Industrial Training on Machine Learning Session 1

Roshni Yadav

What is Machine Learning?

Machine learning (ML) is a rapidly evolving branch of artificial intelligence (AI) that develops algorithms and models to enable computers to perform tasks without explicit instructions. Instead of following predefined rules, ML systems learn from data through experience, allowing them to adapt and improve over time.

A **machine learning model** is a mathematical representation of a real-world process that a computer uses to make predictions or decisions based on input data. It is trained using data, learning patterns, and relationships in that data, which it then applies to make predictions on new, unseen data.

Types of Machine Learning

1. **Supervised Learning:** Involves training a model on labeled data to map inputs to known outputs. Common tasks include:

   - **Classification:** Assigning inputs to categories.
   - **Regression:** Predicting continuous outcomes.

2. **Unsupervised Learning:** Focuses on discovering patterns in unlabeled data without prior outcomes. Key tasks include:

   - **Clustering:** Grouping similar data points.
   - **Association:** Finding rules in data.

3. **Reinforcement Learning**: An agent learns by interacting with its environment, receiving feedback to maximize rewards. It's useful in:

   - **Game AI:** Learning strategies through trial and error.
   - **Robotics:** Enabling autonomous task performance.

Supervised Learning: Classification vs. Regression

Classification and regression are key types of predictive modeling in machine learning, each with distinct purposes.

Classification predicts discrete categories, such as labeling emails as "spam" or "non-spam," and can extend to multi-class problems like categorizing images. Common classification algorithms include Logistic Regression, k-Nearest Neighbors (k-NN), and Support Vector Machines (SVM).

Regression predicts continuous outcomes, such as estimating house prices based on features like location and size. Popular regression methods include Linear Regression, which uses a linear equation to model relationships, and Decision Trees, which can handle non-linear relationships by dividing data into subsets. The choice of method depends on the data characteristics and the specific problem.

LOGISTIC REGRESSION

- **Type**: Supervised (Classification)

- **Purpose**: Used for binary classification tasks, predicting the probability of a categorical outcome.

- **Use Case**: Spam email detection (spam or not spam), Medical diagnosis, Customer Churn.

$$Output = \frac{1}{1+e^{-z}}$$

Logistic Function (Sigmoid Function)
The logistic function is like a special math tool that helps us turn any number into a probability between 0 and 1. This is really useful for deciding if something belongs to one group or another (like yes/no).

➢ **Model**:

- **Probabilities**: It calculates a probability score between 0 and 1. For instance, if it predicts 0.8 for an email, it means there's an 80% chance the email is spam.
- **Threshold**: Usually, a cutoff point of 0.5 is used. If the probability is above 0.5, the prediction is “yes” (or the positive class). If it’s below, the prediction is “no” (or the negative class).

➢ **How It Learns**?

- It uses past data to figure out how different factors (like words in an email or test results) are related to the outcome. For instance, if certain words are more common in spam emails, the model learns this relationship.

➢ **Why Use It?**:

- **Simplicity**: It’s straightforward to understand and implement.
- **Probabilities**: It not only predicts a category but also gives a probability score, which can be useful for understanding confidence levels.

SYNTAX

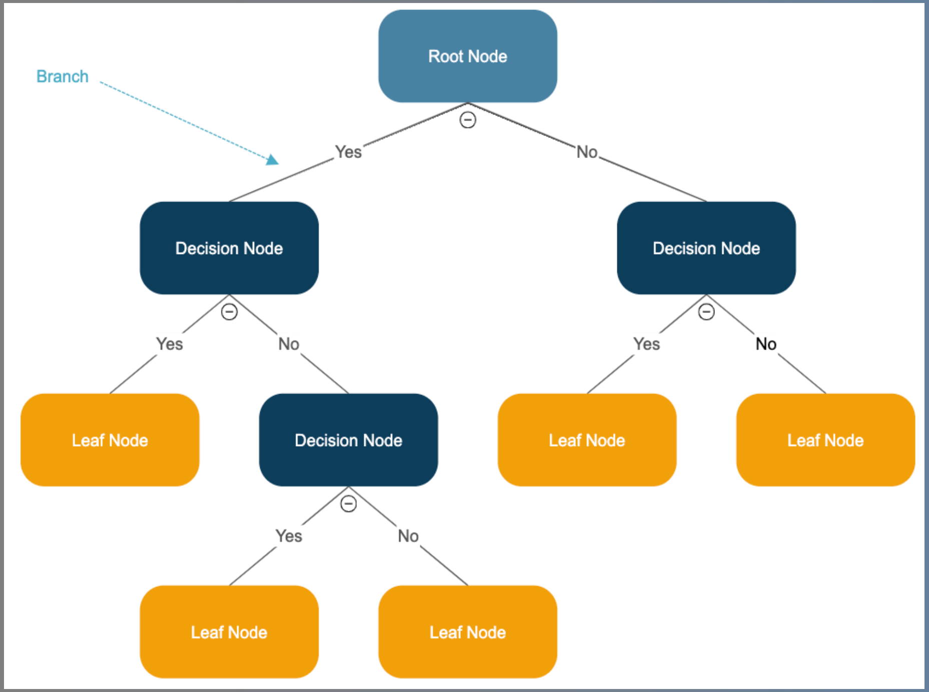
```
#IMPORT LIBRARIES
FROM SKLEARN.LINEAR_MODEL IMPORT LOGISTICREGRESSION
FROM SKLEARN.METRICS IMPORT ACCURACY_SCORE, CONFUSION_MATRIX, CLASSIFICATION_REPORT

# INITIALIZE THE LOGISTIC REGRESSION MODEL
LOGREG = LOGISTICREGRESSION()

# FIT (TRAIN) THE MODEL USING THE TRAINING DATA
LOGREG.FIT(X_TRAIN, Y_TRAIN)

# PREDICT THE TARGET FOR TEST DATA
Y_PRED = LOGREG.PREDICT(X_TEST)

# ACCURACY SCORE
PRINT("ACCURACY:", ACCURACY_SCORE(Y_TEST, Y_PRED))

# CONFUSION MATRIX
PRINT("CONFUSION MATRIX:\N", CONFUSION_MATRIX(Y_TEST, Y_PRED))

# CLASSIFICATION REPORT (PRECISION, RECALL, F1-SCORE)
PRINT("CLASSIFICATION REPORT:\N", CLASSIFICATION_REPORT(Y_TEST, Y_PRED))
```

DECISION TREES

•**Type**: Supervised (Classification/Regression)

•**Purpose**: Splits data into branches to make decisions based on feature values.

•**Use Case**: Predicting whether a patient has a disease based on symptoms.



Decision Trees are like a series of yes/no questions that help you make decisions or predictions based on information you have.

- ➢ **How It Works?**

- ○ **Tree Structure**:
  - ▪ **Root Node**: This is where the decision-making starts. It asks the first question.
  - ▪ **Branches**: These are the paths that show the possible answers to each question.
  - ▪ **Leaf Nodes**: These are the end points of the tree that give the final decision or result.

- ➢ **Limitations:**

- ▪ **Can Overcomplicate**: If not controlled, the tree can become too detailed, fitting the training data too closely and not generalizing well to new data.
- ▪ **Sensitive to Changes**: Small changes in the data can lead to a completely different tree.

SYNTAX

```
#Import libraries
from sklearn.tree import DecisionTreeClassifier # For classification tasks
from sklearn.tree import DecisionTreeRegressor # For regression tasks

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier()

# Fit (train) the model using the training data
clf.fit(X_train, y_train)

# Initialize the Decision Tree Regressor
reg = DecisionTreeRegressor()

# Fit (train) the model using the training data
reg.fit(X_train, y_train)

# Predict the target for test data (classification)
y_pred = clf.predict(X_test)

# Predict the target for test data (regression)
y_pred = reg.predict(X_test)
```

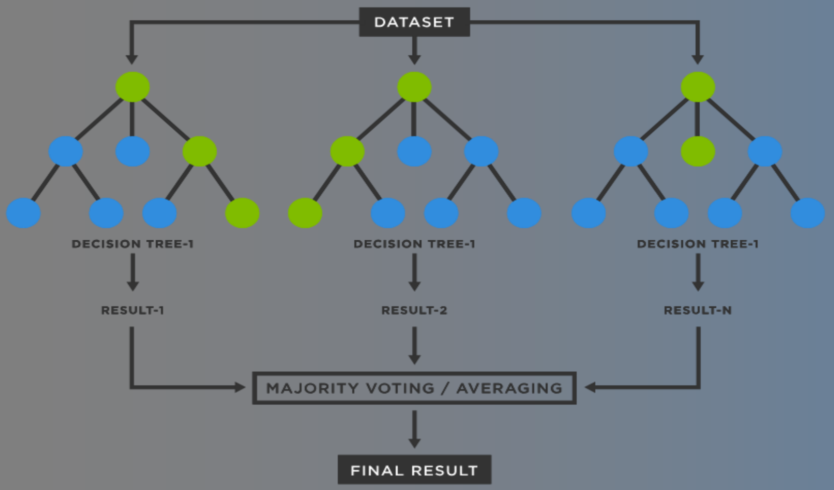
RANDOM FOREST

Type: supervised learning

Purpose: to make more reliable and accurate predictions or classifications by leveraging the collective wisdom of multiple decision trees.

Example: email classification, house prices, etc.

Random forest is an advanced version of decision trees that improves their performance by combining many trees to make better predictions. Imagine it as a group of decision trees working together to make a decision.



➢ **How It Works?**

1. **Forest of Trees**:
    1. **Multiple Trees**: Instead of one decision tree, Random Forest uses many decision trees. Each tree makes its own prediction.
2. **Building the Trees**:
    1. **Random Sampling**: Each tree in the forest is built using a random subset of the data and a random subset of features (questions). This randomness helps make the model more robust.
    2. **Feature Randomness**: During the creation of each tree, only a random subset of features is considered for splitting at each node, reducing correlation between trees.
3. **Making Predictions**:
    1. **Vote or Average**: For classification tasks (e.g., deciding if an email is spam), each tree votes for a class label, and the majority vote determines the final prediction. For regression tasks (e.g., predicting house prices), the final prediction is the average of all trees' predictions.

SYNTAX

```
#Import libraries
from sklearn.ensemble import RandomForestClassifier # For classification tasks
from sklearn.ensemble import RandomForestRegressor # For regression tasks

# Initialize the Random Forest Classifier
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit (train) the model using the training data
rf_clf.fit(X_train, y_train)

# Initialize the Random Forest Regressor
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit (train) the model using the training data
rf_reg.fit(X_train, y_train)

# Predict the target for test data (classification)
y_pred = rf_clf.predict(X_test)

# Predict the target for test data (regression)
y_pred = rf_reg.predict(X_test)
```

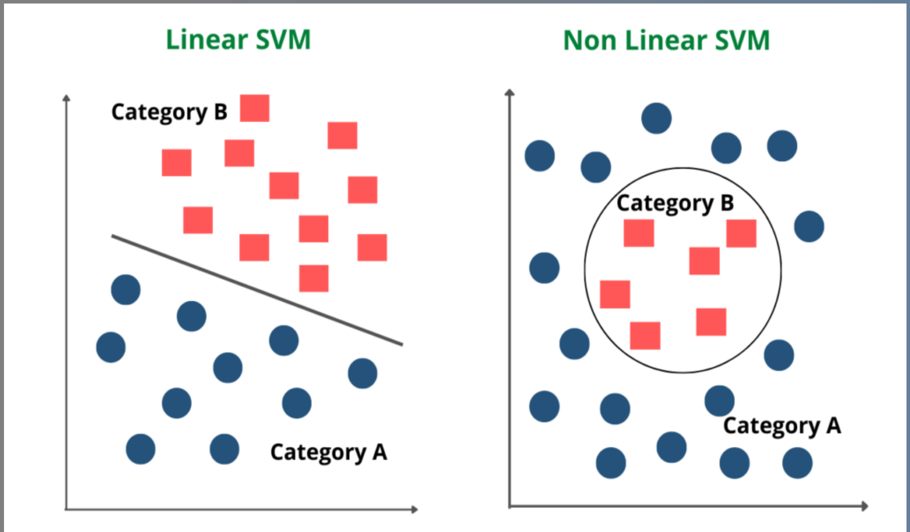
SUPPORT VECTOR MACHINES (SVM)

- **Type**: Supervised (Classification)

- **Purpose**: Finds the optimal hyperplane that separates different classes in the data.

- **Use Case**: Image classification (e.g., cat vs. dog).

- SVM is a method to separate classes by finding the best line (hyperplane) that maximizes the margin between the closest points of the different classes (support vectors). It works great for both linear and non-linear data.



➢ **How It Works:**

1. **Basic Idea**:
   1. **Find the Best Line**: Imagine you have two types of data points (e.g., apples and oranges). SVM tries to draw a line (or a boundary) that best separates these two types of data.
   2. **Maximize Margin**: It looks for the line that has the biggest gap (margin) between the data points of the two categories. This gap helps ensure the line is as clear and reliable as possible.

➢ **Working**:

1. **Plot Data Points**: Place your data points on a graph.
2. **Draw the Line**: SVM calculates the optimal line that separates the two types of data points.
3. **Support Vectors**: These are the data points closest to the line. They are crucial because they determine where the line is placed.

SYNTAX

```
#Import libraries
from sklearn.svm import SVC # For classification tasks
from sklearn.svm import SVR # For regression tasks

# Initialize the Support Vector Classifier
svm_clf = SVC(kernel='linear')  # You can choose different kernels like
'linear', 'poly', 'rbf', etc.

# Fit (train) the model using the training data
svm_clf.fit(X_train, y_train)

# Initialize the Support Vector Regressor
svm_reg = SVR(kernel='rbf')  # Again, you can use different kernels like
'linear', 'poly', 'rbf'

# Fit (train) the model using the training data
svm_reg.fit(X_train, y_train)

# Predict the target for test data (classification)
y_pred = svm_clf.predict(X_test)

# Predict the target for test data (regression)
y_pred = svm_reg.predict(X_test)
```

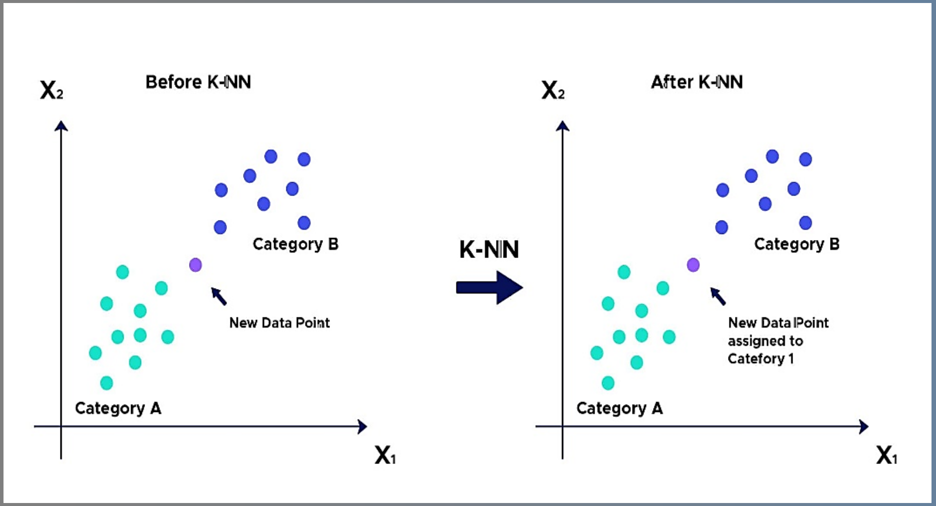
K-NEAREST NEIGHBORS (KNN)

•**Type**: Supervised (Classification/Regression)

•**Purpose**: Classifies data points based on the majority label of the nearest k neighbors.

•**Use Case**: Handwritten digit recognition.

K-nearest neighbors (KNN) is a type of **supervised machine learning algorithm** used for both classification and regression tasks. It works by finding the K data points (neighbors) in the training dataset that are closest to a new data point, based on a chosen distance metric (like euclidean distance).



➢ **How KNN Works:**

1. **Store Data**: KNN doesn't learn any internal model during training. Instead, it memorizes the entire training dataset.
2. **Classify Based on Neighbors**: When a new data point needs to be classified or predicted, KNN looks at the **K** (a number you choose) closest data points from the training dataset.
3. **Majority Vote (for Classification)**: If you're classifying, KNN looks at the labels of these K nearest neighbors. It assigns the label that most of its neighbors have. For example, if 3 out of 5 neighbors are labeled "cat," the new data point will be classified as "cat."
4. **Average (for Regression)**: If you're doing regression, KNN will take the average of the values of the K nearest neighbors and predict that as the output.

SYNTAX

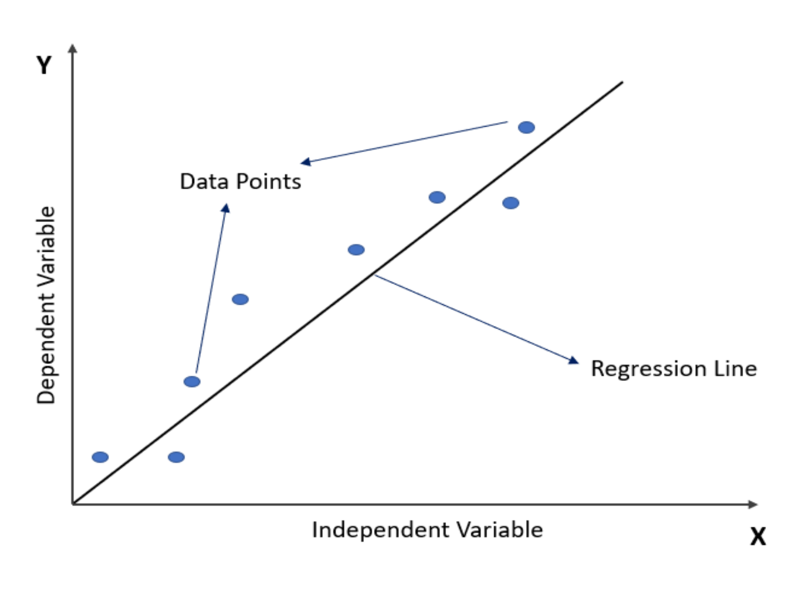
```
#Import libraries
from sklearn.neighbors import KNeighborsClassifier # For classification tasks
from sklearn.neighbors import KNeighborsRegressor # For regression tasks

# Initialize the K-Nearest Neighbors Classifier
knn_clf = KNeighborsClassifier(n_neighbors=5)  # You can set K as desired (e.g., K=5)

# Fit (train) the model using the training data
knn_clf.fit(X_train, y_train)

# Initialize the K-Nearest Neighbors Regressor
knn_reg = KNeighborsRegressor(n_neighbors=5)  # You can set K as desired (e.g., K=5)

# Fit (train) the model using the training data
knn_reg.fit(X_train, y_train)

# Predict the target for test data (classification)
y_pred = knn_clf.predict(X_test)

# Predict the target for test data (regression)
y_pred = knn_reg.predict(X_test)
```

LINEAR REGRESSION

•**Type**: Supervised (Regression)

•**Purpose**: Predicts a continuous target variable based on linear relationships between input features.

•**Use Case**: Predicting house prices based on features like square footage, number of rooms, etc.



Linear Regression is a method used to predict a numerical value based on one or more input features. It tries to find the best line that fits the data points on a graph, so you can make predictions based on new data.

➢ **Training the Model**:

- **Fit the Line**: The model uses the training data to find the best line that minimizes the difference between the predicted values and the actual values. This process is called "fitting" the line.
- **Minimize Errors**: The model tries to minimize the errors (the vertical distance between the data points and the line). This is done using a method called "Least Squares," which finds the line where the sum of these errors is the smallest.

➢ **Making Predictions**:

- **Use the Line**: Once the line is fitted, you can use it to make predictions.

➢ **Evaluating the Model**:

- **Check Accuracy**: After fitting the model, you check how well it performs on new data. Metrics like Mean Absolute Error (MAE) or Mean Squared Error (MSE) are used to measure how close the predicted values are to the actual values.

SYNTAX

```
#Import libraries
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Initialize the Linear Regression model
linreg = LinearRegression()

# Fit (train) the model using the training data
linreg.fit(X_train, y_train)

# Predict the target for test data
y_pred = linreg.predict(X_test)

# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# R-squared (R²)
r2 = r2_score(y_test, y_pred)
print("R-squared:", r2)
```

UNSUPERVISED LEARNING ALGORITHMS

Unsupervised learning algorithms find patterns, structure, or relationships in datasets without pre-existing labels or supervision. They're particularly useful for exploring data, clustering similar data points, reducing dimensionality, and discovering latent features.

Applications of Unsupervised Learning

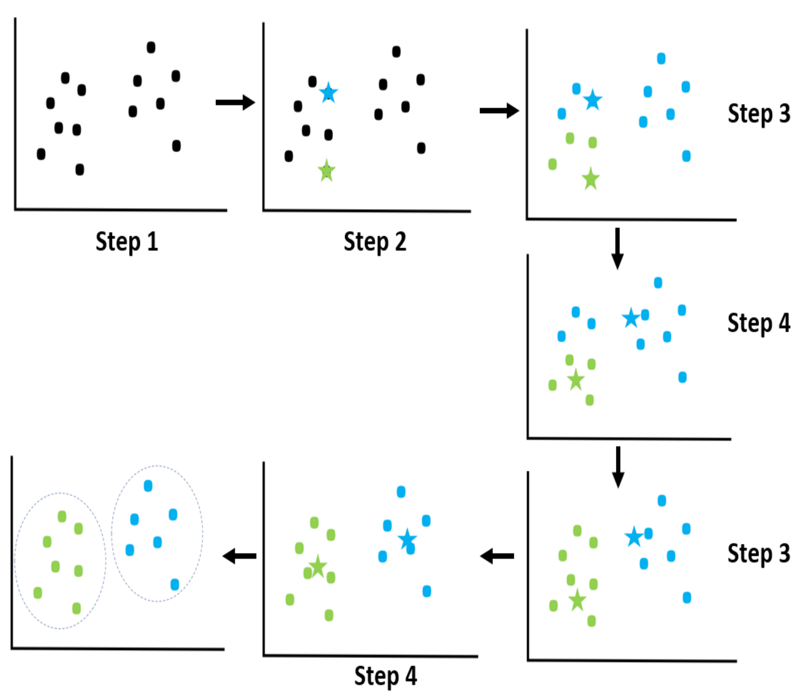
- **Customer Segmentation** in marketing.
- **Anomaly Detection** in finance and cybersecurity.
- **Image Compression** and **Feature Extraction**.
- **Document Organization** through topic modelling.
- **Recommendation Systems** using collaborative filtering.

Types of Unsupervised Learning Algorithm

1. **Clustering Algorithms:** Clustering groups data points into clusters based on similarity. Clustering algorithms are widely used for market segmentation, image segmentation, document organization, and anomaly detection.
2. **Dimensionality Reduction Algorithms:** Dimensionality reduction helps to reduce the number of features or variables in a dataset while preserving as much structure as possible. It's useful for data visualization, noise reduction, and speeding up model training.
3. **Association Rule Learning:** Association algorithms discover interesting relationships (associations) or patterns in large datasets, commonly used in market basket analysis.
4. **Neural Network-Based Unsupervised Algorithms:** Neural networks, traditionally used for supervised learning, can also be adapted for **unsupervised learning** tasks.

K-MEANS CLUSTERING

K-Means Clustering is one of the simplest and most widely used unsupervised machine learning algorithms for clustering. The goal of K-Means is to partition a dataset into kkk clusters (where kkk is a predefined number) by minimizing the variance within each cluster.

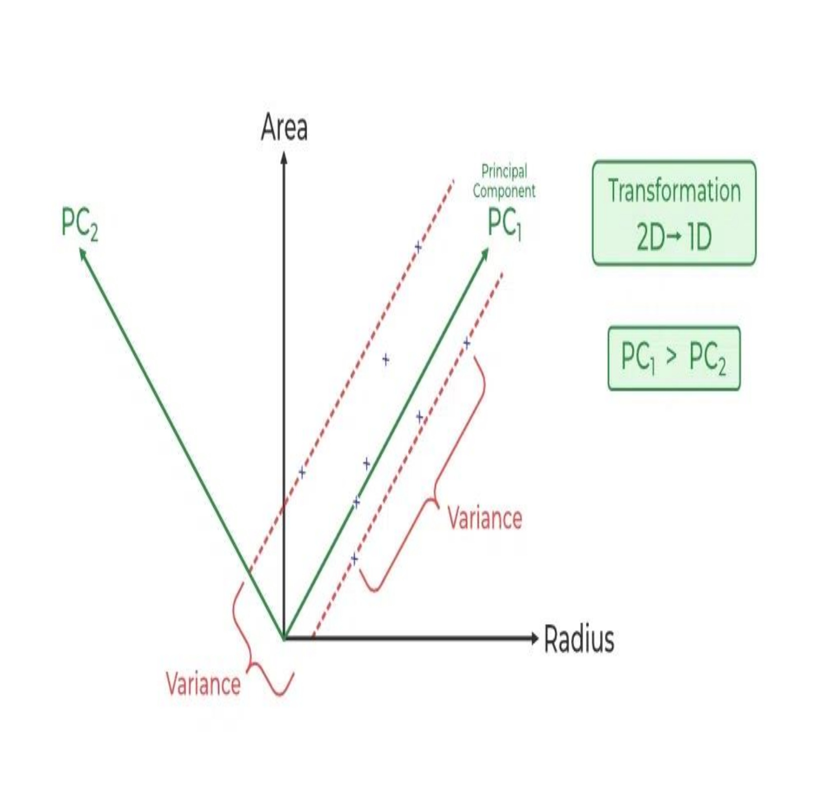


Steps of K-Means Clustering:

1. **Pick the Number of Groups**:
    - First, you decide how many groups you want (let’s call this number "k").
2. **Place Random Group Centers**:
    - K-Means starts by picking some points randomly on the sheet as the "centers" of each group. These are just starting points and will change as we go.
3. **Assign Dots to the Nearest Center**:
    - For each dot, K-Means looks at which center it’s closest to and assigns it to that group. So, all dots near one center become part of the same group.
4. **Recalculate Centers**:
    - After all dots are assigned to a group, the algorithm takes each group and finds the middle point (average location) of all the dots in it. This middle point becomes the new center of the group.
5. **Repeat Until Stable**:
    - Steps 3 and 4 are repeated. Dots might change groups as the centers move around, but eventually, the groups settle down, and the centers stop moving. When that happens, the algorithm stops, and the groups are final.

PRINCIPAL COMPONENT ANALYSIS (PCA)

Principal Component Analysis (PCA) is a way to simplify complex data by reducing the number of features (or dimensions) while keeping the most important information.



How PCA Works ?

1.Find the Main Patterns:

- PCA looks for directions (like axes or lines) in the data where the data varies the most—these directions are where the data "spreads out" the most. Each of these directions is called a **principal component**.

2.Re-orient the Data:

- Imagine rotating your data so it's aligned with these new principal components. Now, you have a new set of axes, where each axis represents one of the most important patterns or features in your data.

3.Keep Only the Important Parts:

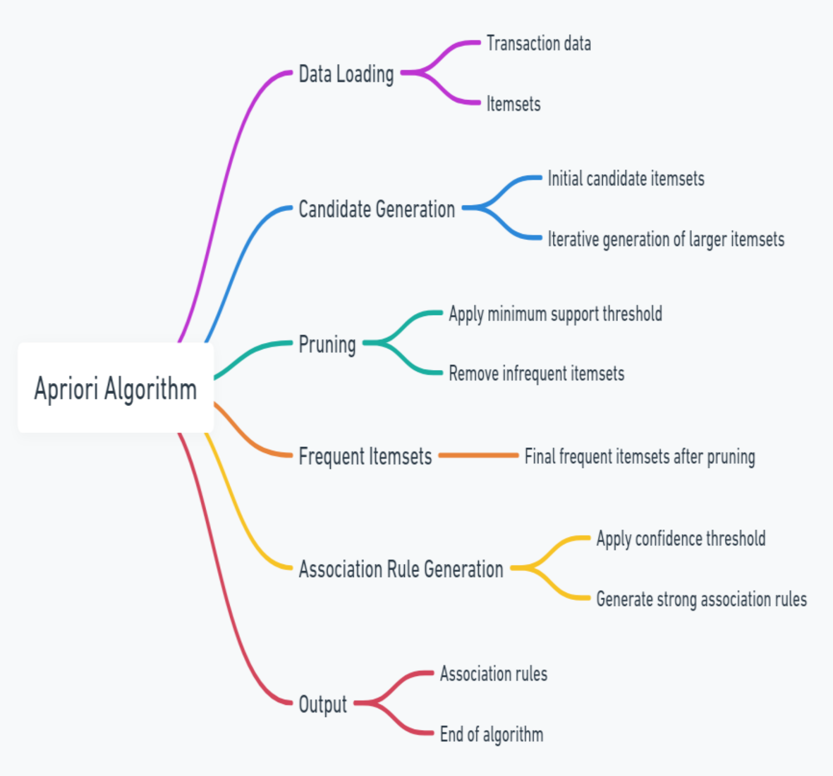
- PCA orders these patterns from the most to the least important. Then, you keep only the top few principal components (the most important ones) and drop the rest. This cuts down the complexity without losing much important information.

4.Result:

- You end up with a simpler version of your data, often with just a few key features instead of the original complex set. This smaller data still contains the key patterns but is now easier to work with or visualize.

Apriori Algorithm

The **Apriori Algorithm** is a way to find patterns in large sets of data by looking for items that frequently appear together. It's mainly used to uncover relationships, like finding items that are often bought together.



Apriori Algorithm

Data Loading

Transaction data

Itemsets

Candidate Generation

Initial candidate itemsets

Iterative generation of larger itemsets

Pruning

Apply minimum support threshold

Remove infrequent itemsets

Frequent Itemsets

Final frequent itemsets after pruning

Association Rule Generation

Apply confidence threshold

Generate strong association rules

Output

Association rules

End of algorithm

How Apriori Works

1.Starts Small and Builds Up:

  1. It starts by finding items that are often chosen individually, then combines those to see if they're also frequently bought together. As it finds frequent item pairs, it moves on to check for even larger groups of items.

2.Focuses on Frequent Patterns:

  1. Apriori keeps track of how often items or combinations of items appear together. It looks for combinations that appear often enough to be considered "frequent."

3.Measures How Strongly Items are Connected:

  1. Apriori measures two things: how often a combination appears in the data (support) and how likely one item is to appear if another item does (confidence). If both are high, the pattern is considered strong.

4.Filters for Relevant Information:

  1. Only combinations that meet a minimum level of frequency and connection are kept, so the output focuses on the most relevant patterns.

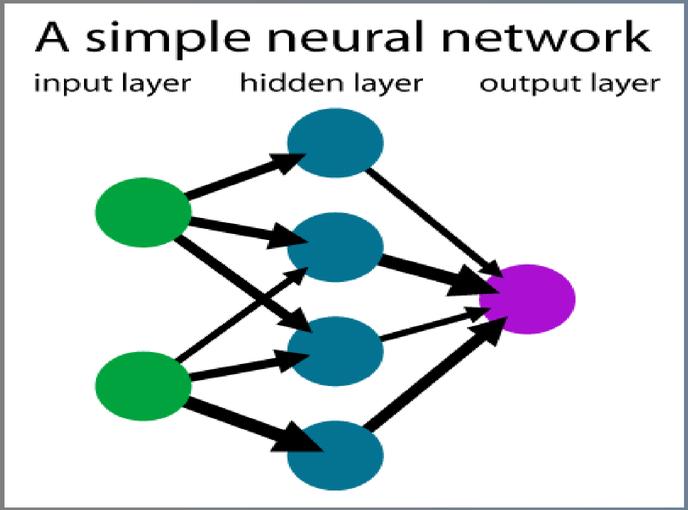
Neural Networks

Neural Networks: The core idea behind deep learning is the use of neural networks. A neural network is made up of layers of **neurons** (simple processing units). Each layer learns something about the data and passes the information to the next layer.

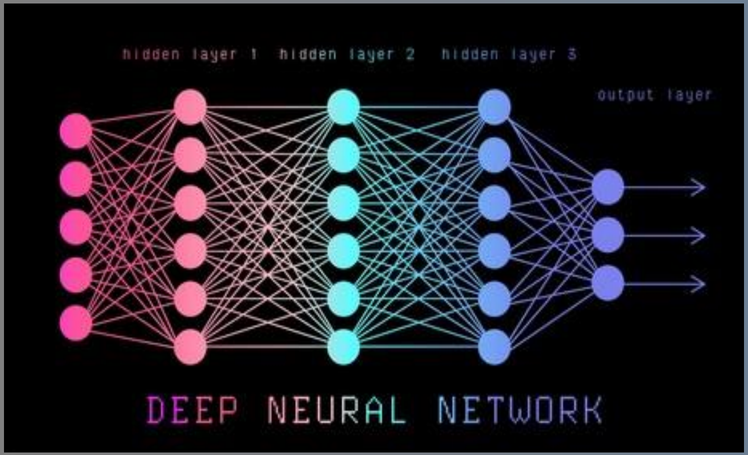
1. **Input Layer**: Takes in the raw data (like images, text, or numbers).
2. **Hidden Layers**: These are the "deep" layers that process the data step by step, extracting more and more complex information.
3. **Output Layer**: Produces the final result, like predicting a category or a number.

Types of Neural Network:

1. Feedforward Neural Networks (FNNs)
2. Convolutional Neural Networks (CNNs)
3. Recurrent Neural Networks (RNNs)
4. Transformer Networks
5. Generative Adversarial Networks (GANs)
6. Autoencoders
7. Deep Belief Networks (DBNs)



A model with one or a few hidden layers that can solve basic problems. It's made up of neurons (simple processing units) arranged in layers. Each neuron takes inputs, processes them, and passes the output to the next layer. Neural networks can learn patterns in data through training.



A deep neural network is just like a regular neural network but with more hidden layers (often called a "deep" network). The "deep" refers to having many layers between the input and output layers. These extra layers allow the network to learn much more complex patterns. It also enables the model to solve much more complex tasks, such as recognizing faces in photos, understanding speech, or playing games like chess.

Recurrent Neural Networks (RNNs)

➢ How RNNs Work:

•In a regular neural network, each input (like an image or a number) is processed independently.

•In an **RNN**, the network remembers **previous inputs** through something called a **hidden state**. This allows it to keep information from earlier steps in the sequence while processing new inputs.

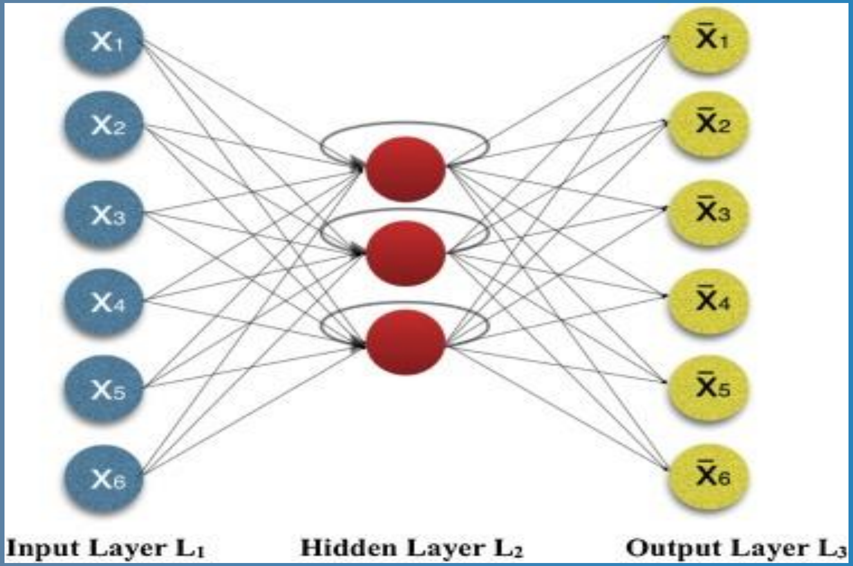
➢ Where RNNs are Used:

•**Language Translation**: Translating a sentence from one language to another, where the order of the words matters.

•**Speech Recognition**: Understanding spoken words by analyzing the sequence of sounds.

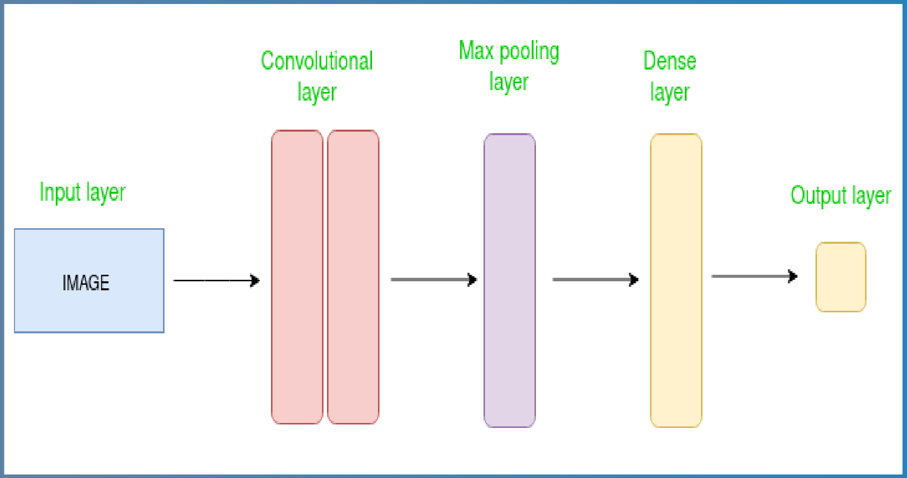
•**Time-Series Forecasting**: Predicting stock prices, weather, or other data that changes over time.

Recurrent Neural Networks (RNNs) are a type of neural network designed to handle sequential data. RNNs are good at understanding patterns in data that has a sequence, like predicting the next word in a sentence or the next step in a time-based process, by remembering what happened before.



Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a special type of neural network designed to process visual data, like images or videos. CNNs are particularly good at recognizing patterns, such as shapes, textures, and objects within images, making them ideal for tasks like image classification, object detection, and facial recognition.



➢ How CNNs Work:

Instead of looking at the entire image at once, CNNs focus on small parts of the image at a time. This allows the network to break down the image into smaller, more manageable pieces and gradually build an understanding of what's in the picture.

- **Convolution Layer:** The network processes small patches of the image (e.g., 3x3 or 5x5 grids) to detect basic features like edges and patterns, using filters that move across the image.
- **Pooling Layer:** The network reduces the size of the image by keeping the most important information (using max pooling), simplifying the data while retaining key features.
- **Fully Connected Layer:** After several convolution and pooling steps, the network combines all the learned features and makes the final prediction (e.g., identifying an image as "cat" or "dog").

Generative Adversarial Networks (GANs)

- **How GANs Work:**

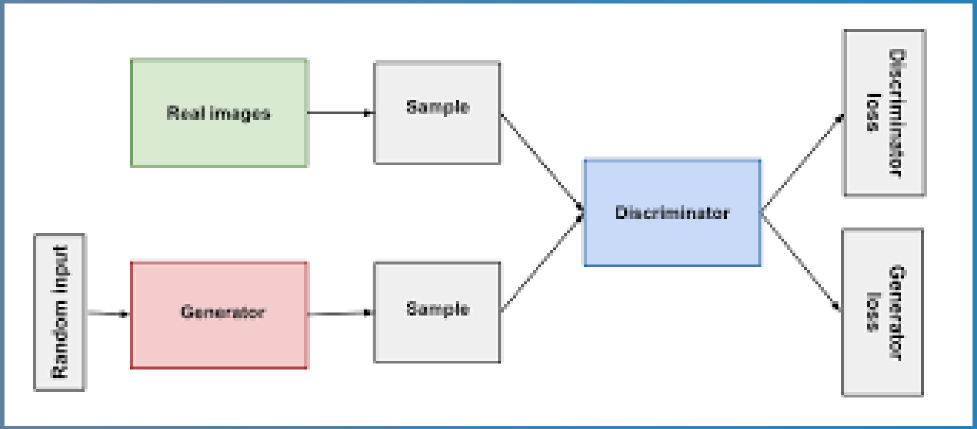
1.Generator: This network's job is to create fake data (e.g., images) that looks as real as possible. It starts by generating random data, like random noise, and tries to improve over time.

2.Discriminator: This network's job is to distinguish between real data (e.g., real images) and fake data created by the Generator. It acts as a judge, giving feedback on whether the data looks real or fake.

3.Training Process:

1. The **Generator** creates fake data and tries to fool the **Discriminator**.
2. The **Discriminator** tries to correctly identify if the data is real or fake.
3. Both networks improve through feedback: the Generator learns to make more realistic data, while the Discriminator gets better at spotting fakes.

Generative Adversarial Networks (GANs) are a type of AI model where two neural networks compete against each other to create realistic data. They are widely used for generating new images, videos, or even text that look or sound like real ones.



Autoencoders

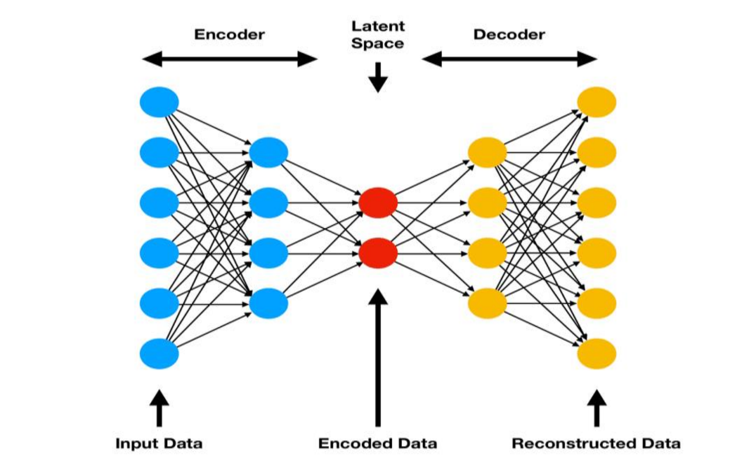
Autoencoders are a type of neural network used to learn how to compress data and then reconstruct it. They are typically used for tasks like data compression, noise reduction, and anomaly detection.

➢ How Autoencoders Work:

1.Encoder: This part of the network takes the input data (like an image or a set of numbers) and compresses it into a smaller, more efficient representation called a **latent space**. The goal is to capture the important features while reducing the data size.

2.Decoder: The decoder takes the compressed data and tries to reconstruct the original input as accurately as possible.

The entire process is trained so that the output of the decoder is as close as possible to the original input.



➢ Applications:

•**Data Compression**: Autoencoders can reduce the size of data while keeping important information, similar to compressing files on a computer.

•**Noise Reduction**: They can clean up noisy data, like removing static from an image.

•**Anomaly Detection**: Autoencoders can identify unusual data by learning what "normal" data looks like. Anything that doesn’t fit the pattern stands out as an anomaly.

THANK YOU !!