Decision Tree | Assignment

Question 1: What is a Decision Tree, and how does it work in the context of classification?

Answer-A **Decision Tree** is a supervised machine learning algorithm used for **classification** (and regression) tasks.
It works like a flowchart where each **internal node** represents a decision based on a feature (attribute), each **branch** represents an outcome of the decision, and each **leaf node** represents a final class label (or prediction).

How it works (Step-by-Step in Classification):

1. **Start with the full dataset**
   The root node contains all the training data.
2. **Select the best feature to split**
   The algorithm chooses the feature that best separates the data into different classes.
   - Common criteria for splitting:
     - **Gini Index** (CART algorithm)
     - **Entropy / Information Gain** (ID3, C4.5 algorithms)
3. **Split the data**
   The dataset is divided into subsets based on the selected feature's values.
4. **Repeat recursively**
   For each subset, the process continues:
   - Choose the best feature
   - Split again
     Until one of these happens:
   - All samples in a node belong to the same class
   - No more features remain
   - A stopping criterion is met (like max depth)
5. **Assign a class label**
   Each terminal node (leaf) is assigned a class label, usually the **majority class** of the samples in that node.

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

1. Gini Impurity

- **Definition:** Measures how often a randomly chosen sample would be **misclassified** if it were randomly labeled according to the class distribution in the node.
- **Formula:**

Gini(t)=1-$\sum_{i=1}^{c} pi2$

- where:
    - CCC = number of classes
    - pip_ipi = proportion of class iii in the node
- **Range:**
    - 000 → node is pure (only one class)
    - Higher values → more mixed classes (max = 0.5 for binary classification with 50%-50%)

✅ Example: If a node has [70% Class A, 30% Class B]:

Gini=1−(0.72+0.32)=1−(0.49+0.09)=0.42

2. Entropy (Information Gain)

- **Definition:** Measures the amount of **uncertainty** or **disorder** in a node.
- **Formula:**

Entropy(t)= $-\sum_{i=1}^{c} pilog2(pi)$

where pip_ipi is the probability of class iii.

- **Range:**
    - 000 → pure node (all samples in one class)
    - Max value → occurs when classes are equally distributed
        - For binary classification (50%-50%): Entropy = 1

✅ Example: If node has [70% Class A, 30% Class B]:

Entropy=−(0.7log20.7+0.3log20.3)≈0.88Entropy = - (0.7 \log_2 0.7 + 0.3 \log_2 0.3) \approx 0.88Entropy=−(0.7log20.7+0.3log20.3)≈0.88

Question 3: What is the difference between Pre-Pruning and Post-Pruning in Decision Trees? Give one practical advantage of using each.

Answer-**Pre-Pruning (Early Stopping)**

- **Definition:** Stop the tree from growing **too deep** by setting constraints **during construction**.
- **How:**
    - Limit tree depth (max_depth)
    - Require a minimum number of samples per node (min_samples_split, min_samples_leaf)
    - Set a threshold for impurity decrease (min_impurity_decrease)

Advantage (Practical):

- **Efficiency** → The tree is smaller and faster to train/predict.
- Example: In real-time fraud detection, pre-pruning avoids building a huge tree that slows down predictions.

Post-Pruning (Cost-Complexity Pruning)

- **Definition:** First grow the tree **fully** (possibly overfitting), then **prune back branches** that do not improve performance on a validation set.
- **How:**
    - CART algorithm uses **Cost Complexity Pruning** with parameter α (controls penalty for complexity).
    - Subtrees are evaluated, and weak branches are cut.

Advantage (Practical):

- **Better Generalization** → The tree first explores all splits, then keeps only those that actually help predictive accuracy.
- Example: In medical diagnosis, post-pruning avoids relying on noise-driven splits and gives a more reliable model.

Question 4: What is Information Gain in Decision Trees, and why is it important for choosing the best split?

Answer-**1. What is Information Gain?**

- **Definition:**
  Information Gain measures the **reduction in impurity (uncertainty)** about the target variable after splitting on a feature.
    - It is based on **Entropy** (from Information Theory).

Formula

IG(S,A)=Entropy(S)-$\sum_{\mu \in values(A/0} \frac{Sv}{s} Entropy(Sv)$

- Where:
    - SSS = current dataset
    - AAA = feature being considered for split
    - SvS_vSv = subset of data where feature A=vA = vA=v
    - |Sv|/|S||S_v|/|S||Sv|/|S| = proportion of samples going into that subset

In simple words: **IG = impurity before split – weighted impurity after split**

2. Why is it important?

- A decision tree must decide: *Which feature should I split on?*
- **Information Gain tells us which feature gives the most "clarity" (reduces the most uncertainty) about the class.**
- The feature with the **highest Information Gain** is chosen at each step.

Question 5: What are some common real-world applications of Decision Trees, and what are their main advantages and limitations?

Answer-**1. Common Real-World Applications of Decision Trees**

1. **Healthcare & Medicine**
    - Disease diagnosis (e.g., predicting diabetes, cancer risk)
    - Treatment decision support systems
    - Example: Splitting based on patient's age, symptoms, and test results.
2. **Finance & Banking**
    - Credit risk assessment (approve/reject loan applications)
    - Fraud detection in transactions
    - Customer segmentation for investment advice.
3. **Marketing & Customer Analytics**
    - Predicting customer churn
    - Recommending products
    - Deciding target audience for campaigns.
4. **Human Resources (HR Analytics)**

- Predicting employee attrition
- Identifying employees eligible for promotion/bonus
- Screening job applicants.

5. **Manufacturing & Quality Control**
    - Fault detection in production lines
    - Predictive maintenance of machinery.
6. **Retail & E-commerce**
    - Product recommendation engines
    - Demand forecasting
    - Predicting if a customer will buy or not (conversion prediction).

2. Advantages of Decision Trees

- **Interpretability** → Easy to visualize and explain to non-technical people (like flowcharts).
- **Handles both categorical & numerical data** → No need for feature scaling.
- **Nonlinear relationships** → Captures complex decision boundaries.
- **Fast & efficient** → Training and prediction are relatively quick.
- **Useful for feature selection** → Splits highlight the most important variables

3. Limitations of Decision Trees

- **Overfitting** → Trees can grow too complex and capture noise (need pruning or ensemble methods like Random Forests).
- **Instability** → Small changes in data can result in very different trees.
- **Bias toward features with many categories** → A feature with many levels may dominate splits (handled better in Random Forests).
- **Not optimal for continuous prediction (regression)** → Can create stepwise predictions rather than smooth outputs.

Question 6: 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.)

Answer-# Question 6: Decision Tree Classifier on Iris Dataset

```
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

# 1. Load the Iris dataset
iris = load_iris()
X = iris.data   # features
y = iris.target # labels

# 2. Split into training & testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# 3. Train Decision Tree Classifier with Gini criterion
clf = DecisionTreeClassifier(criterion="gini", random_state=42)
clf.fit(X_train, y_train)

# 4. Make predictions
y_pred = clf.predict(X_test)

# 5. Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# 6. Get feature importances
feature_importances = clf.feature_importances_

# 7. Print results
```

```
print("Decision Tree Classifier (Gini Criterion)")

print("Accuracy on test data:", accuracy)

print("Feature Importances:")


for feature, importance in zip(iris.feature_names, feature_importances):

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

OUTPUT-

Decision Tree Classifier (Gini Criterion)

Accuracy on test data: 1.0

Feature Importances:

sepal length (cm): 0.0000

sepal width (cm): 0.0200

petal length (cm): 0.4500

petal width (cm): 0.5300

Question 7: 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.)

Answer-

```
# Question 7: Decision Tree Classifier on Iris Dataset (Depth Comparison)


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


# 1. Load the Iris dataset

iris = load_iris()

X, y = iris.data, iris.target
```

```
# 2. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# 3. 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)

# 4. Decision Tree with max_depth=3
clf_pruned = DecisionTreeClassifier(max_depth=3, random_state=42)
clf_pruned.fit(X_train, y_train)
y_pred_pruned = clf_pruned.predict(X_test)
accuracy_pruned = accuracy_score(y_test, y_pred_pruned)

# 5. Print results
print("Decision Tree Accuracy Comparison (Iris Dataset)")
print(f"Fully grown tree accuracy: {accuracy_full:.4f}")
print(f"Pruned tree (max_depth=3) accuracy: {accuracy_pruned:.4f}")
```

OUTPUT-

```
Decision Tree Accuracy Comparison (Iris Dataset)
Fully grown tree accuracy: 1.0000
Pruned tree (max_depth=3) accuracy: 0.9778
```

uestion 8: 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.)

Answer-

```
# Question 8: Decision Tree Regressor on California Housing Dataset

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

# 1. Load the California Housing dataset
housing = fetch_california_housing()
X, y = housing.data, housing.target
feature_names = housing.feature_names

# 2. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# 3. Train Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)

# 4. Predictions
y_pred = regressor.predict(X_test)

# 5. Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)

# 6. Print results
```

```
print("Decision Tree Regressor (California Housing)")
print("Mean Squared Error (MSE):", mse)
print("Feature Importances:")

for feature, importance in zip(feature_names, regressor.feature_importances_):
    print(f"{feature}: {importance:.4f}")
```

OUTPUT-

```
Decision Tree Regressor (California Housing)
Mean Squared Error (MSE): 0.2578
Feature Importances:
MedInc: 0.5442
HouseAge: 0.0511
AveRooms: 0.1224
AveBedrms: 0.0208
Population: 0.0377
AveOccup: 0.0154
Latitude: 0.1123
Longitude: 0.0961
```

Question 9: 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.)

Answer-

```
# Question 9: Hyperparameter Tuning for Decision Tree (Iris Dataset)

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
```

```
# 1. Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# 2. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# 3. Define the Decision Tree Classifier
dt = DecisionTreeClassifier(random_state=42)

# 4. Define parameter grid for tuning
param_grid = {
    "max_depth": [2, 3, 4, 5, None],
    "min_samples_split": [2, 3, 4, 5, 6, 10]
}

# 5. GridSearchCV setup (5-fold cross-validation)
grid_search = GridSearchCV(
    estimator=dt,
    param_grid=param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)

# 6. Fit GridSearchCV
```

```
grid_search.fit(X_train, y_train)

# 7. Get best model
best_dt = grid_search.best_estimator_

# 8. Evaluate on test data
y_pred = best_dt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# 9. Print results
print("Decision Tree Hyperparameter Tuning (Iris Dataset)")
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)
print("Test Accuracy with Best Parameters:", accuracy)
```

OUTPUT-

```
Decision Tree Hyperparameter Tuning (Iris Dataset)
Best Parameters: {'max_depth': 3, 'min_samples_split': 2}
Best Cross-Validation Accuracy: 0.9714
Test Accuracy with Best Parameters: 1.0000
```

Question 10: 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.

Answer-

Step 1: Handle Missing Values

1. **Explore missingness**
    - Use .isnull().sum() or missing value plots.
    - Identify if missing is random (MCAR/MAR) or systematic (MNAR).
2. **Imputation strategies**
    - **Numerical features**: Impute with mean/median, or use advanced methods (e.g., KNNImputer).
    - **Categorical features**: Impute with mode (most frequent category) or create a special category "Unknown".

In healthcare, missing data often has meaning (e.g., a lab test not done), so carefully check before imputing.

Step 2: Encode Categorical Features

- **One-Hot Encoding** → For nominal categories (e.g., blood type: A, B, AB, O).
- **Ordinal Encoding** → For ordered categories (e.g., disease stage: mild < moderate < severe).
- Decision Trees can handle categorical variables better than some models, but scikit-learn requires numerical input → encoding is necessary.

Step 3: Train a Decision Tree Model

- Split data into **train/test sets** (e.g., 70/30).
- Use DecisionTreeClassifier(criterion="gini" or "entropy").
- Fit on preprocessed training data.

Step 4: Tune Hyperparameters

- Use **GridSearchCV** or **RandomizedSearchCV** with cross-validation.
- Important parameters:
    - max_depth: Prevent overfitting
    - min_samples_split, min_samples_leaf: Control minimum samples per node/leaf
    - max_features: Limit number of features considered per split
    - ccp_alpha: Cost-complexity pruning parameter

👉 Example search grid:

```
param_grid = {
   "max_depth": [3, 5, 10, None],
   "min_samples_split": [2, 5, 10],
   "min_samples_leaf": [1, 2, 4],
   "criterion": ["gini", "entropy"]

}
```

Step 5: Evaluate Performance

1. **Classification metrics**:
    - Accuracy (overall correctness)
    - Precision (how many predicted positives are correct)
    - Recall (how many actual positives are caught)
    - F1-score (balance between precision & recall)
    - ROC-AUC (probability ranking quality, especially for imbalanced data).
2. **Confusion Matrix**: Helps understand false positives and false negatives → crucial in healthcare (false negative = missing a disease case).

Step 6: Business Value in Healthcare

- **Early diagnosis & treatment** → Predicting disease risk allows doctors to intervene earlier, improving patient outcomes.
- **Resource allocation** → Hospitals can prioritize high-risk patients (ICU beds, diagnostic tests).
- **Personalized medicine** → Suggests tailored treatments based on patient profile.
- **Cost reduction** → Avoids unnecessary tests for low-risk patients, reducing healthcare costs.