Day - 4, 5, 6

✅ *Algorithm Steps for Decision Tree*

1 Import libraries

2 Load dataset

3 Extract features (X)

4 Extract label (y)

5 Train-test split

6 Create DecisionTreeClassifier

7 Train model using .fit()

8 Predict using .predict()

9 Evaluate with accuracy_score()

10 Visualize tree (optional)

1. Import Required Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

---

2. Import the Dataset

```
data = pd.read_csv('your_dataset.csv')  # Replace with your dataset path
```

---

3. Feature Extraction (Independent Variables)

```
X = data.drop('target_column', axis=1)  # Replace 'target_column' with your label column name
```

---

4. Label Extraction (Dependent Variable)

```
y = data['target_column']  # The column you want to predict
```

---

5. Split the Dataset into Train and Test Sets

```
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

---

6. Create a Decision Tree Classifier

```
 model = DecisionTreeClassifier(criterion='gini', random_state=42)  # or
criterion='entropy'
```

---

7. Train the Model

```
model.fit(X_train, y_train)
```

---

8. Make Predictions on Test Data

```
y_pred = model.predict(X_test)
```

---

9. Evaluate the Model

```
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```

---

10. (Optional) Visualize the Decision Tree

```
from sklearn.tree import plot_tree

import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))

plot_tree(model, filled=True, feature_names=X.columns,
class_names=np.unique(y).astype(str))

plt.show()
```

🔍 What is joblib?

Joblib is a Python library used for:

- **Saving and loading large Python objects** (like machine learning models, numpy arrays, or pipelines).
- **Parallel computing** for tasks like parallel loops or grid searches.
- It is particularly **optimized for performance with NumPy data**.

✅ Most commonly, it's used to **persist trained machine learning models** so you don't need to retrain them every time.

SMOTE

✅ **SMOTE (Synthetic Minority Over-sampling Technique) — In Brief**

SMOTE is a technique used in **machine learning** to handle **imbalanced datasets**, especially in classification problems.

🧠 What is Class Imbalance?

When one class (e.g., "Not Fraud") has **many more samples** than another class (e.g., "Fraud"), models tend to ignore the minority class. This is called **class imbalance**.

🔧 What SMOTE Does:

- SMOTE **increases the number of minority class samples** by **creating synthetic (new) data points**.
- It doesn't copy existing samples — it **generates new ones** by interpolating between existing ones.

📈 Why Use SMOTE?

- To **improve model accuracy** on the minority class.
- To **prevent bias** toward the majority class.
- To make classification models like decision trees, logistic regression, etc., more balanced.

🔍 How SMOTE Works (Basic Idea):

1. For each minority sample, SMOTE:
    - Finds its **nearest neighbors**.
    - Randomly selects one neighbor.
    - **Creates a new sample** between the original point and the selected neighbor.

🛠️ Code Example (Using imblearn library):

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# Original data (X: features, y: labels)
 smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)
```

⚠️ When to Use:

- Only on **training data**, not on test data.
- Works best for **continuous** (numeric) features.

✅ Summary Table

Feature	SMOTE
Goal	Balance dataset
Method	Synthetic sample generation
Target	Minority class
Tool Used	imblearn.over_sampling
Benefit	Better model fairness