INDUSTRIAL TRAINING ON ARTIFICIAL INTELLIGENCE

BY ROSHNI YADAV

ARTIFICIAL INTELLIGENCE

Artificial Intelligence (AI) refers to the simulation of human intelligence in machines that are designed to think and act like humans. AI encompasses a range of technologies that allow machines to perform tasks that typically require human intelligence, such as recognizing speech, learning, problem-solving, decision-making, and understanding natural language.

Core Concepts of AI:

- **Machine Learning (ML):** A subset of AI where machines learn from data to improve their performance on tasks without being explicitly programmed.
- **Natural Language Processing (NLP):** The ability of machines to understand, interpret, and respond to human language.
- **Robotics:** AI applied to machines that can perform physical tasks, often autonomously.
- **Computer Vision:** Enabling machines to interpret and make decisions based on visual input (e.g., images and videos).
- **Deep Learning:** A subset of machine learning involving neural networks with multiple layers that mimic the functioning of the human brain.

APPLICATIONS OF AI

AI IS TRANSFORMING INDUSTRIES WITH SMARTER, MORE EFFICIENT SOLUTIONS.

- **Healthcare**: AI aids in medical diagnosis, personalized treatment, and virtual assistants for patient care.
- **Finance**: Used in fraud detection, algorithmic trading, and personalized financial advice.
- **Autonomous Vehicles**: Powers self-driving cars for navigation and obstacle detection.
- **Retail & E-commerce**: Enhances product recommendations and optimizes supply chains.
- **Education**: Enables personalized learning and automated grading systems.
- **NLP**: Powers virtual assistants, chatbots, and language translation tools.
- **Manufacturing**: Improves predictive maintenance and automates production processes.
- **Entertainment**: AI assists in content creation and video game dynamics.
- **Agriculture**: Supports smart farming with data-driven crop management.
- **Cybersecurity**: Detects threats and prevents cyberattacks through network analysis.

TYPES OF AI

Narrow AI (Weak AI):

•Specializes in specific tasks like language translation, virtual assistants, and image recognition.

•It cannot adapt or perform outside its programmed scope.

•Widely used today (e.g., Siri, self-driving cars).

General AI (AGI):

•Possesses the ability to understand, learn, and perform any task a human can.

•Capable of reasoning, problem-solving, and applying knowledge across different domains.

•Still theoretical and not yet developed.

Super AI:

•Exceeds human intelligence in all aspects, including creativity and decision-making.

•Could solve complex global challenges but poses risks due to its unpredictability.

•Hypothetical and remains in science fiction.

INTRODUCTION TO MACHINE LEARNING

Machine learning (ML) is a subset of artificial intelligence (AI) that enables computers to learn from data and improve their performance on tasks without explicit programming. Instead of being programmed with specific rules, ML algorithms learn patterns and make decisions based on the data they are trained on.

Key Concepts:

- **Data**: The foundation of ML. Machines learn patterns from data, which can be structured (like tabular data) or unstructured (like images, text).
- **Features**: Input variables that are used to make predictions (e.g., age, height, or temperature in a dataset).
- **Labels**: The target variable that the algorithm aims to predict (e.g., whether an email is spam or not).

TYPES OF MACHINE LEARNING

- **Supervised Learning:** The model is trained on labeled data (data with known outputs).

  - **Goal:** Learn a function that maps inputs (features) to outputs (labels).
  - **Examples:** Classification (e.g., spam detection, disease diagnosis).
    Regression (e.g., predicting house prices, stock market trends).

- **Unsupervised Learning:** The model works on unlabeled data (data without known outcomes).

  - **Goal:** Discover hidden patterns or groupings in the data.
  - **Examples:** Clustering (e.g., customer segmentation, grouping news articles).
    Dimensionality Reduction (e.g., reducing features to improve model performance).

- **Reinforcement Learning:** The model learns by interacting with an environment and receiving feedback in the form of rewards or penalties.

  - **Goal:** Learn a sequence of actions that maximize cumulative rewards.
  - **Examples:** Game-playing AI (e.g., AlphaGo, Chess engines).Robotics (e.g., self-learning robots).

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.

- **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))
```

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)
```

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
from sklearn.metrics import accuracy_score, mean_squared_error, classification_report
# 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)

# Accuracy score
print("Accuracy:", accuracy_score(y_test, y_pred))

# Classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

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
from sklearn.metrics import accuracy_score, mean_squared_error, classification_report

# 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)
# Accuracy score
print("Accuracy:", accuracy_score(y_test, y_pred))

# Classification report
print("Classification Report:\n", classification_report(y_test, y_pred))

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

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).

•**Support vector machine (SVM)** is a method used for classification tasks. It helps you separate data into different categories by finding the best boundary (or line) between them.

- **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
from sklearn.metrics import accuracy_score, classification_report, mean_squared_error
# 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)# Accuracy score

# Accuracy score
print("Accuracy:", accuracy_score(y_test, y_pred))
# Classification report (precision, recall, F1-score)
print("Classification Report:\n", classification_report(y_test, y_pred))

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

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
from sklearn.metrics import accuracy_score, classification_report, mean_squared_error

# 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)

# Accuracy score
print("Accuracy:", accuracy_score(y_test, y_pred))
# Classification report (precision, recall, F1-score)
print("Classification Report:\n", classification_report(y_test, y_pred))

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

CONFUSION MATRIX

The confusion matrix is the table used to describe the performance of the classification model. It shows the actual versus predicted classifications.

Structure:

		Predicted	
		Positive	Negative
Actual	Positive	TP	FN
Actual	Negative	FP	TN

- **TP (True Positive)**: Actual positive cases correctly predicted as positive.
- **TN (True Negative)**: Actual negative cases correctly predicted as negative.
- **FP (False Positive)**: Actual negative cases incorrectly predicted as positive.
- **FN (False Negative)**: Actual positive cases incorrectly predicted as negative.

MODEL EVALUATION

- Accuracy: The ratio of correctly predicted instances to the total instances. It tells you how often the classifier is correct.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\textit{Total Number of Predictions}}$$

Example: If you have 100 test cases and your model correctly predicts 90 of them, the accuracy is 90%.

- Precision: The ratio of correctly predicted positive instances to the total predicted positives. It tells you how many predicted positives cases were actually positive.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives+False Positives}}$$

Example: If your model predicts 10 positives and 7 of them are correct, the precision is 70%.

MODEL EVALUATION

- Recall: The ratio of correctly predicted positive instances to all actual positives. It tells you how many of the actual positive cases were captured by the model.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$$

Example: If there are 20 actual positives and your model correctly predicts 15 of them, the recall is 75%.

- F1- Score: The harmonic mean of precision and recall. It gives a balance between precision and recall, especially useful if you need a single measure of performance.

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

Example: If your model has a precision of 70% and a recall of 80%, the F1-score is approximately 74%.

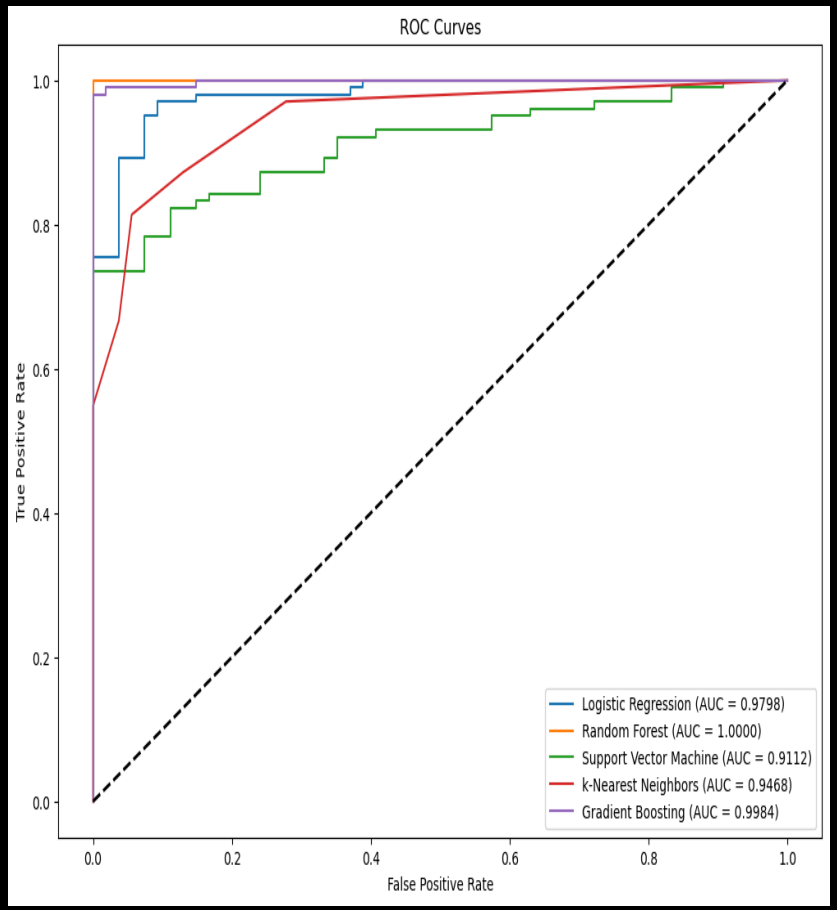
ROC CURVE

- ROC stands for Receiver Operating Characteristic. It is a graphical representation used to evaluate the performance of a binary classification model by illustrating the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) at various threshold settings.

- **Area Under the ROC Curve (AUC - ROC):**

AUC:

  - The Area Under the Curve (AUC) is a single scalar value summarizing the performance of the model.
  - AUC ranges from 0 to 1.
  - A model with perfect discrimination has an AUC of 1.
  - A model with no discrimination ability has an AUC of 0.5 (the ROC curve is the diagonal line).



MODEL EVALUATION FOR REGRESSION

Mean Absolute Error (MAE): Measures the average magnitude of errors in a set of predictions, without considering their direction.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |yi - \widehat{yi}|$$

Mean Squared Error (MSE): Penalizes larger errors more than smaller errors due to squaring the differences.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} \left(yi - \widehat{yi}\right)^2$$

MODEL EVALUATION FOR REGRESSION

Root Mean Squared Error (RMSE): Provides error in the same unit as the output variable. It's more sensitive to outliers.

$$\text{MAE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(yi - \widehat{yi})^2}$$

R-squared (R^2) : Measures the proportion of variance in the dependent variable that is predictable from the independent variables. Values range from 0 to 1, with 1 being a perfect fit.

$$R^2 = 1 - \frac{\sum_{i-1}^{n}(yi - \hat{y}i)^2}{\sum_{i=1}^{n}(yi - \bar{y}i)^2}$$

THANK YOU !!