Q1 What is a Decision Tree, and how does it work in the context of classification?

Ans A **Decision Tree** is a **supervised machine learning algorithm** used for both classification and regression, but most commonly for **classification tasks**. It works by splitting the dataset into subsets based on feature values, creating a **tree-like structure** of decisions that lead to a predicted class label.

**Structure of a Decision Tree**

1. **Root Node** – Represents the entire dataset and the first splitting attribute.
2. **Decision Nodes** – Intermediate nodes where the dataset is split further based on conditions.
3. **Branches** – Outcomes of the decisions (e.g., Yes/No).
4. **Leaf Nodes** – Terminal nodes that assign the final class label.

**Working of Decision Tree in Classification**

1. **Feature Selection for Splitting**:
   * At each node, the algorithm chooses the "best" feature to split the data.
   * Metrics like **Information Gain (ID3), Gain Ratio (C4.5), or Gini Index (CART)** are used.
2. **Recursive Partitioning**:
   * The dataset is divided into subsets based on feature values.
   * This process is repeated recursively until one of the stopping conditions is met:
     + All samples in a node belong to the same class.
     + No remaining features.
     + Tree reaches maximum depth.
3. **Classification Rule Formation**:
   * Each path from root to leaf represents a **decision rule** (if-else conditions).
   * A new instance is classified by traversing the tree from root to leaf.

**Example**

Suppose we are classifying whether a person will **play cricket** based on **Weather (Sunny, Rainy, Overcast)** and **Humidity (High, Normal)**:

* If **Weather = Overcast → Play = Yes**.
* If **Weather = Sunny & Humidity = High → Play = No**.
* If **Weather = Rainy → Play = Yes**.

This illustrates how the decision tree predicts outcomes through conditions.

**Advantages**

* Easy to understand and interpret.
* Handles both categorical and numerical data.
* Requires little data preprocessing (no scaling/normalization).

**Limitations**

* Prone to **overfitting** if the tree grows too deep.
* Sensitive to small changes in data.
* Biased towards features with many levels.

Q 2 Explain the concepts of Gini Impurity and Entropy as impurity measures. How do they impact the splits in a Decision Tree?

Ans **Gini Impurity and Entropy in Decision Trees**

**1. Concept of Impurity Measures**

When constructing a **Decision Tree**, the goal is to split the dataset into subsets that are as **pure** as possible, i.e., containing samples mostly from a single class.  
To measure this **impurity (or disorder)**, two common metrics are used: **Gini Impurity** and **Entropy**.

**2. Gini Impurity**

* **Definition**: Gini Impurity measures the probability of **incorrectly classifying** a randomly chosen sample if it was labeled according to the class distribution in that node.
* **Formula**:

Gini(t)=1−∑i=1kpi2Gini(t) = 1 - \sum\_{i=1}^{k} p\_i^2Gini(t)=1−i=1∑k​pi2​

where pip\_ipi​ = proportion of class *i* in node *t*.

* **Range**: 0 (pure node, only one class) to 0.5 (for binary classification with equal distribution).
* **Interpretation**: Lower Gini = higher purity.

**3. Entropy (Information Gain)**

* **Definition**: Entropy measures the **uncertainty or disorder** in a node. It comes from Information Theory.
* **Formula**:

Entropy(t)=−∑i=1kpilog⁡2(pi)Entropy(t) = - \sum\_{i=1}^{k} p\_i \log\_2(p\_i)Entropy(t)=−i=1∑k​pi​log2​(pi​)

where pip\_ipi​ = proportion of class *i* in node *t*.

* **Range**: 0 (pure node) to 1 (maximum impurity in binary classification).
* **Information Gain**:
  + Decision Trees using Entropy choose splits that **maximize Information Gain (reduction in entropy)**.

IG = Entropy(parent) - \sum\_{j} \frac{N\_j}{N} Entropy(child\_j)  
]

**4. Impact on Splits in Decision Trees**

* At each node, the algorithm evaluates **all possible splits** and selects the one that results in the **largest reduction in impurity**.
* **Using Gini**:
  + Splits that **maximize class separation** are preferred.
  + Gini tends to isolate the **most frequent class** quickly.
* **Using Entropy**:
  + Focuses more on **overall distribution** of classes.
  + Leads to more **balanced splits** when classes are close in proportion.
* In practice, both usually yield similar trees, but **Gini is computationally faster**, while **Entropy is more theoretically grounded** in information theory.

**5. Example**

Suppose a node has 10 samples: 6 of Class A and 4 of Class B.

* pA=0.6,pB=0.4p\_A = 0.6, p\_B = 0.4pA​=0.6,pB​=0.4.
* **Gini** = 1 – (0.6² + 0.4²) = 0.48.
* **Entropy** = –(0.6 log₂0.6 + 0.4 log₂0.4) ≈ 0.97.  
  ➡️ Both values show impurity, and the algorithm will try to split this node to reduce it.

**6. Advantages & Limitations**

* **Gini**: Faster to compute, often leads to similar results as Entropy.
* **Entropy**: Provides a stronger theoretical basis using information theory.

**7. Conclusion**

Both **Gini Impurity** and **Entropy** are measures of node impurity used to guide Decision Tree splits. They determine which feature and threshold provide the **purest child nodes**, thereby making the classification more accurate. The choice between them often has little effect on performance, though Gini is preferred for efficiency, while Entropy is valued for its theoretical foundation.

Q3 What is the difference between Pre-Pruning and Post-Pruning in Decision Trees? Give one practical advantage of using each.

Ans **Pre-Pruning vs. Post-Pruning in Decision Trees**

**1. Need for Pruning**

* Decision Trees tend to grow very deep and capture **noise**, leading to **overfitting**.
* **Pruning** techniques simplify the tree to improve generalization.

**2. Pre-Pruning (Early Stopping)**

* **Definition**: Pre-pruning stops the tree from growing beyond a certain point during its construction.
* **How it works**:
  + Set conditions such as:
    - Maximum depth of tree.
    - Minimum number of samples required at a node to split.
    - Minimum information gain required for a split.
  + If these conditions are not satisfied, the split is not made.
* **Advantage**: Saves **time and computation** because unnecessary splits are avoided early.
* **Example**: Stop splitting a node if it has fewer than 5 samples.

**3. Post-Pruning (Prune After Full Growth)**

* **Definition**: Post-pruning allows the tree to grow fully, and then prunes back branches that do not improve accuracy significantly.
* **How it works**:
  + Grow a complete tree.
  + Use a **validation set** or **statistical test** (e.g., reduced error pruning, cost-complexity pruning).
  + Remove branches that add little predictive power.
* **Advantage**: Produces a **more accurate and simplified tree**, as pruning decisions are based on actual performance.
* **Example**: Removing branches that increase training accuracy but reduce validation accuracy.

**4. Key Differences**

| **Aspect** | **Pre-Pruning** | **Post-Pruning** |
| --- | --- | --- |
| **When applied** | During tree construction | After tree is fully grown |
| **Approach** | Prevents splits early using thresholds | Removes unhelpful branches later |
| **Computation** | Faster, less resource-intensive | More computation, but better accuracy |
| **Risk** | May underfit (stops too early) | Less underfitting, more balanced |

**5. Conclusion**

* **Pre-Pruning** is useful when **speed and efficiency** are important (e.g., real-time systems).
* **Post-Pruning** is useful when **accuracy and generalization** are prioritized (e.g., medical diagnosis, fraud detection).
* Both techniques aim to reduce overfitting and improve model interpretability.

Q4 What is Information Gain in Decision Trees, and why is it important for choosing the best split?

Ans **Information Gain in Decision Trees**

**1. Definition**

* **Information Gain (IG)** is a measure used in Decision Trees to determine the **best attribute for splitting** the data at each node.
* It quantifies the **reduction in impurity (or uncertainty)** after splitting the dataset based on a feature.
* It is based on **Entropy** from Information Theory.

**2. Formula**

IG(S,A)=Entropy(S)−∑v∈Values(A)∣Sv∣∣S∣⋅Entropy(Sv)IG(S, A) = Entropy(S) - \sum\_{v \in Values(A)} \frac{|S\_v|}{|S|} \cdot Entropy(S\_v)IG(S,A)=Entropy(S)−v∈Values(A)∑​∣S∣∣Sv​∣​⋅Entropy(Sv​)

Where:

* SSS = parent dataset.
* AAA = attribute used for split.
* SvS\_vSv​ = subset of SSS for value vvv of attribute AAA.
* Entropy(S)Entropy(S)Entropy(S) = impurity before split.
* Weighted entropy of children is subtracted from parent entropy.

**3. How it Works in Splitting**

1. **Calculate Entropy of the parent node** (before split).
2. **Split dataset** based on an attribute.
3. **Calculate weighted entropy of child nodes**.
4. **Information Gain = Reduction in entropy** (higher is better).
5. The attribute with the **highest Information Gain** is chosen for splitting.

**4. Example**

Suppose we classify whether students **Pass** or **Fail** based on “Hours of Study.”

* Parent node entropy = 0.97 (mix of Pass/Fail).
* After splitting:
  + Students with >5 hours → mostly Pass (low entropy).
  + Students with ≤5 hours → mixed results (higher entropy).
* Weighted child entropy = 0.65.
* **IG = 0.97 – 0.65 = 0.32.**  
  ➡️ Since IG is positive, the split reduces impurity, making classification more accurate.

**5. Importance of Information Gain**

* **Guides splitting**: Ensures that the most informative attribute is chosen.
* **Improves purity**: Leads to child nodes that are closer to containing a single class.
* **Prevents randomness**: Avoids arbitrary splits by using a quantitative measure.
* **Foundation of ID3 & C4.5 algorithms**: Core concept in building decision trees.

**6. Conclusion**

Information Gain measures how much an attribute improves the **classification ability** of the tree. By choosing the split with the **highest Information Gain**, the tree becomes more accurate, interpretable, and effective in reducing uncertainty.

Q5What are some common real-world applications of Decision Trees, and what are their main advantages and limitations?

Ans **Real-World Applications of Decision Trees**

**1. Common Applications**

1. **Finance & Banking** – Credit scoring, loan approval, and fraud detection.
2. **Healthcare** – Disease diagnosis, treatment recommendations, predicting patient survival rates.
3. **Marketing & Sales** – Customer segmentation, predicting churn, product recommendation.
4. **Human Resources** – Employee attrition prediction, recruitment decision support.
5. **Manufacturing & Engineering** – Fault detection, quality control, predictive maintenance.
6. **Education** – Predicting student performance, personalizing learning paths.

**2. Main Advantages of Decision Trees**

* **Easy to Understand & Interpret** – Produces intuitive if-else rules and visual trees.
* **Handles Both Categorical & Numerical Data** – Flexible with different types of inputs.
* **No Need for Feature Scaling** – Unlike algorithms like SVM or Logistic Regression.
* **Fast to Train & Predict** – Computationally efficient for smaller datasets.
* **Helps in Feature Selection** – Identifies the most important features automatically.

**3. Main Limitations of Decision Trees**

* **Overfitting** – Trees can grow too deep and capture noise in training data.
* **Unstable** – Small changes in data may lead to very different trees.
* **Bias Towards Attributes with Many Levels** – Features with more categories may dominate splits.
* **Lower Accuracy Compared to Ensembles** – Alone, they may perform worse than Random Forests or Gradient Boosted Trees.

from sklearn.datasets import load\_iris

import pandas as pd

# Load dataset

iris = load\_iris()

# Create DataFrame

df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

df['target'] = iris.target

df['species'] = df['target'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})

print(df.head())

output:

| **sepal length (cm)** | **sepal width (cm)** | **petal length (cm)** | **petal width (cm)** | **target** | **species** |
| --- | --- | --- | --- | --- | --- |
| 5.1 | 3.5 | 1.4 | 0.2 | 0 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | 0 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | 0 | setosa |

from sklearn.datasets import fetch\_california\_housing

import pandas as pd

# Load dataset

housing = fetch\_california\_housing()

# Convert to DataFrame

df = pd.DataFrame(housing.data, columns=housing.feature\_names)

df['target'] = housing.target

print(df.head())

output:

MedInc HouseAge AveRooms AveBedrms Population AveOccup Latitude \

0 8.3252 41.0 6.984127 1.023810 322.0 2.555556 37.88

1 8.3014 21.0 6.238137 0.971880 2401.0 2.109842 37.86

2 7.2574 52.0 8.288136 1.073446 496.0 2.802260 37.85

3 5.6431 52.0 5.817352 1.073059 558.0 2.547945 37.85

4 3.8462 52.0 6.281853 1.081081 565.0 2.181467 37.85

Longitude target

0 -122.23 4.526

1 -122.22 3.585

2 -122.24 3.521

3 -122.25 3.413

4 -122.25 3.422

Q6 Write a Python program to: ● Load the Iris Dataset ● Train a Decision Tree Classifier using the Gini criterion ● Print the model’s accuracy and feature importances (Include your Python code and output in the code box below.)

Ans # Import libraries

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train Decision Tree Classifier using Gini criterion

clf = DecisionTreeClassifier(criterion="gini", random\_state=42)

clf.fit(X\_train, y\_train)

# Make predictions

y\_pred = clf.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

# Print results

print("Decision Tree Classifier (Gini Index)")

print("Accuracy:", accuracy)

print("Feature Importances:")

for feature, importance in zip(iris.feature\_names, clf.feature\_importances\_):

print(f"{feature}: {importance:.4f}")

output:

Decision Tree Classifier (Gini Index)

Accuracy: 1.0

Feature Importances:

sepal length (cm): 0.0000

sepal width (cm): 0.0191

petal length (cm): 0.8933

petal width (cm): 0.0876

Q7 Write a Python program to: ● Load the Iris Dataset ● Train a Decision Tree Classifier with max\_depth=3 and compare its accuracy to a fully-grown tree. (Include your Python code and output in the code box below.)

Ans # Import libraries

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=42

)

# Decision Tree with max\_depth=3

clf\_limited = DecisionTreeClassifier(max\_depth=3, random\_state=42)

clf\_limited.fit(X\_train, y\_train)

y\_pred\_limited = clf\_limited.predict(X\_test)

accuracy\_limited = accuracy\_score(y\_test, y\_pred\_limite

# Fully grown Decision Tree

clf\_full = DecisionTreeClassifier(random\_state=42)

clf\_full.fit(X\_train, y\_train)

y\_pred\_full = clf\_full.predict(X\_test)

accuracy\_full = accuracy\_score(y\_test, y\_pred\_full)

# Print results

print("Decision Tree with max\_depth=3 Accuracy:", accuracy\_limited)

print("Fully-grown Decision Tree Accuracy:", accuracy\_full)

**example Output (results may vary slightly)**

Decision Tree with max\_depth=3 Accuracy: 0.9778

Fully-grown Decision Tree Accuracy: 1.0

Q8 Write a Python program to: ● Load the California Housing dataset from sklearn ● Train a Decision Tree Regressor ● Print the Mean Squared Error (MSE) and feature importances (Include your Python code and output in the code box below.)

Ans # Import libraries

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

import pandas as pd

# Load the California Housing dataset

housing = fetch\_california\_housing()

X = housing.data

y = housing.target

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=42

)

# Train Decision Tree Regressor

reg = DecisionTreeRegressor(random\_state=42)

reg.fit(X\_train, y\_train)

# Make predictions

y\_pred = reg.predict(X\_test)

# Calculate Mean Squared Error

mse = mean\_squared\_error(y\_test, y\_pred)

# Print results

print("Decision Tree Regressor Results")

print("Mean Squared Error (MSE):", mse)

print("\nFeature Importances:")

for feature, importance in zip(housing.feature\_names, reg.feature\_importances\_):

print(f"{feature}: {importance:.4f}")

output :

Decision Tree Regressor Results

Mean Squared Error (MSE): 0.2671

Feature Importances:

MedInc: 0.6432

HouseAge: 0.0501

AveRooms: 0.1193

AveBedrms: 0.0079

Population: 0.0205

AveOccup: 0.0410

Latitude: 0.0625

Longitude: 0.0555

Q9 : Write a Python program to: ● Load the Iris Dataset ● Tune the Decision Tree’s max\_depth and min\_samples\_split using GridSearchCV ● Print the best parameters and the resulting model accuracy (Include your Python code and output in the code box below.)

Ans # Import libraries

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=42

)

# Define Decision Tree Classifier

clf = DecisionTreeClassifier(random\_state=42)

# Define parameter grid for GridSearchCV

param\_grid = {

"max\_depth": [2, 3, 4, 5, None],

"min\_samples\_split": [2, 3, 4, 5, 10]

}

# Apply GridSearchCV (5-fold cross-validation)

grid\_search = GridSearchCV(

estimator=clf,

param\_grid=param\_grid,

cv=5,

scoring="accuracy",

n\_jobs=-1

)

# Fit the model

grid\_search.fit(X\_train, y\_train)

# Get the best model

best\_clf = grid\_search.best\_estimator\_

# Predict on test set

y\_pred = best\_clf.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

# Print results

print("Best Parameters:", grid\_search.best\_params\_)

print("Best Cross-Validation Accuracy:", grid\_search.best\_score\_)

print("Test Set Accuracy:", accuracy)

output: Best Parameters: {'max\_depth': 3, 'min\_samples\_split': 2}

Best Cross-Validation Accuracy: 0.9714

Test Set Accuracy: 0.9778

Q10 : Imagine you’re working as a data scientist for a healthcare company that wants to predict whether a patient has a certain disease. You have a large dataset with mixed data types and some missing values. Explain the step-by-step process you would follow to: ● Handle the missing values ● Encode the categorical features ● Train a Decision Tree model ● Tune its hyperparameters ● Evaluate its performance And describe what business value this model could provide in the real-world setting.

Ans **1. Handle the Missing Values**

* **Why?** Medical datasets often have incomplete records (e.g., missing blood pressure readings, lab results).
* **How?**
  + For **numerical features**:
    - Use **mean/median imputation** (if missing at random).
    - Or use **KNN imputer** (considers similar patients).
  + For **categorical features**:
    - Replace missing values with the **most frequent category** or a new category like "Unknown".
  + Advanced approach: Use **model-based imputation** (e.g., regression to estimate missing lab values).

**2. Encode the Categorical Features**

* Decision Trees can handle categorical data in some libraries, but scikit-learn requires **numeric encoding**.
* Options:
  + **One-Hot Encoding**: For nominal categories (e.g., gender, blood type).
  + **Ordinal Encoding**: For ordered categories (e.g., disease stage I < II < III < IV).
  + Use **Target/Mean Encoding** if high-cardinality categorical features exist (e.g., hospital IDs).

**3. Train a Decision Tree Model**

* Split the dataset:
  + **Training (70–80%)** and **Testing (20–30%)**.
* Train a DecisionTreeClassifier:

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X\_train, y\_train)

**4. Tune its Hyperparameters**

* Important parameters for Decision Trees:
  + max\_depth: Prevents overfitting.
  + min\_samples\_split: Minimum samples to split a node.
  + min\_samples\_leaf: Minimum samples in a leaf node.
  + criterion: "gini" or "entropy".
* Use **GridSearchCV** or **RandomizedSearchCV**:

from sklearn.model\_selection import GridSearchCV

param\_grid = {

"max\_depth": [3, 5, 7, None],

"min\_samples\_split": [2, 5, 10],

"min\_samples\_leaf": [1, 2, 4]

}

grid\_search = GridSearchCV(clf, param\_grid, cv=5, scoring="accuracy")

grid\_search.fit(X\_train, y\_train)

best\_model = grid\_search.best\_estimator\_

**5. Evaluate its Performance**

* **Metrics**:
  + **Accuracy** (overall correctness).
  + **Precision** (how many predicted positives are correct).
  + **Recall (Sensitivity)** (how many actual positives are caught).
  + **F1-score** (balance between precision & recall).
  + **ROC-AUC** (how well model distinguishes diseased vs. healthy).

from sklearn.metrics import classification\_report, roc\_auc\_score

y\_pred = best\_model.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

print("ROC-AUC:", roc\_auc\_score(y\_test, best\_model.predict\_proba(X\_test)[:,1]))