AI/ML CheatSheet

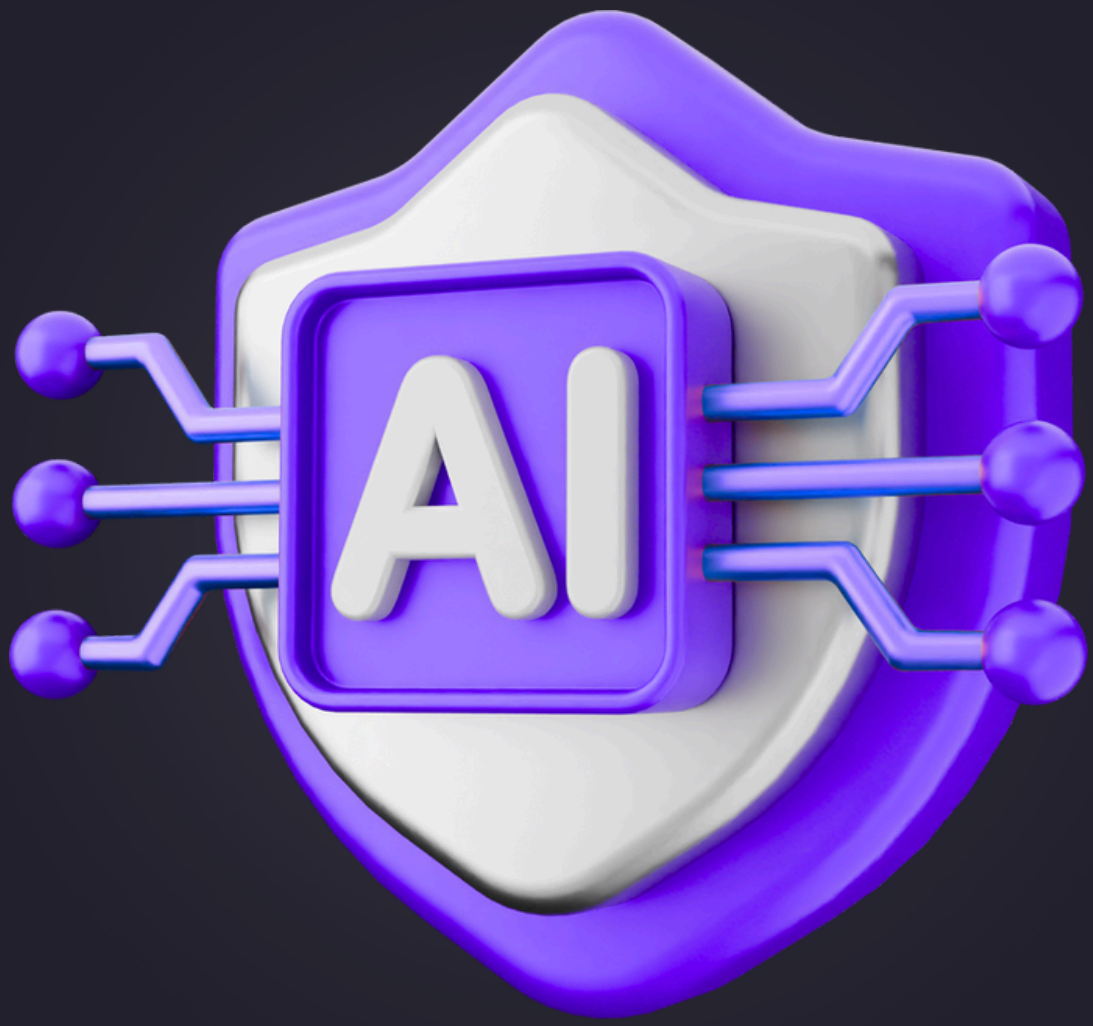


TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

1. INTRODUCTION TO AI AND ML

1.1 What is Artificial Intelligence (AI)?

- Artificial Intelligence (AI) is the ability of machines or computer programs to perform tasks that typically require human intelligence. These tasks can include understanding language, recognizing patterns, solving problems, and making decisions.

Simple explanation:

- AI is when machines are made smart enough to think and act like humans.

Examples:

- Voice assistants like Alexa
- Image recognition systems
- Chatbots
- Self-driving cars

1.2 Types of AI: Narrow AI vs. General AI

Narrow AI (Weak AI):

- Designed to perform one specific task
- Cannot do anything beyond its programming
- Examples: Email spam filters, facial recognition, recommendation systems

General AI (Strong AI):

- Still under research and development
- Can learn and perform any intellectual task a human can do
- Would have reasoning, memory, and decision-making abilities similar to a human

1.3 What is Machine Learning (ML)?

- Machine Learning is a subfield of AI that allows machines to learn from data and improve their performance over time without being explicitly programmed.

Simple explanation:

- Instead of writing rules for everything, we give the machine data, and it figures out the rules on its own.

Example:

- A machine learns to identify spam emails by studying thousands of examples.

1. INTRODUCTION TO AI AND ML

1.4 Supervised vs. Unsupervised vs. Reinforcement Learning

Supervised Learning:

- The training data includes both input and the correct output (labels)
- The model learns by comparing its output with the correct output
- Example: Predicting house prices based on features like size, location, etc.

Unsupervised Learning:

- The data has no labels
- The model tries to find patterns or groupings in the data
- Example: Grouping customers based on purchasing behavior

Reinforcement Learning:

- The model learns through trial and error
- It receives rewards or penalties based on its actions
- Example: A robot learning to walk or a program learning to play chess

1.5 Key Terminologies

Model:

- A program or function that makes predictions or decisions based on data.

Feature:

- An input variable used in making predictions (e.g., age, income, temperature).

Target:

- The value the model is trying to predict (e.g., house price, spam or not).

Algorithm:

- A step-by-step method or set of rules used to train the model.

Training:

- The process of teaching the model using a dataset.

Testing:

- Evaluating the trained model on new, unseen data to measure its accuracy.

1.6 Applications of AI and ML

- Recommendation systems (YouTube, Amazon, Netflix)
- Fraud detection in banking
- Language translation
- Healthcare diagnosis
- Self-driving vehicles
- Stock market prediction
- Chatbots and customer support
- Social media content moderation

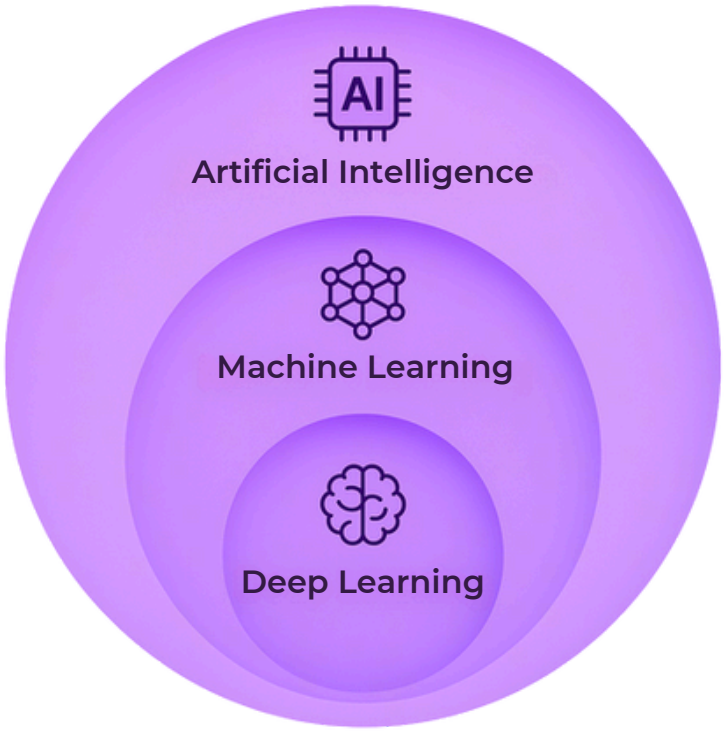
1. INTRODUCTION TO AI AND ML

1.7 Difference Between AI, ML, and Deep Learning (DL)

Term	Description
Artificial Intelligence (AI)	The overall field focused on creating intelligent machines
Machine Learning (ML)	A subset of AI where machines learn from data
Deep Learning (DL)	A specialized type of ML that uses neural networks inspired by the human brain

Visual analogy:

- AI is the broader concept, ML is a part of AI, and DL is a part of ML.



2. MATHEMATICS FOR ML/AI

Mathematics is the foundation of AI and Machine Learning. It helps us understand how algorithms work under the hood and how to fine-tune models for better performance.

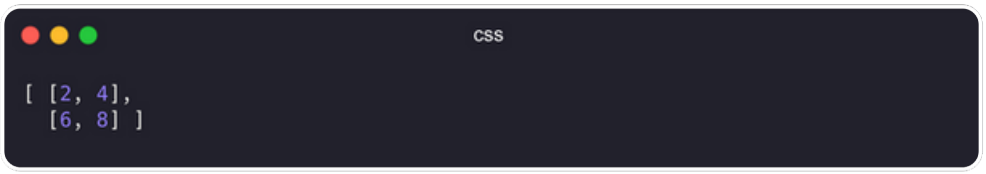
2.1 Linear Algebra

- Linear Algebra deals with numbers organized in arrays and how these arrays interact. It is used in almost every ML algorithm.

2.1.1 Vectors, Matrices, and Tensors

- Vector: A 1D array of numbers. Example: [3, 5, 7]
- Used to represent features like height, weight, age.
- Matrix: A 2D array (rows and columns).

Example:



- Used to store datasets or model weights.

- Tensor: A generalization of vectors and matrices to more dimensions (3D or higher).
- Example: Used in deep learning models like images (3D tensor: width, height, color channels).

2.1.2 Matrix Operations

- Addition/Subtraction: Add or subtract corresponding elements of two matrices.
- Multiplication: Used to combine weights and inputs in ML models.
- Transpose: Flip a matrix over its diagonal.
- Dot Product: Fundamental in calculating output in neural networks.

2.1.3 Eigenvalues and Eigenvectors

- Eigenvector: A direction that doesn't change during a transformation.
- Eigenvalue: Tells how much the eigenvector is stretched or shrunk.

These are used in algorithms like Principal Component Analysis (PCA) for dimensionality reduction.

2. MATHEMATICS FOR ML/AI

2.2 Probability and Statistics

- Probability helps machines make decisions under uncertainty, and statistics helps us understand data and model performance.

2.2.1 Mean, Variance, Standard Deviation

- Mean: The average value.
- Variance: How spread out the values are from the mean.
- Standard Deviation: The square root of variance. It measures how much the values vary.

Used to understand the distribution and behavior of features in datasets.

2.2.2 Bayes Theorem

A mathematical formula to calculate conditional probability:

```
P(A|B) = [P(B|A) * P(A)] / P(B)
```

Used in Naive Bayes classifiers for spam detection, document classification, etc.

2.2.3 Conditional Probability

- The probability of one event occurring given that another event has already occurred.

Example:

- Probability that a user clicks an ad given that they are between 20-30 years old.

2.2.4 Probability Distributions

- Normal Distribution: Bell-shaped curve. Common in real-world data like height, exam scores.
- Binomial Distribution: Used for yes/no type outcomes. Example: Flipping a coin 10 times.
- Poisson Distribution: For events happening over a time period. Example: Number of customer calls per hour.

These distributions help in modeling randomness in data.

2. MATHEMATICS FOR ML/AI

2.3 Calculus for Optimization

- Calculus helps in training models by optimizing them to reduce errors.

2.3.1 Derivatives and Gradients

- Derivative: Measures how a function changes as its input changes.
- Gradient: A vector of derivatives that tells the slope of a function in multi-dimensions.

Used to find the direction in which the model should adjust its weights.

2.3.2 Gradient Descent

- An optimization algorithm used to minimize the loss (error) function.

How it works:

- Start with random values
- Calculate the gradient (slope)
- Move slightly in the opposite direction of the gradient
- Repeat until the loss is minimized

Gradient Descent is the core of many training algorithms in ML and DL.

3. DATA PREPROCESSING

Before feeding data into any machine learning model, it must be cleaned, transformed, and prepared. This step is called data preprocessing, and it is one of the most important stages in building accurate ML models.

3.1 Data Cleaning

- Real-world data is often messy. Data cleaning means identifying and fixing errors in the dataset.

3.1.1 Missing Values

- Missing values can be due to incomplete forms, sensor errors, etc.
- Techniques to handle missing data:
  - Remove rows/columns with too many missing values
  - Fill (impute) missing values using:
    - Mean/Median/Mode
    - Forward/Backward fill
    - Predictive models (like KNN)

3.1.2 Outliers

- Outliers are data points that are very different from others.
- They can distort results and reduce model performance.
- Detection methods:
  - Box plot, Z-score, IQR method
- Handling outliers:
  Remove them
  Transform data (e.g., log scaling)
  Cap them (set a maximum/minimum)

3.2 Data Normalization and Standardization

Helps scale numeric data so that features contribute equally to the model.

- Normalization (Min-Max Scaling):
  - Scales all values between 0 and 1
  - Formula:

```
(x - min) / (max - min)
```

Used when the data is not normally distributed

- Standardization (Z-score Scaling):
- Centers the data around mean = 0 and standard deviation = 1
- Formula:

```
(x - mean) / std
```

Used when data is normally distributed

3. DATA PREPROCESSING

3.3 Encoding Categorical Variables

- ML models work with numbers, not text. Categorical data needs to be converted into numerical form.

Label Encoding:

- Assigns each unique category a number.

Example:

```
Red → 0, Blue → 1, Green → 2
```

One-Hot Encoding:

- Creates new binary columns for each category.

Example:

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

- Label encoding is good for ordinal data (ranked), while one-hot encoding is best for nominal data (non-ranked).

3.4 Feature Scaling

- Ensures features are on the same scale so the model can learn effectively.

Min-Max Scaling:

- Scales features between 0 and 1.
- Good for algorithms like KNN, neural networks.

Z-score Scaling (Standardization):

- Useful for models that assume normality, like linear regression or logistic regression.

Scaling is crucial for models that use distance or gradient-based optimization.

3.5 Feature Engineering

- Creating new features or modifying existing ones to improve model performance.

Polynomial Features:

- Create new features by raising existing features to a power.
- Example: From x, create x^2, x^3

Binning (Discretization):

- Convert continuous data into categories.
- Example: Age → [0–18], [19–35], [36–60], 60+

Feature engineering can significantly boost the predictive power of a model.

3. DATA PREPROCESSING

3.6 Handling Imbalanced Data

- In classification, if one class dominates (e.g., 95% non-fraud, 5% fraud), models may ignore the minority class. This is called class imbalance.

SMOTE (Synthetic Minority Oversampling Technique):

- Creates synthetic examples of the minority class using nearest neighbors.

Undersampling:

- Remove some samples from the majority class.

Oversampling:

- Duplicate or generate more samples of the minority class.

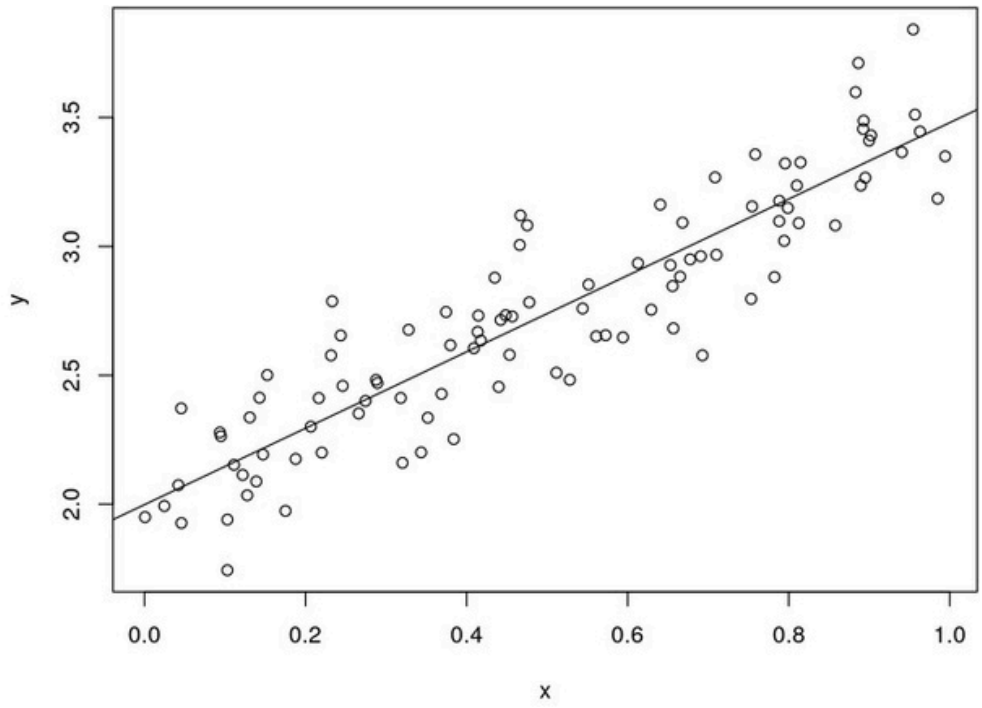
Balancing data improves the ability of the model to correctly predict both classes.

4. SUPERVISED LEARNING ALGORITHMS

Supervised learning uses labeled data, meaning the model learns from input-output pairs (X → y). The algorithm tries to map inputs (features) to correct outputs (targets/labels).

4.1 Linear Regression

- Used for predicting continuous values (e.g., predicting house price, temperature).



4.1.1 Simple vs. Multiple Linear Regression

- Simple Linear Regression: One input (X) to predict one output (Y).
  - Example: Predicting salary from years of experience.
- Multiple Linear Regression: Multiple inputs (X1, X2, ..., Xn).
  - Example: Predicting price based on area, location, and age.

4.1.2 Gradient Descent and Normal Equation

- Gradient Descent: Iterative method to minimize error (cost function).
- Normal Equation: Direct way to find weights using linear algebra:

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

- Works for small datasets.

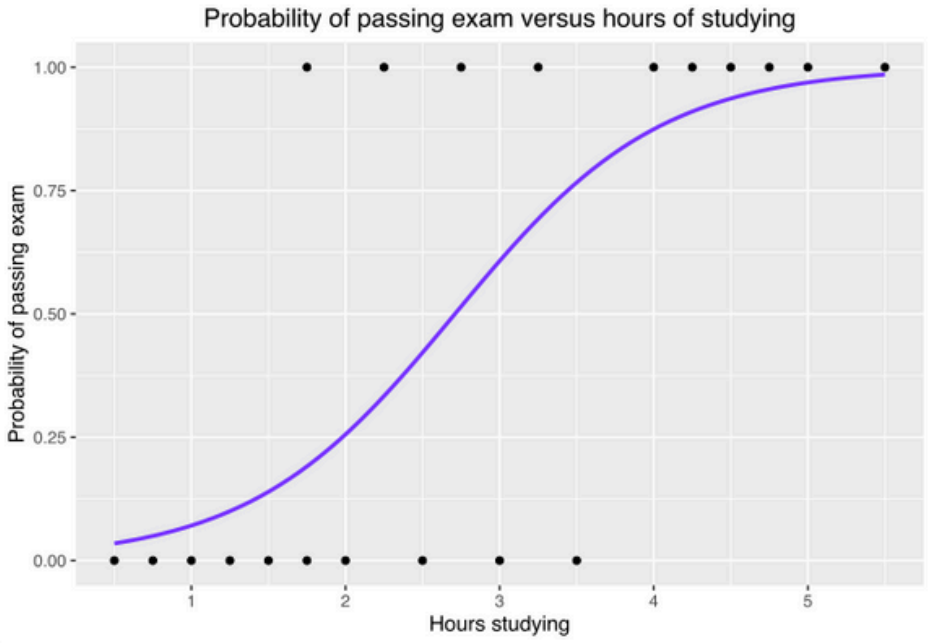
4. SUPERVISED LEARNING ALGORITHMS

4.1.3 Regularization (L1, L2)

- Prevents overfitting by adding a penalty:
  - L1 (Lasso): Can reduce coefficients to 0 (feature selection).
  - L2 (Ridge): Shrinks coefficients but doesn't make them 0.

4.2 Logistic Regression

- Used for classification problems (e.g., spam vs. not spam).



4.2.1 Binary vs. Multiclass Classification

- Binary: 2 outcomes (e.g., 0 or 1)
- Multiclass: More than 2 classes (handled using One-vs-Rest or Softmax)

4.2.2 Sigmoid and Cost Function

- Sigmoid Function: Converts outputs to values between 0 and 1.

```
sigmoid(z) = 1 / (1 + e⁻ᶻ)
```

- Cost Function: Log loss used to measure prediction error.

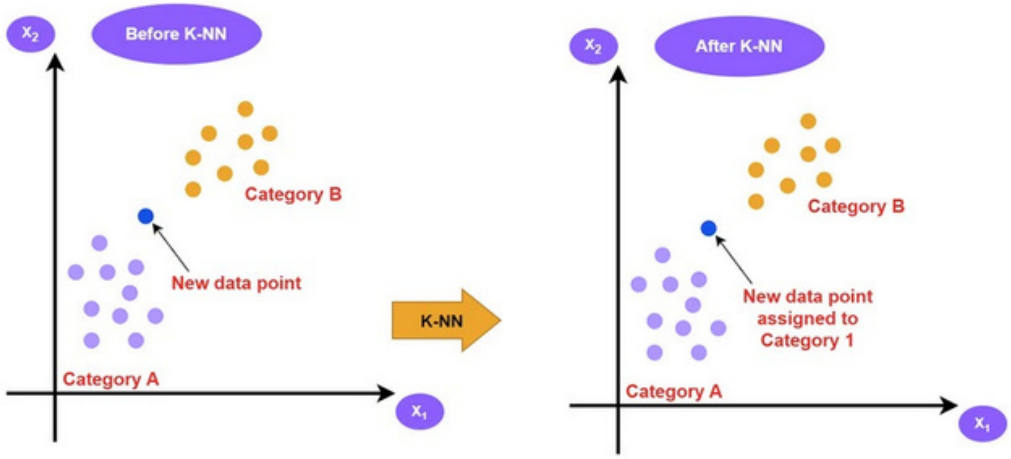
4.2.3 Regularization

- L1 and L2 regularization help prevent overfitting in logistic regression as well.

4. SUPERVISED LEARNING ALGORITHMS

4.3 K-Nearest Neighbors (KNN)

- A simple classification (or regression) algorithm that uses proximity.



4.3.1 Distance Metrics

- Euclidean Distance: Straight line between two points.
- Manhattan Distance: Sum of absolute differences.

4.3.2 Choosing K

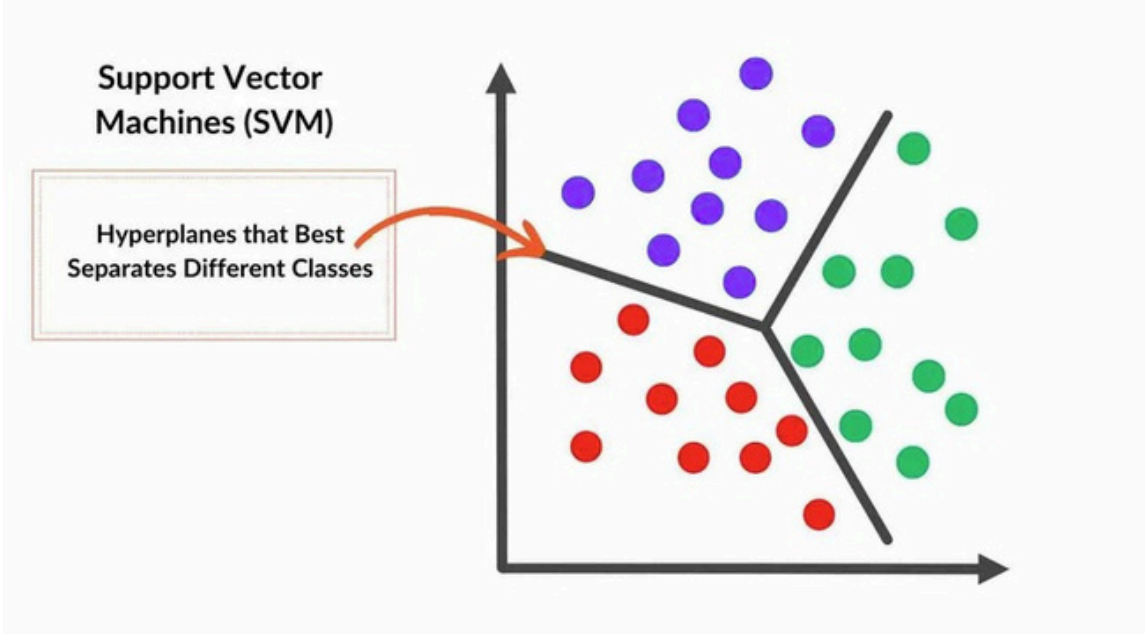
- K is the number of neighbors to consider.
- Too low K → sensitive to noise
- Too high K → model becomes less flexible

4.3.3 Advantages & Disadvantages

- Simple and easy to implement
- Slow for large datasets, sensitive to irrelevant features

4.4 Support Vector Machines (SVM)

- Powerful classification model for small to medium-sized datasets.



4. SUPERVISED LEARNING ALGORITHMS

4.4.1 Hyperplanes and Margins

- SVM finds the best hyperplane that separates data with maximum margin.

4.4.2 Linear vs. Non-Linear SVM

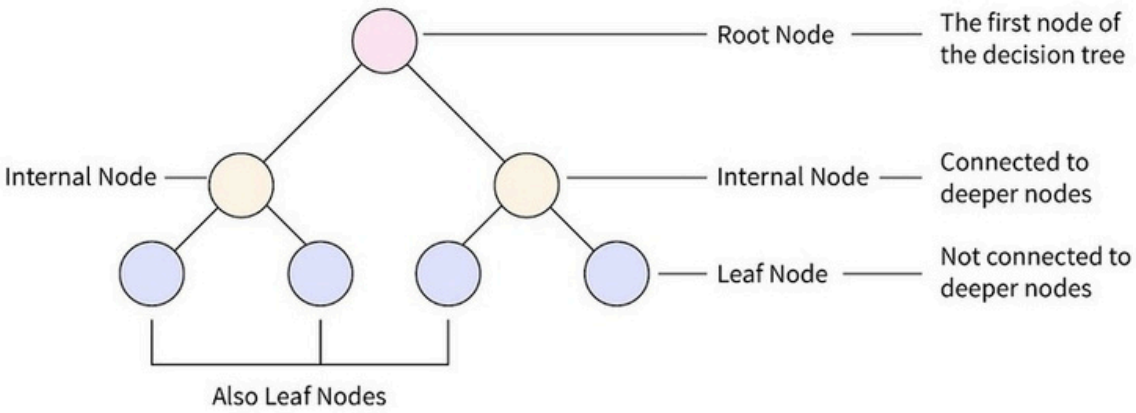
- Linear SVM: Works when data is linearly separable.
- Non-linear SVM: Uses kernel trick for complex datasets.

4.4.3 Kernel Trick

- Transforms data into higher dimensions to make it separable.
  - Common kernels: RBF (Gaussian), Polynomial, Sigmoid

4.5 Decision Trees

- Tree-like structure used for classification and regression.



4.5.1 Gini Impurity and Entropy

- Measures how pure a node is:
  - Gini Impurity: Probability of misclassification.
  - Entropy: Measure of randomness/information.

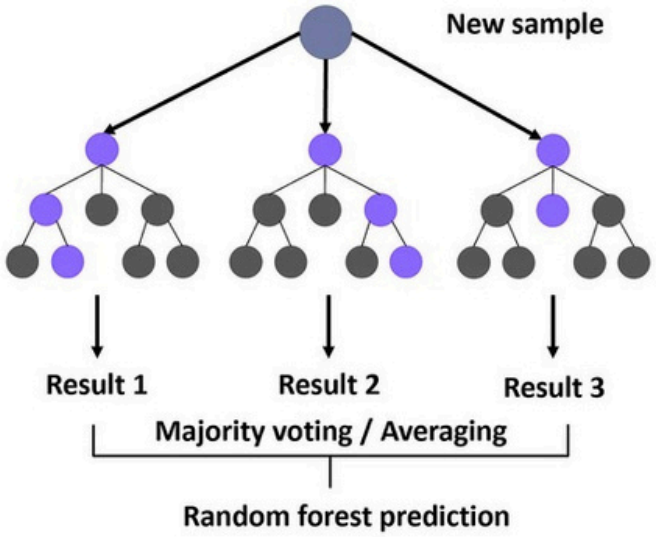
4.5.2 Overfitting and Pruning

- Overfitting: Tree memorizes training data.
- Pruning: Removes unnecessary branches to reduce overfitting.

4. SUPERVISED LEARNING ALGORITHMS

4.6 Random Forest

- An ensemble of decision trees to improve accuracy and reduce overfitting.



4.6.1 Bootstrapping

- Randomly selects subsets of data to train each tree.

4.6.2 Bagging

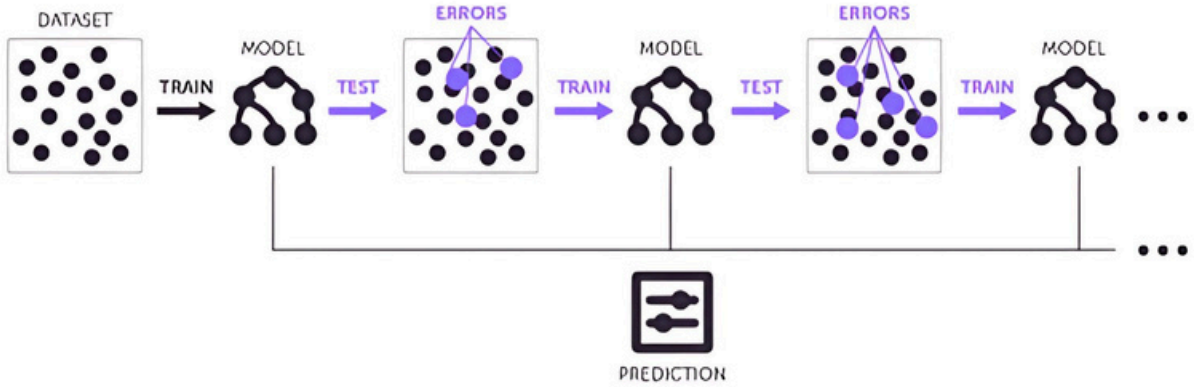
- Combines predictions of multiple trees (majority vote or average).

4.6.3 Feature Importance

- Measures which features contribute most to model prediction.

4.7 Gradient Boosting Machines (GBM)

- Boosting is an ensemble method where models are trained sequentially.



4.7.1 XGBoost, LightGBM, CatBoost

- Advanced boosting libraries:
  - XGBoost: Popular, fast, and accurate
  - LightGBM: Faster, uses leaf-wise growth
  - CatBoost: Handles categorical features automatically

4. SUPERVISED LEARNING ALGORITHMS

4.7.2 Hyperparameter Tuning

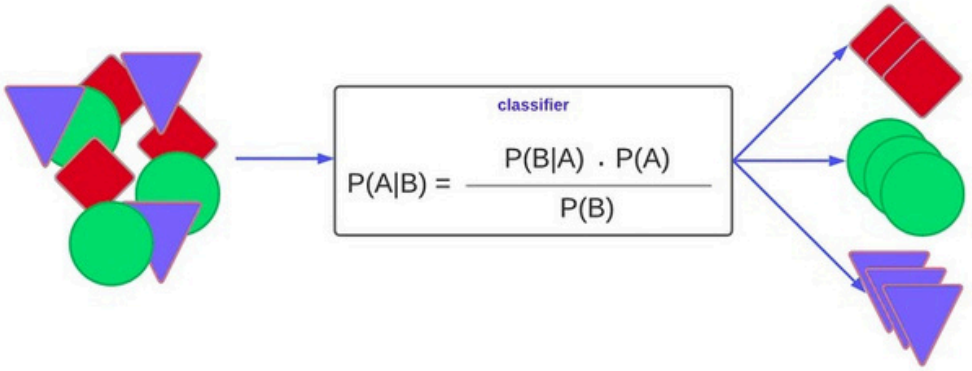
- Adjust parameters like:
  - Learning rate
  - Number of estimators (trees)
  - Max depth
- Tools: GridSearchCV, RandomSearchCV

4.7.3 Early Stopping

- Stops training if the model stops improving on validation set.

4.8 Naive Bayes

- Probabilistic classifier based on Bayes' Theorem and strong independence assumption.



4.8.1 Gaussian, Multinomial, Bernoulli

- Gaussian NB: For continuous features (assumes normal distribution)
- Multinomial NB: For text data, counts of words
- Bernoulli NB: For binary features (0/1)

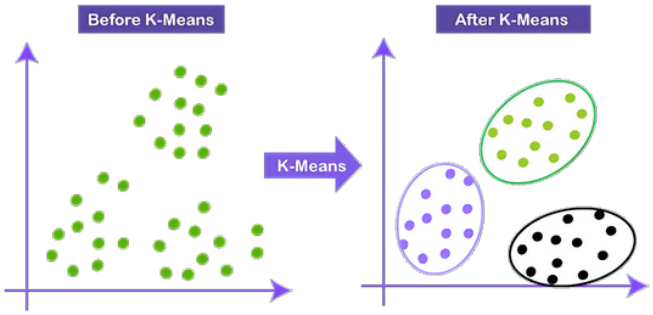
4.8.2 Assumptions and Applications

- Assumes all features are independent (rarely true but still works well)
- Commonly used in spam detection, sentiment analysis, document classif ication

5. UNSUPERVISED LEARNING ALGORITHMS

Unsupervised learning finds hidden patterns in unlabeled data. Unlike supervised learning, it doesn't rely on labeled outputs (no predefined target).

5.1 K-Means Clustering



5.1.1 Algorithm Overview

- K-Means is a clustering algorithm that divides data into K clusters based on similarity.
- It works by:
  a. Selecting K random centroids.
  b. Assigning each point to the nearest centroid.
  c. Updating the centroid to the mean of its assigned points.
  d. Repeating steps 2–3 until the centroids stop changing.

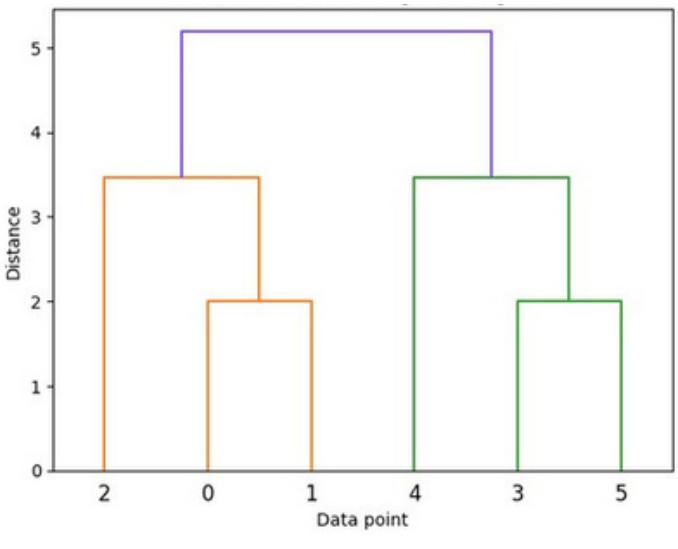
5.1.2 Elbow Method

- Used to choose the optimal number of clusters (K).
- Plot the number of clusters (K) vs. Within-Cluster-Sum-of-Squares (WCSS).
- The point where the WCSS curve bends (elbow) is the best K.

5.1.3 K-Means++ Initialization

- Improves basic K-Means by smartly selecting initial centroids, reducing the chance of poor clustering.
- Starts with one random centroid, then selects the next ones based on distance from the current ones (probabilistically).

5.2 Hierarchical Clustering



5. UNSUPERVISED LEARNING ALGORITHMS

5.2.1 Agglomerative vs. Divisive Clustering

- Agglomerative (bottom-up): Start with each point as its own cluster and merge the closest clusters.
- Divisive (top-down): Start with one large cluster and recursively split it.

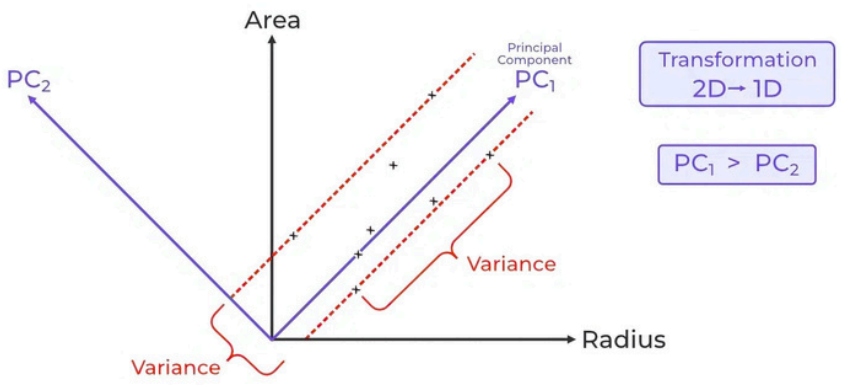
Agglomerative is more commonly used.

5.2.2 Dendrogram and Optimal Cut

- A dendrogram is a tree-like diagram that shows how clusters are formed at each step.
- The height of branches represents the distance between clusters.
- Cutting the dendrogram at a certain height gives the desired number of clusters.

5.3 Principal Component Analysis (PCA)

- PCA is a dimensionality reduction technique used to simplify datasets while retaining most of the important information.



5.3.1 Dimensionality Reduction

- PCA transforms the data into a new coordinate system with fewer dimensions (called principal components).
- Useful for visualization, speeding up algorithms, and avoiding the curse of dimensionality.

5.3.2 Eigenvalue Decomposition

- PCA is based on eigenvectors and eigenvalues of the covariance matrix of the data.
- The eigenvectors define the new axes (principal components).
- The eigenvalues indicate the amount of variance each component captures.

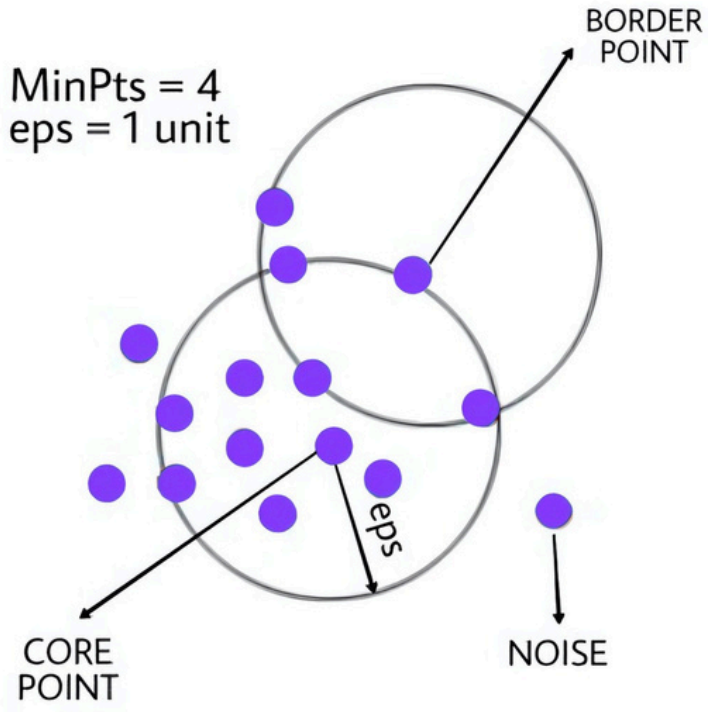
5.3.3 Scree Plot and Explained Variance

- Scree Plot: A plot of eigenvalues to help decide how many components to keep.
- The explained variance ratio shows how much of the data's variance is captured by each component.

5. UNSUPERVISED LEARNING ALGORITHMS

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

- DBSCAN is a density-based clustering algorithm that groups closely packed points and marks outliers as noise.



5.4.1 Density-Based Clustering

- Unlike K-Means, DBSCAN doesn't require specifying the number of clusters.
- Clusters are formed based on dense regions in the data.

5.4.2 Epsilon and MinPts Parameters

- Epsilon (ε): Radius around a point to search for neighbors.
- MinPts: Minimum number of points required to form a dense region.
- Points are classified as:
  - Core Point: Has MinPts within ε.
  - Border Point: Not a core but within ε of a core.
  - Noise: Neither core nor border.

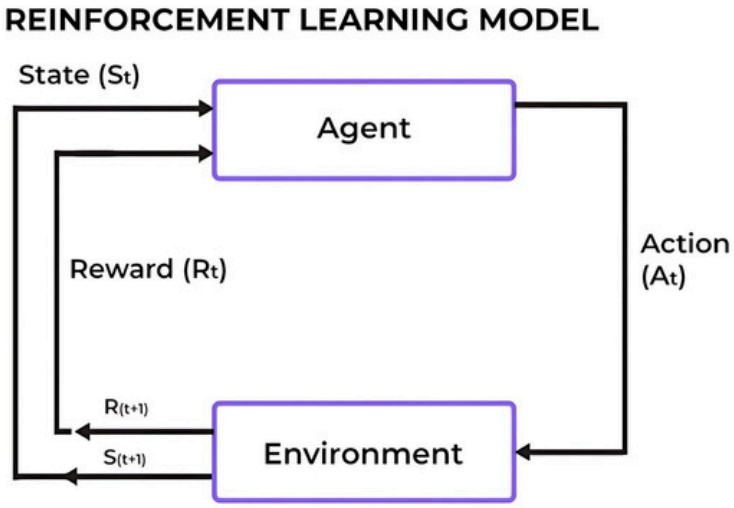
6. REINFORCEMENT LEARNING (RL)

6.1 Introduction to Reinforcement Learning

- Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment. It is inspired by how humans learn from experience — by trial and error.

- The agent performs an action
- The environment responds with a reward
- The agent uses this feedback to learn better actions over time

Unlike supervised learning, RL doesn't rely on labeled data. Instead, it uses rewards or penalties to guide learning.



6.2 Key Concepts

- Agent: The learner or decision maker (e.g., a robot, self-driving car).
- Environment: Everything the agent interacts with (e.g., a maze, a game).
- State: A snapshot of the current situation (e.g., position in a maze).
- Action: A move or decision made by the agent (e.g., turn left).
- Reward: Feedback from the environment (e.g., +1 for reaching goal, -1 for hitting a wall).

The goal of the agent is to maximize cumulative rewards over time.

6.3 Markov Decision Process (MDP)

- RL problems are often modeled as Markov Decision Processes (MDPs).

An MDP includes: S: Set of states A: Set of actions P: Transition probabilities (P(s' | s, a)) R: Reward function γ (gamma): Discount factor (how much future rewards are valued) The "Markov" property means the next state only depends on the current state and action, not on previous ones.

6. REINFORCEMENT LEARNING (RL)

6.4 Q-Learning and Deep Q-Networks (DQN)

Q-Learning:

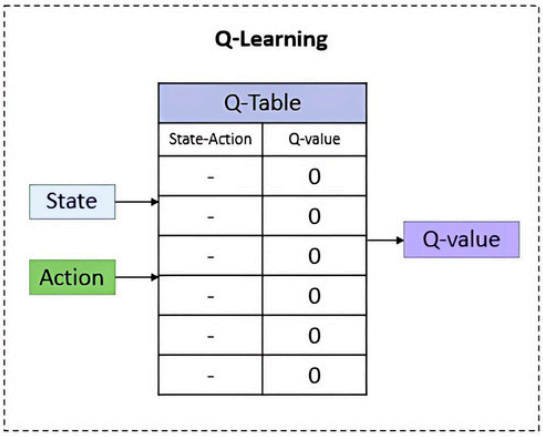
- A model-free algorithm that learns the value (Q-value) of taking an action in a state.
- Uses the formula:

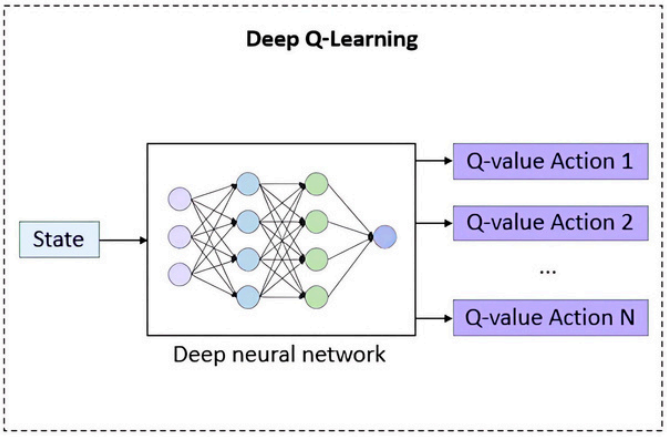
```
Q(s, a) ← Q(s, a) + α [r + γ max Q(s', a') - Q(s, a)]
```

- Where:
  - s: current state
  - a: action
  - r: reward
  - s': next state
  - α: learning rate
  - γ: discount factor

Deep Q-Networks (DQN):

- When the state/action space is too large, a neural network is used to approximate Q-values.
- Combines Q-Learning with Deep Learning.
- Used in Atari games and robotics.





6.5 Policy Gradients and Actor-Critic Methods

Policy Gradients:

- Instead of learning value functions, learn the policy directly (probability distribution over actions).
- Use gradient ascent to improve the policy based on the reward received.

Actor-Critic Methods:

- Combine the best of both worlds:
  - Actor: chooses the action
  - Critic: evaluates how good the action was (value function)
- More stable and efficient than pure policy gradients.

6. REINFORCEMENT LEARNING (RL)

6.6 Exploration vs. Exploitation

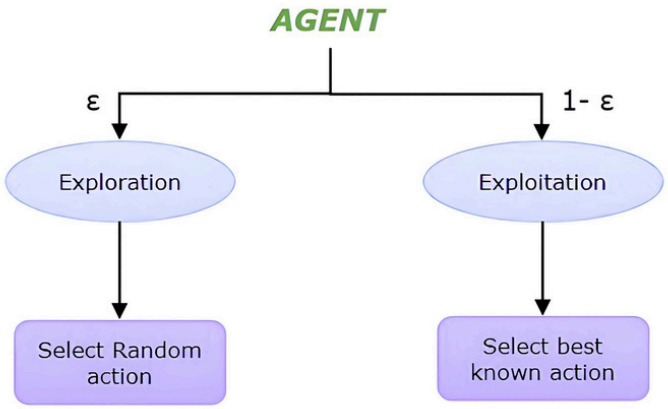
- Exploration: Trying new actions to discover their effects (important early in training).
- Exploitation: Choosing the best-known action for maximum reward.

RL must balance both:

- Too much exploration = slow learning
- Too much exploitation = stuck in local optimum

Common strategy: ε-greedy

- Choose a random action with probability ε
- Otherwise, choose the best-known action



7. NEURAL NETWORKS & DEEP LEARNING

7.1 Introduction to Neural Networks

- A neural network is a computer model inspired by the human brain. It consists of neurons (nodes) organized in layers, capable of learning patterns from data.

7.1.1 Perceptrons

- The perceptron is the simplest type of neural network, with:
  - Inputs → Weights → Summation → Activation Function → Output
- It's like a yes/no decision maker (binary classification).

7.1.2 Activation Functions

These introduce non-linearity, allowing the network to learn complex functions:

- Sigmoid: Outputs between 0 and 1. Good for probability-based outputs.
- ReLU (Rectified Linear Unit): Most popular. Fast, reduces vanishing gradient.
  - ReLU(x) = max(0, x)
- Tanh: Like sigmoid but outputs between -1 and 1.

7.1.3 Forward Propagation and Backpropagation

- Forward Propagation: Input data flows through the network to produce an output.
- Backpropagation: Calculates the error and updates weights using gradients (from loss function).

This is how neural networks learn from data.

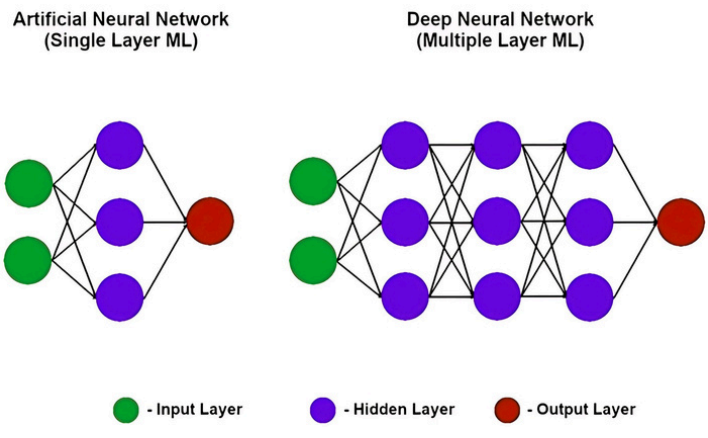
7.1.4 Loss Functions

They measure how far off the prediction is from the actual result.

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

7.2 Deep Neural Networks (DNN)

- A Deep Neural Network has multiple hidden layers between input and output.



7. NEURAL NETWORKS & DEEP LEARNING

7.2.1 Architecture and Layers

- Input Layer: Where the data comes in
- Hidden Layers: Where computation happens (many neurons per layer)
- Output Layer: Final predictions

7.2.2 Training Process and Optimizers

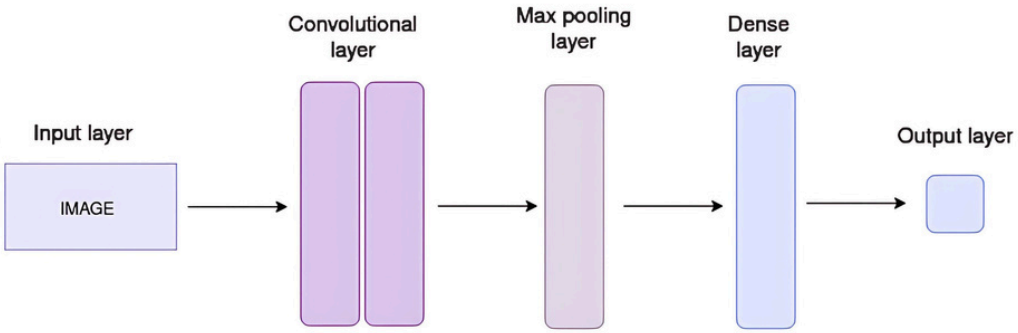
- During training, the network:
  - Makes predictions
  - Calculates the loss
  - Updates weights via optimizers like:
    - SGD (Stochastic Gradient Descent)
    - Adam (adaptive learning rate)
    - RMSProp

7.2.3 Overfitting and Regularization

- Overfitting happens when the model learns noise instead of patterns.
- Regularization techniques help:
  - Dropout: Randomly turns off neurons during training.
  - L2 Regularization: Penalizes large weights (weight decay).

7.3 Convolutional Neural Networks (CNN)

- CNNs are specialized for image data.



7.3.1 Convolutional Layers, Pooling Layers

- Convolutional Layers: Apply filters to detect features (edges, corners).
- Pooling Layers: Reduce size of feature maps (e.g., Max Pooling).

7.3.2 Filters/Kernels and Strides

- Filters: Small matrix to slide over input to extract features.
- Strides: Step size of the filter as it moves.

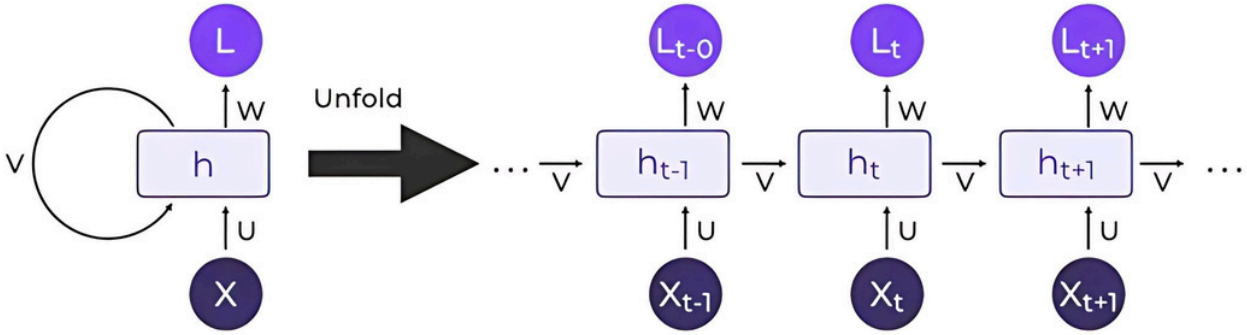
7.3.3 Applications

- Image Classification
- Face Recognition
- Object Detection

7. NEURAL NETWORKS & DEEP LEARNING

7.4 Recurrent Neural Networks (RNN)

- RNNs are designed for sequential data (time series, text, etc.).



7.4.1 Basic RNN vs. LSTM vs. GRU

- Basic RNN: Loops through time steps but suffers from memory issues.
- LSTM (Long Short-Term Memory): Handles long dependencies well.
- GRU (Gated Recurrent Unit): Similar to LSTM but faster.

7.4.2 Time-Series Prediction and NLP Applications

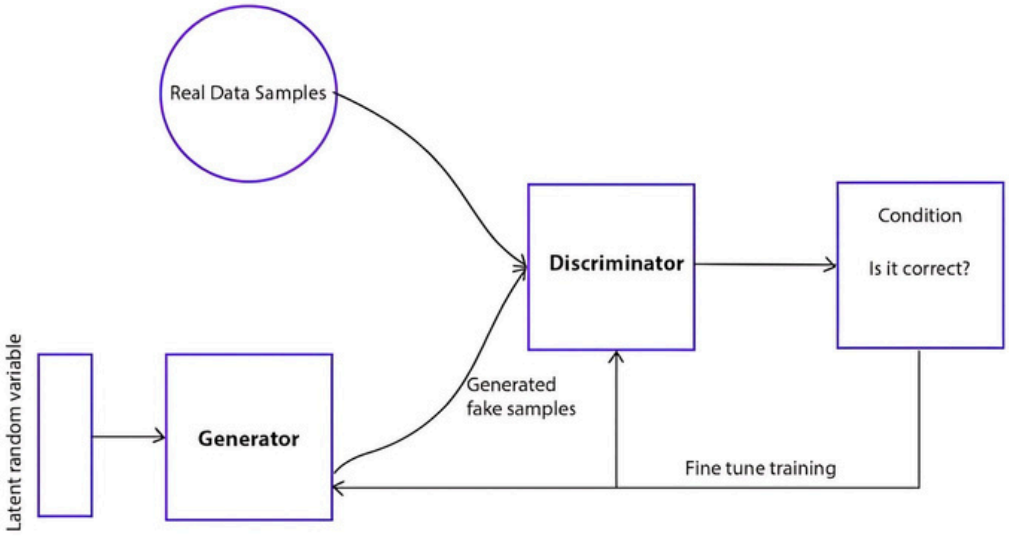
- Predict stock prices, weather, or language sequences.
- Used in chatbots, translation, and speech recognition.

7.4.3 Vanishing and Exploding Gradients

- Problem during training of RNNs where gradients shrink (vanish) or explode.
- LSTM and GRU solve this with gate mechanisms.

7.5 Generative Adversarial Networks (GANs)

- GANs are powerful models for generating new data.



7. NEURAL NETWORKS & DEEP LEARNING

7.5.1 Generator and Discriminator

- Generator: Creates fake data
- Discriminator: Tries to distinguish real from fake data

They compete with each other (like a forger and a detective).

7.5.2 Training Process

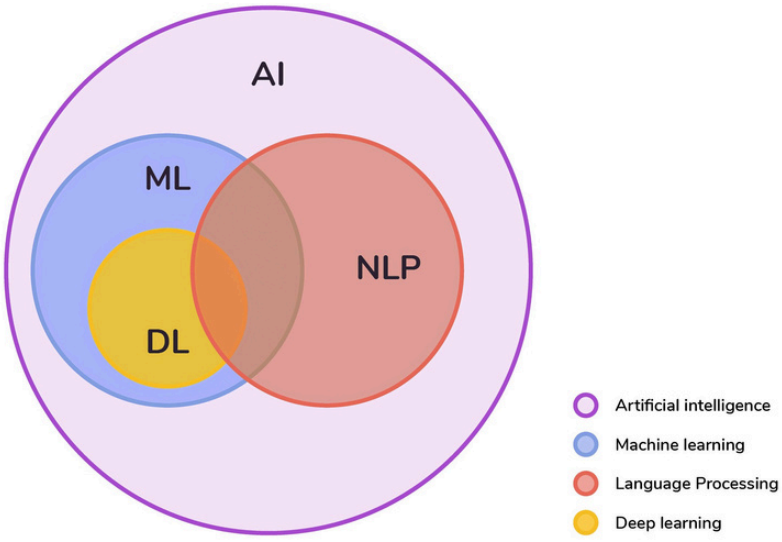
- Generator tries to fool the discriminator
- Discriminator improves to detect fakes
- They both improve over time — leading to realistic generated data

7.5.3 Applications

- Image Generation (e.g., fake faces)
- Art and Style Transfer
- Data Augmentation for training other ML models

8. NATURAL LANGUAGE PROCESSING (NLP)

NLP helps computers understand, interpret, and generate human language. It's widely used in applications like chatbots, translation tools, and voice assistants.



8.1 Text Preprocessing

- Before using text in machine learning models, we need to clean and convert it into a format the computer understands.

8.1.1 Tokenization

- Breaking text into smaller parts like words or sentences.

Example: "I love AI" → ["I", "love", "AI"]

8.1.2 Stopwords

- Removing common words that do not add much meaning (like “is”, “the”, “and”).

8.1.3 Stemming

- Cutting words down to their root form.

Example: “playing”, “played” → “play”

8.1.4 Lemmatization

- Similar to stemming but uses grammar to find the proper base word.

Example: “better” → “good”

8.1.5 Bag of Words (BoW)

- Converts text into numbers based on word counts in a document.

8.1.6 TF-IDF

- Gives importance to words that appear often in one document but not in others. Helps identify keywords.

8. NATURAL LANGUAGE PROCESSING (NLP)

8.2 Word Embeddings

- Word embeddings turn words into vectors (numbers) so that a machine can understand their meaning and context.

8.2.1 Word2Vec

- A model that learns how words are related based on their surrounding words.

8.2.2 GloVe

- Learns word meanings by looking at how often words appear together.

8.2.3 FastText

- Similar to Word2Vec but also looks at parts of words, which helps with unknown words.

8.2.4 Sentence Embeddings (BERT, RoBERTa, GPT)

- These models convert full sentences into vectors. They understand context much better than older models.

8.3 Sequence Models

- These models are good for processing data where order matters, like text.

8.3.1 RNN (Recurrent Neural Networks)

- Good for learning from sequences, such as sentences.

8.3.2 LSTM (Long Short-Term Memory)

- An advanced RNN that remembers long-term information.

8.3.3 GRU (Gated Recurrent Unit)

- A simpler version of LSTM that works faster and often just as well.

8.4 Transformer Architecture

- Transformers are a powerful model used in almost all modern NLP systems.

8.4.1 Self-Attention Mechanism

- This allows the model to focus on important words in a sentence, no matter where they appear.

8.4.2 Encoder-Decoder Model

- Used in tasks like translation where the model reads input (encoder) and generates output (decoder).

8.4.3 Examples:

- BERT: Great for understanding text.
- GPT: Great for generating text.
- T5: Can both understand and generate text for many tasks.

8. NATURAL LANGUAGE PROCESSING (NLP)

8.5 Text Classification

- Classify text into categories.

Examples:

- Sentiment Analysis: Is a review positive or negative?
- Named Entity Recognition (NER): Find names, places, dates, etc. in text.

8.6 Language Generation

- Generate new text from existing input.

8.6.1 Text Summarization

- Shortens a long document while keeping important points.

8.6.2 Machine Translation

- Translates text from one language to another (like English to Hindi).

9. MODEL EVALUATION AND METRICS

Model evaluation helps us check how well our machine learning models are performing. We use different metrics depending on whether it's a classification or regression problem.

9.1 Classification Metrics

- Used when your model predicts categories or classes (e.g., spam or not spam).

9.1.1 Accuracy

- How often the model is correct.

Formula: (Correct Predictions) / (Total Predictions)

9.1.2 Precision

- Out of all predicted positives, how many were actually positive?

Used when false positives are costly.
Formula: TP / (TP + FP)

9.1.3 Recall (Sensitivity)

- Out of all actual positives, how many were predicted correctly?

Used when missing positives is costly.
Formula: TP / (TP + FN)

9.1.4 F1-Score

- Balance between precision and recall.

Formula: 2 * (Precision * Recall) / (Precision + Recall)

9.1.5 Confusion Matrix

- A table showing True Positives, False Positives, False Negatives, and True Negatives.

9.1.6 ROC Curve (Receiver Operating Characteristic)

- Shows the trade-off between True Positive Rate and False Positive Rate.

9.1.7 AUC (Area Under the Curve)

- Measures the entire area under the ROC curve.

Higher AUC = better performance.

9. MODEL EVALUATION AND METRICS

9.2 Regression Metrics

- Used when the model predicts continuous values (like house price, temperature).

9.2.1 Mean Absolute Error (MAE)

- Average of the absolute errors.
- Easy to understand.

9.2.2 Mean Squared Error (MSE)

- Average of squared errors.
- Penalizes large errors more than MAE.

9.2.3 R-Squared (R^2)

- Explains how much variance in the output is explained by the model.
- Ranges from 0 to 1 (higher is better).

9.2.4 Adjusted R-Squared

- Like R^2, but adjusts for the number of predictors (features).
- Useful when comparing models with different numbers of features.

9.3 Cross-Validation

- Used to test model performance on different splits of the data.

9.3.1 K-Fold Cross-Validation

- Split data into k equal parts. Train on k-1 and test on the remaining part. Repeat k times.

9.3.2 Leave-One-Out Cross-Validation (LOOCV)

- A special case of K-Fold where k = number of data points. Very slow but thorough.

9.3.3 Stratified K-Fold

- Same as K-Fold, but keeps the ratio of classes the same in each fold.
- Useful for imbalanced datasets.

9. MODEL EVALUATION AND METRICS

9.4 Hyperparameter Tuning

- Hyperparameters are settings that control how a model learns (like learning rate, depth of a tree, etc.).

9.4.1 Grid Search

- Tests all combinations of given hyperparameter values.

9.4.2 Random Search

- Randomly selects combinations. Faster than Grid Search.

9.4.3 Bayesian Optimization

- Uses past results to pick the next best combination. Smart and efficient.

10. ADVANCED TOPICS

These are modern machine learning methods used in advanced real-world applications such as chatbots, recommendation systems, self-driving cars, and privacy-focused AI.

10.1 Transfer Learning

- Instead of training a model from scratch, we use a model that has already been trained on a large dataset and apply it to a new, similar task.

Pre-trained Models

- These are models trained on huge datasets.

Examples:

- VGG, ResNet – for images
- BERT – for text

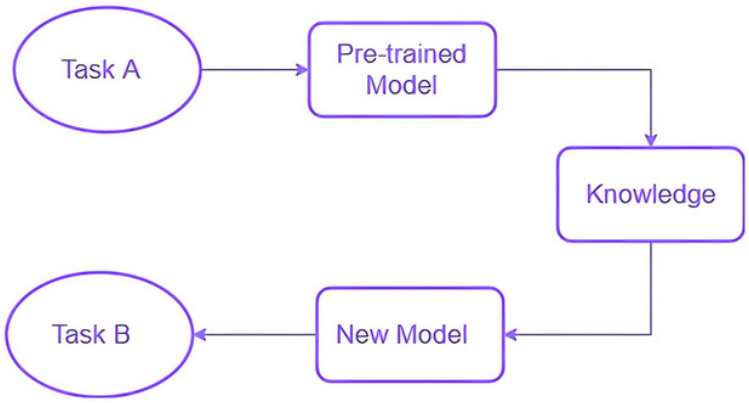
Fine-Tuning

- Slightly updating the pre-trained model using your own smaller dataset.

Feature Extraction

- Using the pre-trained model to extract useful features and then using those features for your own model or task.

Benefit: Saves training time and works well even with limited data.



10.2 Attention Mechanism

- This helps the model decide which parts of the input data are most important.

Self-Attention

- Every part of the input focuses on every other part to understand context better.
- Used in NLP (Natural Language Processing) and transformers.

Multi-Head Attention

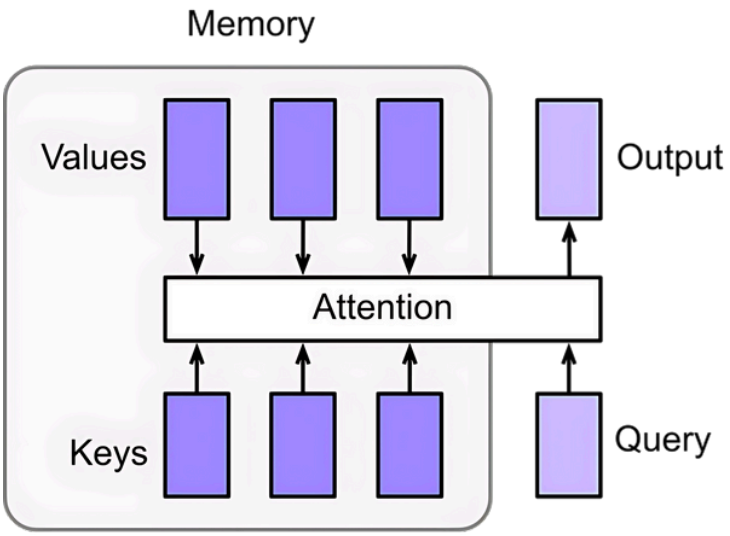
- Applies attention multiple times in parallel to capture different relationships within the data.

Applications

- In NLP: translation, summarization, question-answering
- In vision: image recognition with Vision Transformers

10. ADVANCED TOPICS

10.2 Attention Mechanism



10.3 Reinforcement Learning in Deep Learning

- Combining deep learning with reinforcement learning for decision-making tasks.

Actor-Critic

Two models work together:

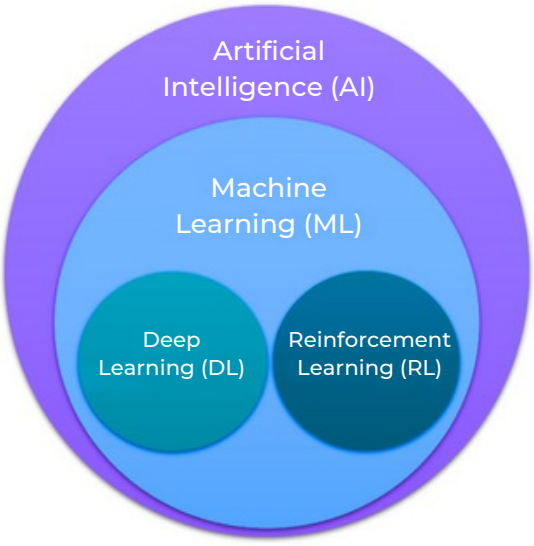
- Actor: selects the best action
- Critic: evaluates how good the action was

A3C (Asynchronous Advantage Actor-Critic)

- Uses multiple agents to learn in parallel, which speeds up learning and increases stability.

PPO (Proximal Policy Optimization)

- An improved and stable way to train reinforcement learning agents. Used in games, robotics, etc.



10. ADVANCED TOPICS

10.4 Federated Learning

- Model training happens across many devices without collecting data in a central server. Each device keeps its data private and only sends model updates.

Distributed Learning Frameworks

- Used when data is spread across users, hospitals, or devices. Examples include Google's keyboard predictions.

Privacy-Preserving ML

- Since data never leaves the device, user privacy is protected. This is useful in healthcare, banking, and personal mobile applications.

11. TOOLS AND LIBRARIES FOR AI/ML

These tools help you build, train, and deploy AI/ML models more efficiently. They provide ready-to-use functions so you don't need to code everything from scratch.

11.1 Python Libraries

- Python is the most popular language in AI/ML. These libraries make your work faster and easier:

NumPy

- Used for numerical computing.
- Supports arrays, matrices, and linear algebra operations.

Pandas

- Used for data manipulation and analysis.
- You can load data, clean it, and analyze it in tabular formats (DataFrames).

Scikit-learn

- A powerful machine learning library.
- Includes ready-to-use models like Linear Regression, SVM, Random Forest, KNN, and more.

TensorFlow & Keras

Used for building deep learning models.

- TensorFlow: low-level control
- Keras: high-level interface, easier to use

PyTorch

- An alternative to TensorFlow, widely used in research and development.
- It's flexible, fast, and dynamic (supports on-the-fly computation graphs).

OpenCV

- Used for computer vision tasks like image processing, object detection, face recognition, etc.

NLTK & SpaCy

Natural Language Processing (NLP) libraries.

- NLTK: good for learning, includes many basic NLP tasks
- SpaCy: industrial-strength NLP, faster and more efficient

11. TOOLS AND LIBRARIES FOR AI/ML

11.2 Cloud Platforms

- Cloud platforms help you run your models on powerful servers without needing your own hardware.

Google Colab

- Free online Jupyter Notebook
- Supports GPU/TPU
- Great for students and beginners

AWS SageMaker

- Amazon's cloud ML platform
- Supports training, tuning, and deploying models at scale
- Used in enterprise-level applications

Azure ML

- Microsoft's machine learning platform
- Integrates well with other Microsoft tools (e.g., Excel, Power BI)
- Provides autoML, drag-and-drop pipelines, and more

12. DEPLOYMENT AND PRODUCTION

12.1 Model Serialization

What it means:

- After training your machine learning model, you save (serialize) it to use later without retraining.

Popular tools:

- Pickle – A Python library to serialize and deserialize Python objects.
- Joblib – Similar to Pickle but better for large NumPy arrays.

Example:

```python
import pickle

# Save model
with open('model.pkl', 'wb') as f:
    pickle.dump(model, f)

# Load model
with open('model.pkl', 'rb') as f:
    model = pickle.load(f)
```

12.2 Flask/Django for Model Deployment

- These are web frameworks that let you expose your model as an API endpoint, so other apps or users can access it via the internet.
- Flask: Lightweight and easier for quick ML model APIs.
- Django: Heavier but better for large web applications with built-in admin, security, and ORM.

Flask Example:

```python
from flask import Flask, request, jsonify
import pickle

app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json
    prediction = model.predict([data['features']])
    return jsonify({'prediction': prediction.tolist()})

if __name__ == '__main__':
    app.run()
```

12. DEPLOYMENT AND PRODUCTION

12.3 Serving Models with TensorFlow Serving, FastAPI

TensorFlow Serving:

- Used to deploy TensorFlow models in production. It supports versioning and high-performance serving with REST/gRPC.

FastAPI:

- A modern, fast (high-performance) framework for building APIs with automatic docs, great for production-grade ML APIs.

FastAPI Example:

```python
from fastapi import FastAPI
import pickle

app = FastAPI()
model = pickle.load(open("model.pkl", "rb"))

@app.post("/predict")
def predict(features: list):
    result = model.predict([features])
    return {"prediction": result.tolist()}
```

12.4 Monitoring and Maintaining Models in Production

- Once your model is live, you need to ensure it continues to perform well.

What to monitor:

- Model accuracy degradation (due to data drift)
- Response time
- Error rates
- System metrics (CPU, memory)

Tools:

- Prometheus + Grafana for system and application monitoring
- MLflow or Evidently.ai for tracking model performance over time

13. PRACTICE & COMMON BEGINNER MISTAKES

13.1 Practice Tasks

To strengthen understanding, here are simple practice tasks for each core concept:

Topic	Mini Practice Task
Linear Regression	Predict house prices using a dataset with features like area, rooms, and location.
Logistic Regression	Build a binary classifier to detect spam emails.
K-Means Clustering	Group customers by shopping behavior (customer segmentation).
PCA	Reduce dimensions in the Iris dataset and visualize clusters.
Decision Trees	Classify if a person will buy a product based on age, income, etc.
CNNs	Classify hand-written digits using the MNIST dataset.
RNNs	Predict the next word in a sentence using a small corpus.
GANs	Generate new handwritten digits after training on MNIST.
NLP	Build a sentiment analysis model on product reviews.

13.2 Common Beginner Mistakes

General ML Mistakes

- Using test data during training
- Not normalizing/scaling features
- Ignoring class imbalance in classification tasks
- Forgetting to check for data leakage
- Not splitting the dataset correctly (Train/Validation/Test)

Neural Network Mistakes

- Using too many/too few layers without tuning
- Choosing wrong activation/loss functions
- Ignoring overfitting (no dropout or regularization)

NLP Mistakes

- Feeding raw text without preprocessing
- Using TF-IDF on small datasets without context
- Confusing stemming with lemmatization

Deployment Mistakes

- Not checking model performance after deployment
- Ignoring real-time latency
- No monitoring/logging in place