared Euclidean distance is an example of non-metric, but it provides the same ranking as the Euclidean distance metric.
- One example of a non-metric similarity function is k-median distance between two vectors.
- Given x=(x(1),x(2), ….,x(l)), and y= (y(1), y(2), …..y(l)), the formula for the k-median distance is
  D(x,y)=k-median{|x(1)-y(1)|,….|x(n)-y(n)|},

k-Median Distance

• The k-Median distance is a non-metric distance function.
• This finds the distance between two vectors.
• The difference between the values for each element of the vector is found.

• Putting this together, we get the difference vector.
• If the values in the difference vector are put in non-decreasing order, the k th value gives the k-Median distance.

If the two vectors are {p1, p2, ...pn} and {q1, q2, ..., qn}, Then the difference vector will be D = {| p1 − q1 |, | p2 − q2 |, ..., | pn − qn |} Then d (P, Q) = k-median (D)

As an example, let us take P = (10, 56, 23, 32, 78, 99, 65, 83, 1, 53) and Q = (34, 87, 94, 21, 49, 81, 23, 77, 34, 61)
Then the difference vector will be D =(| 10 − 34 |, | 56 − 87 |, | 23 − 94 |, | 32 − 21 |, | 78 − 49 |, | 99 − 81 |, | 65 − 23 |, | 83 − 77 |, | 1 − 34 |, | 53 − 61 |) = (24, 31, 71, 11, 29, 18, 42, 6, 33, 8)
If D is arranged in non-decreasing order we get D' = (6, 8, 11, 24, 29, 21, 31, 33, 42, 71) If k = 5, then d(P,Q) = 29

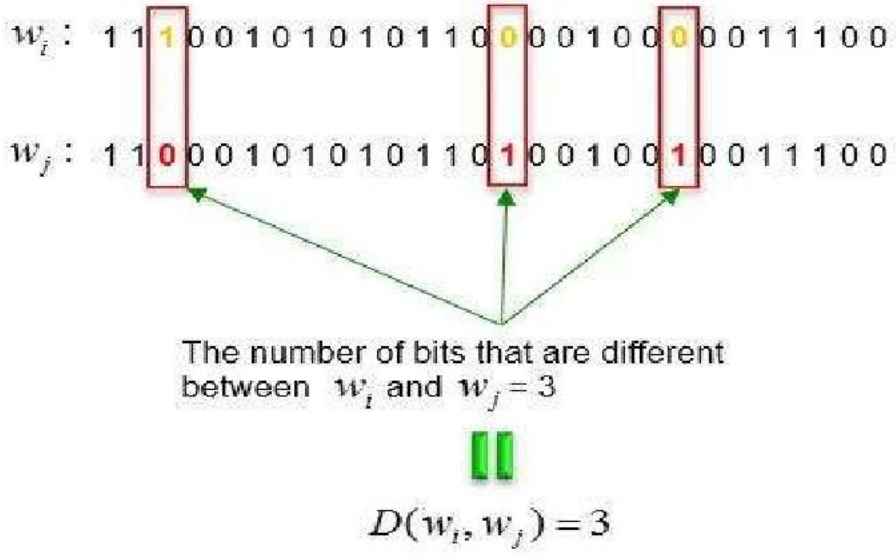
4. Proximity Between Binary Patterns

Binary Patterns are nothing but binary strings of length l.
Some of the popular proximity measures on such binary patterns are:

1. Hamming Distance
2. Simple Matching Coefficient
3. Jaccard Coefficient

Hamming Distance

The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different. It is used in various applications, such as error detection and correction, DNA sequence analysis, and cryptography.



Formula for Hamming Distance

For two strings a and b of equal length, the Hamming distance dH(a,b) is calculated as follows:

where:

- n is the length of the strings.
- ai and bi are the symbols at the i-th position of strings a and b, respectively.
- The expression (ai≠bi) evaluates to 1 if ai is not equal to bi, and 0 otherwise.

Applications of Hamming Distance

1. **Error Detection and Correction**: Hamming distance is used in coding theory to detect and correct errors in data transmission. Hamming codes, for example, are a class of error-correcting codes that use Hamming distance to detect and correct single-bit errors.
2. **DNA Sequence Analysis**: In bioinformatics, Hamming distance is used to compare DNA sequences and identify mutations or similarities between genetic codes.
3. **Cryptography**: Hamming distance is used to measure the similarity between cryptographic keys or hash values.

Simple Matching Coefficient (SMC)

The **Simple Matching Coefficient (SMC)** is a similarity measure used to compare two binary (0/1) vectors. It considers the proportion of matching components (both 0s and 1s) between the vectors.

Formula:

$$\text{SMC} = \frac{(a+d)}{(a+b+c+d)}$$

Where:

- a: Number of positions where both vectors are **1**.
- d: Number of positions where both vectors are **0**.
- b: Number of positions where the first vector is **1** and the second is **0**.
- c: Number of positions where the first vector is **0** and the second is **1**.

Key Points:

- Values range from 0 (completely dissimilar) to 1 (completely similar).
- It equally weights match for both **1s** and **0s**.

Jaccard Similarity

Jaccard Similarity is a measure of similarity between two asymmetric binary vectors or we can say a way to find the similarity between two sets. It is a common proximity measurement used to compute the similarity of two items, such as two text documents. The index ranges from 0 to 1. Range closer to 1 means more similarity in two sets of data.

It is denoted by J and it is also referred as Jaccard Index, Jaccard Coefficient, Jaccard Dissimilarity, and Jaccard Distance. It is frequently used in Data Science and Machine Learning such as Text Mining, E-Commerce, Recommendation System, etc.

It is calculated by the formula:
Jaccard Similarity = (number of observations in both sets) / (number in either set)
or mathematically,
$J(A, B) = |A \cap B| / |A \cup B|$

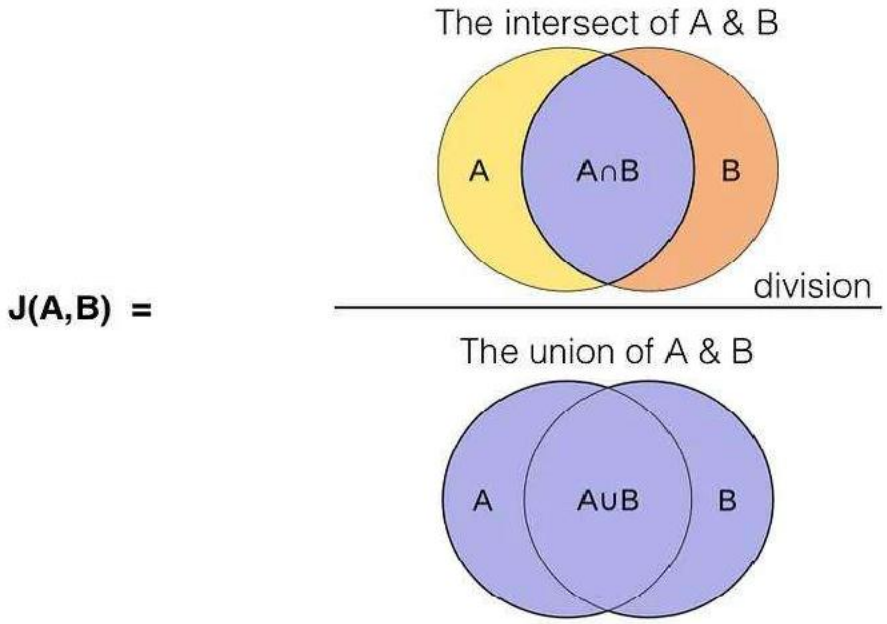
If two datasets share exact same members then their Jaccard Similarity Index will be 1 and if there are no common members then Jaccard Similarity index will be 0.
Jaccard Similarity will tell us that how much features are similar to each other in the dataset.

The Jaccard Distance
It measures the dissimilarity between two sets. It is calculated as:
Jaccard Distance = 1 — Jaccard Similarity

Example to calculate Jaccard Similarity and Distance
Example 1:
Suppose we have two sets:
Set A = {1,3,5,7,9}
Set B = {1,2,3,4,5,6,7,8}
Then, to compute the Jaccard Similarity between them, we first count the total number of observations in both sets, then divide that total number by the total number of observations in either set.
i.e.
Number of observations in both: $(A \cap B) = \{1,3,5,7\} = 4$
Number of observations in either: $(A \cup B) = \{1,2,3,4,5,6,7,8,9\} = 9$
Hence, **J (A, B) =** $|A \cap B| / |A \cup B| = |4| / |9| = 0.44$
As this number is close to one we can say that the two sets are quite similar.
Jaccard Distance = 1–0.44 = 0.56
Example 2:
As similar to Example 1, if we have text instead of numbers,
Set A = {'Lion' , 'Tiger' , 'Cheetah' , 'Leopard' , 'Rhino'}
Set B = {'Lion' , 'Monkey' , 'Cheetah' , 'Cat' , 'Dog'}
Same as we did in Example 1,
Number of observation in both: $(A \cap B)$ = {'Lion' , 'Cheetah'} = 2

Number of observation in either: (A∪B) = {'Lion' , 'Tiger' , 'Cheetah' , 'Leopard' , 'Rhino' , 'Monkey' , 'Cat' , 'Dog'} = 8
Hence, **J(A, B)** = |A∩B| / |A∪B| = |2| / |8| = 0.25
Since this number is fairly low and close to 0, it indicates that the two sets are quite dissimilar.
Jaccard Distance = 1–0.25 = 0.75

Visualize the Jaccard Similarity
Jaccard Similarity can be easily visualized using Venn diagrams. Making it one of the easiest machine learning formulae to understand.



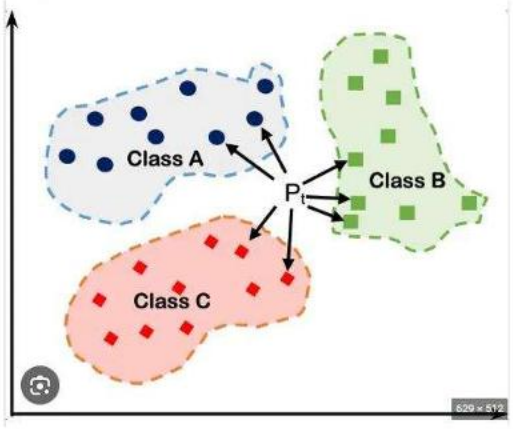
5. Different Classification Algorithms Based on the Distance Measures: K-Nearest Neighbour Classifier

- A K Nearest Neighbour classifier is a machine learning model that makes predictions based on the majority class of the K nearest data points in the feature space.
- The KNN algorithm assumes that similar things exist in close proximity, making it intuitive and easy to understand.
- K-Nearest Neighbours (KNN) is a simple, yet powerful, supervised machine learning algorithm for classification and regression task.
- It is a non-parametric and lazy learning algorithm, meaning that it makes predictions based on the entire training dataset and does not make any assumptions about the underlying data distribution.

- Given a new, unseen data point, KNN identifies the k-nearest neighbors to that point from the training dataset. The term "nearest" is typically defined by a distance metric, such as Euclidean distance.
- The distance metric used depends on the specific problem, such as Euclidean distance, Manhattan distance, cosine distance.
- The algorithm looks at the class labels of the k-nearest neighbors for classification tasks and assigns the majority class to the new data point. This is known as majority voting.

How the algorithm works?

- Given a test data point we calculate its distance between every other datapoint in the train data set and find the top k smallest distance and perform a majority voting on the data points corresponding to the smallest distance.

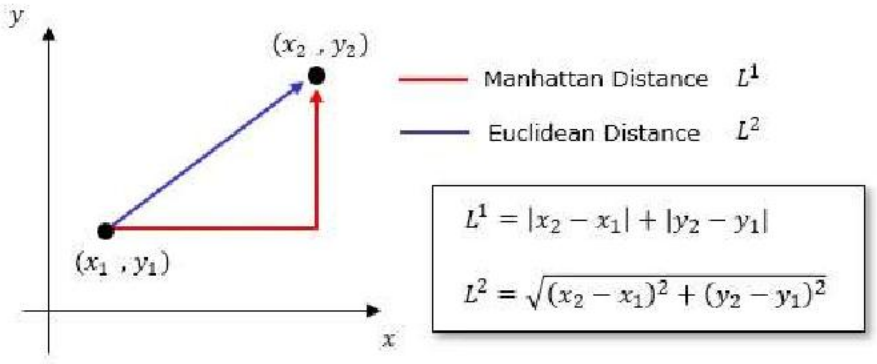


- In the above image, P_t is a test data point. Now we need to find the class it belongs to.
- There are three classes — red, green and blue.
- Here we have considered K = 7. So, we take 7 nearest neighbours and perform majority voting. We can observe that 2 data points belong to class blue, 2 are class red and 3 from class green.
- Now, according to KNN the data point Pt belongs to class green as majority of its neighbours belongs to class green.
- The value 'k' in KNN can be any integer ≥ 1, but its always better to choose k as an odd number as we perform majority voting.
- For example, if we take k as 4, there can be 2 datapoints belonging to class 1 and 2 datapoints belonging to class 2.
- In such a case we cannot perform majority voting. So, it is always best to choose k as an odd number since we will always have a majority class.

Step 1 - Assign a value to K.
Step 2 - Calculate the distance between the new data entry and all other existing data entries. Arrange them in ascending order.
Step 3 - Find the K nearest neighbors to the new entry based on the calculated distances.
Step 4 - Assign the new data entry to the majority class in the nearest neighbors.

How to calculate the distance?

- Till now we have seen that we calculate the distance between the points. Now lest understand what distance means and different types of distance.
- The distance between data points is a critical component for determining the "closeness" or similarity between instances.
- The choice of distance metric plays a crucial role in the performance of the KNN algorithm. Different distance metrics may be suitable for different types of data and tasks.
- Now let us consider two datapoints X_1 (x_{11},y_{12}) and X_2 (x_{21},y_{22}) where x_{11} represents data point 1 and feature 1,y_{12} represents data point 1 and feature 2.



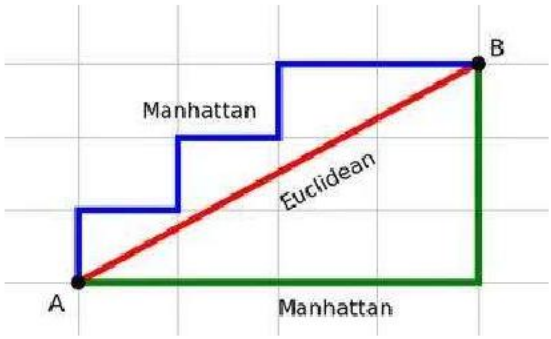
Euclidean and Manhattan distance explained with a image

1. Euclidean distance: It's the shortest line between any two points/vector. Euclidean distance is also called as L2 norm.

distance = $\sqrt{(x_{12}-x_{22})^2+(y_{12}-y_{22})^2} = ||x_1-x_2||_2$
We can also rewrite this as
$||x_1-x_2||_2 = (\sum(x_{1i}-x_{2i})^2)^{1/2}$ where i=1 to n
The right side of this equation is called L2 Norm

2. Manhattan distance: Also known as Taxicab norm or L1 norm. L1 Norm is the sum of the magnitudes of the vectors in a space. It is the most natural way of measuring distance between vectors, that is the sum of absolute difference of the components of the vectors. In this norm, all the components of the vector are weighted equally.

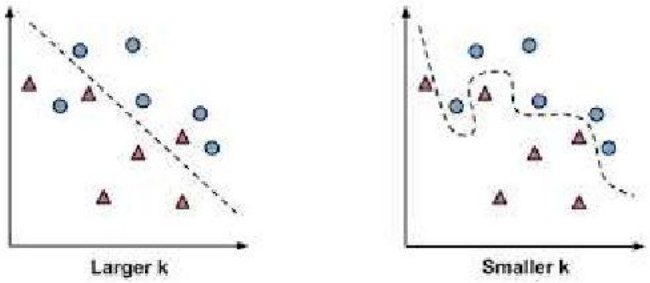
distance =$|x_{12}-x_{22}| + |y_{12}-y_{22}| = ||x_1-x_2||_1$
which can also be rewritten as
$||x_1-x_2||_1 = \sum abs(x_{1i}-x_{2i})$ where i=1 to n
This is called L1 norm



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

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

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.
- The impact of selecting a smaller or larger K value on the model
- **Larger K value:** The case of underfitting occurs when the value of k is increased. In this case, the model would be unable to correctly learn on the training data.
- **Smaller k value:** The condition of overfitting occurs when the value of k is smaller. The model will capture all of the training data, including noise. The model will perform poorly for the test data in this scenario



Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data.
- It can be more effective if the training data is large.
- No training is required for classification.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex sometimes.
- The computation cost is high because of calculating the distance between the data points for all the training samples.
- A lot of memory is required for processing large data sets.
- KNN is also called as a lazy classifier as it does nothing but just stores /memorizes the data in the training phase. The major computation happens only during the test phase.

Real-time applications of K-Nearest Neighbour

Real-Life Examples

Here are concise examples of real-life use of K-nearest neighbors (KNN):

1. **Netflix**: Uses KNN for recommending movies and TV shows based on user preferences by identifying similar users.
2. **Amazon**: Utilizes KNN in its recommendation engine to suggest products based on customers' browsing and purchase histories.
3. **Spotify**: Employs KNN to create personalized music playlists and recommend songs by analyzing user listening habits.
4. **Zillow**: Uses KNN to estimate property values by comparing similar properties in terms of location, size, and features.
5. **Healthcare Institutions**: Apply KNN to predict diseases and assist in diagnosis by analyzing patient data and finding similar cases.

KNN is used in many areas:

- Text categorization: Classifying documents into predefined topics
- Image classification: Identifying objects or scenes in images
- Customer segmentation: Grouping customers based on purchasing behavior

Example

We have data from the questionnaires survey (to ask people opinion) and objective testing with two attributes (acid durability and strength) to classify whether a special paper tissue is good or not. Here are four training samples

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Y = Classification
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

Now the factory produces a new paper tissue that pass laboratory test with X1 = 3 and X2 = 7.

Without another expensive survey, can we guess what the classification of this new tissue is?

1. Determine parameter K = number of nearest neighbours

Suppose use K = 3

2. Calculate the distance between the query-instance and all the training samples

Coordinate of query instance is (3, 7), instead of calculating the distance we compute square distance which is faster to calculate

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Square Distance to query instance (3, 7)
7	7	$(7-3)^2+(7-7)^2=16$
7	4	$(7-3)^2+(4-7)^2=25$
3	4	$(3-3)^2+(4-7)^2=9$
1	4	$(1-3)^2+(4-7)^2=13$

3. Sort the distance and determine nearest neighbours based on the K-th minimum distance

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Square Distance to query instance (3, 7)	Rank minimum distance	Is it included in 3-Nearest neighbors?
7	7	$(7-3)^2+(7-7)^2=16$	3	Yes
7	4	$(7-3)^2+(4-7)^2=25$	4	No
3	4	$(3-3)^2+(4-7)^2=9$	1	Yes
1	4	$(1-3)^2+(4-7)^2=13$	2	Yes

4. Gather the category (Y) of the nearest neighbours. Notice in the second row last column that the category of nearest neighbour (Y) is not included because the rank of this data is more than 3 (=K).

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Square Distance to query instance (3, 7)	Rank minimum distance	Is it included in 3-Nearest neighbors?	Y = Category of nearest Neighbor
7	7	$(7-3)^2+(7-7)^2=16$	3	Yes	**Bad**
7	4	$(7-3)^2+(4-7)^2=25$	4	No	-
3	4	$(3-3)^2+(4-7)^2=9$	1	Yes	**Good**
1	4	$(1-3)^2+(4-7)^2=13$	2	Yes	**Good**

5. Use simple majority of the category of nearest neighbours as the prediction value of the query instance

We have 2 good and 1 bad, since 2>1 then we conclude that a new paper tissue that pass laboratory test with X1 = 3 and X2 = 7 is included in Good category.

6. Radius Distance Nearest Neighbour Algorithm

- Radius Nearest Neighbours implements the nearest neighbours vote, where the neighbours are selected from within a given radius. For regression, the target is predicted by local interpolation of the targets associated of the nearest neighbours in the training set.
- It is an extension to the k-nearest neighbours' algorithm that makes predictions using all examples in the radius of a new example rather than the k-closest neighbours.
- Radius Nearest Neighbours: A technique for finding data points in close proximity within a specified radius.
- It is useful in various applications, such as clustering, classification, and anomaly detection, and helps uncover patterns and trends within the data for more accurate predictions and insights.
- Radius Nearest Neighbours can help uncover patterns and trends within the data, enabling more accurate predictions and insights.
- One of the main challenges in implementing Radius Nearest Neighbours is the computational complexity involved in searching for nearest neighbours, especially in high-dimensional spaces.
- Radius Neighbours is also one of the techniques based on instance-based learning.
- Models based on instance-based learning generalize beyond the training examples.
- To do so, they store the training examples first. When it encounters a new instance (or test example), then they instantly build a relationship between stored training examples and this new instant to assign a target function value for this new instance.
- Instance-based methods are sometimes called lazy learning methods because they postponed learning until the new instance is encountered for prediction.

Radius Neighbors Classifier:

Basic Assumptions

1. All instances correspond to points in the n-dimensional space where n represents the number of features in any instance.
2. The neighbors of an instance are defined in terms of the Euclidean distance.

An instance can be represented by $< x_1, x_2, ..., x_n >$.

Euclidean distance between two instances x^a and x^b is given by $d(x^a, x^b)$:

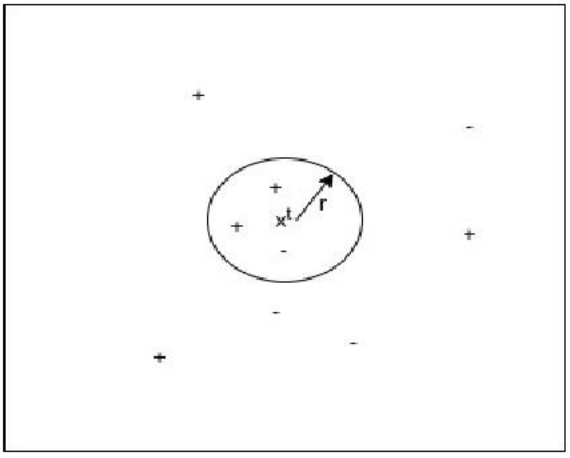
$$\sum j=1n(xja-xjb)2 \ \sum j=1n(xja-xjb)2$$

How does Radius Nearest Neighbors work?

- Radius Nearest Neighbors works by calculating the distance between a given data point and all other data points in the dataset.
- It then identifies the data points that are within a specified radius of the given point.
- The distance can be calculated using various metrics, such as Euclidean distance, Manhattan distance, or cosine similarity.

- Radius Neighbours Classifier first stores the training examples.
- During prediction, when it encounters a new instance (or test example) to predict, it finds the number of neighbours from training instances within a fixed radius of *r* centre at test instance, where r is a floating-point value specified by the user.
- Then assigns the most common class among the training instances within that radius to the test instance.

The optimal choice for r is by validating errors on test data.

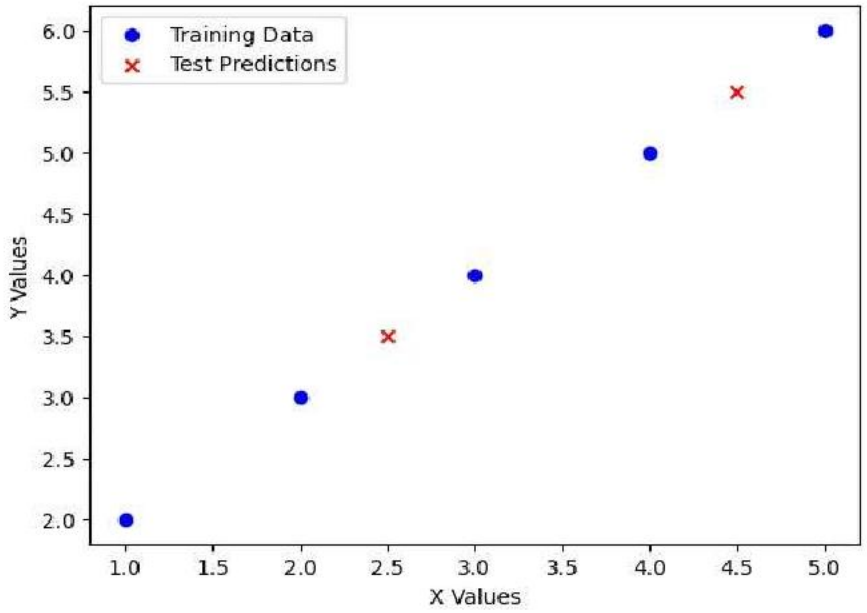


- In the above figure, "+" denotes training instances labelled with 1. "-" denotes training instances with 0.
- Here we classified for the test instance x_t as the most common class among training instances within the circle.

- Here, r is a user-specified choice. In the above figure, positive instances are in majority in the circle, so x^t is classified as "+" or 1.

7. KNN Regression

- KNN regression is a non-parametric method used for predicting continuous values.
- The core idea is to predict the target value for a new data point by averaging the target values of the K nearest neighbours in the feature space.
- The distance between data points is typically measured using Euclidean distance, although other distance metrics can be used.



- K-Nearest Neighbours (KNN) is a non-parametric machine learning algorithm that can be used for both classification and regression tasks.
- In the context of regression, KNN is often referred to as "K-Nearest Neighbours Regression" or "KNN Regression."
- It's a simple and intuitive algorithm that makes predictions by finding the K nearest data points to a given input and averaging their target values (for numerical regression) or selecting the majority class (for classification).

Here's an overview of how KNN Regression works:

1. **Data Collection:** You start with a dataset that includes both input features and target values. In regression tasks, the target values are continuous and represent the output you want to predict.
2. **Choosing the Number of Neighbors (K):** You need to choose the number of nearest neighbors, K, that will be used to make predictions. This is a hyperparameter that you can tune based on the characteristics of your data. A small K (e.g., 1 or 3) may lead to noisy predictions, while a large K may lead to overly smoothed predictions.

3. **Distance Metric:** KNN relies on a distance metric (e.g., Euclidean distance) to measure the similarity between data points. Different distance metrics can be used depending on the nature of your data.
4. **Prediction:** When you want to make a prediction for a new input data point, KNN calculates the distance between this point and all other data points in the dataset. It then selects the K data points with the smallest distances.
5. **Regression Prediction:** For regression, the predicted value for the new data point is the average of the target values of the K nearest neighbors. This could be a simple arithmetic mean.

Example:

- Let us now discuss a numerical example to understand the KNN regression algorithm in a better way.
- To discuss the numerical example of N-Nearest Neighbours Regression, we will use the following dataset.

Length	Weight	Cost
10	15	45
11	6	37
12	14	48
7	9	33
9	14	38
8	12	40
6	11	35
15	10	50
14	8	46
7	12	35
10	6	36
13	8	44
9	7	32

5	8	30
5	10	30

Dataset for KNN Regression

- In the above dataset, we have 15 data points. The dataset contains the length and weight of metal rods along with their cost.
- **Now, suppose that we want to calculate the cost for a rod with a length of 7 and a weight of 8. For this, we will use the following steps.**
- First, we will decide on the value of K. We will take 3 as the number of closest neighbours used to decide the cost of the input data point.
- Next, we will calculate the distance of the new data point i.e. (7, 8) to all the existing points in the dataset. Here, we will use the Euclidean distance measure. I have tabulated the distances in the below table.

Point	**Distance from (7,8)**
(10, 15)	7.61
(11, 6)	4.47
(12, 14)	7.81
(7, 9)	1.0
(9, 14)	6.32
(8, 12)	4.12
(6, 11)	3.16
(15, 10)	8.24
(14, 8)	7.0
(7, 12)	4.0
(10, 6)	3.60
(13, 8)	6.0
(9, 7)	2.23
(5, 8)	2.0

(5, 10)	2.82

Distance Table

- Now, we have found the distance of all the data points in the dataset from the point (7, 8). Next, we have to find the three closest points in the dataset. For this, we will sort the points according to their distances from (7, 8). The result is tabulated below.

Point	**Distance from (7,8)**
(7, 9)	1
(5, 8)	2
(9, 7)	2.23
(5, 10)	2.82
(6, 11)	3.16
(10, 6)	3.6
(7, 12)	4
(8, 12)	4.12
(11, 6)	4.47
(13, 8)	6
(9, 14)	6.32
(14, 8)	7
(10, 15)	7.61
(12, 14)	7.81
(15, 10)	8.24

Sorted Distance Table

- In the above table, you can observe that the three points closest to (7, 8) are (7, 9), (5, 8), and (9, 7). These points have costs of 33, 30, and 32.
- To calculate the cost of a rod with length 7 and weight 8, we can take the average of the above costs. Hence, the cost for (7, 8) will be 31.67.
- Thus, we have found the cost of the rod using the given dataset and the KNN regression algorithm in this numerical example.

- Now, we will discuss some of the applications, advantages, and disadvantages of the K-Nearest Neighbors (KNN) regression algorithm.

Applications of K-Nearest Neighbors Regression

KNN regression is a simple yet powerful machine-learning algorithm that has a wide range of applications. Some of the areas where KNN regression is commonly used are as follows.

1. **Real-time prediction**: KNN regression can be used for real-time prediction. It is fast and efficient during the prediction stage. This makes it suitable for applications such as stock price prediction, weather forecasting, and demand forecasting.
2. **Recommender systems**: You can use KNN regression for recommender systems, such as movie and product recommenders. In these applications, the KNN algorithm is used to identify the nearest neighbors in the training set for a given user, and the prediction is made based on the preferences of the nearest neighbors.
3. **Customer Segmentation**: K-nearest neighbors' regression can be a great tool to implement customer segmentation. Based on the features like recency, frequency, average monetary value, and variety, each customer can be assigned a score. The scores and features of the customer in the dataset can then be used to find the score for new customers.
4. **Medical diagnosis**: KNN regression can be used for medical diagnosis. Based on the metrics in the test reports, it can identify the probability of any disease in the patient by matching it with the historical data. In a similar way, we can use KNN regression to find similar protein structures based on their chemical behavior. This makes the KNN regression algorithm useful for applications such as disease diagnosis and drug discovery.
5. **Quality control:** KNN regression can be used for quality control, as it is able to identify patterns in the data that are relevant for quality control. Based on the features of the goods, it can easily identify the quality by looking for similar goods data in the dataset. This makes it useful for applications such as process monitoring and quality inspection.

Advantages of KNN Regression

The K-nearest neighbors' algorithm has various advantages as discussed below.

- **Simple to understand and implement**: KNN regression is one of the simplest machine learning algorithms. It is easy to understand and implement, making it accessible to both practitioners and researchers.
- **No assumptions about data distribution**: Unlike many other regression algorithms, KNN regression does not make any assumptions about the distribution of the data. This makes it suitable for a wide range of datasets, including those with complex or non-linear relationships between features and target variables.
- **Handle noisy data well**: KNN regression is able to handle noisy data well, as it is less sensitive to outliers and extreme values in the data.
- **Versatile**: KNN regression can be used for both regression and classification problems and can handle both continuous and categorical target variables.

- **Can be used for online learning**: KNN regression can be used for online learning, which means it can be updated incrementally as new data becomes available. This can give accurate results in real time.
- **Can work well with small datasets**: Unlike some other algorithms that require a large amount of data to work well, KNN regression can still produce good results with small datasets, as long as the data is representative of the problem space.

Disadvantages of KNN Regression

Apart from its advantages, the KNN regression algorithm also has many disadvantages. Some of them are discussed below.

- **Computational cost**: While predicting the results, the KNN algorithm needs to find the distance between all the data points in the existing dataset and the new dataset. Due to this, computational costs keep increasing as the data size increases. Thus, it might not be computationally efficient for use cases with large data sets.
- **High memory usage**: KNN regression requires a lot of memory to store the entire training dataset, which can be a problem for very large datasets.
- **Hyperparameter sensitivity**: The performance of KNN regression is highly dependent on the choice of the hyperparameter K, which determines the number of nearest neighbors used to make the prediction. Choosing the wrong value of K can result in overfitting or underfitting.
- **Sensitivity to irrelevant features**: KNN regression is sensitive to irrelevant features, as they can have a large impact on the distance metric used to identify the nearest neighbors. This can result in poor performance if the features are not carefully pre-processed.
- **Non-parametric nature**: Unlike other regression algorithms, KNN regression does not provide a model that can be used to make predictions for new data points. Every time, it calculates the distance and then gives the results. This can make it more difficult to interpret the results and understand the relationships between the features and target variables.
- **Not suitable for large datasets with many features**: KNN regression can become computationally infeasible for datasets with a large number of features and data points. In these cases, you can use algorithms like multiple regression or polynomial regression.

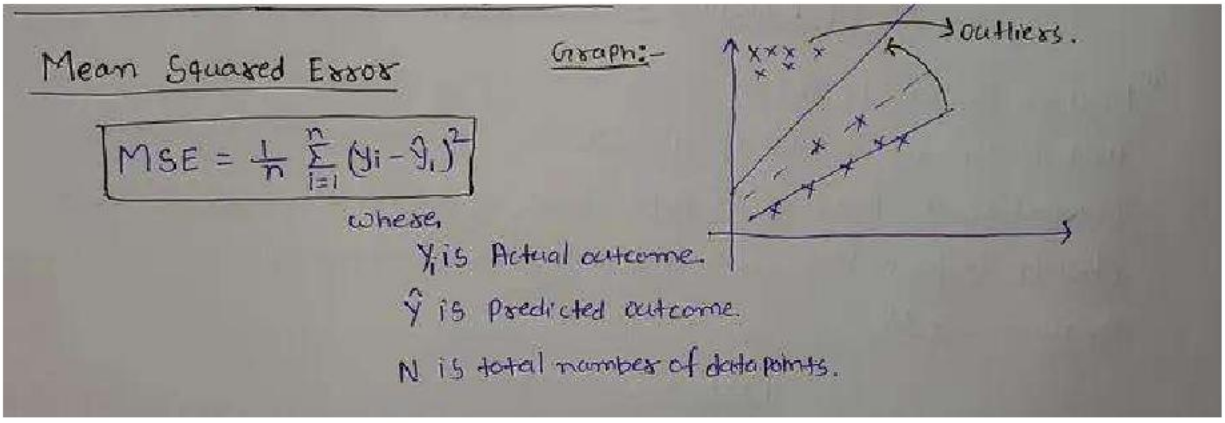
8. Performance of the Regression Algorithms

- Performance metrics are measurements that are used to evaluate the performance of machine learning algorithms.
- Performance metrics play a crucial role in evaluating the effectiveness and accuracy of machine learning models.
- Performance metrics are used in a wide range of machine learning tasks, including classification, regression, clustering, and recommendation systems.
- Evaluating performance of a regression model, though, requires a different approach and different metrics from evaluating classification models.
    1. Mean Squared Error (MSE)

2. Mean Absolute Error (MAE)
3. Root Mean Squared Erro (RMSE)

1. Mean Squared Error (MSE)

- Mean Squared Error (MSE) is a widely used metric and means the average of the squares of the errors between estimated and accurate values.
- It provides a measure of the model's accuracy in terms of mean squared difference between its predictions and real-world observations.



Advantages:

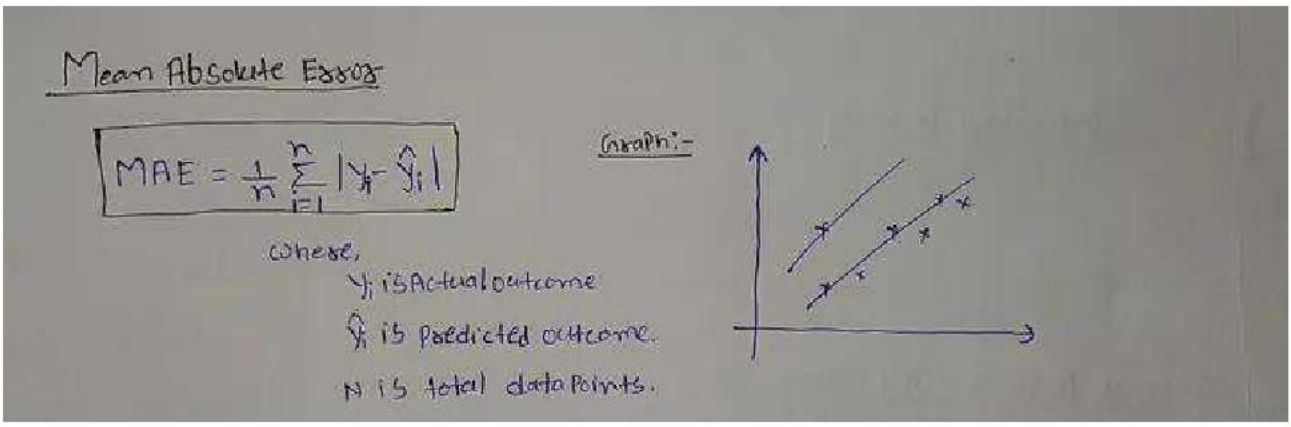
- Differentiable
- It has One Local & Global Minima

Disadvantages:

- Not Robust to Outlier
- It Change its Unit

2. Mean Absolute Error (MAE)

- Mean Absolute Error (MAE) is yet another widely employed measure of regression model precision.
- Unlike MSE, MAE computes the average of absolute differences between predicted and actual values.
- It offers an average measure of the relative magnitude errors, ignoring their direction.



Advantages:

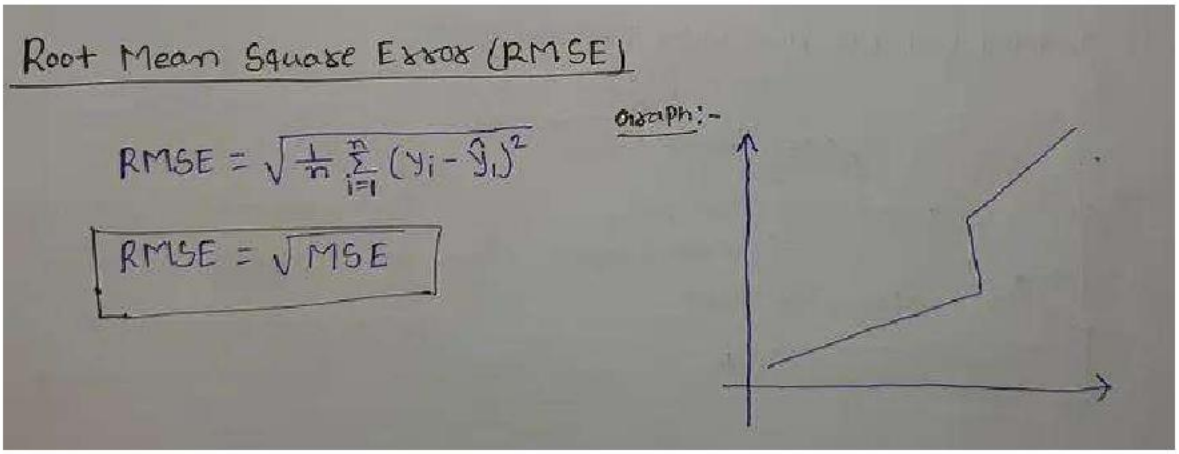
- It is Robust to Outlier
- It will be same Unit

Disadvantages:

- Convergence usually takes more time
- In Optimization take complex process

3. Root Mean Squared Erro (RMSE)

- The Root Mean Squared Error (RMSE) comes from MSE and is a measure of the standard deviation of errors, or residuals, in other words mismatches between predicted and true values.
- RMSE turns out to be particularly handy because its error metric is expressed on the same scale as our target variable, making for a more intuitive interpretation of what really is going on here.



Advantages:

- The value of MSE is same as output unit, which makes the interpretation of loss easy.

Disadvantages:

- Not Robust to Outlier

Supplementary Material

Metric Data

- **Definition**: Metric data, also known as quantitative data, consists of numerical values that can be measured and quantified.
- **Characteristics**:
- **Continuous or Discrete**: Metric data can be continuous (e.g., height, weight, temperature) or discrete (e.g., number of students, count of cars).
- **Meaningful Arithmetic Operations**: You can perform meaningful mathematical operations on metric data, such as addition, subtraction, averaging, etc.

- **Examples**: Temperature (in Celsius or Fahrenheit), distance (in meters or miles), time (in seconds or hours), sales figures.

Non-Metric Data

- **Definition**: Non-metric data, often referred to as qualitative or categorical data, consists of values that represent categories or qualities rather than quantities.
- **Characteristics**:
- **Categorical**: Non-metric data can be nominal (unordered categories, e.g., colors, types of fruit) or ordinal (ordered categories, e.g., satisfaction ratings like poor, fair, good).
- **Limited Arithmetic Operations**: Mathematical operations are generally not meaningful for non-metric data. You can count occurrences or calculate percentages, but you cannot average categories.
- **Examples**: Gender (male, female), marital status (single, married, divorced), survey responses (agree, neutral, disagree).

KNN Regression Algorithm

The KNN regression algorithm can be broken down into the following steps:

1. **Choose a value for K**: We first choose a value for K. This determines the number of nearest neighbors used to make the prediction.
2. **Calculate the distance**: After choosing K, we calculate the distance between each data point in the training set and the target data point for which a prediction is being made. For this, we can use a variety of distance metrics, including Euclidean distance, Manhattan distance, or Minkowski distance.
3. **Find the K nearest neighbors**: After calculating the distance between the existing data points and the new data point, we identify K nearest neighbors by selecting the K data points nearest to the new data point.
4. **Calculate the prediction**: After finding the neighbors, we calculate the value of the dependent variable for the new data point. For this, we take the average of the target values of the K nearest neighbors.

References

https://builtin.com/articles/euclidean-distance
https://www.turing.com/kb/how-to-decide-perfect-distance-metric-for-machine-learning-model
https://builtin.com/machine-learning/cosine-similarity

https://medium.com/geekculture/cosine-similarity-and-cosine-distance-48eed889a5c4
https://medium.com/@mayurdhvajsinhjadeja/jaccard-similarity-34e2c15fb524
https://medium.com/@weidagang/essential-math-for-machine-learning-jaccard-similarity-195040755fd4
https://www.learndatasci.com/glossary/jaccard-similarity/
https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/
https://medium.com/@kunal_gohrani/different-types-of-distance-metrics-used-in-machine-learning-e9928c5e26c7
https://medium.com/@gshriya195/top-5-distance-similarity-measures-implementation-in-machine-learning-1f68b9ecb0a3
https://medium.com/@sayalisureshkumbhar/machine-learning-distance-measures-a-beginners-guide-964348608d95
https://medium.com/analytics-vidhya/role-of-distance-metrics-in-machine-learning-e43391a6bf2e
https://medium.com/@sujathamudadla1213/how-does-the-simple-matching-coefficient-smc-differ-from-hamming-distance-in-measuring-binary-65368624bdd5
https://medium.com/@conniezhou678/understanding-jaccard-similarity-a-powerful-tool-for-data-analysis-42abaaafd782

https://medium.com/towards-data-science/k-nearest-neighbor-classifier-explained-a-visual-guide-with-code-examples-for-beginners-a3d85cad00e1
https://www.freecodecamp.org/news/k-nearest-neighbors-algorithm-classifiers-and-model-example/
https://medium.com/@pritioli/implementing-k-nearest-neighbor-classifier-20a0d469cec6
https://medium.com/@pingsubhak/machine-learning-basics-k-nearest-neighbors-9e8e2d46db75
https://www.youtube.com/watch?v=iCJmTYu6Inw
https://www.analyticsvidhya.com/articles/knn-algorithm/
https://www.linkedin.com/pulse/introduction-knn-divey-anand/
https://vtupulse.com/machine-learning/knn-solved-example-diabetic-patient/
https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
https://www.geeksforgeeks.org/implementation-of-radius-neighbors-from-sratch-in-python/
https://www.activeloop.ai/resources/glossary/radius-nearest-neighbors/
https://medium.com/@nandiniverma78988/understanding-k-nearest-neighbors-knn-regression-in-machine-learning-c751a7cf516c
https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/
https://codinginfinite.com/knn-regression-numerical-example/
https://alok05.medium.com/performance-metrics-or-loss-function-in-machine-learning-for-regression-7fae992577f0
https://www.saedsayad.com/k_nearest_neighbors_reg.htm
https://www.jeremyjordan.me/k-nearest-neighbors/
https://codinginfinite.com/knn-regression-numerical-example/#google_vignette

https://keylabs.ai/blog/k-nearest-neighbors-knn-real-world-applications/

UNIT – III

Models Based on Decision Trees: Decision Trees for Classification, Impurity Measures, Properties, Regression Based on Decision Trees, Bias–Variance Trade-off, Random Forests for Classification and Regression.

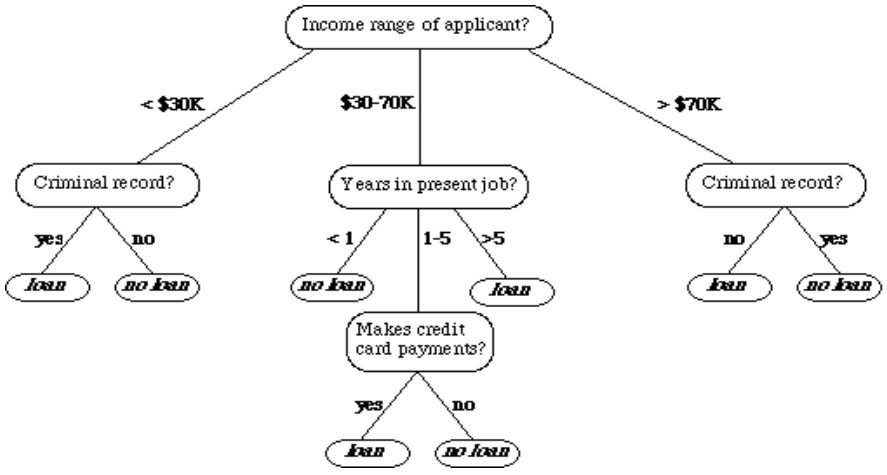
The Bayes Classifier: Introduction to the Bayes Classifier, Bayes' Rule and Inference, The Bayes Classifier and its Optimality, Multi-Class Classification | Class Conditional Independence and Naive Bayes Classifier (NBC)

1. Decision Trees for Classification

- Decision tree is one of the most popular Supervised machine learning algorithms.
- Decision trees are used for classification and regression problems, this story we talk about classification.
- Decision Trees are the foundation for many classical machine learning algorithms like **Random Forests, Bagging**, and **Boosted** Decision Trees.

Why Decision trees?

1. Decision tress often mimic the human level thinking so it's so simple to understand the data and make some good interpretations.
2. Decision trees actually make you see the logic for the data to interpret (not like black box algorithms like SVM, NN, etc.)



For example: if we are classifying *bank loan* application for a customer, the decision tree may look like this

Here we can see the logic how it is making the decision.

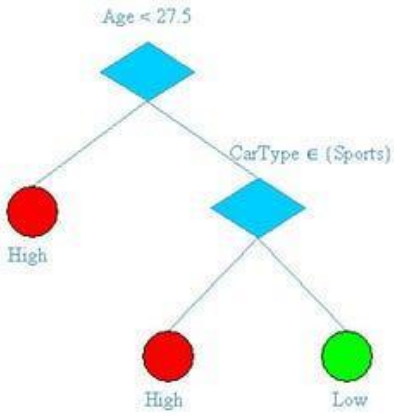
what is the decision tree?

A decision tree is a tree where each node represents a feature(attribute), each link(branch) represents a decision(rule) and each leaf represents an outcome (categorical or continues value).

The whole idea is to create a tree like this for the entire data and process a single outcome at every leaf (or minimize the error in every leaf).

Tid	Age	Car Type	Class
0	23	Family	High
1	17	Sports	High
2	43	Sports	High
3	68	Family	Low
4	32	Truck	Low
5	20	Family	High

Numeric Categorical



1) Age < 27.5 ⇒ High

2) Age >= 27.5 and
CarType = Sports ⇒ High

3) Age >= 27.5 and
CarType ≠ Sports ⇒ High

Types of Decision Tree Algorithms

There are several algorithms for building decision trees, each with slight differences in how they handle data and make splits:

1. ID3 (Iterative Dichotomiser 3): This algorithm picks the feature that gives the largest information gain (i.e., the feature that best separates the data) and builds a multi-way tree. It's *mainly used for categorical data*.
2. C4.5: An improvement over ID3, C4.5 can *handle both categorical and continuous data* by dynamically defining thresholds for continuous variables.
3. CART (Classification and Regression Trees): Used in scikit-learn, CART builds binary trees and works for *both classification and regression* tasks. It doesn't create rule sets like C4.5 but focuses on maximizing the information gained at each split.

Terminology of Decision Trees

The various terminologies that we need to understand before diving deep into decision trees are:

1. **Root Node**: The top node of the tree, that represents the entire dataset. It is the starting point for the decision-making process. Here, the genre of the movie is our lead node as it has the lowest entropy (i.e. least randomness and more certainty) among all features.
2. **Decision Node** (Internal Node): Nodes in the tree that are not leaf nodes. They represent decision points based on specific features.
3. **Leaf Node** (Terminal Node): The end nodes of the tree where the final decision or prediction is made. Each leaf node corresponds to a class label (for classification) or a numerical value (for regression).
4. **Branch**: The path between nodes in the tree. It represents the decision path based on the values of features.
5. **Split:** A decision point at an internal node where the dataset is divided into subsets based on the value of a specific feature.
6. **Decision Criterion:** The criteria used at each internal node to determine the splitting of data. Common criteria include Gini impurity for classification and mean squared error for regression.

7. **Pruning:** The process of removing unnecessary branches from the tree to improve its generalization on new, unseen data and avoid overfitting.
8. **Level:** The depth or distance of a node from the root node. The root node is considered to be at level 0, its child nodes are at level 1, and so on.

Classification with using the ID3 algorithm.

- Let's just take a famous dataset in the machine learning world which is weather dataset (playing game Y or N based on weather condition).

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

- We have four X values (outlook, temp, humidity and windy) being categorical and one y value (play Y or N) also being categorical.
- So, we need to learn the mapping (what machine learning always does) between X and y.
- This is a binary classification problem, let's build the tree using the *ID3* algorithm
- To create a tree, we need to have a root node first and we know that nodes are features/attributes (outlook, temp, humidity and windy),

Which one do we need to pick first?

Answer: determine the attribute that best classifies the training data; use this attribute at the root of the tree. Repeat this process at for each branch.

This means we are performing top-down, greedy search through the space of possible decision trees.

At each level, the algorithm evaluates all features and picks the one that results in the greatest improvement in accuracy. This is called the greedy approach.

How do we choose the best attribute?

Use the attribute with the highest ***information gain*** in ***ID3.***

In order to define information, gain precisely, we begin by defining a measure commonly used in information theory, called entropy that characterizes the (im)purity of an arbitrary collection of examples."

Entropy

Entropy $H(S)$ is a measure of the amount of uncertainty in the (data) set S (i.e. entropy characterizes the (data) set S).

$$H(S) = \sum_{c \in C} -p(c) \log_2 p(c)$$

Where,

- S – The current (data) set for which entropy is being calculated (changes every iteration of the ID3 algorithm)
- C – Set of classes in S C={ yes, no }
- $p(c)$ – The proportion of the number of elements in class c to the number of elements in set S

When $H(S) = 0$, the set S is perfectly classified (i.e. all elements in S are of the same class).

In ID3, entropy is calculated for each remaining attribute. The attribute with the **smallest** entropy is used to split the set S on this iteration. The higher the entropy, the higher the potential to improve the classification here.

For a binary classification problem

- If all examples are positive or all are negative then entropy will be *zero* i.e., low.
- If half of the examples are of positive class and half are of negative class then entropy is one i.e., high.

Information gain

Information gain $IG(A)$ is the measure of the difference in entropy from before to after the set S is split on an attribute A. In other words, how much uncertainty in S was reduced after splitting set S on attribute A.

$$IG(A, S) = H(S) - \sum_{t \in T} p(t) H(t)$$

Where,

- $H(S)$ – Entropy of set S
- T – The subsets created from splitting set S by attribute A such that $S = \bigcup_{t \in T} t$
- $p(t)$ – The proportion of the number of elements in t to the number of elements in set S
- $H(t)$ – Entropy of subset t

In ID3, information gain can be calculated (instead of entropy) for each remaining attribute. The attribute with the **largest** information gain is used to split the set S on this iteration.

Let's apply these metrics to our dataset to split the data (getting the root node)

1.compute the entropy for data-set2.for every attribute/feature:

1. calculate entropy for all categorical values
2. take average information entropy for the current attribute
3. calculate gain for the current attribute3. pick the highest gain attribute.

4. Repeat until we get the tree we desired.

<u>Compute the entropy for the weather data set:</u>

$$H(S) = \sum_{c \in C} -p(c)\log_2 p(c)$$

C ={yes,no}

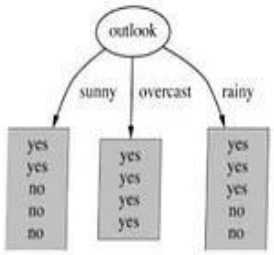
Out of 14 instances, 9 are classified as yes, and 5 as no

pyes = −(9/14)*log2(9/14) = 0.41

pno = −(5/14)*log2(5/14) = 0.53

H (S) = pyes + pno = 0.94

For every feature calculate the entropy and information gain



E (Outlook=sunny) = $-\frac{2}{5}\log\left(\frac{2}{5}\right)-\frac{3}{5}\log\left(\frac{3}{5}\right) = 0.971$

E (Outlook=overcast) = $-1\log(1) - 0\log(0) = 0$

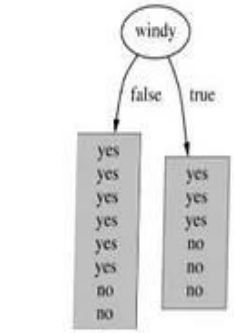
E (Outlook=rainy) = $-\frac{3}{5}\log\left(\frac{3}{5}\right)-\frac{2}{5}\log\left(\frac{2}{5}\right) = 0.971$

} H(S,Outlook)

Average Entropy information for Outlook

I (Outlook) = $\frac{5}{14} * 0.971 + \frac{4}{14} * 0 + \frac{5}{14} * 0.971 = 0.693$ } $\sum_{t \in T} p(t)H(t)$

Gain (Outlook) = E(S) – I (outlook) = 0.94-.693 = 0.247 ⇨ $IG(A,S) = H(S) - \sum_{t \in T} p(t)H(t)$



E (Windy=false) = $-\frac{6}{8}\log\left(\frac{6}{8}\right)-\frac{2}{8}\log\left(\frac{2}{8}\right) = 0.811$

E (Windy=true) = $-\frac{3}{6}\log\left(\frac{3}{6}\right)-\frac{3}{6}\log\left(\frac{3}{6}\right) = 1$

Average entropy information for Windy

I (Windy) = $\frac{8}{14} * 0.811 + \frac{6}{14} * 1 = 0.892$

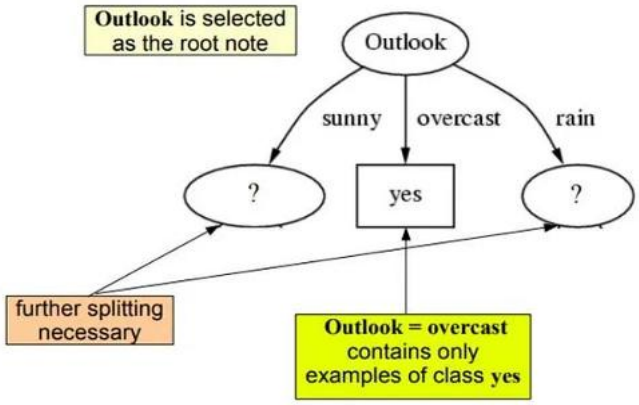
Gain (Windy) = E(S) – I (Windy) = 0.94-0.892=0.048

Similarity we can calculate for other two attributes (Humidity and Temp).

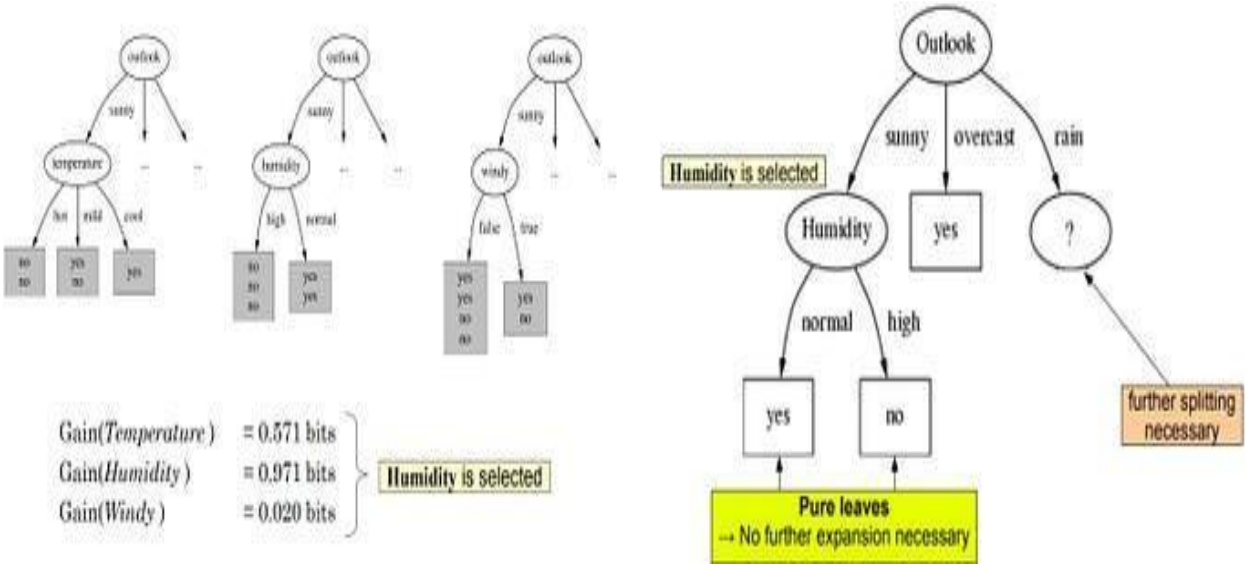
Pick the highest gain attribute

Outlook		Temperature	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	0.247	Gain: 0.940-0.911	0.029
Humidity		**Windy**	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048

So, our root node is Outlook.

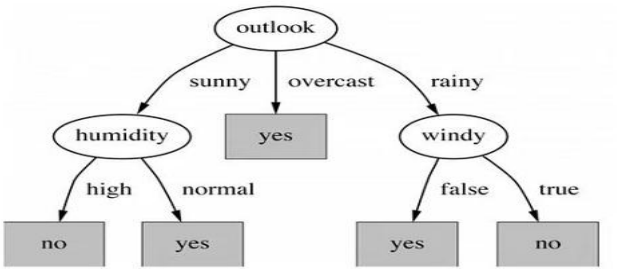


Repeat the same thing for sub-trees till we get the tree.



Finally, we get the tree something like this.

Final decision tree



Classification with using the CART algorithm.

In CART we use Gini index as a metric,
We use the Gini Index as our cost function used to evaluate splits in the dataset.
our target variable is Binary variable which means it take two values (Yes and No). There can be 4 combinations.

Actual=1 predicted 1
1 0 , 0,1, 0 0P(Target=1).P(Target=1) + P(Target=1).P(Target=0) + P(Target=0).P(Target=1) + P(Target=0).P(Target=0) = 1P(Target=1).P(Target=0) + P(Target=0).P(Target=1) = 1 — P^2(Target=0) — P^2(Target=1)
Gini Index for Binary Target variable is
= 1 — P^2(Target=0) — P^2(Target=1)

$$= 1 - \sum_{t=0}^{t=1} P_t^2$$

A Gini score gives an idea of how good a split is by how mixed the classes are in the two groups created by the split. A perfect separation results in a Gini score of 0, whereas the worst case split those results in 50/50 classes.
We calculate it for every row and split the data accordingly in our binary tree. We repeat this process recursively.
For Binary Target variable, Max Gini Index value

= 1 — (1/2) ^2 — (1/2) ^2
= 1–2*(1/2) ^2
= 1- 2*(1/4)
= 1–0.5
= 0.5
Similarly, if Target Variable is categorical variable with multiple levels, the Gini Index will be still similar. If Target variable takes k different values, the Gini Index will be

$$1 - \sum_{t=0}^{t=k} P_t^2$$

Maximum value of Gini Index could be when all target values are equally distributed.
Similarly for Nominal variable with k level, the maximum value Gini Index is
= 1–1/k
Minimum value of Gini Index will be 0 when all observations belong to one label.

1.compute the gini index for data-set2.for every attribute/feature:

1. calculate gini index for all categorical values
2. take average information entropy for the current attribute
3. calculate the gini gain3. pick the best gini gain attribute.

4. Repeat until we get the tree we desired

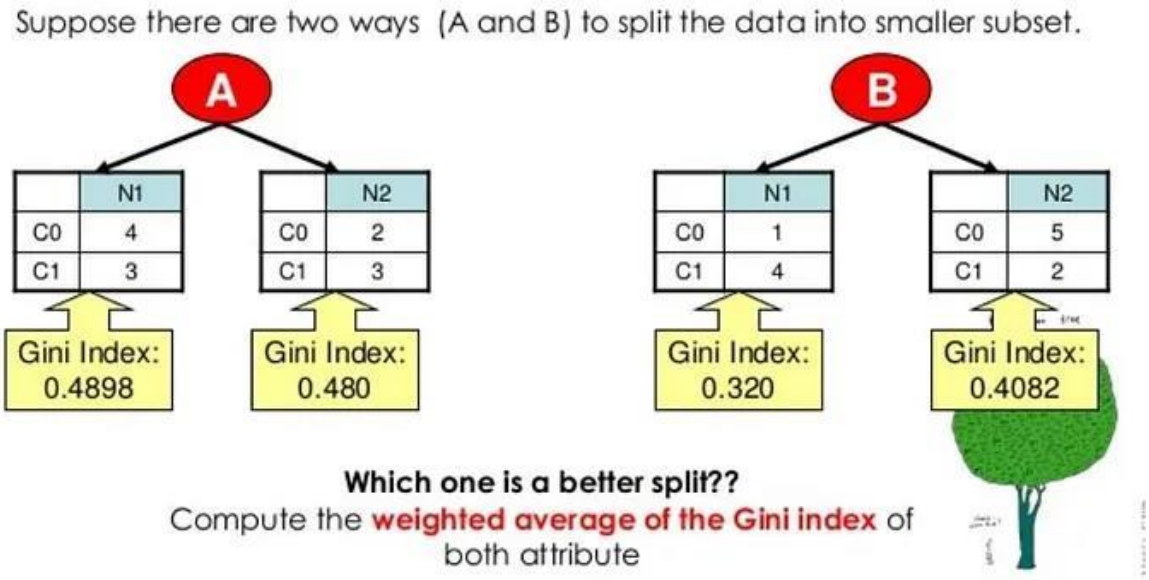
The calculations are similar to ID3, except the formula changes.
for example: compute Gini index for dataset

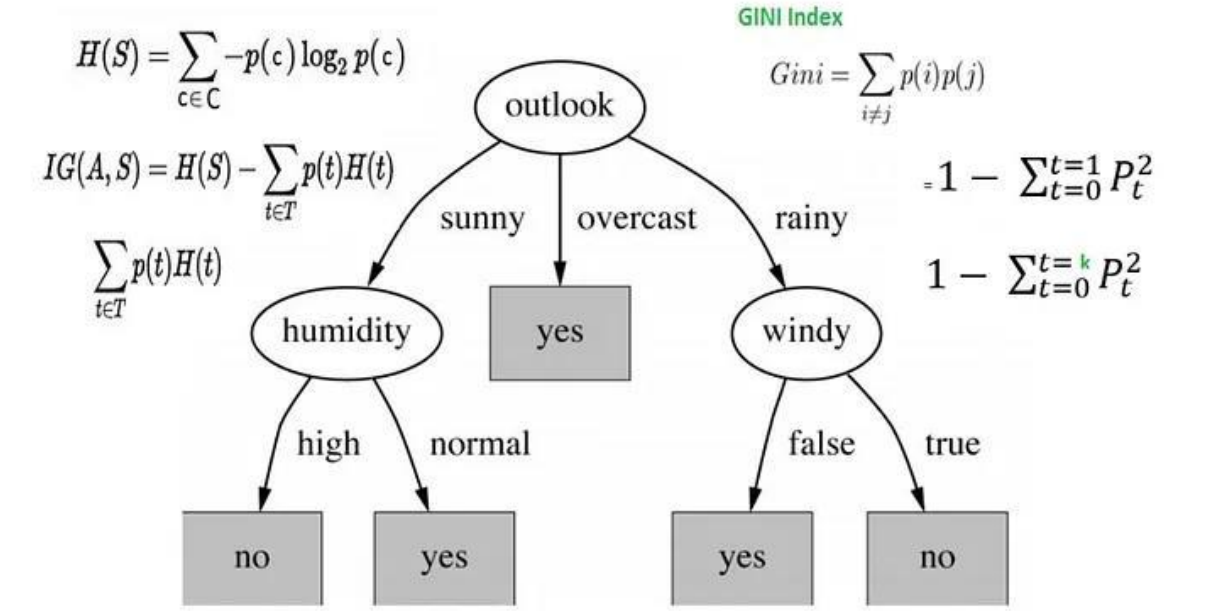
$$= 1 - \sum_{t=0}^{t=1} P_t^2$$

Out of 14 instances ,
yes=9,no=5
1 - (9/14)^2- (5/14)^2

1-0.413-0.127= 0.46
Gini = 0.46

Similarly, we can follow other steps to build the tree





2. Impurity Measures

The Gini index, Entropy, Misclassification error, and Information Gain are used in machine learning algorithms such as decision trees to measure the impurity of a node and to find the best attribute for splitting the dataset. Here's how to calculate each of them:

Gini index: The Gini index measures the impurity of a dataset or a node. A smaller Gini index indicates a better split.
Formula: Gini(t) = 1 — ∑(p_i)²
where t is the node and p_i is the probability of class i at that node.
To find the best attribute to split on, calculate the weighted Gini index for each attribute and choose the one with the lowest weighted Gini index.

Entropy: Entropy measures the randomness or disorder in a dataset. A smaller entropy value indicates a better split.
Formula: Entropy(t) = — ∑(p_i * log2(p_i))
where t is the node and p_i is the probability of class i at that node.
To find the best attribute to split on, calculate the weighted entropy for each attribute and choose the one with the lowest weighted entropy.

Misclassification error: Misclassification error measures the proportion of incorrect predictions in a node. A smaller misclassification error indicates a better split.

Formula: Misclassification error(t) = 1 — max(p_i)
where t is the node and p_i is the probability of class i at that node.
To find the best attribute to split on, calculate the weighted misclassification error for each attribute and choose the one with the lowest weighted misclassification error.

Information Gain: Information Gain measures the reduction in entropy or the increase in purity obtained by splitting a dataset on a particular attribute.
Formula: Information Gain = Entropy(parent) — Weighted Entropy(children)
where Entropy(parent) is the entropy of the parent node and Weighted Entropy(children) is the sum of the entropies of the child nodes, weighted by the proportion of instances in each child node.

To find the best attribute to split on, calculate the information gain for each attribute and choose the one with the highest information gain.

To use these measures in a decision tree algorithm, follow these steps:

1. Calculate the impurity (Gini index, entropy, or misclassification error) for the current node.
2. For each attribute, calculate the weighted impurity for the possible splits and choose the attribute with the lowest weighted impurity (or the highest information gain).
3. Split the dataset on the chosen attribute.
4. Recursively apply steps 1–3 to each child node until a stopping criterion is met (e.g., maximum depth, minimum number of samples in a leaf node, or no further reduction in impurity).

3. Properties of Decision Tree Classifier

1. Splitting rule

Node splitting, or simply splitting, divides a node into multiple sub-nodes to create relatively pure nodes. This is done by finding the best split for a node and can be done in multiple ways. The ways of splitting a node can be broadly divided into two categories based on the type of target variable:

1. Continuous Target Variable: Reduction in Variance
2. Categorical Target Variable: Gini Impurity, Information Gain, and Chi-Square

2. Criterion for splitting

Splitting criteria and Algorithm selection is based on the type of target variable. The splitting criteria used by the regression tree and the classification tree are different. Like the regression tree, the goal of the classification tree is to divide the data into smaller, more homogeneous groups. Homogeneity means that most of the samples at each node are from one class. The original CART algorithm uses Gini impurity as the splitting criterion; The later ID3, C4.5, and C5.0 use entropy.

3. **Binary or non-binary**

Binary vs. non-binary conditions

Conditions with two possible outcomes (for example, true or false) are called **binary conditions**. Decision trees containing only binary conditions are called **binary decision trees**.

Non-binary conditions have more than two possible outcomes. Therefore, non-binary conditions have more discriminative power than binary conditions. Decisions containing one or more non-binary conditions are called **non-binary decision trees**.

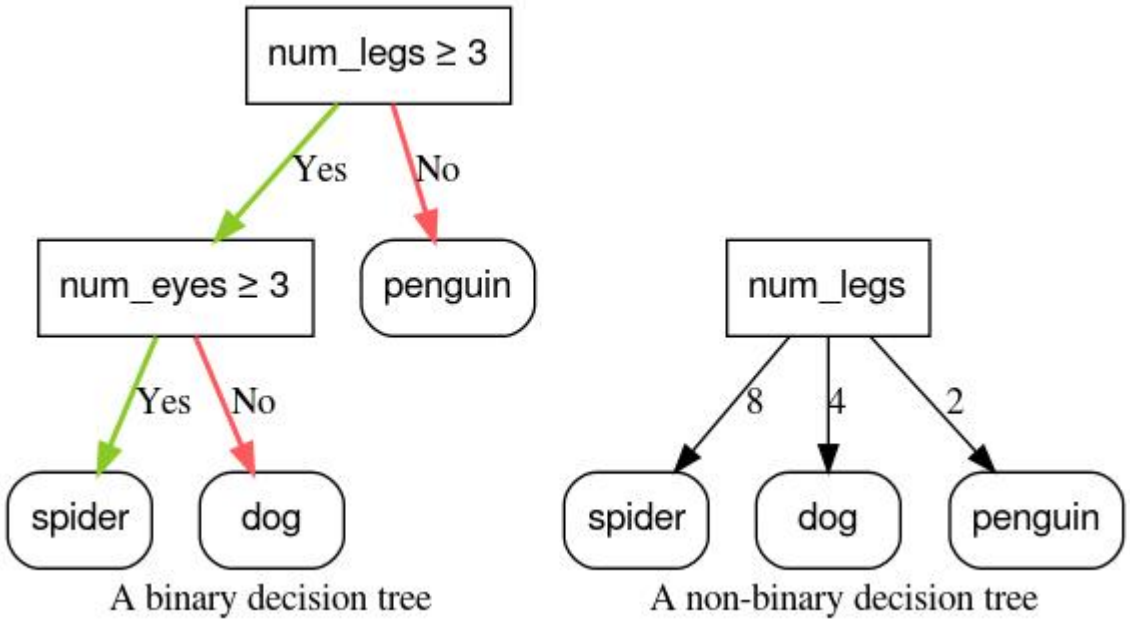


Figure 6: Binary versus non-binary decision trees.

Conditions with too much power are also more likely to overfit. For this reason, decision forests generally use binary decision trees, so this course will focus on them.

4. Termination Condition

Termination Conditions
For Classification problems: When a node comprises samples belonging to the same class or partitioning isn't feasible based on a minimum threshold.
For Regression problems: The process halts upon reaching the minimum number of observations.

5. Class labels
6. Classification
7. Transparency
8. Handling mixed datatypes
9. Eliminating irrelevant features
10. Pruning a decision tree

Real-time Applications of Decision trees

1. Decision Making

In daily life, we use decision trees without even realizing it. For instance, imagine you're trying to decide what to wear in the morning. You might ask yourself:

- "What's the weather like?" (root node)
- If it's hot, you might choose to wear a t-shirt (leaf node).
- If it's cold (decision node), you might ask yourself:
- "Will it rain?" (another decision node)
- If yes, you'd wear a jacket and take an umbrella (leaf node).
- If no, you might just wear a sweater (leaf node).

2. Health Care

In healthcare, decision trees can help medical professionals with diagnoses. For example, based on symptoms (decision nodes), a doctor can narrow down the possible conditions (leaf nodes). This can be particularly useful for initial screening or in rural areas where there is a lack of specialized healthcare professionals.

3. Financial Analysis

In the financial sector, decision trees are used in options pricing and strategy development. They can model possible future price movements based on different market conditions to help investors make informed decisions.

4. Customer Relationship Management (CRM)

Companies use decision trees to predict customer behavior, such as whether a customer will churn or respond positively to a marketing campaign. Based on different characteristics (e.g., age, purchase history, browsing behavior), a company can categorize customers and tailor their marketing strategies accordingly.

5. Quality Control

In manufacturing and quality control, decision trees can be used to predict whether a product will fail a quality assurance test based on different measurements and conditions during the manufacturing process.

6. Fraud Detection

Decision trees can help detect fraud by identifying patterns in transactions. Based on parameters like transaction frequency, amount, and location, a decision tree model can flag suspicious activities for further investigation.

7. Recommendation Systems

Many online platforms use decision trees as part of their recommendation algorithms. For example, Netflix or Spotify may use decision trees to determine what movies or songs to

recommend based on a user's past viewing or listening habits, demographic information, and preferences.

Key Differences and Use Cases

The primary distinction between these tree types is their output. Classification trees predict class labels, whereas regression trees forecast numerical values. Here's a detailed comparison:

Feature	Classification Trees	Regression Trees
Data Type	Categorical	Continuous
Output	Class Labels	Numerical Values
Example Use Case	**Fraud Detection**	Price Prediction
Splitting Criteria	Gini Impurity/Entropy	Mean Squared Error
Leaf Node	Most Common Class	Average of Values

Advantage of Decision Tree

- Easy to use and understand.
- Compared to other algorithms decision trees requires less effort for data preparation during pre-processing.
- Can handle both categorical and numerical data.
- Resistant to outliers, hence require little data pre-processing. Missing values in the data also do NOT affect the process of building a decision tree to any considerable extent.

Disadvantage of Decision Tree

- Prone to over fitting.
- For a Decision tree sometimes, calculation can go far more complex compared to other algorithms.
- Require some kind of measurement as to how well they are doing.
- Need to be careful with parameter tuning. A small change in the data can cause a large change in the structure of the decision tree causing instability.
- Decision tree often involves higher time to train the model.
- The Decision Tree algorithm is inadequate for applying regression and predicting continuous values

4. Regression Based on Decision Trees

What is a Regression Decision Tree?

A Regression Decision Tree is a type of decision tree that predicts continuous values. Unlike classification trees, which predict categories, regression trees predict a numerical value for a given set of features.

Key Concepts:

- **Nodes and Leaves**: Each internal node represents a decision based on a feature, and each leaf node represents a predicted continuous value.
- **Splitting**: The process of dividing a node into two or more sub-nodes based on certain conditions.
- **Loss Function**: Measures the difference between the actual and predicted values. In regression trees, the most common loss function is the mean squared error (MSE).

Building Regression Trees from Scratch

Let's break down the process step-by-step:

Step 1: Start with the Entire Dataset

Begin with the entire dataset and treat it as the root of the tree.

Step 2: Select the Best Split

For each node, consider splitting the data on every feature. The goal is to find the split that minimizes the loss function (e.g., MSE).

How to Find the Best Split:

1. **Calculate MSE for Each Split**: For every possible split, calculate the mean squared error (MSE) for the resulting sub-nodes.

- For a node t, the MSE is given by:

$$\mathrm{MSE}(t) = \frac{1}{N_t} \sum_{i \in t} (y_i - \bar{y}_t)^2$$

where N_t is the number of observations in node t, y_i is the actual value, and $\bar{y}_t$ is the mean value of node t.

2. Choose the Split with the Lowest MSE: Select the feature and value that result in the lowest MSE for the children's nodes.

Step 3: Split the Node

Divide the node into two or more sub-nodes based on the selected feature and value.

Step 4: Repeat the Process

Recursively apply steps 2 and 3 to each sub-node until a stopping criterion is met (e.g., maximum depth of the tree, minimum number of samples per leaf).

Step 5: Assign Values to Leaves

Once the stopping criteria are met, assign each leaf node a value, typically the mean value of the observations in that node.

Example:

How Decision Trees Work for Regression

Regression trees aim to **split the dataset** into smaller subsets, making the predictions in each subset as **homogeneous (as close to other values)** as possible in terms of the target variable. Here's how it works:

1. Choosing the Best Split:

- The algorithm evaluates all possible splits for each feature and selects the one that minimizes the variance in the target variable within the subsets. To choose the best split, the concept of **variance reduction** is used.
- This splitting continues recursively until the stopping criteria are met, such as reaching a maximum depth or a minimum number of samples per leaf.

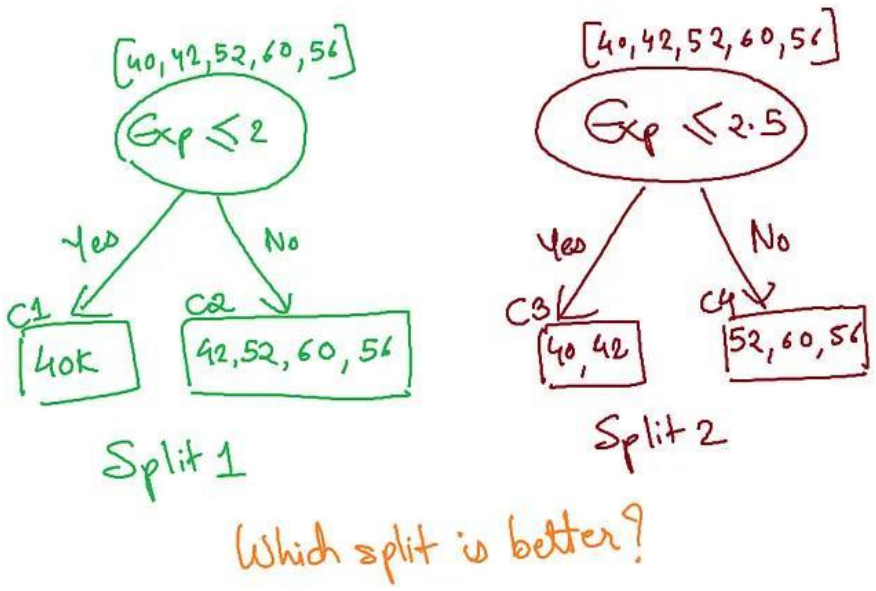
2. Prediction:

- Once the tree has grown to its maximum size (or an optimal size determined by hyperparameters), the prediction for a new data point is made by traversing the tree from the root to a leaf.
- The prediction is the **average** value of the target variable in the leaf node that the data point lands in.

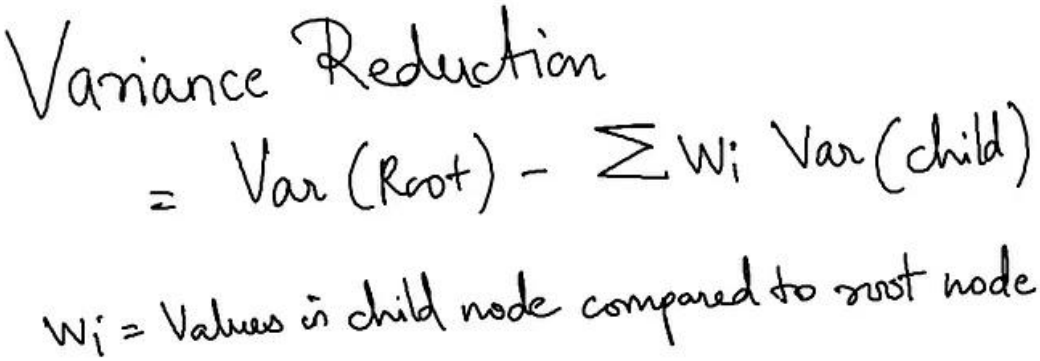
DATASET

Exp	Gap	Salary₹
2	Yes	40K
2.5	Yes	42K
3	No	52K
4	No	60K
4.5	Yes	56K

Avg (salary) = 50K



Which split is better — Split1 or Split2? **To answer this question, we will use the concept of variance reduction**



Variance reduction formula

$$\text{Variance of root node} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$
$$= \frac{1}{5}\left((40-50)^2 + (42-50)^2 + (52-50)^2 + (60-50)^2 + (56-50)^2\right)$$
$$= 60.8$$

$$\text{Variance of } c_1 = \frac{1}{1}\left((40-50)^2\right) = 100$$
$$\text{Variance of } c_2 = \frac{1}{4}\left((42-50)^2 + (52-50)^2 + (60-50)^2 + (56-50)^2\right) = 51$$
$$\text{Variance of } c_3 = \frac{1}{2}\left((40-50)^2 + (42-50)^2\right) = 82$$
$$\text{Variance of } c_4 = \frac{1}{3}\left((52-50)^2 + (60-50)^2 + (56-50)^2\right) = 46.67$$

Calculations for variance reduction

$$\text{Variance Reduction for split 1} = 60.8 - \left[\frac{1}{5}\times 100 + \frac{4}{5}\times 51\right]$$
$$= 0$$

$$\text{Variance Reduction for split 2} = 60.8 - \left[\frac{2}{5}\times 82 + \frac{3}{5}\times 46.67\right]$$
$$= 0.304$$

Variance reduction calculations for split1 and split2

The split whose variance reduction is more will be chosen.
Since here variance reduction of split2 is more than that of split1, we will go with split2.
Now, let's say we want to predict the income of the person whose exp is 3, then the answer will be here (considering split2 as the final tree) avg of values present in the leaf node which will be (52+60+56)/3 = 56K.

5. Bias-Variance Trad-off

Bias (Underfitting)
Bias refers to the error due to overly simplistic assumptions in the learning algorithm. A model with high bias tends to underfit the data, meaning it oversimplifies the underlying patterns. This leads to poor predictive performance as the model cannot capture the complexity of the real-world problem.
Example of High Bias:

Suppose you are training a linear regression model to predict housing prices. If the model assumes that the relationship between the features (e.g., square footage, number of bedrooms) and the price is strictly linear, it may perform poorly when the true relationship is more complex.
In machine learning, bias refers to a systematic error or tendency within a model that leads to inaccurate or unfair predictions due to flawed assumptions or data representation.

Bias is typically measured by evaluating the performance of a model on a training dataset.
One common way to calculate bias is to use performance metrics such as mean squared error (MSE) or mean absolute error (MAE), which determine the difference between the predicted and real values of the training data.
Bias is a systematic error that occurs due to incorrect assumptions in the machine learning process, leading to the misrepresentation of data distribution.

High-bias model features

1. **Underfitting**. High-bias models often underfit the data, meaning they oversimplify the solution based on generalization. As a result, the proposed distribution does not correspond to the actual distribution.
2. **Low training accuracy**. The lack of proper processing of training data results in high training loss and low training accuracy.
3. **Oversimplification.** The oversimplified nature of high-bias models limits their ability to identify complex features in the training data, making them inefficient for solving complicated problems.

How to reduce high bias?

There are several ways to overcome high bias:

1. Incorporating additional features from data to improve the model's accuracy.
2. Increasing the number of training iterations to allow the model to learn more complex data.
3. Avoiding high-bias algorithms such as linear regression, logistic regression, discriminant analysis, etc. and instead using nonlinear algorithms such as k-nearest neighbors, SVM, decision trees, etc.
4. Decreasing regularization at various levels to help the model learn the training set more effectively and prevent underfitting.

Variance (Overfitting)

Variance, on the other hand, is the error due to excessive complexity in the learning algorithm. A model with high variance captures not only the underlying patterns but also the noise in the training data. This leads to poor generalization to unseen data.
Example of High Variance:
Imagine training a decision tree with a very deep structure on a dataset of handwritten digits. While the tree can fit the training data perfectly, it might perform poorly on new, unseen digits because it has essentially memorized the training examples, including their individual quirks.

Variance stands in contrast to bias; it measures how much a distribution on several sets of data values differs from each other. The most common approach to measuring variance is by performing cross-validation experiments and looking at how the model performs on different random splits of your training data.

High-variance model features

- **Low testing accuracy.** Despite high accuracy on training data, high variance models tend to perform poorly on test data.
- **Overfitting.** A high-variance model often leads to overfitting as it becomes too complex.
- **Overcomplexity.** As researchers, we expect that increasing the complexity of a model will result in improved performance on both training and testing data sets. However, when a model becomes too complex and a simpler model may provide the same level of accuracy, it's better to choose the simpler one.

How to reduce high variance?

The following methods can be used to overcome high variance:

- Reducing the number of features in the model.
- Replacing the current model with a simpler one.
- Increasing the training data diversity to balance out the complexity of the model and the data structure.
- Avoiding high-variance algorithms (support vector machines, decision trees, k-nearest neighbors, etc.) and opt for low-variance ones such as linear regression, logistic regression, and linear discriminant analysis.
- Performing hyperparameter tuning to avoid overfitting.
- Increasing regularization on inputs to decrease the complexity of the model and prevent overfitting.
- Using a new model architecture. (Like with the high bias, this should be considered a last resort if other methods are not effective.)

What bias-variance scenarios are possible?

Now let's take a look at the different combinations of bias and variance in machine learning models and the results they provide.

1. **Low bias, low variance: ideal model**

A machine learning model with low bias and low variance is considered ideal but is not often the case in the machine learning practice, so we can speak of "reasonable bias" and "reasonable variance."

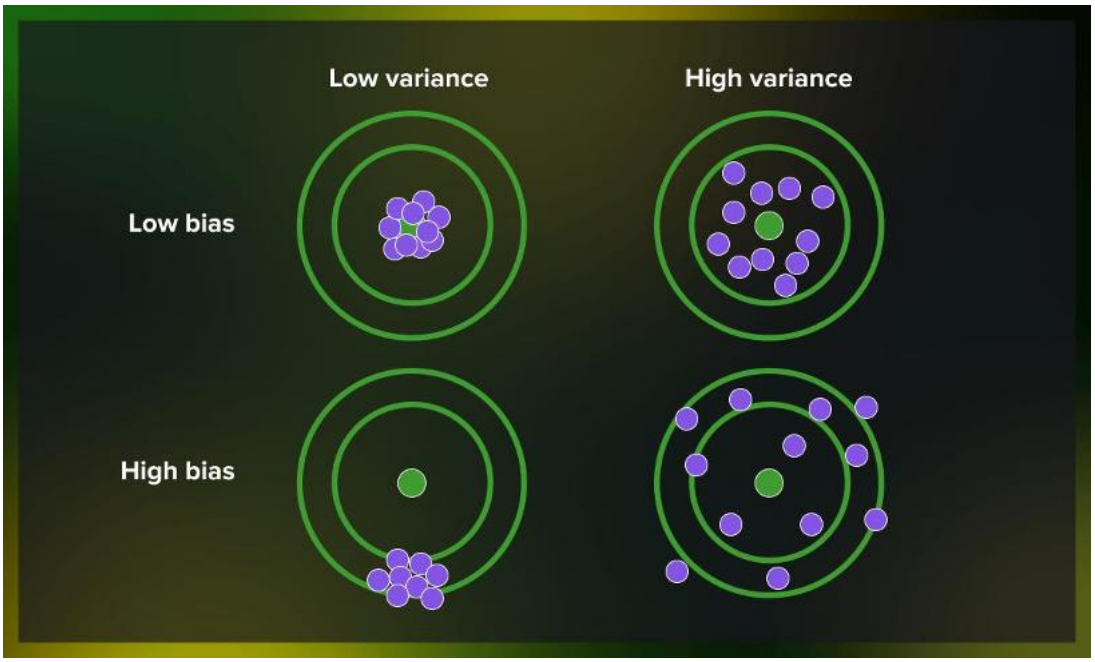
2. **Low bias, high variance: results in overfitting**

This combination results in inconsistent predictions that are accurate on average. It occurs when a model has too many parameters and fits too closely to the training data.

3. **High bias, low variance: results in underfitting**

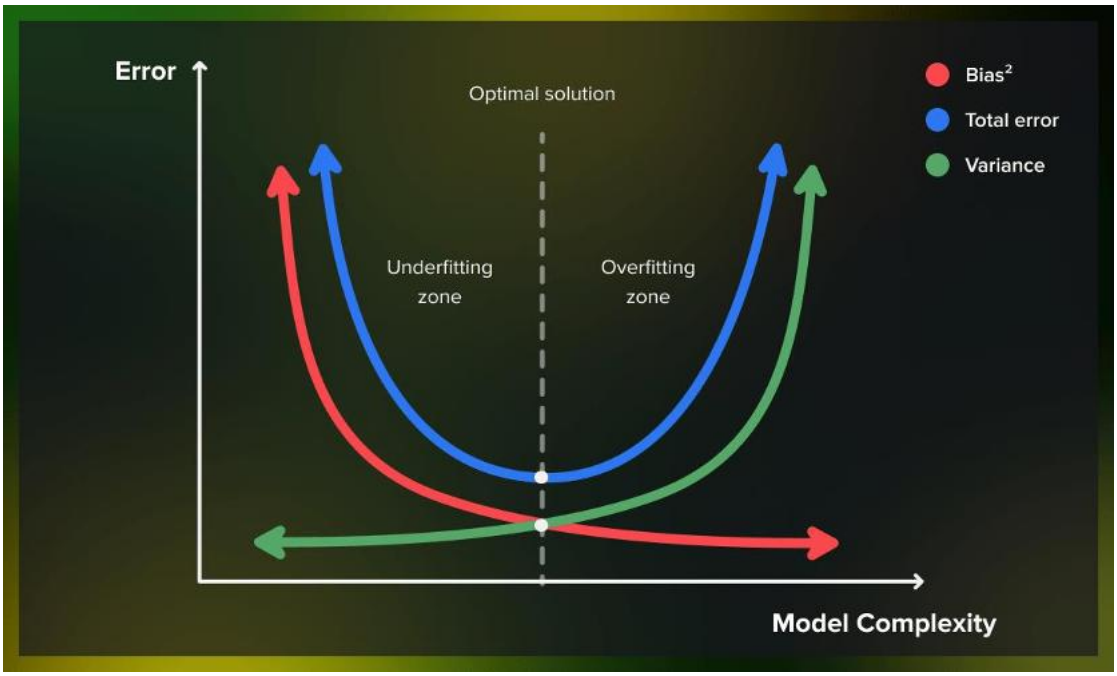
Predictions are consistent but inaccurate on average in this scenario. This happens when the model doesn't learn well from the training data or has too few parameters, leading to underfitting issues.

4. **High bias, high variance: results in inaccurate predictions**



How to achieve a bias-variance tradeoff?

As we have learned, bias and variance are interdependent. In other words, lowering a model's bias leads to an increase in its variance and vice versa. This relationship between bias and variance is known as the bias-variance tradeoff.



The balance between bias and variance can be adjusted in specific algorithms by modifying parameters, as seen in the following examples:

- For k-nearest neighbors, a low bias and high variance can be corrected by increasing the value of k, which increases the bias and decreases the variance.
- For support vector machines, a low bias and high variance can be altered by adjusting the C parameter, which increases the bias but decreases the variance.

Unfortunately, it's impossible to determine the actual bias and variance error terms while we are trying to predict the target function. However, bias and variance serve as useful frameworks to understand the performance of machine learning algorithms in making predictions.

What is the difference between bias-variance decomposition and bias-variance tradeoff?

Bias-variance decomposition and bias-variance tradeoff are closely related concepts.

Bias-variance decomposition is a mathematical technique that divides the generalization error in a predictive model into two components: bias and variance.

In machine learning, as you try to minimize one component of the error (e.g., bias), the other component (e.g., variance) tends to increase, and vice versa. Finding the right balance of bias and variance is key to creating an effective and accurate model. This is called the bias-variance tradeoff.

Identifying Underfitting and Overfitting

Now that we understand bias and variance, let's discuss how to recognize whether a model is underfitting or overfitting.

Underfitting

Underfitting occurs when a model is too simplistic to capture the underlying patterns in the data. You can identify underfitting by analyzing the model's performance on both the training and test datasets:

- Training Set: A model with high bias will struggle to fit the training data, resulting in low training accuracy (e.g., 60%).
- Test Set: The same model will also perform poorly on the test set, leading to low test accuracy (e.g., 65%).

In summary, underfitting results in a model that fails to grasp even the fundamental relationships in the data.

Overfitting

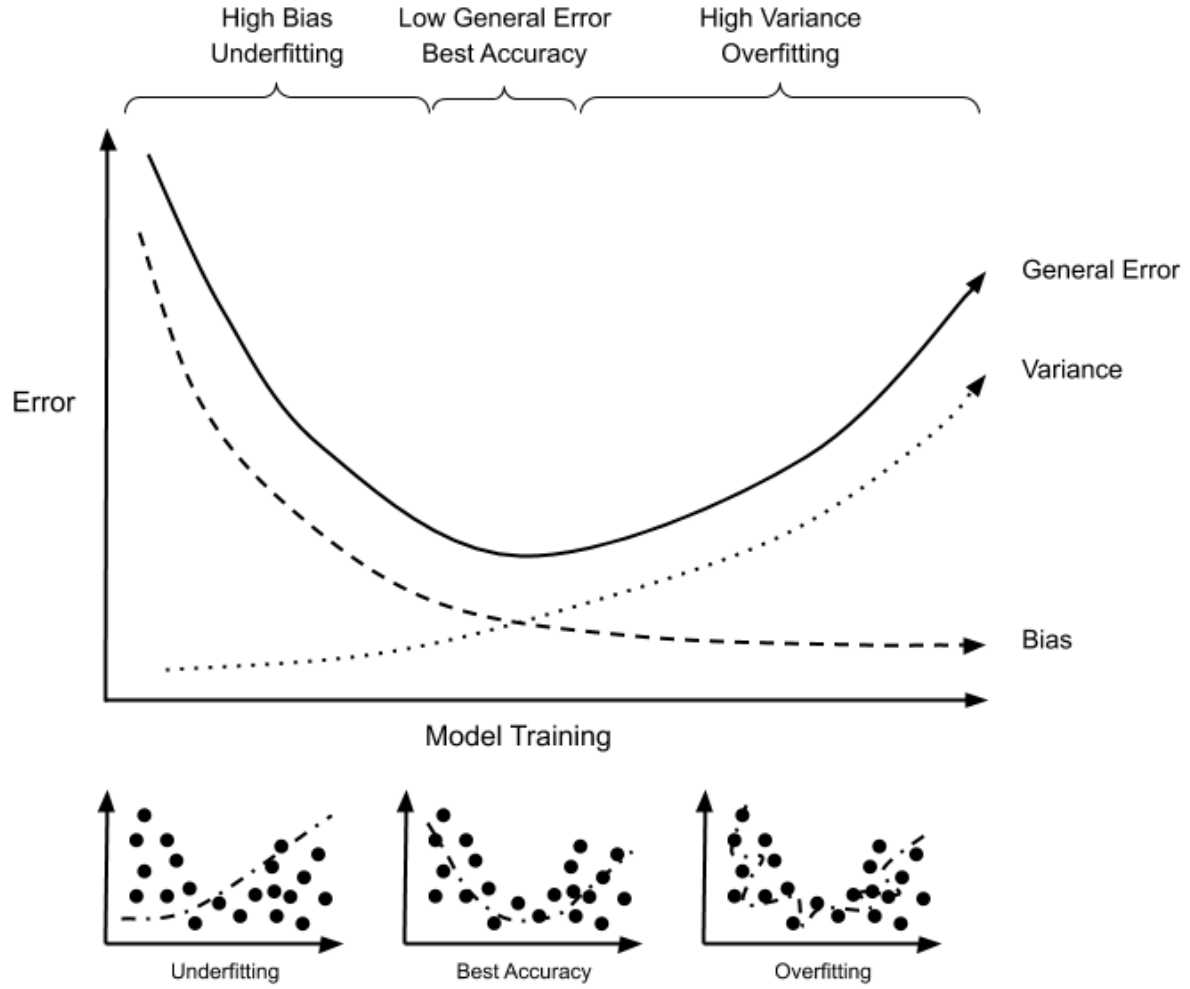
Overfitting happens when a model is excessively complex, capturing not just the patterns but also the noise in the training data. Identifying overfitting is a bit more nuanced:

- Training Set: A model with high variance can fit the training data exceedingly well, achieving high training accuracy (e.g., 95%).
- Test Set: However, this model will perform significantly worse on the test set, exhibiting lower test accuracy (e.g., 75%).

In essence, an overfit model "memorizes" the training data but struggles to generalize to new, unseen data.

The Tradeoff: Balancing Bias and Variance

The bias-variance tradeoff is the delicate equilibrium between underfitting and overfitting. The goal is to find the optimal level of complexity that allows a model to generalize effectively to unseen data. This tradeoff is often visualized as a curve, known as the validation error curve:



Validation Error Curve:

The validation error curve is a graphical representation that illustrates the relationship between model complexity (or flexibility) and the error on a validation dataset. It typically looks like an inverted U-shape or a curve with two distinct components:

1. **Bias (Underfitting) Region:** On the left side of the curve, you have the region associated with high bias or underfitting. In this area, the model's complexity is too low to capture the underlying patterns in the data. As a result, both the training and validation errors are high.
2. **Variance (Overfitting) Region:** On the right side of the curve, you enter the region associated with high variance or overfitting. Here, the model's complexity is excessively high, and it starts fitting not only the underlying patterns but also the noise in the training data. In this region, the training error is very low, but the

validation error starts to increase significantly because the model fails to generalize to unseen data.

3. **Optimal Region (Tradeoff):** The point of optimal model complexity lies between the bias and variance regions, often referred to as the "optimal region." This is where the model generalizes well to both the training and validation data, resulting in the lowest validation error.

Solutions to Address Bias and Variance

To strike the right balance and address bias and variance issues, consider the following solutions:

1. **Regularization:** Regularization techniques like L1 (Lasso) and L2 (Ridge) can help mitigate overfitting. These methods add penalty terms to the model's cost function, discouraging it from becoming overly complex.
2. **Feature Engineering**: Thoughtful feature selection and engineering can reduce both bias and variance. By including relevant features and excluding noisy ones, you can improve model performance.
3. **Cross-Validation:** Utilize cross-validation to assess your model's performance on different subsets of the data. This helps you gauge how well your model generalizes across various data splits, providing valuable insights into bias and variance.
4. **Ensemble Methods:** Ensemble techniques such as Random Forests and Gradient Boosting combine multiple models to achieve better performance. They can effectively reduce overfitting while improving predictive accuracy.
5. **Collect More Data:** If your model suffers from high bias (underfitting), acquiring more data can help it capture more complex patterns. Additional data can be especially beneficial when dealing with deep neural networks.

Algorithm	**Bias**	**Variance**
Linear Regression	High Bias	Less Variance
Decision Tree	Low Bias	High Variance
Bagging	Low Bias	High Variance (Less than Decision Tree)
Random Forest	Low Bias	High Variance (Less than Decision Tree and Bagging)

What is Bias and Variance Tradeoff:

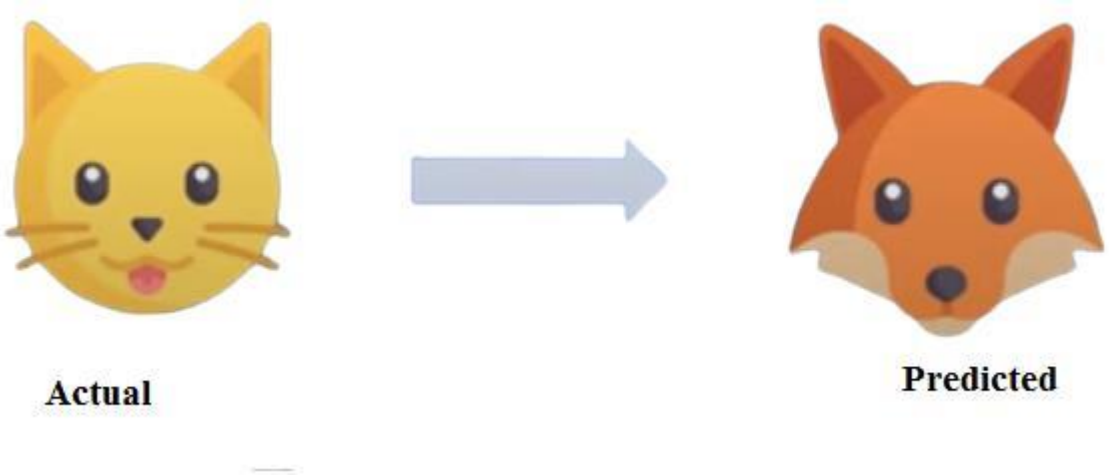
The bias-variance trade off in machine learning refers to the inherent tension between a model's ability to fit the training data well (low bias) and its ability to generalize to unseen data (low variance). The goal is to find the optimal model complexity that minimizes both errors.

- **Bias:**
  - Refers to the error caused by a model making strong, simplifying assumptions about the data, leading to an oversimplified representation.
  - High bias means the model is too simple and fails to capture the underlying patterns in the data, resulting in underfitting.
- **Variance:**
  - Refers to the model's sensitivity to fluctuations in the training data, leading to an overly complex model that captures noise instead of genuine patterns.
  - High variance means the model is too complex and fits the training data too closely, resulting in overfitting, where the model performs poorly on new, unseen data.
- **The Tradeoff:**
  - As model complexity increases, bias typically decreases (the model can fit the training data better), but variance increases (the model becomes more sensitive to training data variations).
  - Conversely, as model complexity decreases, variance typically decreases (the model becomes more stable), but bias increases (the model becomes too simple to capture the underlying patterns).
- **Finding the Balance:**
  - The ideal scenario is to find a model that strikes a balance between bias and variance, achieving good generalization performance on both training and unseen data.
  - This can be achieved through techniques like regularization, which penalizes model complexity, or by using ensemble methods, which combine multiple models to improve overall performance.
- **Examples:**
  - **Underfitting:** A linear regression model might have high bias and low variance, failing to capture complex relationships in the data.
  - **Overfitting:** A very complex model, like a decision tree with many branches, might have low bias and high variance, fitting the training data perfectly but performing poorly on new data.

https://www.shiksha.com/online-courses/articles/bias-and-variance/

Bias is the error that calculates the difference between the average prediction of our model and the actual value that we are trying to predict.

A model suffering from high bias is a simple model which pays very little attention to the training data. This type of model always leads to a high error on both **training and test data**. Let's take an example. Suppose we want our model to predict the animal by showing photos of animals. We trained the model on only one attribute **poiniting_ears.** Then we showed the image of a cat to the model. **So the model predicted it as a fox also has pointed ears.**



This shows the model is not able to capture other details while predicting as it has bias.
Characteristics of a high bias model include:

- Not able to capture proper data trends
- Trained over noise also. So giving less accurate results
- Suffers from underfitting
- A more general or simple model

What is variance?
Variance is the opposite of Bias. Variance is also an error that measures the randomness of the predicted value from the actual value.

Variance can be defined as the model's sensitivity to fluctuations in the data. if we model is allowed to view the data too many times, it will learn very well for only that data. It will capture most patterns in the data, but it will also learn from the unnecessary data present, or from the noise. When we train our model with too much data or allow it to view the data too many times, it will learn the data including noise, which will cause our model to consider trivial features as important. In this case, our model is overfitted. Now let's continue the

above example of **animal prediction**. If we consider **fur** as a feature then that will be noise as many animals have fur.

Note: *Noise* here means *irrelevant details* which are not required for the predicting output.

If you will train the model with some 100 images of cat and dog and again show the same images to it. It will predict correctly. But if you will some different cat and dog images the model will not be able to predict it correctly. This model performs well during the training phase but not during a test phase. And it might be looking at specific features like the nose and ears also. When the variance is high our model will capture all the features of the data given to it will tune itself to the data and predict it very well.

A model should have less variation in the predicted values with changes in the training data set. *Continuing the same cat example,* now this time we gave more features to the model for training

Variance errors are either the **low variance or high variance.**

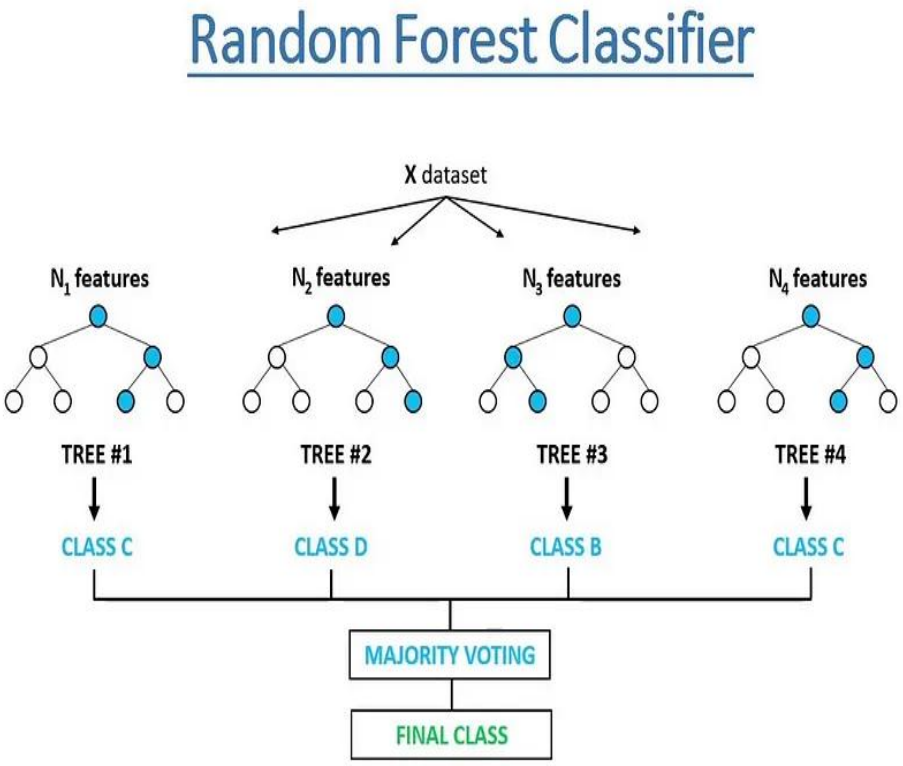
- ***Low variance: A model has a small variation*** in the predicted values with changes in the training data set.
- ***High variance: A model has a high variation*** in the predicted values with changes in the training data set. A model having high variance learns everything shown to it and performs well with the training dataset, but not on test data.

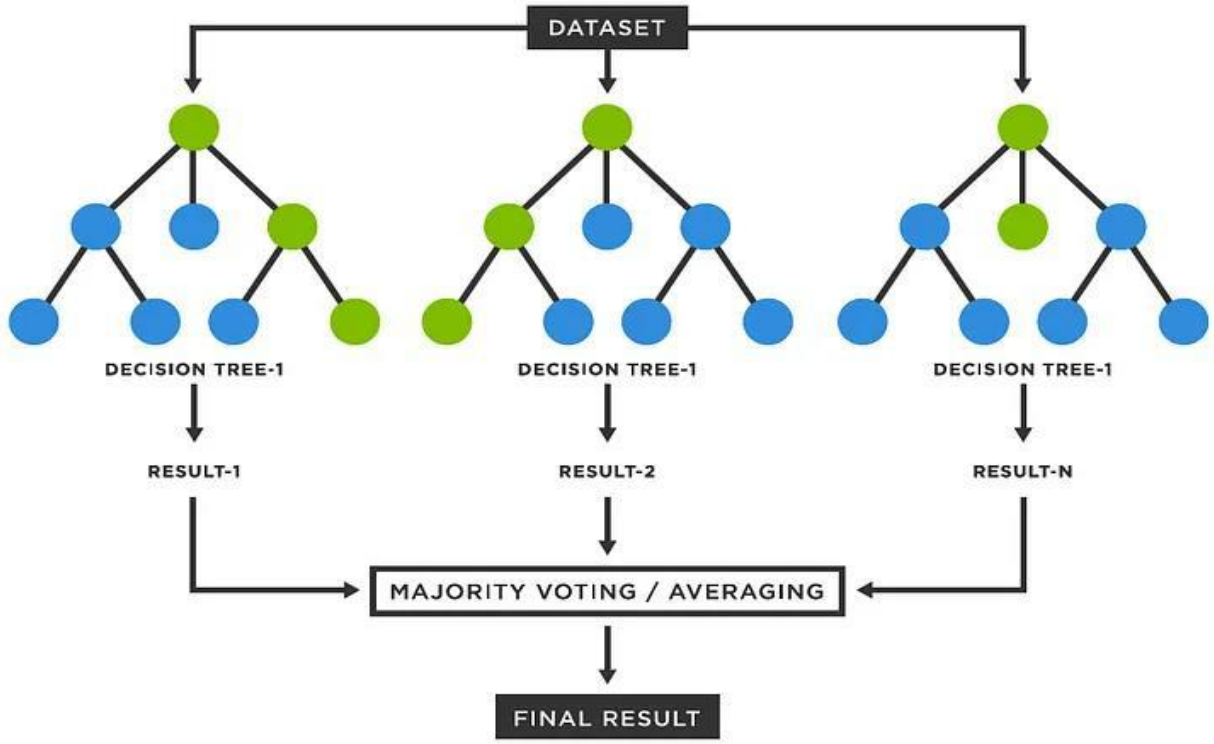
6. Random Forests for Classification and Regression.

A random forest is a supervised machine learning algorithm that is constructed from decision tree algorithms. This algorithm is applied in various industries such as banking and e-commerce to predict behaviour and outcomes.

What is a random forest?

A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems.





- A random forest algorithm consists of many decision trees. The 'forest' generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms.
- The (random forest) algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome.
- A random forest eradicates the limitations of a decision tree algorithm. It reduces the overfitting of datasets and increases precision. It generates predictions without requiring many configurations in packages

Features of a Random Forest Algorithm

- It's more accurate than the decision tree algorithm.
- It provides an effective way of handling missing data.
- It can produce a reasonable prediction without hyper-parameter tuning.
- It solves the issue of overfitting in decision trees.
- In every random forest tree, a subset of features is selected randomly at the node's splitting point.

In Random Forest, "voting" and "averaging" are two different methods used to combine the predictions made by the individual decision trees within the ensemble. The choice between

voting and averaging depends on whether you're working on a classification or regression problem.

1. Voting (Classification Problems): In classification tasks, Random Forest uses a voting mechanism to determine the final prediction. Each decision tree in the ensemble provides its own class prediction, and the class that receives the majority of votes among the trees is selected as the final predicted class.

- For example, if you have a Random Forest with 100 decision trees, and 70 of them predict Class A while 30 predict Class B for a particular data point, the ensemble's final prediction will be Class A, as it received the majority of votes.
- This voting mechanism helps improve the overall classification accuracy and makes the model less prone to making incorrect predictions due to individual tree variations or noise in the data.

2. Averaging (Regression Problems): In regression tasks, Random Forest uses an averaging method to determine the final prediction. Each decision tree in the ensemble provides its own numerical prediction, and the final prediction is obtained by averaging these numerical predictions.

- For instance, if you have a Random Forest with 100 decision trees, and each tree predicts a different numerical value for a specific data point, the ensemble's final prediction is the average of all these numerical predictions.
- Averaging helps smooth out the predictions and reduce the variance of the model, resulting in more stable and accurate regression predictions.

Key Challenges

- Time-consuming process: Since random forest algorithms can handle large data sets, they can be providing more accurate predictions, but can be slow to process data as they are computing data for each individual decision tree.
- Requires more resources: Since random forests process larger data sets, they'll require more resources to store that data.
- More complex: The prediction of a single decision tree is easier to interpret when compared to a forest of them.

Random forest applications

The random forest algorithm has been applied across a number of industries, allowing them to make better business decisions. Some use cases include:

- Finance: It is a preferred algorithm over others as it reduces time spent on data management and pre-processing tasks. It can be used to evaluate customers with high credit risk, to detect fraud, and option pricing problems.
- Healthcare: The random forest algorithm has applications within computational biology (link resides outside ibm.com), allowing doctors to tackle problems such as gene expression classification, biomarker discovery, and sequence annotation. As a result, doctors can make estimates around drug responses to specific medications.

- E-commerce: It can be used for recommendation engines for cross-sell purposes

Advantages of Random Forests

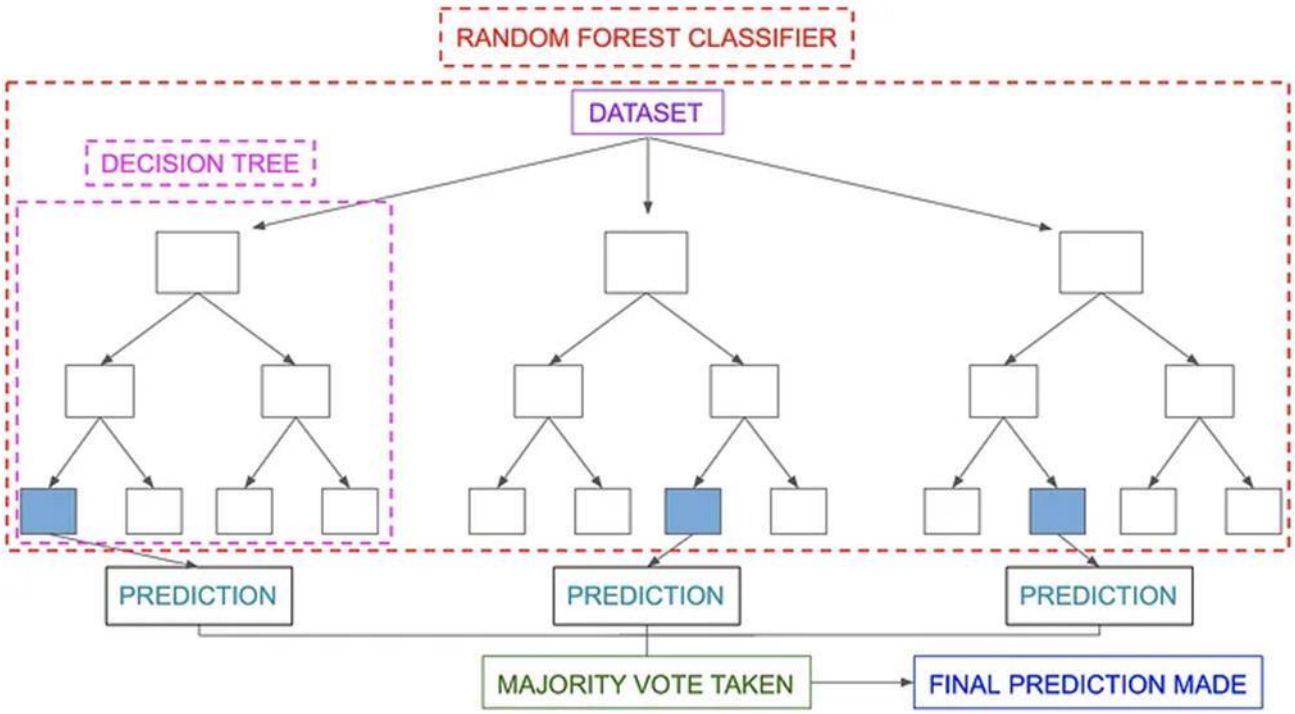
- **Robustness**: Random forests are resistant to overfitting, even with noisy datasets, because they aggregate many decision trees.
- **Versatility**: They can be used for both classification and regression problems.
- **Handles Missing Data**: Random forests can maintain good performance even when some data is missing.
- **Feature Importance**: Random forests can estimate the importance of each feature, making them valuable for feature selection.
- **Scalability**: Random forests are easily parallelizable, allowing them to scale to large datasets.

Disadvantages of Random Forests

- **Interpretability**: While decision trees are easy to interpret, random forests are harder to interpret since they involve multiple trees. It can be difficult to understand the model as a whole.
- **Computationally Intensive**: Building a large number of trees can be computationally expensive, especially for large datasets.
- **Memory Usage**: Random forests require significant memory, as each tree needs to be stored.
- **Slow Predictions**: Making predictions with random forests can be slower than with a single decision tree, as it requires querying each tree in the forest.

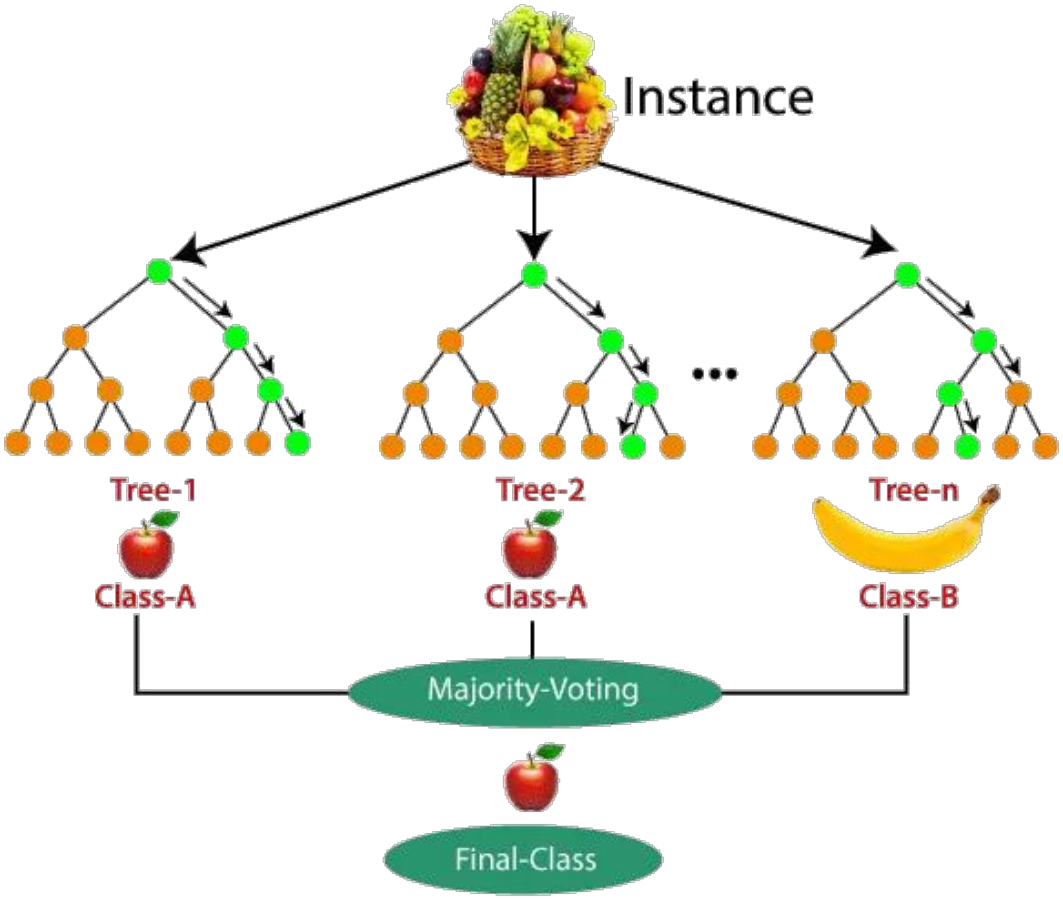
Random Forest Classification:

- Classification in random forests employs an ensemble methodology to attain the outcome.
- The training data is fed to train various decision trees. This dataset consists of observations and features that will be selected randomly during the splitting of nodes.
- A rain forest system relies on various decision trees.
- Every decision tree consists of decision nodes, leaf nodes, and a root node.
- The leaf node of each tree is the final output produced by that specific decision tree.
- The selection of the final output follows the majority-voting system.
- In this case, the output chosen by the majority of the decision trees becomes the final output of the rain forest system. The diagram below shows a simple random forest classifier.



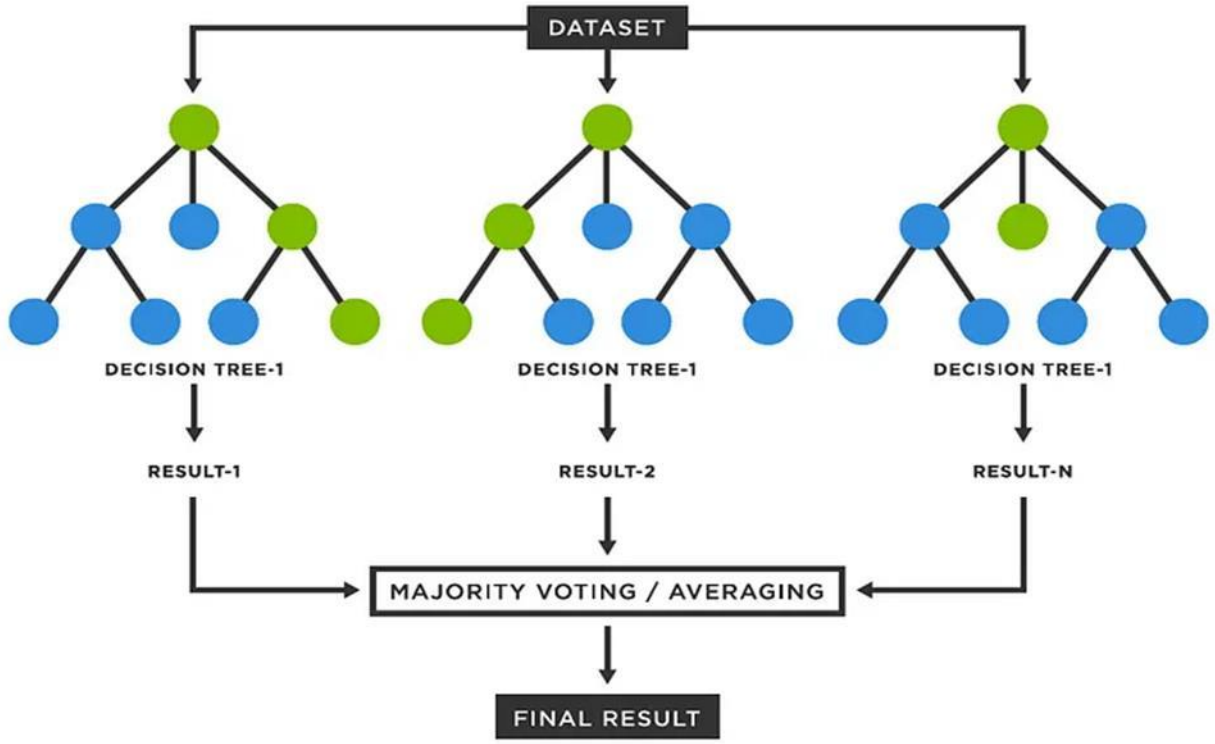
Example:

- Let's take an example of a training dataset consisting of various fruits such as bananas, apples, pineapples, and mangoes.
- The random forest classifier divides this dataset into subsets.
- These subsets are given to every decision tree in the random forest system. Each decision tree produces its specific output. For example, the prediction for trees 1 and 2 is *apple*.
- Another decision tree (n) has predicted *banana* as the outcome.
- The random forest classifier collects the majority voting to provide the final prediction.
- The majority of the decision trees have chosen *apple* as their prediction.
- This makes the classifier choose *apple* as the final prediction.



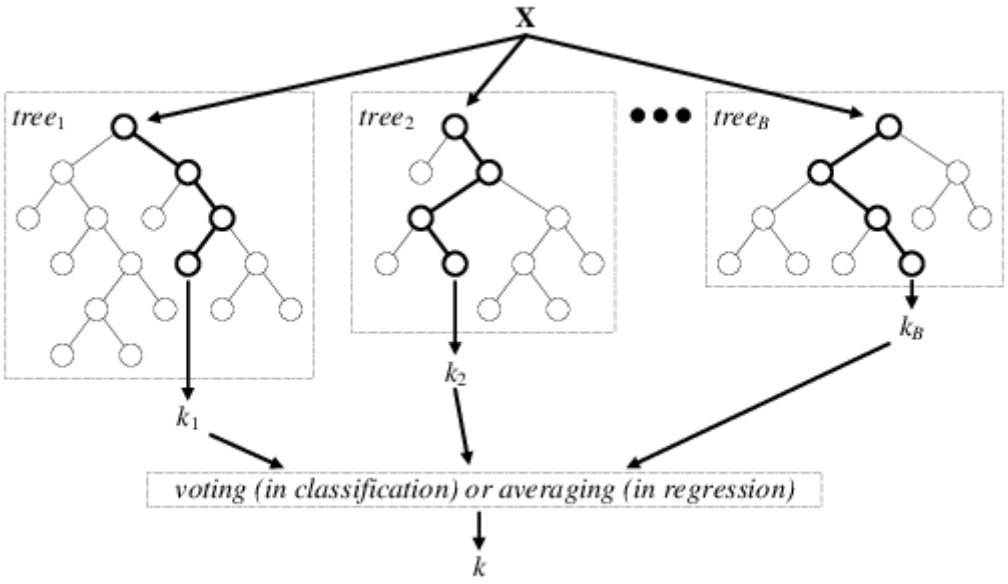
Random Forest Regression

- Regression is the other task performed by a random forest algorithm.
- A random forest regression follows the concept of simple regression.
- Values of dependent (features) and independent variables are passed in the random forest model.
- In a random forest regression, each tree produces a specific prediction.
- The mean prediction of the individual trees is the output of the regression.
- This is contrary to random forest classification, whose output is determined by the mode of the decision trees' class.
- Although random forest regression and linear regression follow the same concept, they differ in terms of functions.
- The function of linear regression is y=bx + c, where y is the dependent variable, x is the independent variable, b is the estimation parameter, and c is a constant.
- The function of a complex random forest regression is like a Blackbox.



7. Introduction to the Bayes Classifier

- Bayesian classifiers are the systems that are based on Bayes' decision theory.
- This theory is a fundamental statistical approach.
- The idea behind these classifiers is that if the class is known, the values of the other features can be predicted.
- If the class is not known, then Bayes' rule can be used to predict the class label according to the given feature values.
- In Bayesian classifiers, probabilistic models of the features are built to predict the class label of a new sample.
- Bayesian classifiers, which are one of the widely used methods for pattern recognition problems, are utilized in most of the recent studies.
- The types of Bayesian classifiers utilized for ECG classification are the Bayesian network, naïve Bayes, and Bayes maximum likelihood classifier.
- A classifier that is purely based on probability (posterior and prior) is called a Bayesian classifier.
- Bayesian classifier assists in finding the probability that a given instance (a record) belongs to a particular class.
- This statistical classifier exhibits high accuracy and speed in mining large databases. The working principle of the Bayesian classifier is based on the assumption (class condition independence) that the impact of the attribute value for a given class does not rely on the values of other attributes.



What are the applications of Random Forest Regressor?

Random forest regression is a powerful machine learning technique that can be used in a variety of applications. Some common applications of random forest regression include:

1. **Predicting continuous outcomes**: Random forest regression can be used to predict continuous outcomes such as the price of a stock, the temperature at a given time, or the weight of a fruit.
2. **Environmental modeling**: Random forest regression can be used to model the relationship between environmental variables such as temperature, precipitation, soil characteristics, and the growth of trees or other vegetation.
3. **Energy consumption prediction**: Random forest regression can be used to predict energy consumption in buildings, industrial plants or energy systems, which helps optimize energy consumption and reduce costs.
4. **Healthcare**: Random forest regression can be used to predict the progression of a disease, the likelihood of treatment success, or the expected lifespan of a patient.
5. **Financial forecasting**: Random forest regression can be used to predict stock prices, currency exchange rates, or other financial outcomes.
6. **Production Quality Control**: Random forest regression can be used to predict the quality of a production process, predict the amount of waste generated or predict the lifetime of a mechanical part.
7. **Time Series Forecasting**: Random Forest Regression also have been used for time-series forecasting, for example, forecasting product demand or financial time-series data.

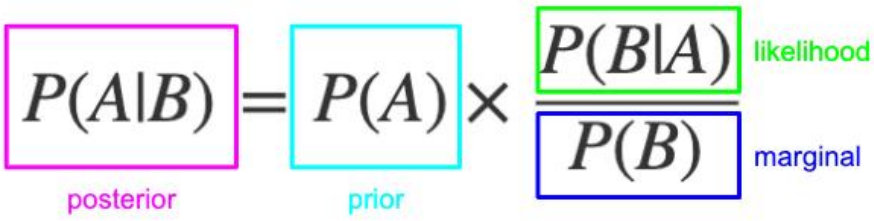
What are the limitations of Random Forest Regressor?

There are several limitations to ensemble methods:

1. Random forest does not work well for models with categorical variables as it might generate biased results.
2. Random forest has a high training time, especially for very large datasets.
3. Random forest is not ideal for models with a lot of sparse features.

8.Bayes' Rule and Inference

- **Bayes' theorem** describes the probability of occurrence of an event related to any condition. It is also considered for the case of conditional probability. Bayes theorem is also known as the formula for the probability of "causes".
- For example: if we have to calculate the probability of taking a blue ball from the second bag out of three different bags of balls, where each bag contains three different colour balls viz. red, blue, black.
- In this case, the probability of occurrence of an event is calculated depending on other conditions is known as conditional probability.
- Bayes' Rule is the most important rule in data science. It is the mathematical rule that describes how to update a belief, given some evidence. In other words – it describes the act of learning.



The equation: Posterior = Prior x (Likelihood over Marginal probability)
There are four parts:

- **Posterior probability** (updated probability after the evidence is considered)
- **Prior probability** (the probability before the evidence is considered)
- **Likelihood** (probability of the evidence, given the belief is true)
- **Marginal probability** (probability of the evidence, under any circumstance)

Bayes' Rule tells you how to calculate a conditional probability with information you already have.
It is helpful to think in terms of two events – a hypothesis (which can be true or false) and evidence (which can be present or absent).

$$P(Hypothesis \mid Evidence) = P(Hypothesis) \times \frac{P(Evidence \mid Hypothesis)}{P(Evidence)}$$

- Bayes' Rule lets you calculate the **posterior (or "updated") probability**. This is a conditional probability. It is the probability of the hypothesis being true, if the evidence is present.
- Think of the **prior (or "previous") probability** as your belief in the hypothesis before seeing the new evidence. If you had a strong belief in the hypothesis already, the prior probability will be large.
- The prior is multiplied by a fraction. Think of this as the "strength" of the evidence. The posterior probability is greater when the top part (numerator) is big, and the bottom part (denominator) is small.
- The numerator is the **likelihood**. This is another conditional probability. It is the probability of the evidence being present, given the hypothesis is true.
- Remember, the "probability of the evidence being present given the hypothesis is true" is not the same as the "probability of the hypothesis being true given the evidence is present".

- Now look at the denominator. This is the **marginal probability** of the evidence. That is, it is the probability of the evidence being present, whether the hypothesis is true or false. The smaller the denominator, the more "convincing" the evidence.

Worked example of Bayes' Rule

Here's a simple worked example.
Your neighbor is watching their favorite football (or soccer) team. You hear them cheering, and want to estimate the probability their team has scored.
Step 1 – write down the posterior probability of a goal, given cheering
Step 2 – estimate the prior probability of a goal as 2%
Step 3 – estimate the likelihood probability of cheering, given there's a goal as 90% (perhaps your neighbor won't celebrate if their team is losing badly)
Step 4 – estimate the marginal probability of cheering – this could be because:

- a goal has been scored (2% of the time, times 90% probability)
- or any other reason, such as the other team missing a penalty or having a player sent off (98% of the time, times perhaps 1% probability)

Now, piece everything together:

$$P(Goal \mid Cheer) = P(Goal) \times \frac{P(Cheer \mid Goal)}{P(Cheer \mid Goal) + P(Cheer \mid No\ goal)}$$

$$= 0.02 \times \frac{0.9}{(0.02 \times 0.9) + (0.98 \times 0.01)}$$

$$= 64.7\%$$

Use cases for Bayes' Rule

Bayes' Rule has use cases in many areas:

- Understanding probability problems (including those in medical research)
- Statistical modelling and inference
- Machine learning algorithms (such as Naive Bayes, Expectation Maximization)
- Quantitative modelling and forecasting

9.Naive Bayes Classifier

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

Why is it called Naïve Bayes?

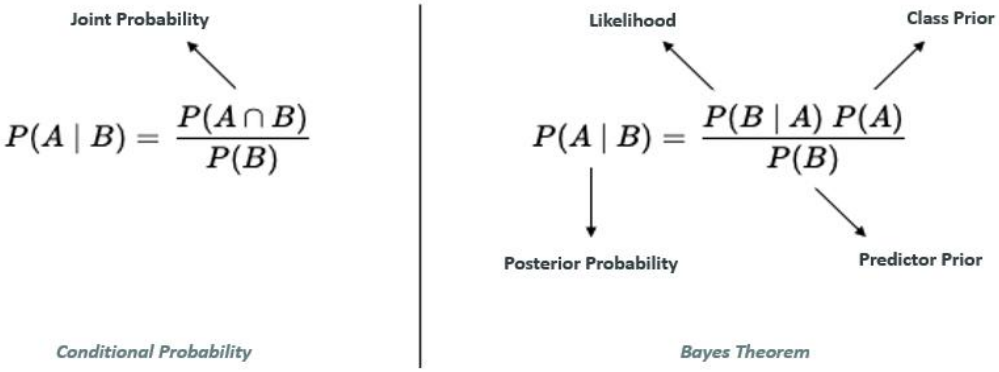
The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

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

Bayes' Theorem:

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

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$



- **P(A) or Class Prior** is the prior probability of the response variable
- **P(B) or Predictor Prior** is the evidence or the probability of training data
- **P(A|B) or Posterior Probability** is the conditional probability of the response variable being of a particular value given the input attributes
- **P(B|A) or Likelihood** is basically the reverse of the posterior probability or the likelihood of training data

Stepwise Bayes Theorem

Step 1 – Collect raw data

Day	Outlook	Humidity	Wind	Run
D1	Sunny	High	Weak	No
D2	Overcast	High	Strong	No
D3	Rainy	High	Weak	Yes
D4	Rainy	Normal	Weak	No
D5	Sunny	Normal	Weak	Yes
D6	Sunny	High	Weak	Yes
D7	Sunny	High	Weak	Yes
D8	Rainy	Normal	Strong	No
D9	Overcast	High	Weak	Yes
D10	Sunny	High	Weak	Yes
D11	Rainy	High	Weak	No
D12	Overcast	Normal	Strong	No
D13	Overcast	High	Weak	Yes
D14	Sunny	High	Weak	Yes

Next, you need to create a frequency table for each attribute of your dataset.

Step 2 – Convert data to a frequency table(s)

Frequency Table		Run	
		Yes	No
Outlook	Sunny	5	1
	Overcast	2	2
	Rainy	1	3

Frequency Table		Run	
		Yes	No
Humidity	High	7	3
	Normal	1	3

Frequency Table		Run	
		Yes	No
Wind	Strong	0	3
	Weak	9	2

Then, for each frequency table, you will create a likelihood table.

Step 3 – Calculate prior probability and evidence

Likelihood Table		Run		
		Yes	No	
Outlook	Sunny	5/8	1/6	6/14
	Overcast	2/8	2/6	4/14
	Rainy	1/8	3/6	4/14
		8/14	6/14	

Likelihood Table for Outlook

Likelihood Table		Run		
		Yes	No	
Humidity	High	7/8	3/6	10/14
	Normal	1/8	3/6	4/14
		8/14	6/14	

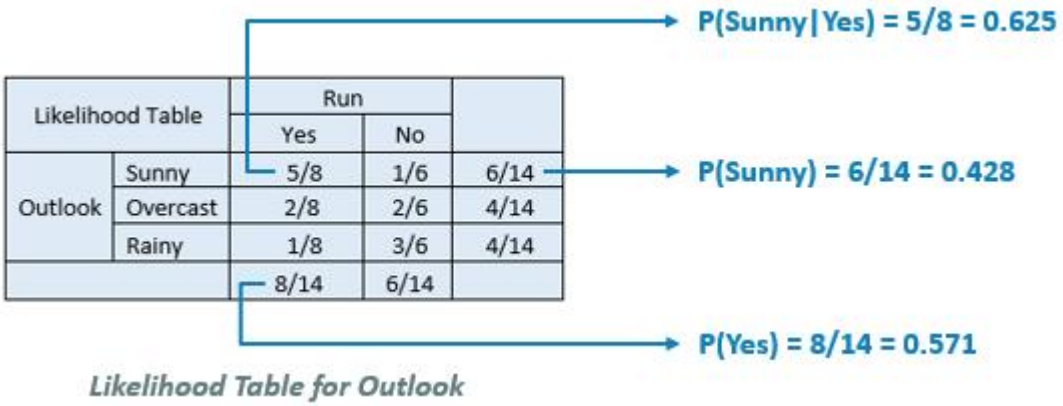
Likelihood Table for Humidity

Likelihood Table		Run		
		Yes	No	
Wind	Strong	0/9	3/5	3/14
	Weak	9/9	2/5	11/14
		9/14	5/14	

Likelihood Table for Wind

Step 4 – Apply probabilities to Bayes' Theorem equation

Let's say you want to focus on the likelihood that you go for a run given that it's sunny outside.



Likelihood Table		Run		
		Yes	No	
Outlook	Sunny	5/8	1/6	6/14
	Overcast	2/8	2/6	4/14
	Rainy	1/8	3/6	4/14
		8/14	6/14	

Likelihood Table for Outlook

P(Yes|Sunny) = P(Sunny|Yes) * P(Yes) / P(Sunny) = 0.625 * 0.571 / 0.428 = **0.834**

Applications of Naïve Bayes classifier

- **Text Classification:** Naïve Bayes algorithm is almost always used as a classifier and is an excellent choice for spam filtering of your emails or news categorization on your smartphone.
- **Recommendation Systems:** Naïve Bayes is used with collaborative filtering to build recommendation systems for you. The '*Because you watched* ____' section on Netflix is exactly that.
- **Sentiment Analysis:** Naïve Bayes is an effective algorithm for the identification of positive or negative sentiments of a target group (customers, audience, etc.). Think of feedback forms and IMDb reviews.
- **Spam filtering**
- Weather Prediction
- Medical Diagnosis
- Face Recognition

Advantages of Naïve Bayes

- Easy to work with when using binary or categorical input values.
- It is simple and easy to implement.
- It does not require as much training data.
- Handles both continuous and discrete data.
- Fast and reliable for making real-time predictions.

Limitations of Naïve Bayes

- Assumes that all the features are independent, which is highly unlikely in practical scenarios.
- Unsuitable for numerical data.
- The number of features must be equal to the number of attributes in the data for the algorithm to make correct predictions.
- Encounters 'Zero Frequency' problem: If a categorical variable has a category in the test dataset that wasn't included in the training dataset, the model will assign it a 0 probability and will be unable to make a prediction. This problem can be resolved using smoothing techniques which are out of the scope of this article.
- Computationally expensive when used to classify a large number of items.

Unit-4- Linear Discriminants for Machine Learning

Linear Discriminant Analysis (LDA) in Machine Learning

Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).

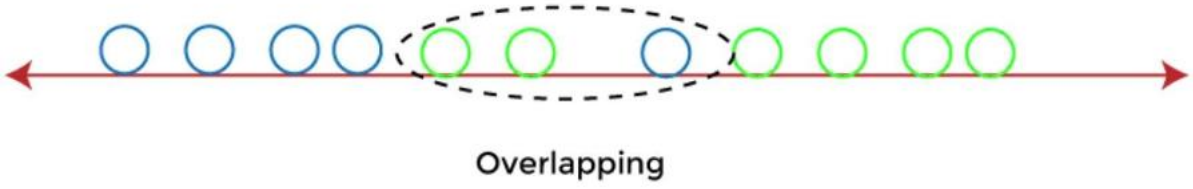
This can be used to project the features of higher dimensional space into lower-dimensional space in order to reduce resources and dimensional costs.

What is Linear Discriminant Analysis (LDA)?

Although the logistic regression algorithm is limited to only two-class, linear Discriminant analysis is applicable for more than two classes of classification problems.

Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning. It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification.

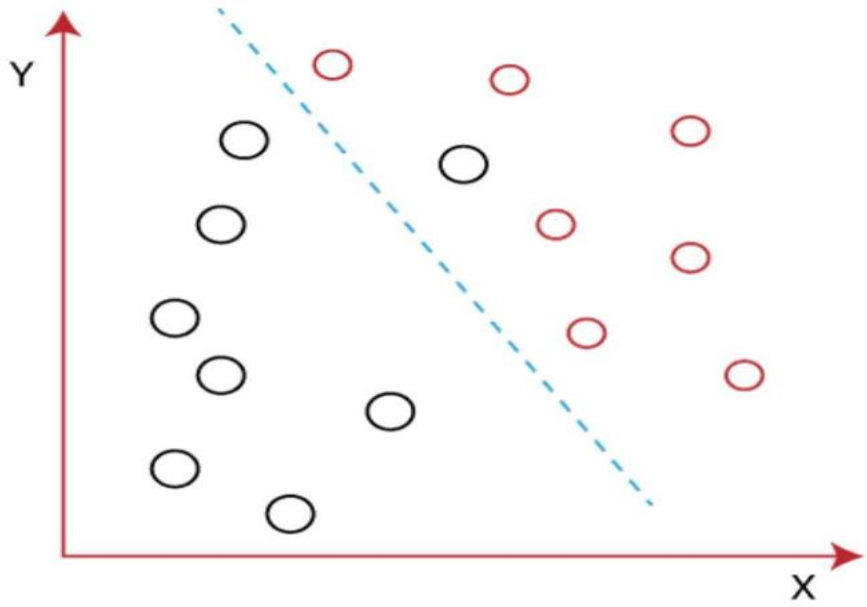
Whenever there is a requirement to separate two or more classes having multiple features efficiently, the Linear Discriminant Analysis model is considered the most common technique to solve such classification problems. For e.g., if we have two classes with multiple features and need to separate them efficiently. When we classify them using a single feature, then it may show overlapping.



To overcome the overlapping issue in the classification process, we must increase the number of features regularly.

Example:

Let's assume we have to classify two different classes having two sets of data points in a 2-dimensional plane as shown below image:



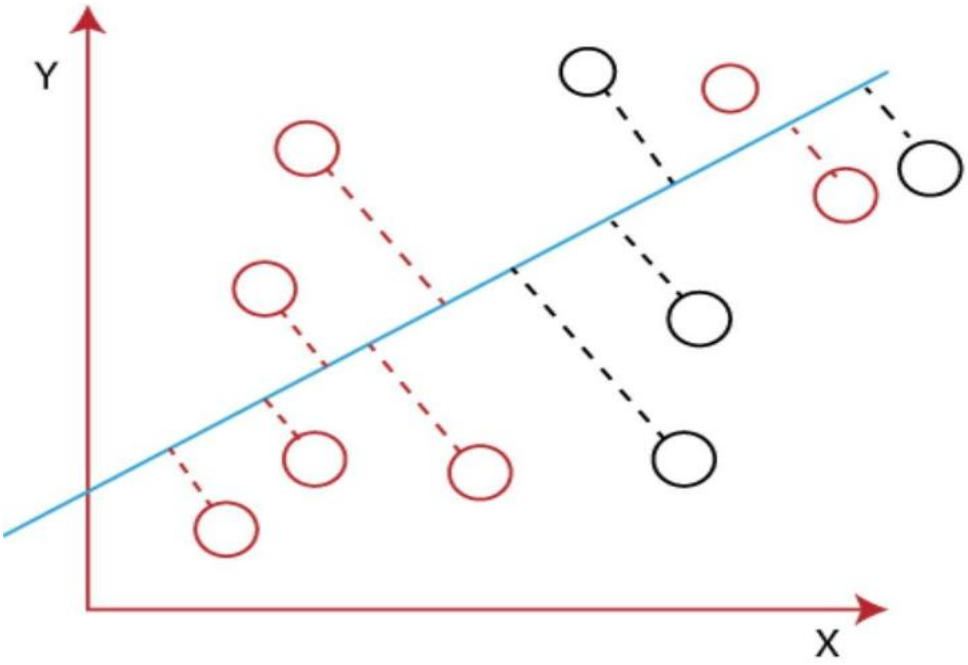
However, it is impossible to draw a straight line in a 2-d plane that can separate these data points efficiently but using linear Discriminant analysis; we can dimensionally reduce the 2-D plane into the 1-D plane. Using this technique, we can also maximize the separability between multiple classes.

How Linear Discriminant Analysis (LDA) works?

Linear Discriminant analysis is used as a dimensionality reduction technique in machine learning, using which we can easily transform a 2-D and 3-D graph into a 1-dimensional plane.

Let's consider an example where we have two classes in a 2-D plane having an X-Y axis, and we need to classify them efficiently. As we have already seen in the above example that LDA enables us to draw a straight line that can completely separate the two classes of the data points. Here, LDA uses an X-Y axis to create a new axis by separating them using a straight line and projecting data onto a new axis.

Hence, we can maximize the separation between these classes and reduce the 2-D plane into 1-D.



To create a new axis, Linear Discriminant Analysis uses the following criteria:

- It maximizes the distance between means of two classes.
- It minimizes the variance within the individual class.

Using the above two conditions, LDA generates a new axis in such a way that it can maximize the distance between the means of the two classes and minimizes the variation within each class.

In other words, we can say that the new axis will increase the separation between the data points of the two classes and plot them onto the new axis.

Why LDA?

- Logistic Regression is one of the most popular classification algorithms that perform well for binary classification but falls short in the case of multiple classification problems with well-separated classes. At the same time, LDA handles these quite efficiently.
- LDA can also be used in data pre-processing to reduce the number of features, just as PCA, which reduces the computing cost significantly.
- LDA is also used in face detection algorithms. In Fisher faces, LDA is used to extract useful data from different faces. Coupled with eigenfaces, it produces effective results.

Drawbacks of Linear Discriminant Analysis (LDA)

Although, LDA is specifically used to solve supervised classification problems for two or more classes which are not possible using logistic regression in machine learning. But LDA also fails in some cases where the Mean of the distributions is shared. In this case, LDA fails to create a new axis that makes both the classes linearly separable.

To overcome such problems, we use **non-linear Discriminant analysis** in machine learning.

Extension to Linear Discriminant Analysis (LDA)

Linear Discriminant analysis is one of the most simple and effective methods to solve classification problems in machine learning. It has so many extensions and variations as follows:

1. **Quadratic Discriminant Analysis (QDA):** For multiple input variables, each class deploys its own estimate of variance.
2. **Flexible Discriminant Analysis (FDA):** it is used when there are non-linear groups of inputs are used, such as splines.
3. **Flexible Discriminant Analysis (FDA):** This uses regularization in the estimate of the variance (actually covariance) and hence moderates the influence of different variables on LDA.

Real-world Applications of LDA

Some of the common real-world applications of Linear discriminant Analysis are given below:

- **Face Recognition** Face recognition is the popular application of computer vision, where each face is represented as the combination of a number of pixel values. In this case, LDA is used to

minimize the number of features to a manageable number before going through the classification process. It generates a new template in which each dimension consists of a linear combination of pixel values. If a linear combination is generated using Fisher's linear discriminant, then it is called Fisher's face.

- **Medical**
  In the medical field, LDA has a great application in classifying the patient disease on the basis of various parameters of patient health and the medical treatment which is going on. On such parameters, it classifies disease as mild, moderate, or severe. This classification helps the doctors in either increasing or decreasing the pace of the treatment.
- **Customer Identification** In customer identification, LDA is currently being applied. It means with the help of LDA; we can easily identify and select the features that can specify the group of customers who are likely to purchase a specific product in a shopping mall. This can be helpful when we want to identify a group of customers who mostly purchase a product in a shopping mall.
- **For Predictions** LDA can also be used for making predictions and so in decision making. For example, "will you buy this product" will give a predicted result of either one or two possible classes as a buying or not.
- **In Learning** Nowadays, robots are being trained for learning and talking to simulate human work, and it can also be considered a classification problem. In this case, LDA builds similar groups on the basis of different parameters, including pitches, frequencies, sound, tunes, etc.

Example:

Suppose that a bank is deciding whether to approve or reject loan applications. The bank uses two features to make this decision: the applicant's credit score and annual income.

Here, the two features or classes are plotted on a 2-dimensional (2D) plane with an X-Y axis. If we tried to classify approvals using just one feature, we might observe overlap. By applying LDA, we can draw a straight line that completely separates these two class data points. LDA achieves this by using the X–Y axis to create a new axis, separating the different classes with a straight line and projecting data onto the new axis.

To create this new axis and reduce dimensionality, LDA follows these criteria:

Maximize the distance between the means of two classes.
Minimize the variance within individual classes.

Properties and Assumptions of LDA

LDAs operate by projecting a feature space, that is, a dataset with n-dimensions, onto a smaller space "k", where k is less than or equal to n – 1, without losing class information. An LDA model comprises the statistical properties that are calculated for the data in each class. Where there are

multiple features or variables, these properties are calculated over the multivariate Gaussian distribution

The multivariates are:

- Means
- Covariance matrix, which measures how each variable or feature relates to others within the class

The statistical properties that are estimated from the data set are fed into the LDA function to make predictions and create the LDA model. There are some constraints to bear in mind, as the model assumes the following:

- The input dataset has a Gaussian distribution, where plotting the data points gives a bell shaped curve.
- The data set is linearly separable, meaning LDA can draw a straight line or a decision boundary that separates the data points.
- Each class has the same covariance matrix.

For these reasons, LDA may not perform well in high-dimensional feature spaces.

Perceptron Learning Algorithm

Perceptron is a linear supervised machine learning algorithm. It is used for binary classification. This article will introduce you to a very important binary classifier, the perceptrons, which forms the basis for the most popular machine learning models nowadays – the neural networks.

Introduction

Perceptron Learning Algorithm is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence. The perceptron learning algorithm is treated as the most straightforward Artificial Neural network. It is a supervised learning algorithm of binary classifiers. Hence, it is a single-layer neural network with four main parameters, **i.e.**, input values, weights and Bias, net sum, and an activation function.

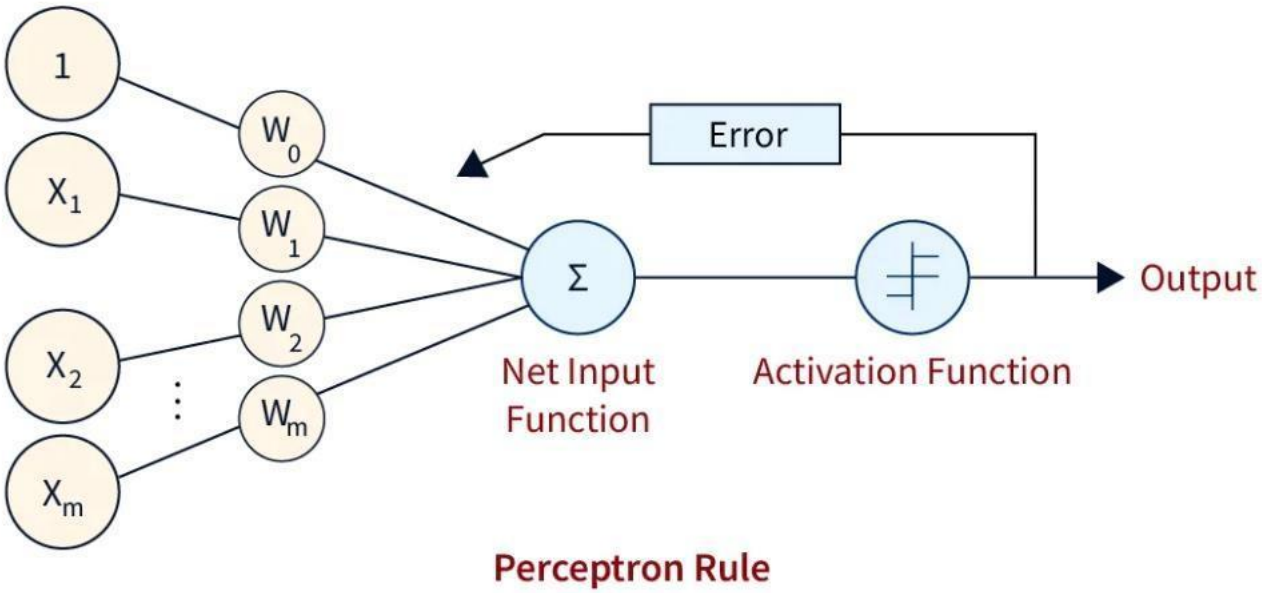
What is the Perceptron Learning Algorithm?

There are four significant steps in a perceptron learning algorithm:

1. First, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows: $\sum wi*xi = x1*w1 + x2*w2 + ... + wn*xn$ $\sum wi * xi = x1 * w1 + x2 * w2 + \ldots + wn * xn$. Add another essential term called bias 'b' to the weighted sum to improve the model performance. $\sum wi*xi + b$ $\sum wi * xi + b$.
2. Next, an activation function is applied to this weighed sum, producing a binary or a continuous-valued output. $Y = f(\sum wi*xi + b)$ $Y = f(\sum wi * xi + b)$

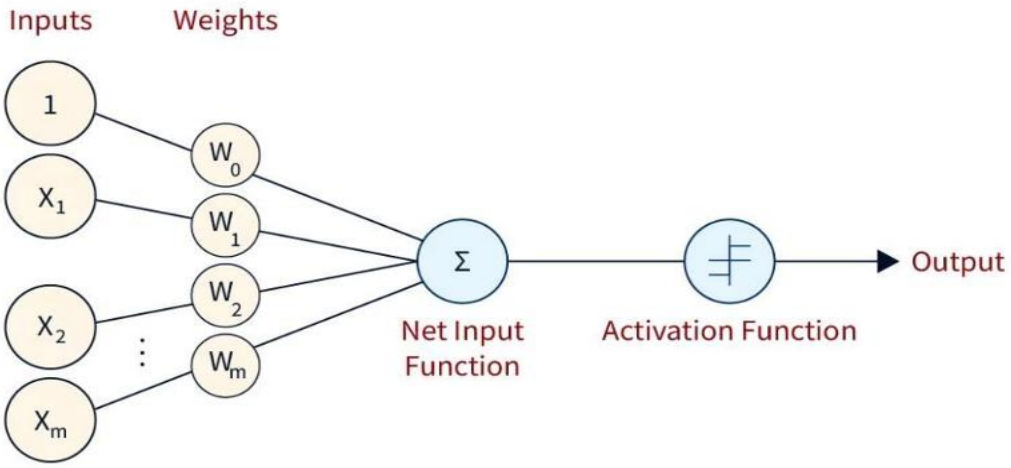
3. Next, the difference between this output and the actual target value is computed to get the error term, E, generally in terms of mean squared error. The steps up to this form the forward propagation part of the algorithm. $E=(Y-Yactual)2E=(Y-Yactual\ )2$
4. We optimize this error (loss function) using an optimization algorithm. Generally, some form of gradient descent algorithm is used to find the optimal values of the hyperparameters like learning rate, weight, Bias, etc. This step forms the backward propagation part of the algorithm.

An overview of this algorithm is illustrated in the following Figure:

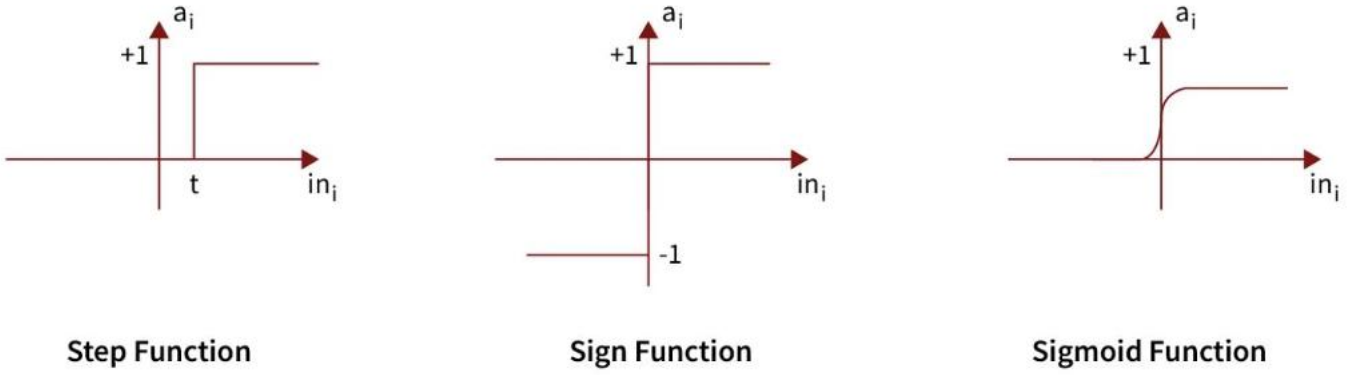


Basic Components of Perceptron

Frank Rosenblatt invented the perceptron learning algorithm.



It is a binary classifier and consists of three main components. These are:

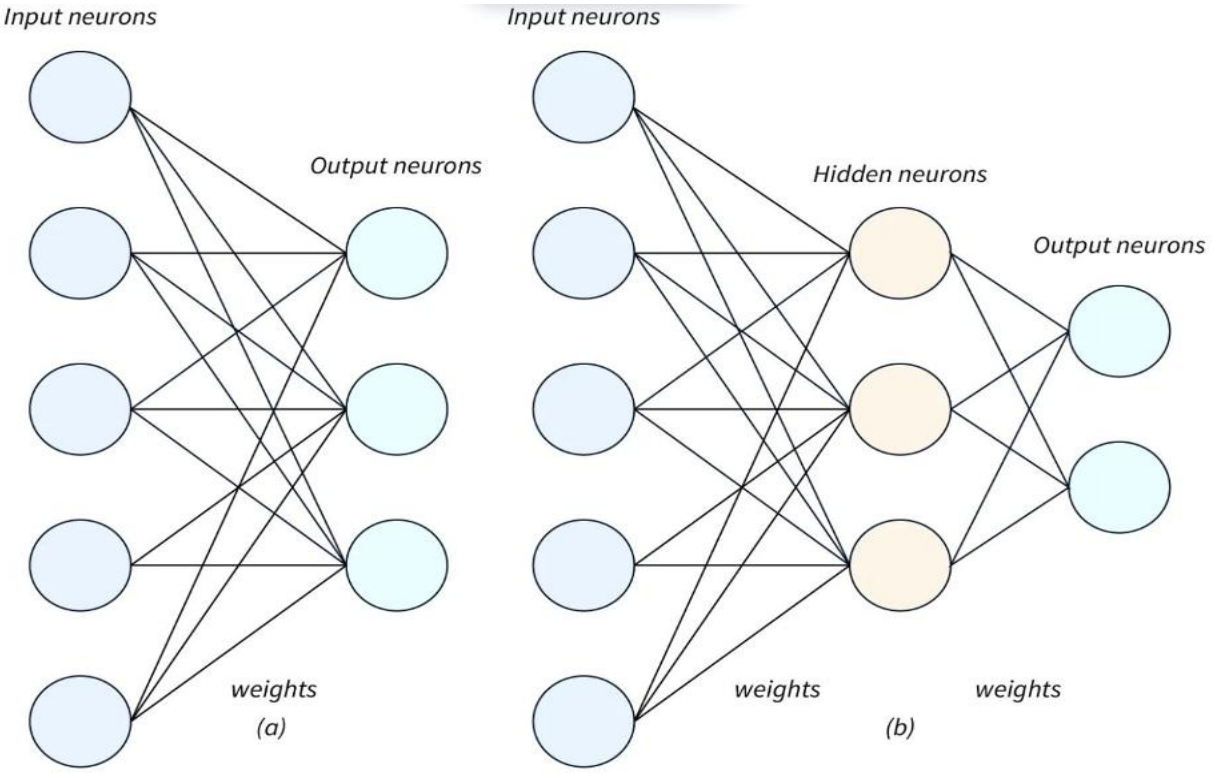


1. **Input Nodes or Input Layer:** Primary component of Perceptron learning algorithm, which accepts the initial input data into the model. Each input node contains an actual value.
2. **Weight and Bias:** The weight parameter represents the strength of the connection between units. Bias can be considered as the line of intercept in a linear equation.
3. **Activation Function:** Final and essential components help determine whether the neuron will fire. The activation function can be primarily considered a step function. There are various types of activation functions used in a perceptron learning algorithm. Some of them are the sign function, step function, sigmoid function, etc.

Types of Perceptron Models

Based on the number of layers, perceptrons are broadly classified into two major categories:

1. **Single Layer Perceptron Model:**
   It is the simplest Artificial Neural Network (ANN) model. A single-layer perceptron model consists of a feed-forward network and includes a threshold transfer function for thresholding on the Output. The main objective of the single-layer perceptron model is to classify linearly separable data with binary labels.
2. **Multi-Layer Perceptron Model:**
   The multi-layer perceptron learning algorithm has the same structure as a single-layer perceptron but consists of an additional one or more hidden layers, unlike a single-layer perceptron, which consists of a single hidden layer. The distinction between these two types of perceptron models is shown in the Figure below.



(a) Architecture of a single layer perceptron. The architecture consists of a layer on input neurons fully connected to a single layer of output neurons.

(b) Extension to a multi-layer perceptron including more than one layer of trainable weights. In this example, the network includes 3 layers: input, hidden and output layer. Each connection between two neurons is given by a certain weight.

Perceptron Function

Perceptron learning algorithm function $f(x)f(x)$ is represented as the product of the input vector (x) and the learned weight vector (w). In mathematical notion, it can be described as:

$f(x)=1, if\ w.x+b>0\ f(x)=0, otherwise\ f(x)=0$

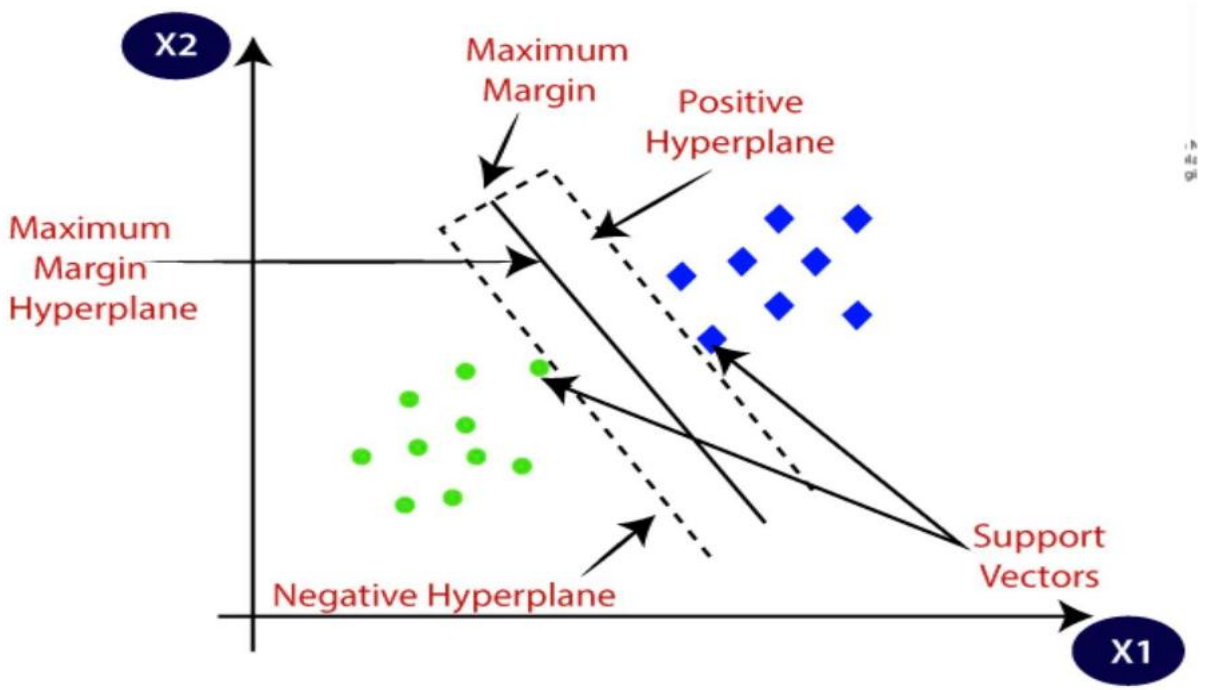
Where

- w represents the weight vector which consists of a set of real-valued weights.
- b represents the bias vector.
- x represents the input vector which consists of the input feature values.

Support Vector Machine Algorithm

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

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



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

Types of SVM

SVM can be of two types:

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

Hyperplane and Support Vectors in the SVM algorithm:

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

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

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

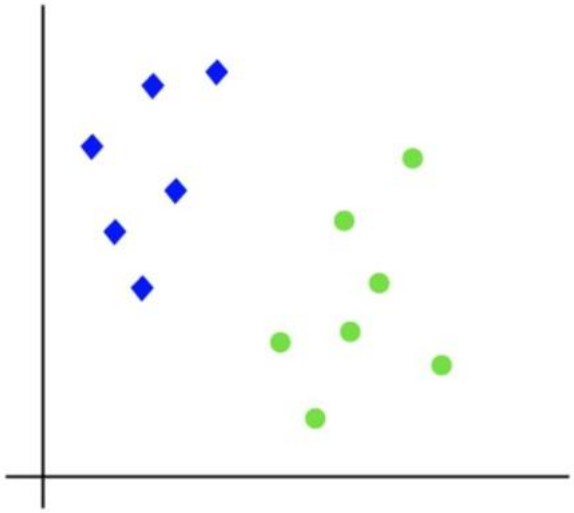
Support Vectors:

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

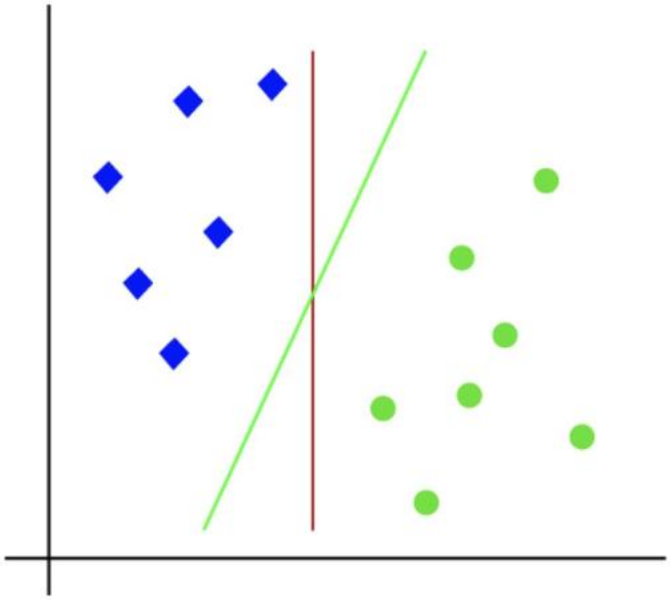
How does SVM works?

Linear SVM:

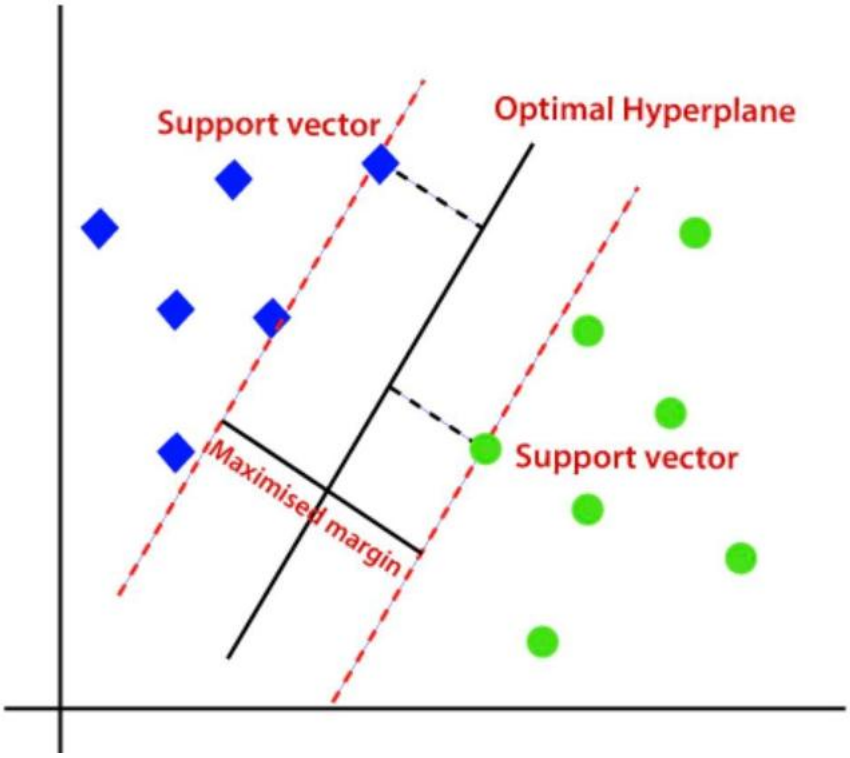
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



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

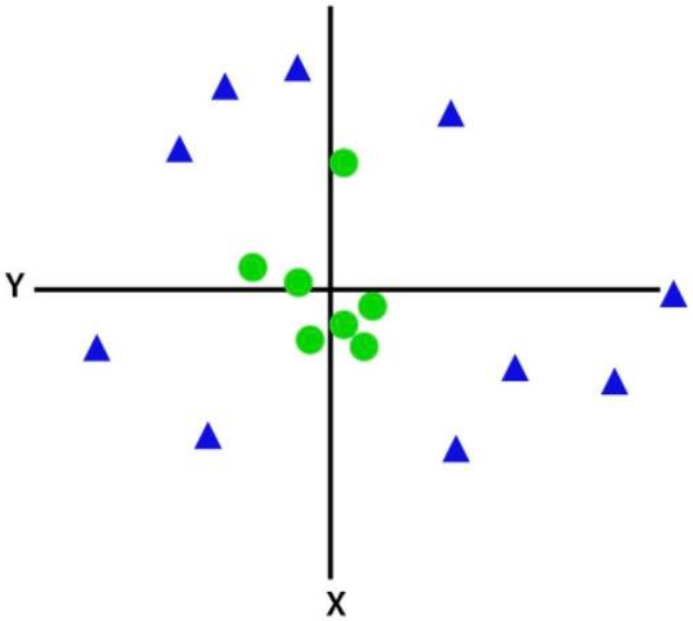


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane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



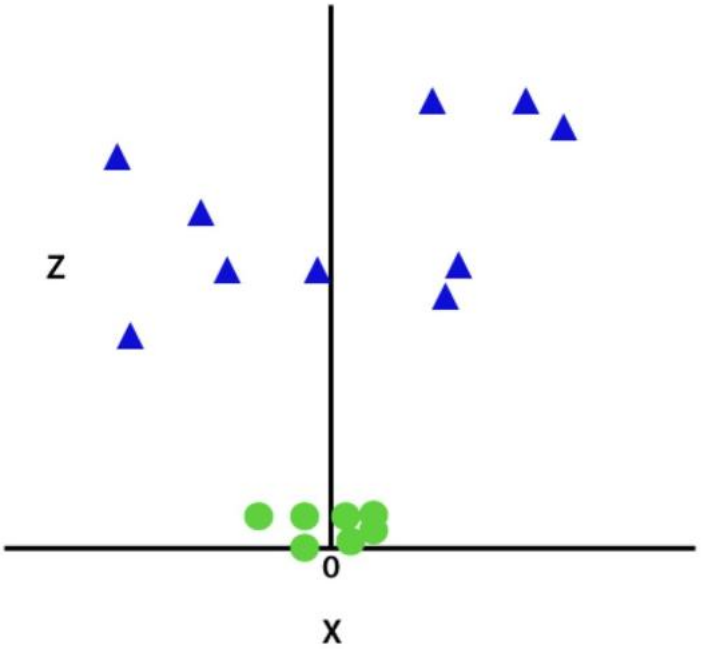
Non-Linear SVM:

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

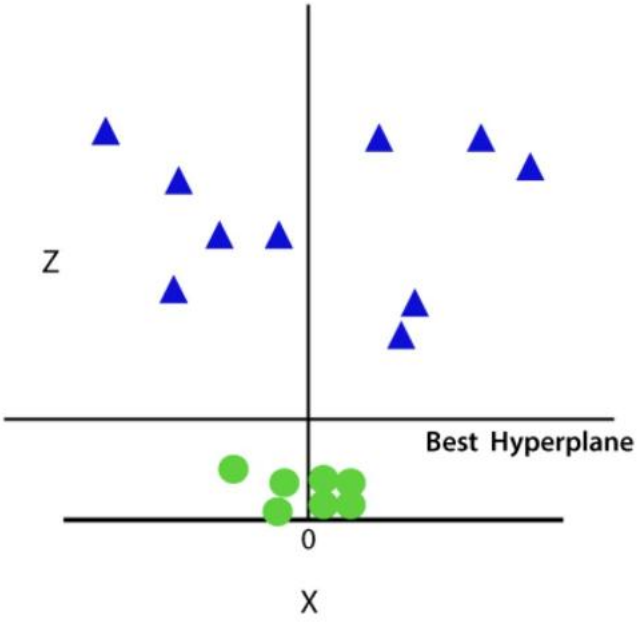


So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:
z=x^2+y^2

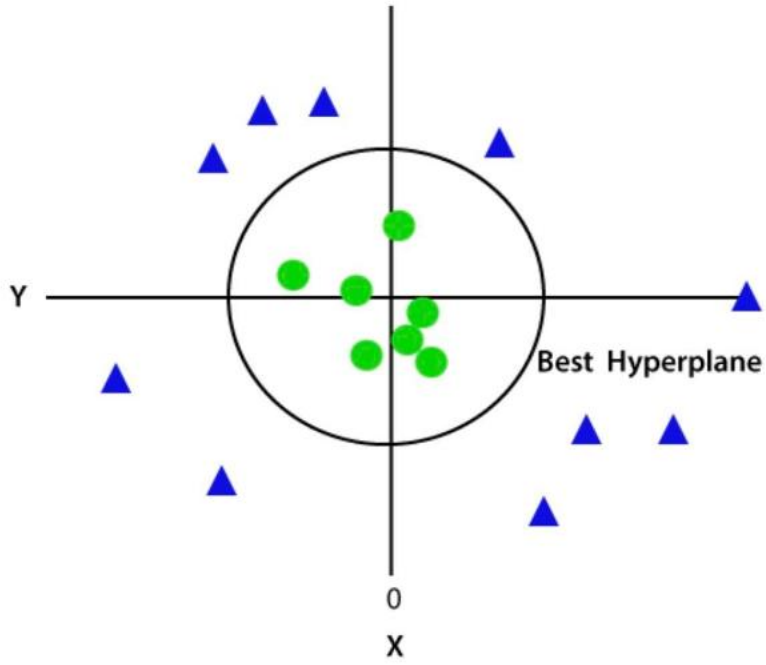
By adding the third dimension, the sample space will become as below image:



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



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



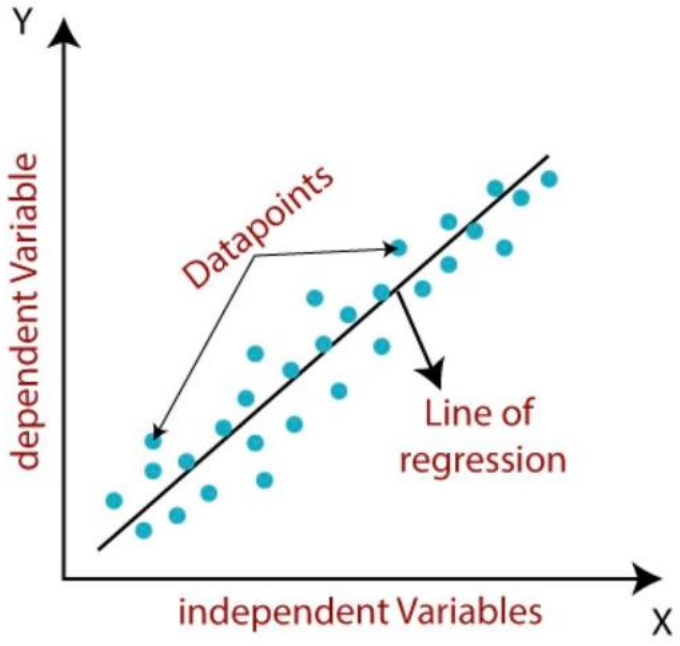
Hence we get a circumference of radius 1 in case of non-linear data.

Linear Regression in Machine Learning

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



$y= a_0+a_1x+ \varepsilon$

Here,

Y= Dependent Variable (Target Variable)

X= Independent Variable (predictor Variable)

a0= intercept of the line (Gives an additional degree of freedom)

a1 = Linear regression coefficient (scale factor to each input value).

ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

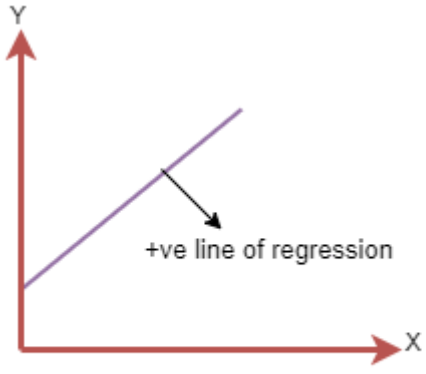
Linear regression can be further divided into two types of the algorithm:

- **Simple Linear Regression:** If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.
- **Multiple Linear regression:** If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

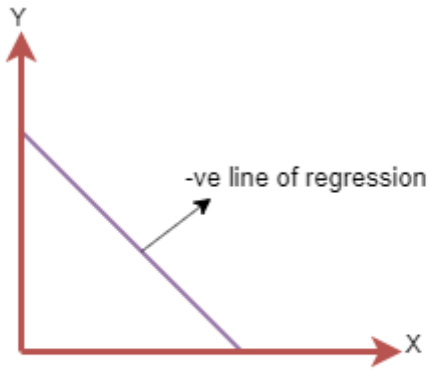
A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **Positive Linear Relationship:** If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $\mathbf{Y= a_0+a_1x}$

- **Negative Linear Relationship:** If the dependent variable decreases on the Y-axis and independent variable increases on the X- axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y= -a_0+a_1x$

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a_0, a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function-

- The different values for weights or coefficient of lines (a_0, a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

For the above linear equation, MSE can be calculated as:

$$MSE= 1\frac{1}{N}\sum_{i=1}^{n}(y_i - (a_1x_i + a_0))^2$$

Where,

N=Total number of observation

Y_i = Actual value $(a1x_i+a_0)$= Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.

- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

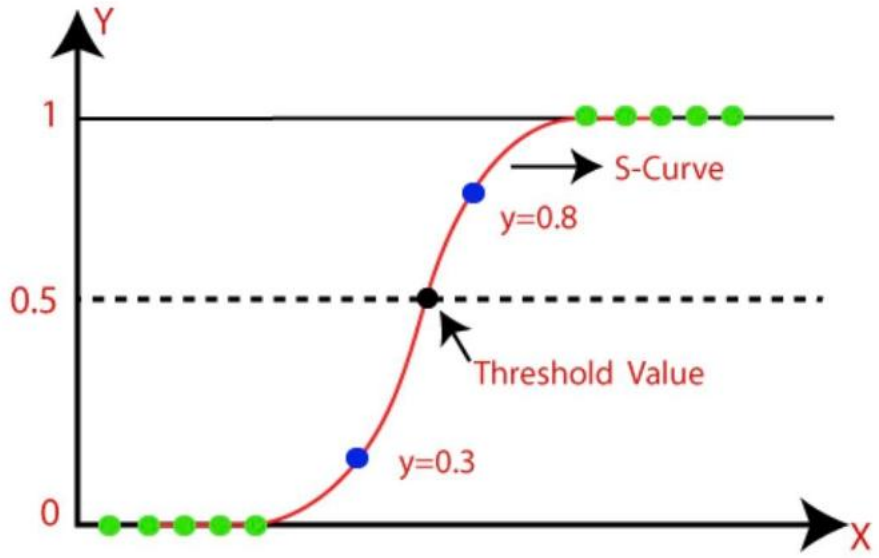
1. R-squared method:

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Logistic Regression in Machine Learning

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function.



Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

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

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

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

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

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

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

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

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

Multi-Layer Perceptron Learning in Tensorflow

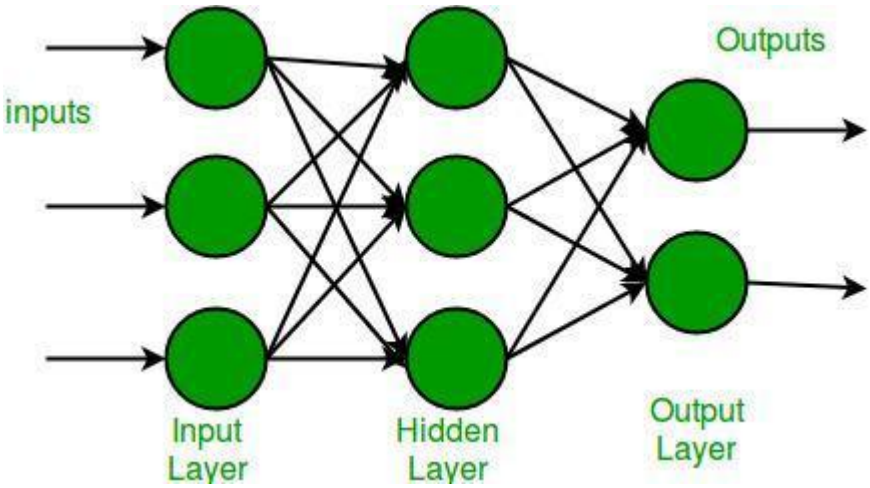
Multi-Layer Perceptron (MLP) is an artificial neural network widely used for solving classification and regression tasks.

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, making it a powerful tool for various machine learning tasks.

Pre-requisites: *Neural Network*, *Artificial Neural Network*, *Perceptron*

Key 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**: An MLP can have any number of hidden layers, with each layer containing any number of nodes. These layers process the information received from the input layer.
- **Output Layer**: The output layer generates the final prediction or result. If there are multiple outputs, the output layer will have a corresponding number of neurons.



Every connection in the diagram is a representation of the fully connected nature of an MLP. This means that every node in one layer connects to every node in the next layer. As the data moves through the network, each layer transforms it until the final output is generated in the output layer.

Working of Multi-Layer Perceptron

Let's delve in to the working of the multi-layer perceptron. The key mechanisms such as forward propagation, loss function, backpropagation, and optimization.

Step 1: Forward Propagation

In **forward propagation**, the data flows from the input layer to the output layer, passing through any hidden layers. Each neuron in the hidden layers processes the input as follows:

1. **Weighted Sum**: The neuron computes the weighted sum of the inputs:
   - $z = \sum_i w_i x_i + b$
     - Where:
       - x_i is the input feature.
       - w_i is the corresponding weight.
       - b is the bias term.
2. Activation Function: The weighted sum z is passed through an activation function to introduce non-linearity. Common activation functions include:
   - **Sigmoid**: $\sigma(z) = \frac{1}{1+e^{-z}}$
   - **ReLU (Rectified Linear Unit)**: $f(z) = \max(0, z)$
   - **Tanh (Hyperbolic Tangent)**: $\tanh(z) = \frac{2}{1+e^{-2z}} - 1$

Step 2: Loss Function

Once the network generates an output, the next step is to calculate the loss using a loss function. In supervised learning, this compares the predicted output to the actual label.

For a classification problem, the commonly used **binary cross-entropy** loss function is:

$$L = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1-y_i) \log(1-\hat{y}_i)]$$

Where:

- y_i is the actual label.
- $\hat{y}_i$ is the predicted label.
- N is the number of samples.

For regression problems, the **mean squared error (MSE)** is often used:

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

Step 3: Backpropagation

The goal of training an MLP is to minimize the loss function by adjusting the network's weights and biases. This is achieved through **backpropagation**:

1. **Gradient Calculation**: The gradients of the loss function with respect to each weight and bias are calculated using the chain rule of calculus.
2. **Error Propagation**: The error is propagated back through the network, layer by layer.
3. **Gradient Descent**: The network updates the weights and biases by moving in the opposite direction of the gradient to reduce the loss:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

- Where:
  - w is the weight.
  - η is the learning rate.
  - $\frac{\partial L}{\partial w}$ is the gradient of the loss function with respect to the weight.

Step 4: Optimization

MLPs rely on optimization algorithms to iteratively refine the weights and biases during training. Popular optimization methods include:

- **Stochastic Gradient Descent (SGD)**: Updates the weights based on a single sample or a small batch of data: $w = w - \eta \cdot \frac{\partial L}{\partial w}$
- **Adam Optimizer**: An extension of SGD that incorporates momentum and adaptive learning rates for more efficient training:
  - $m_t = \beta_1 m_{t-1} + (1-\beta_1) \cdot g_t$
  - $v_t = \beta_2 v_{t-1} + (1-\beta_2) \cdot g_t^2$
- Here, g_t represents the gradient at time t, and β_1, β_2 are decay rates.

1. **Introduction to Linear Discriminants:**

Linear Discriminant Analysis in Machine Learning

Linear Discriminant Analysis (LDA) also known as Normal Discriminant Analysis is supervised classification problem that helps separate two or more classes by converting higher-dimensional data space into a lower-dimensional space. It is used to identify a linear combination of features that best separates classes within a dataset.

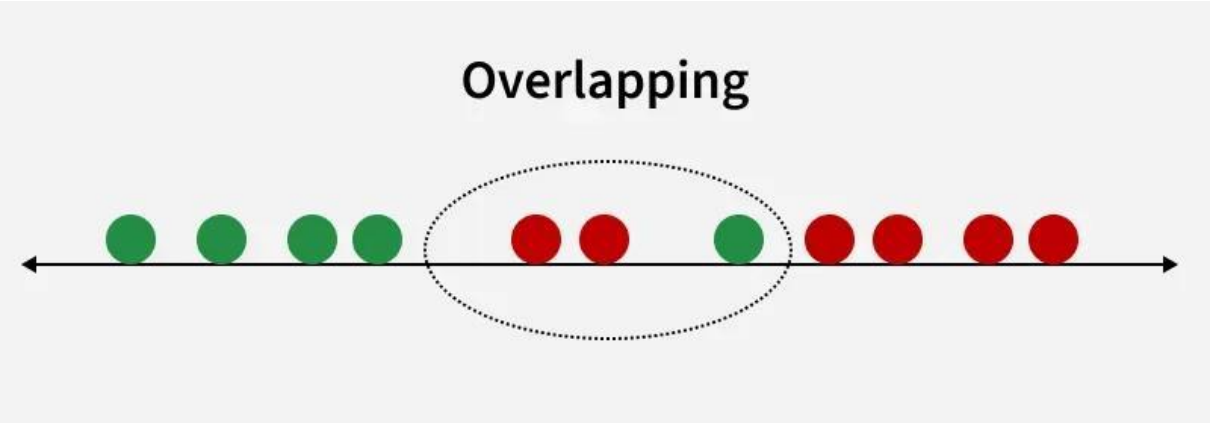


Fig: Overlapping

For example we have two classes that need to be separated efficiently. Each class may have multiple features and using a single feature to classify them may result in overlapping. To solve this LDA is used as it uses multiple features to improve classification accuracy. LDA works by some assumptions and we are required to understand them so that we have a better understanding of its working.

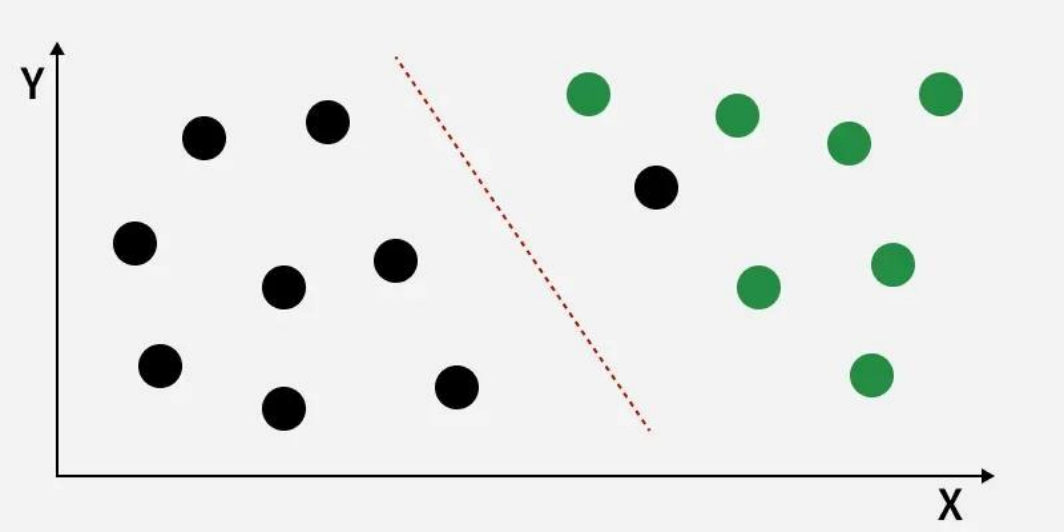
Key Assumptions of LDA

For LDA to perform effectively, certain assumptions are made:

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

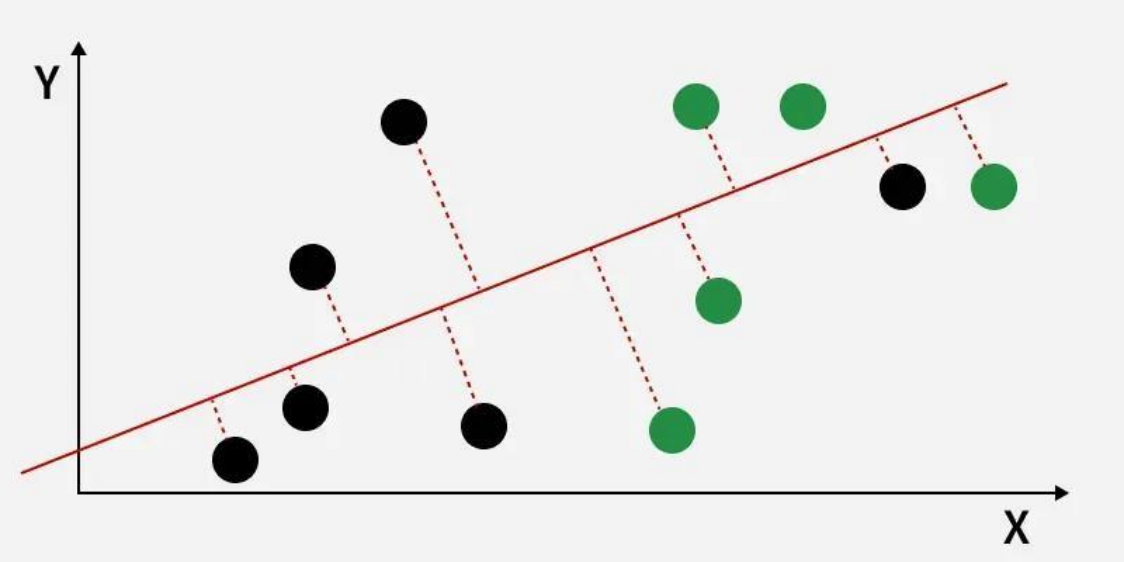
LDA can produce very good results if it meets these assumptions. For example when data points belonging to two classes are plotted, if they are not linearly

separable LDA will attempt to find a projection that maximizes class separability.



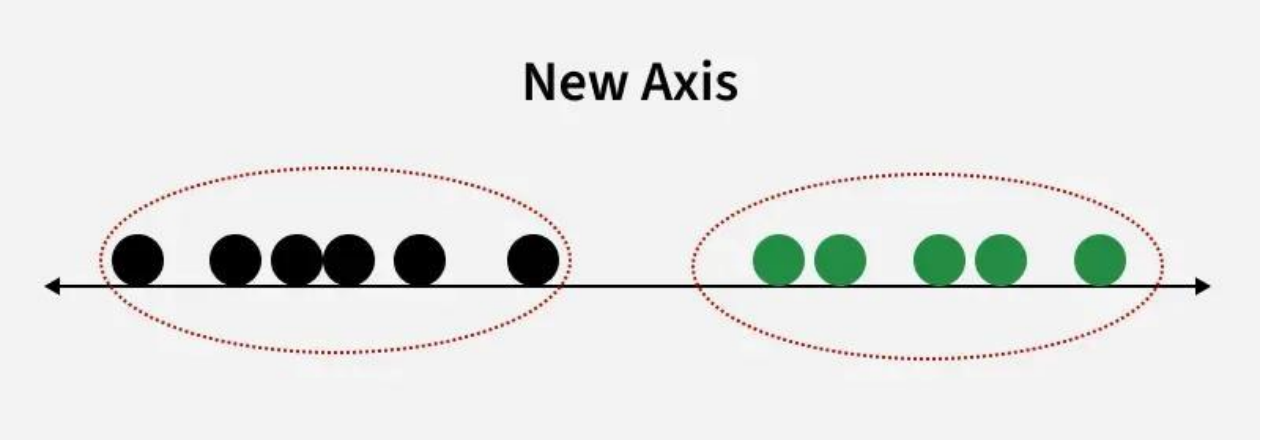
Linearly Separable Dataset

The image shows classes (black and green) that are not linearly separable. LDA finds a new axis (red dashed line) that maximizes the distance between class means while minimizing within-class variance, improving class separation for better classification.



The perpendicular distance between the line and points

The perpendicular distance from the decision boundary to the data points shows how LDA reduces within-class variation and increases class separability. The data points are then projected onto the new axis, as shown in the figure below.



New Axis

This shows how LDA creates a new axis to project the data and separate two classes along a linear path. However, when class distributions share the same mean, LDA cannot find a separating axis and non-linear discriminant analysis is needed.

How LDA work

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

Let's assume we have two classes with d-dimensional samples such as $x_1, x_2, \ldots x_n$ where:

- n_1 samples belong to class c_1
- n_2 samples belong to class c_2.

If x_i represents a data point its projection onto the line represented by the unit vector v is $v^T x_i$. Let the means of class c_1 and class c_2 before projection be μ_1 and μ_2 respectively. After projection the new means are $\hat{\mu}_1 = v^T \mu_1$ and $\hat{\mu}_2 = v^T \mu_2$.

Our aim to normalize the difference $| \hat{\mu}_1 - \hat{\mu}_2 |$ to maximize the class separation. The scatter for samples of class c_1 is calculated as:

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

Similarly for class c_2:

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

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

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

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

Extensions to LDA

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

Why LDA?

• Logistic Regression is one of the most popular classification algorithms that perform well for binary classification but falls short in the case of multiple classification problems with well separated classes. At the same time, LDA handles these quite efficiently.

• LDA can also be used in data pre-processing to reduce the number of features, just as PCA, which reduces the computing cost significantly.

• LDA is also used in face detection algorithms. In Fisher faces, LDA is used to extract useful data from different faces. Coupled with eigenfaces, it produces effective results.

Drawbacks of Linear Discriminant Analysis (LDA)

Although, LDA is specifically used to solve supervised classification problems for two or more classes which are not possible using logistic regression in machine learning. But LDA also fails in some cases where the Mean of the distributions is shared. In this case, LDA fails to create a new axis that makes

both the classes linearly separable. To overcome such problems, we use **non-linear Discriminant analysis** in machine learning.

Real-world Applications of LDA:

Some of the common real-world applications of Linear discriminant Analysis are given below:

1. Face Recognition
2. Medical
3. Customer Identification
4. For Predictions
5. **In Learning:** Nowadays, robots are being trained for learning and talking to simulate human work, and it can also be considered a classification problem. In this case, LDA builds similar groups on the basis of different parameters, including pitches, frequencies, sound, tunes, etc.

Example:

Suppose that a bank is deciding whether to approve or reject loan applications. The bank uses two features to make this decision: the applicant's credit score and annual income. Here, the two features or classes are plotted on a 2-dimensional (2D) plane with an X-Y axis. If we tried to classify approvals using just one feature, we might observe overlap. By applying LDA, we can draw a straight line that completely separates these two class data points. LDA achieves this by using the X–Y axis to create a new axis, separating the different classes with a straight line and projecting data onto the new axis. To create this new axis and reduce dimensionality,

LDA follows these criteria:

Maximize the distance between the means of two classes.

Minimize the variance within individual classes.

MACHINE LEARNING
UNIT-V

Introduction to Clustering, Partitioning of Data, Matrix Factorization |
Clustering of Patterns, Divisive Clustering, Agglomerative Clustering, Partitional Clustering, K-Means Clustering, Soft Partitioning, Soft Clustering, Fuzzy C-Means Clustering, Rough Clustering, Rough K-Means Clustering Algorithm, Expectation Maximization-Based Clustering, Spectral Clustering

---

Introduction to Clustering:

Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset. It can be defined as "A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."

It does it by finding some similar patterns in the unlabelled dataset such as shape, size, color, behavior, etc., and divides them as per the presence and absence of those similar patterns.

It is an unsupervised learning method, hence no supervision is provided to the algorithm, and it deals with the unlabeled dataset.

After applying this clustering technique, each cluster or group is provided with a cluster-ID. ML system can use this id to simplify the processing of large and complex datasets.

The clustering technique is commonly used for statistical data analysis.

Clustering refers to the process of arranging or organizing objects according to specific criteria.

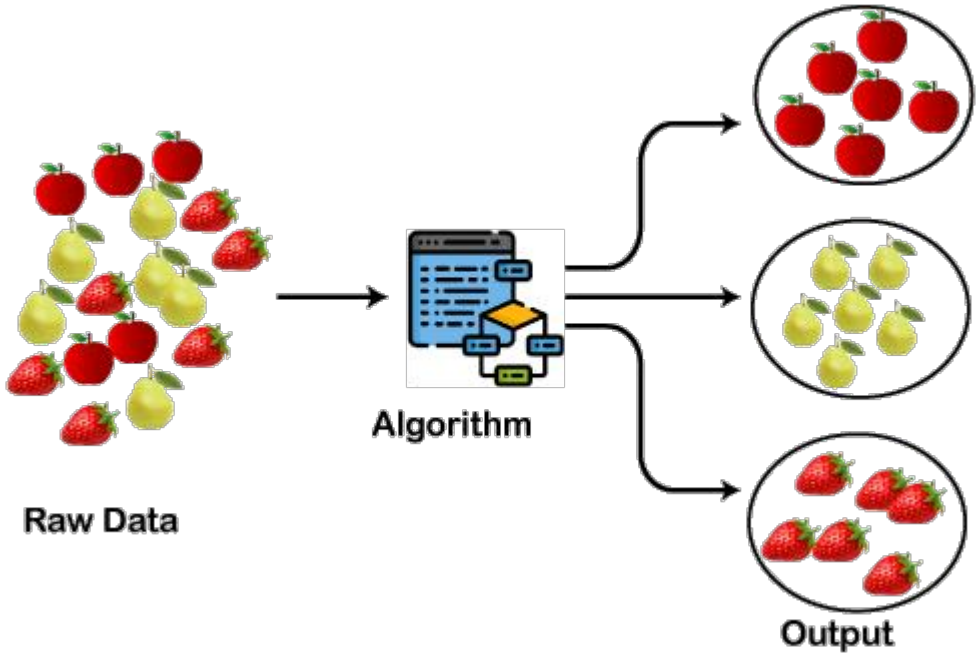
• Partitioning of data-Grouping of data in database application improves the data access

Ex:

Let's understand the clustering technique with the real-world example of Mall: When we visit any shopping mall, we can observe that the things with similar usage are grouped together. Such as the t-shirts are grouped in one section, and trousers are at other sections, similarly, at vegetable sections, apples, bananas, Mangoes, etc., are grouped in separate sections, so that we can easily find out the things. The clustering technique also works in the same way. Other examples of clustering are grouping documents according to the topic.

Ex:

The below diagram explains the working of the clustering algorithm. We can see the different fruits are divided into several groups with similar properties.



Ex:

- Data reorganization

Pattern	f1	f2	f3	f4	f5	f6
1	1	0	1	0	1	0
2	0	1	0	1	0	1
3	1	0	1	0	1	0
4	0	1	0	1	0	1

Row data

Pattern	f1	f2	f3	f4	f5	f6
1	1	0	1	0	1	0
3	1	0	1	0	1	0
2	0	1	0	1	0	1
4	0	1	0	1	0	1

Data re-organization using row as a criteria

- Data compression

Pattern	f1	f2	f3	f4	f5	f6	count
1,3	1	0	1	0	1	0	2
2 ,4	0	1	0	1	0	1	2

Compressed data

Applications of Clustering

Below are some commonly known applications of clustering technique in Machine Learning:

- In Identification of Cancer Cells: The clustering algorithms are widely used for the identification of cancerous cells. It divides the cancerous and non-cancerous data sets into different groups.
- In Search Engines: Search engines also work on the clustering technique. The search result appears based on the closest object to the search query. It does it by grouping similar data objects in one group that is far from the other dissimilar objects. The accurate result of a query depends on the quality of the clustering algorithm used.
- Customer Segmentation: It is used in market research to segment the customers based on their choice and preferences.
- In Biology: It is used in the biology stream to classify different species of plants and animals using the image recognition technique.
- In Land Use: The clustering technique is used in identifying the area of similar lands use in the GIS database. This can be very useful to find that for what purpose the particular land should be used, that means for which purpose it is more suitable.

Partitioning of data:

Partitioning of data in clustering refers to dividing a dataset into k distinct, non-overlapping groups or clusters based on certain criteria, usually by minimizing the distance between points within the same cluster and maximizing the distance between points in different clusters.

How Partitioning Works:

- Choose the number of clusters k (usually predefined in partitioning methods like K-means).
- Randomly initialize k cluster centroids.
- Assign each data point to the nearest cluster (based on a distance metric like Euclidean distance).
- Update centroids by calculating the mean of all points in each cluster.

- Repeat steps 3 and 4 until centroids don't change or a set number of iterations is reached.

Ex: Partitioning with K-means

Imagine you have the following dataset of points in 2D space:

{(2,3),(3,3),(4,4),(10,12),(11,11),(12,13)}

Steps:

Choose k=2 (we want 2 clusters).

Initialize centroids:

Let C1=(2,3)

Let C2=(10,12)

Assign points to clusters:

Points closer to C1: (2,3),(3,3),(4,4)

Points closer to C2C_2C2: (10,12),(11,11),(12,13)

Update centroids:

New C1=mean of (2,3),(3,3),(4,4)=(3,3.33)

New C2=mean of (10,12),(11,11),(12,13)=(11,12)

Repeat until centroids stop changing.

Result:

Cluster 1: (2,3),(3,3),(4,4)

Cluster 2: (10,12),(11,11),(12,13)

Matrix Factorization:

Matrix factorization is a powerful technique in clustering, often used to reduce dimensionality and reveal latent structures in data.

- Let there be n data points in an l-dimensional space. We can represent it as a matrix $X_{n\times l}$.
- It is possible to approximate X as a product of two matrices $B_{n\times K}$ and $C_{K\times l}$.
- So, $X \approx BC$, where B is the cluster assignment matrix and C is the representatives matrix

Key points:

· Noise reduction: Removes irrelevant features or noise by capturing only essential patterns.

· Visualization: Projects high-dimensional data into 2D or 3D spaces for better cluster visualization.

· Scalability: Efficiently handles large datasets compared to distance-based methods like K-means.

Clustering of Patterns:

Patterns refer to sets of features or attributes that describe objects or events — like customer behavior, image pixel intensities, or DNA sequences.

Clustering patterns means grouping these patterns into meaningful clusters based on their similarities, without prior labels.

Explanation:

- Data (Patterns):

  Each handwritten digit (28x28 pixel image) is flattened into a 784-dimensional vector — a pattern of pixel intensities.

- Clustering:

  We apply K-means to divide the 1000 samples into 10 clusters (expecting them to correspond to digits 0–9).

- Visualization:

  The cluster centroids (averages of patterns in each group) are reshaped back into 28x28 images and plotted.

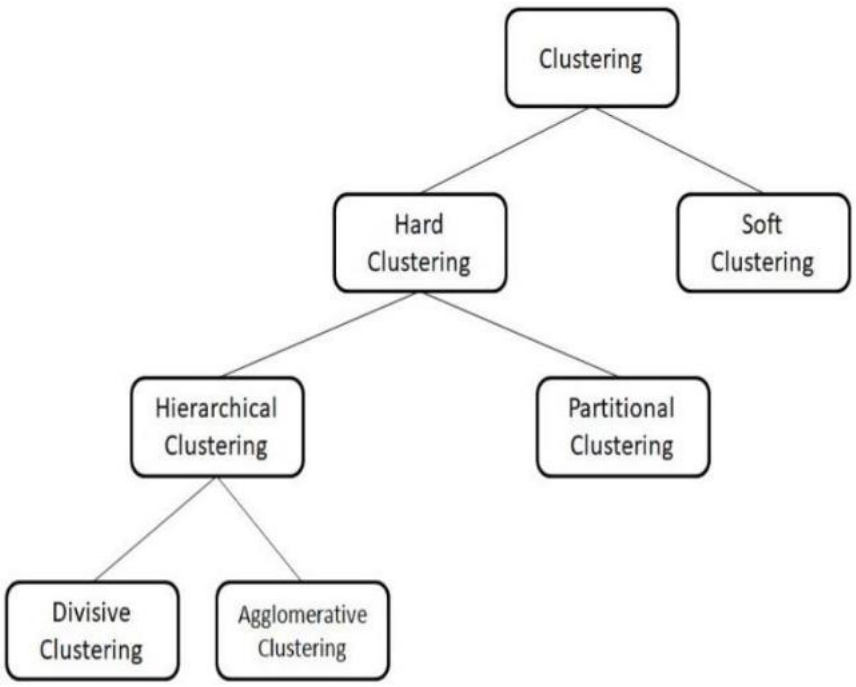
Why cluster patterns?

Image processing: Group similar images together for compression or recognition.

Anomaly detection: Identify outliers (e.g., fraudulent patterns in transaction data).

Genomics: Cluster gene expression patterns to understand biological processes.

<u>Clustering Algorithms:</u>



Divisive Clustering:

Divisive clustering on the other hand, is a top-down hierarchical clustering approach where: It starts with all data points in a single cluster and recursively splits the clusters into smaller sub-clusters based on their dissimilarity until each data point is in its own individual cluster. This approach is more computationally intensive, as it requires splitting the data rather than merging it.

Key steps in divisive clustering:

- Start with a single cluster containing all the data points.
- Split the cluster into two sub-clusters based on their dissimilarity.
- Recursively apply the same process to the resulting sub-clusters.
- Repeat until each data point is in its own cluster.

Divisive clustering's complexity can vary depending on the implementation it generally requires more computational power due to the recursive splitting process. However because it operates on sub-clusters it can sometimes reduce the computational cost when compared to agglomerative clustering on very large datasets

Divisive clustering is more complex to implement and requires a careful choice of splitting criteria making it less commonly used than agglomerative clustering that's why it is not available in SciPy and Scikit learn.

If a collection of patterns (data points) is split into two clusters with p patterns $x_1, \cdots, x_p$ in one cluster and q patterns $y_1, \cdots, y_q$ in the other cluster with the centroids of the two clusters being C1 and C2 respectively, then the sum of the sample variances will be

$$\sum_{i=1}^{p}(x_i - C1)^2 + \sum_{j=1}^{q}(y_j - C2)^2$$

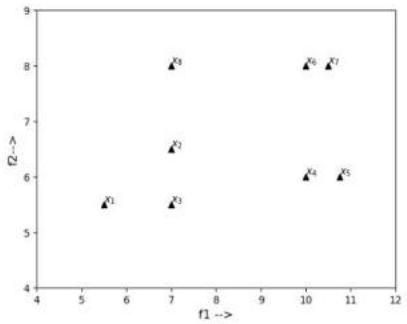
Key Points:

· **Better granularity:** Good for detecting outliers or small clusters.
· **Global perspective:** Starts with a bird's-eye view of all data points, which can be useful when you want to prioritize large differences early on.
· **Applications:** Used in text clustering, bioinformatics (e.g., gene expression analysis), and anomaly detection.

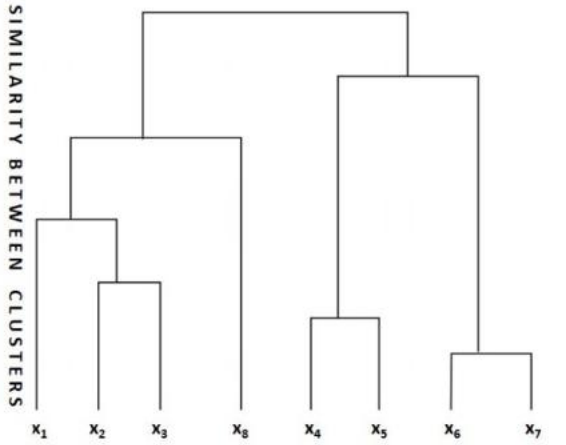
Ex:

Patterns	f1	f2
x_1	5.5	5.5
x_2	7	6.5
x_3	7	5.5
x_4	10	6
x_5	10.75	6
x_6	10	8
x_7	10.5	8
x_8	7	8

Example data patterns with two features



Visual representation of data patterns



Divisive clustering for polythetic clustering for the data points

---

Agglomerative Clustering:

Agglomerative clustering is a bottom-up approach where each data point starts as its own individual cluster. The algorithm iteratively merges the most similar pairs of clusters until all the data points belong to a single cluster. It's widely used due to its simplicity and efficiency in many clustering tasks.

Key steps in agglomerative clustering:

- Treat each data point as a separate cluster.
- Calculate the similarity (distance) between all pairs of clusters.
- Merge the two most similar clusters.
- Repeat steps 2-3 until all points belong to a single cluster.

This method can be computationally expensive especially for large datasets. The algorithm needs to compute the distance between every pair of points leading to a time complexity of $O(n^3)$ for large datasets.

More computationally expensive due to pairwise distance calculations.

Scikit-learn provides multiple linkage methods such as "ward," "complete," "average," and "single."

Applications	Image segmentation, customer segmentation, document clustering, etc.

Ex:

Patterns	f1	f2
x_1	5.5	5.5
x_2	7	6.5
x_3	7	5.5
x_4	10	6
x_5	10.75	6
x_6	10	8
x_7	10.5	8
x_8	7	8

Example data patterns with two features

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
x_1	0	2.5	1.5	5	5.5	7	7.5	4
x_2	2.5	0	1	3.5	4.25	4.5	5	1.5
x_3	1.5	1	0	3.5	4.25	5.5	6	2.5
x_4	5	3.5	3.5	0	0.75	2	2.5	5
x_5	5.75	4.25	4.25	0.75	0	2.75	2.25	5.75
x_6	7	4.5	5.5	2	2.75	0	0.5	3
x_7	7.5	5	6	2.5	2.25	0.5	0	3.5
x_8	4	1.5	2.5	5	5.75	3	3.5	0

City-block distance between pairs of data points

	x_1	x_2	x_3	x_4	x_5	C1=$\{x_6, x_7\}$	x_8
x_1	0	2.5	1.5	5	5.5	7	4
x_2	2.5	0	1	3.5	4.25	4.5	1.5
x_3	1.5	1	0	3.5	4.25	5.5	2.5
x_4	5	3.5	3.5	0	0.75	2	5
x_5	5.75	4.25	4.25	0.75	0	2.75	5.75
$C1 = \{x_6, x_7\}$	7	4.5	5.5	2	2.25	0	3
x_8	4	1.5	2.5	5	5.75	3	0

City-block distance among the data points after merging X6 and X7 as one cluster

---

Partitional Clustering:

Partitional clustering directly divides a dataset into k non-overlapping clusters.

Unlike hierarchical clustering, it doesn't create a nested structure (no dendrograms!).

The most popular method is K-means clustering.

 Key characteristics:

Input: Number of clusters k.

Objective: Minimize intra-cluster variance (or distance between points and their cluster centroids).

Output: k disjoint clusters.

Why use Partitional Clustering?

Efficiency: Faster than hierarchical clustering for large datasets.

Flexibility: Works well when you know the number of clusters in advance.

Interpretability: Simple, especially with K-means, since centroids clearly represent cluster centers.

 Real-world Applications:

Customer segmentation: Grouping customers by purchasing behavior.

Image compression: Clustering similar pixel colors.

Document clustering: Organizing texts by topic.

K-Means Clustering:

K-means is a **partitional clustering algorithm** that:

- Divides n data points into k **clusters**.
- Minimizes the sum of squared distances between points and their cluster centroids.

Algorithm:

Initialize: Choose k random points as initial centroids.

Assign points: Each point is assigned to the cluster with the nearest centroid.

Update centroids: Compute the mean of all points in each cluster — this becomes the new centroid.

Repeat: Steps 2–3 until centroids stabilize or max iterations are reached.

The objective function minimizes:

$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} ||x_j - \mu_i||^2$$

where:

- C_i is cluster i.
- μ_i is the centroid of C_i.
- $||x_j - \mu_i||^2$ is the squared Euclidean distance.

Real-world Applications:

Customer segmentation: Grouping customers by behavior or demographics.

Image compression: Quantizing colors by clustering pixel intensities.

Anomaly detection: Isolating points that don't fit any cluster.

Soft Partitioning/Soft clustering:

- In **hard partitioning** (like K-means), each data point belongs to **exactly one cluster** — no overlap allowed.
- In **soft partitioning** (or **fuzzy clustering**), each point can belong to **multiple clusters with varying degrees of membership**.

The most common soft clustering algorithm is **Fuzzy C-means (FCM)**.
Instead of assigning a point x_i strictly to one cluster, it calculates a **membership value** u_{ij} for each cluster j:

$$u_{ij} \in [0,1] \quad \text{with} \quad \sum_{j=1}^{k} u_{ij} = 1$$

The objective function to minimize is:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{k} (u_{ij})^m ||x_i - c_j||^2$$

where:

- m controls the fuzziness (usually $m > 1$).
- c_j is the cluster centroid.
- $||x_i - c_j||$ is the Euclidean distance between point i and cluster j.

Fuzzy C-Means Clustering:

- In this type, each data point x_i can be assigned to a cluster C_j with a membership value μ_{ij}, which represents the degree to which x_i belongs to C_j.
- The membership value μ_{ij} is assumed to range between 0 and 1, and for each data point x_i, the sum of all μ_{ij} values across clusters is equal to 1.
- The fuzzy c-means clustering algorithm follows an iterative scheme similar to the k-means algorithm.
- It begins by identifying c initial cluster centers from the dataset or membership values between each data point and the cluster. The following steps are then performed iteratively:

1. *Compute membership values:*

$$\mu_{ij} = \frac{\mathrm{d}(x_j, C_i)^{-1/(M-1)}}{\sum_{l=1}^{c} \mathrm{d}(x_j, C_l)^{-1/(M-1)}}$$

2. *Compute fuzzy cluster centroid:*

$$C_i = \frac{\sum_{j=1}^{m} (\mu_{ij})^M \times x_j}{\sum_{j=1}^{m} (\mu_{ij})^M}$$

In above expressions

- μ_{ij} represents the membership value of data point x_j in cluster C_i, where j $= 1, 2, \ldots, n$ and i $= 1, 2, \ldots, c$.
- $\mathrm{d}(x_j, C_i)$ denotes the Euclidean distance between x_j and C_i.
- C_i represents the centroid of the ith cluster.
- M is the fuzzyifying constant which determines the behavior of the algorithm. When $M = 1$, the fuzzy algorithm functions similar to the hard k-means algorithm, where $\mu_{ij} = 1$ when X_j is assigned to C_i. As M increases or tends to infinity, μ_{ij} approaches $\frac{1}{c}$, as the exponents in both the numerator and denominator tend towards 0.

3. Repeat the above steps, and the algorithm terminates when there is no significant change in the computed values for membership and cluster centroid.

Key differences from K-means:

Soft clustering: A point can belong to multiple clusters with varying degrees.

Fuzziness control: m controls how "soft" the clustering is — higher m means more overlap between clusters.

Convergence: Like K-means, FCM can converge to local minima, so initialization matters.

Rough Clustering:

Rough clustering is based on Rough Set Theory (proposed by Zdzisław Pawlak in 1982). Unlike traditional clustering methods, rough clustering allows for overlapping clusters by dividing each cluster into two regions:

Lower Approximation: Contains elements that definitely belong to the cluster.

Upper Approximation: Contains elements that possibly belong to the cluster.

This approach helps deal with uncertain boundaries between clusters.

The sets are defined as:

- **Lower Approximation** $\text{Lower}(C)$ = set of points that belong to **only** cluster C.
- **Upper Approximation** $\text{Upper}(C)$ = set of points that might belong to **either** cluster C or others.

The **boundary region** is the difference between upper and lower approximations:

$$\text{Boundary}(C) = \text{Upper}(C) - \text{Lower}(C)$$

We can define rough set using the lower approximation $(\underline{R}(S))$ and the upper approximation $(\overline{R}(S))$ where $S \subseteq \mathfrak{X}$, a set of data points as follows:

$$\underline{R}(S) = \bigcup_i G_i, where \quad G_i \subseteq S$$

$$\overline{R}(S) = \bigcup_i G_i, where\ G_i \cap S \neq \emptyset$$

Ex:

Consider **6 points**:

$$(1,2), (2,1), (3,3), (8,8), (9,9), (10,10)$$

We want to group them into **2 clusters**:

1. **Initial partitioning:**
   - Cluster 1 (C1): points close to (2, 2)
   - Cluster 2 (C2): points close to (9, 9)
2. **Define lower and upper approximations:**
   - **Lower(C1):** Points definitely in Cluster 1 → $(1,2), (2,1), (3,3)$
   - **Upper(C1):** Points possibly in Cluster 1 → $(1,2), (2,1), (3,3), (8,8)$
   - **Lower(C2):** Points definitely in Cluster 2 → $(9,9), (10,10)$
   - **Upper(C2):** Points possibly in Cluster 2 → $(8,8), (9,9), (10,10)$
3. **Boundary regions:**
   - **Boundary(C1):** $(8,8)$
   - **Boundary(C2):** $(8,8)$

Rough K-Means Clustering Algorithm:

Let n be the number of data points, $\mathfrak{X} = \{x_1, x_2, \cdots, x_n\}$ having l features, k be the number of clusters, w_l and w_u are the weights associated with lower and upper approximation, ϵ is the threshold value.

1. Randomly assign each data object to exactly one lower approximation $\underline{R}(k)$. Note that by definition, the data object also belongs to upper approximation $\overline{R}(k)$ of the same cluster.
2. Compute the cluster centroid, C_j as follows:
   - If $\underline{R}(k) \neq \emptyset$ and $\overline{R}(k) - \underline{R}(k) = \emptyset$

     $C_j = \sum_{x_i \in \underline{R}(k)} \frac{x_i}{|\underline{R}(k)|}$
   - Else If $\underline{R}(k) = \emptyset$ and $\overline{R}(k) - \underline{R}(k) \neq \emptyset$

     $C_j = \sum_{x_i \in \overline{R}(k) - \underline{R}(k)} \frac{x_i}{|\overline{R}(k) - \underline{R}(k)|}$
   - Else

     $C_j = w_l \times \sum_{x_i \in \underline{R}(k)} \frac{x_i}{|\underline{R}(k)|} + w_u \times \sum_{x_i \in \overline{R}(k) - \underline{R}(k)} \frac{x_i}{|\overline{R}(k) - \underline{R}(k)|}$
3. For each data point in $\mathfrak{X}$, compute Euclidean distance with each cluster, C_j, $1 \leq j \leq k$ [i.e., $\mathrm{d}(x_i, C_j)$, $\forall i \in m$, $\forall j \in k$].
   Let $\mathrm{d}(X, C_p) = min_{1 \leq j \leq k}(x, Cj)$.
4. Use the ratio, $\frac{\mathrm{d}(x, C_j)}{\mathrm{d}(x, C_p)}$, where $1 \leq i, p \leq k$ to determine the membership of X as follows:
   - If the ratio $\nleq \epsilon$ then the corresponding data point, x will not be part of the lower approximation.
   - Else, the corresponding data point, x will be a part of the lower approximation
5. Repeat steps from 2 until convergence (i.e., Old centroid = New centroid)

Expectation Maximization-Based Clustering:

EM clustering is a soft clustering technique often used for Gaussian Mixture Models (GMM). Unlike K-means, which assigns each point to a single cluster, EM calculates probabilities for each point's cluster membership.

The main goal is to:

Estimate cluster parameters (means, variances, and weights for each Gaussian distribution).

Assign probabilities of each point belonging to each cluster.

Steps:

1. **Initialization:**
   - Randomly initialize means μ_k, covariances Σ_k, and mixing coefficients π_k for each cluster k.
2. **Expectation step (E-step):**
   - Calculate the probability (responsibility) that each point x_i belongs to cluster k:

$$r_{ik} = \frac{\pi_k \cdot \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_i|\mu_j, \Sigma_j)}$$

where $\mathcal{N}(x_i|\mu_k, \Sigma_k)$ is the Gaussian probability density function.

3. **Maximization step (M-step):**
   - Update the cluster parameters:

$$\mu_k = \frac{\sum_{i=1}^{N} r_{ik} \cdot x_i}{\sum_{i=1}^{N} r_{ik}}$$

$$\Sigma_k = \frac{\sum_{i=1}^{N} r_{ik}(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{N} r_{ik}}$$

$$\pi_k = \frac{1}{N} \sum_{i=1}^{N} r_{ik}$$

4. **Repeat:**
   - Iterate until convergence — when changes in parameters are very small.

Aspect	K-means	EM (GMM)
Clusters	Hard assignments	Soft assignments (probabilistic)
Shape	Spherical	Elliptical (uses covariance)
Optimization	Minimizes distance to centroids	Maximizes likelihood function
Handling overlap	No overlap	Allows overlap (through probabilities)
Model assumption	Implicit spherical clusters	Explicit Gaussian distributions

---

Spectral Clustering:

Spectral clustering is a powerful method for finding clusters in non-linearly separable data — data where traditional clustering (like K-means) struggles.

Working Procedure:

- Instead of clustering directly in the original space, spectral clustering:
- Constructs a similarity graph: Nodes are data points, and edges represent similarity (based on distance or kernel functions).
- Computes the Laplacian matrix: Derived from the graph's adjacency matrix.
- Finds eigenvectors: The smallest eigenvalues' corresponding eigenvectors map the data to a lower-dimensional space.
- Applies K-means: Finally, K-means clustering is used on these transformed points.

1. **Construct a similarity graph:**
   - Create a graph $G = (V, E)$, where each node is a data point.
   - Edges E are weighted by a similarity function (like a Gaussian kernel):

$$w(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

2. **Compute the Laplacian matrix L:**
   - D: degree matrix (diagonal matrix with node degrees)
   - A: adjacency matrix

$$L = D - A$$

3. **Eigen decomposition:**
   - Find the **k smallest eigenvectors** of L.
   - Stack them as rows in a new matrix U.
4. **K-means on eigenvectors:**
   - Use K-means to cluster the rows of U.

Aspect	K-means	Spectral Clustering
Cluster shapes	Spherical	Arbitrary (non-linear)
Distance metric	Euclidean	Graph-based similarity
Dimensionality	Works in original space	Operates in eigenspace
Initialization	Sensitive to start	More robust