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 that models decision-making as a tree-like structure. It works by repeatedly splitting the dataset into smaller, more homogeneous groups based on feature values, enabling clear and interpretable predictions.

How a Decision Tree Works in Classification:

- Root Node:
  Represents the entire dataset and initiates the first split based on the most informative feature.
- Decision Nodes:
  Apply conditional rules (e.g., *Is Age > 30?*) to partition data, optimizing class separation using metrics such as Gini Impurity or Information Gain (Entropy).
- Leaf Nodes:
  Store the final class labels, representing the model's prediction outcome.

Operational Flow:

1. The algorithm selects the best feature to split the data based on impurity reduction.
2. Data is recursively divided into branches until a stopping condition is met (pure nodes or max depth).
3. A new data point traverses the tree based on its feature values and lands in a leaf node that defines its class.

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

Answer:

Gini Impurity and **Entropy** are **impurity measures** used in **Decision Tree algorithms** to evaluate how well a dataset is split at each node. Their core objective is to quantify **how mixed the class labels are** and guide the algorithm toward **cleaner, more informative splits**.

---

1. Gini Impurity

- Measures the **probability of incorrectly classifying** a randomly chosen data point if labels were assigned randomly based on class distribution.

- Formula:

$$\text{Gini} = 1 - \sum_{i=1}^{n} p_i^2$$

where $p_i$is the probability of class i.

Key Characteristics:

- Lower Gini value → **purer node**
- Gini = 0 → all samples belong to a single class
- Computationally efficient and widely used (default in CART)

---

2. Entropy

- Measures the **level of disorder or uncertainty** in the data.
- Formula:

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Key Characteristics:

- Entropy = 0 → completely pure node
- Higher entropy → higher randomness
- More sensitive to class distribution changes

---

3. Impact on Decision Tree Splits

- At each node, the algorithm evaluates all possible splits.
- The split that results in the **maximum reduction in impurity** is selected:
  - **Gini Gain** for Gini Impurity
  - **Information Gain** for Entropy

    $$\text{Information Gain} = \text{Parent Impurity} - \text{Weighted Child Impurity}$$

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 and **Post-Pruning** are optimization strategies used in **Decision Trees** to control model complexity and prevent overfitting. The key difference lies in **when** the tree growth is restricted.

Pre-Pruning (Early Stopping)

- Limits tree growth **during the training phase**.
- The algorithm stops splitting a node if certain conditions are met (e.g., maximum depth, minimum samples per split, minimum impurity decrease).

Practical Advantage:

- **Faster training and lower computational cost**, making it suitable for large datasets and time-sensitive modeling scenarios.

Post-Pruning (Late Pruning)

- Allows the tree to grow fully and then **prunes unnecessary branches after training** based on validation performance.
- Removes branches that do not contribute significantly to predictive accuracy.

Practical Advantage:

- **Improved generalization performance**, as pruning decisions are based on actual model behavior rather than early assumptions.

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

Answer:

Information Gain is a key metric used in **Decision Tree algorithms** to determine the **most effective feature for splitting the data** at each node. It measures how much **uncertainty (impurity)** in the dataset is reduced after performing a split on a particular feature.

Core Concept:

Information Gain is calculated using **Entropy**, which quantifies randomness in the data.

$$\text{Information Gain} = \text{Entropy (Parent)} - \sum \text{Entropy (Children)}$$

Why Information Gain Is Important:

- Identifies the feature that delivers the **maximum reduction in uncertainty**
- Ensures each split results in **more homogeneous (pure) child nodes**
- Helps build **efficient, compact, and accurate** decision trees

- Directly impacts **model performance and interpretability**

Strategic Value:

By prioritizing splits with the highest Information Gain, decision trees optimize their **decision-making pathway**, reduce overfitting risks, and accelerate convergence toward meaningful outcomes.

In summary, Information Gain acts as a **decision-quality accelerator**, enabling the model to choose splits that maximize clarity, predictive strength, and overall analytical efficiency.

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

Answer:

Decision Trees are widely adopted across industries due to their **interpretability, flexibility, and decision-centric structure**. They are used for both **classification and regression** tasks in real-world scenarios.

Common Real-World Applications of Decision Trees

1. **Finance & Banking**
   - Credit risk assessment
   - Loan approval and fraud detection
   - Helps institutions make **rule-based, auditable decisions**
2. **Healthcare**
   - Disease diagnosis and treatment recommendation
   - Patient risk stratification
   - Enables **transparent clinical decision support**
3. **Marketing & Sales**
   - Customer segmentation
   - Churn prediction and targeted campaigns
   - Drives **data-backed personalization strategies**
4. **Human Resources**
   - Employee performance evaluation
   - Attrition prediction
   - Supports **talent analytics and workforce planning**

5. **Manufacturing & Operations**
    - Fault detection and quality control
    - Process optimization
    - Improves **operational efficiency and reliability**

Main Advantages of Decision Trees

- **Easy to Understand and Interpret**
  Results are explainable to both technical and non-technical stakeholders.
- **Minimal Data Preprocessing**
  No strict requirement for data normalization or scaling.
- **Handles Both Numerical and Categorical Data**
  Highly versatile across diverse datasets.
- **Captures Non-Linear Relationships**
  Effective in modeling complex decision boundaries.

Main Limitations of Decision Trees

- **Prone to Overfitting**
  Especially when the tree becomes too deep and complex.
- **Sensitive to Small Data Changes**
  Minor variations can lead to significantly different tree structures.
- **Lower Predictive Accuracy Compared to Ensembles**
  Single trees often underperform compared to Random Forests or Gradient Boosting.
- **Bias Toward Dominant Features**
  Can favor attributes with more levels.

Dataset Info:

● Iris Dataset for classification tasks (sklearn.datasets.load_iris() or provided CSV).

● Boston Housing Dataset for regression tasks (sklearn.datasets.load_boston() or provided CSV).

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:

```
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

iris = load_iris()
X = iris.data
y = iris.target

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

model = DecisionTreeClassifier(criterion="gini", random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("Model Accuracy:", accuracy)

print("\nFeature Importances:")
for feature, importance in zip(iris.feature_names, model.feature_importances_):
    print(f"{feature}: {importance:.4f}")
```

O/P:

```
Model Accuracy: 1.0
Feature Importances:
sepal length (cm): 0.0000
sepal width (cm): 0.0167
petal length (cm): 0.9061
petal width (cm): 0.0772
```

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:

```python
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

dt_limited = DecisionTreeClassifier(max_depth=3, random_state=42)
dt_limited.fit(X_train, y_train)
y_pred_limited = dt_limited.predict(X_test)
accuracy_limited = accuracy_score(y_test, y_pred_limited)

dt_full = DecisionTreeClassifier(random_state=42)
```

```
dt_full.fit(X_train, y_train)
y_pred_full = dt_full.predict(X_test)
accuracy_full = accuracy_score(y_test, y_pred_full)

print("Decision Tree with max_depth=3 Accuracy:", accuracy_limited)
print("Fully-grown Decision Tree Accuracy:", accuracy_full)
```

O/P :

```
Decision Tree with max_depth=3 Accuracy: 1.0
Fully-grown Decision Tree Accuracy: 1.0
```

Question 8: Write a Python program to:

● Load the Boston Housing Dataset

● 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:

```
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

data = fetch_california_housing()
X = pd.DataFrame(data.data, columns=data.feature_names)
Y = data.target

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

model = DecisionTreeRegressor(random_state=42)
```

```
model.fit(X_train, Y_train)


Y_pred = model.predict(X_test)


mse = mean_squared_error(Y_test, Y_pred)


print("Mean Squared Error (MSE):", mse)
print("\nFeature Importances:")
for feature, importance in zip(X.columns, model.feature_importances_):
    print(f"{feature}: {importance:.4f}")
```

O/P:

```
Mean Squared Error (MSE): 0.495235205629094

Feature Importances:
MedInc: 0.5285
HouseAge: 0.0519
AveRooms: 0.0530
AveBedrms: 0.0287
Population: 0.0305
AveOccup: 0.1308
Latitude: 0.0937
Longitude: 0.0829
```

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:

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

dt = DecisionTreeClassifier(random_state=42)

param_grid = {
    'max_depth': [2, 3, 4, 5, None],
    'min_samples_split': [2, 3, 4, 5]
}

grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5,
scoring='accuracy')
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print("Best Parameters:", best_params)
print("Test Set Accuracy:", accuracy)
```

O/P:

```
Best Parameters: {'max_depth': 3, 'min_samples_split': 2}

Test Set Accuracy: 1.0
```

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:

1. **Handle Missing Values**

- **Identify missing data** using methods like isnull() or info().
- **Impute missing values** depending on the type of data:
  - **Numerical features:** Replace with mean, median, or use predictive imputation.
  - **Categorical features:** Replace with mode or create a new category like "Unknown."
- Optionally, **drop columns** with excessive missingness (>50%) if they are not critical.

```
from sklearn.impute import SimpleImputer


# Example: numerical imputer

num_imputer = SimpleImputer(strategy='median')

X_num = num_imputer.fit_transform(X_num)


# Example: categorical imputer

cat_imputer = SimpleImputer(strategy='most_frequent')

X_cat = cat_imputer.fit_transform(X_cat)
```

2. Encode Categorical Features

1.Convert categorical variables into numeric representations suitable for ML.

2. **One-Hot Encoding:** For nominal categories with no order.

3. Label Encoding: For ordinal categories with inherent order.

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_cat_encoded = encoder.fit_transform(X_cat)
```

3.Train a Decision Tree Model

- Split the data into **training and testing sets**.
- Initialize a **Decision Tree Classifier** and train on the preprocessed data.

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

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

model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

4. Tune Hyperparameters

- Optimize model performance and avoid overfitting using **Grid Search or Randomized Search**.
- Key hyperparameters:
  - max_depth → maximum depth of the tree
  - min_samples_split → minimum samples required to split a node
  - min_samples_leaf → minimum samples at a leaf node
  - criterion → “gini” or “entropy” for splitting

```
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(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
```

5. **Evaluate Performance**
   - Use metrics suitable for classification:
     - **Accuracy** → overall correctness
     - **Precision & Recall** → especially important for healthcare (false positives vs false negatives)
     - **F1-Score** → balance between precision and recall
     - **ROC-AUC** → probability ranking performance

```
from sklearn.metrics import classification_report, roc_auc_score

y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred))

y_prob = best_model.predict_proba(X_test)[:,1]
roc_auc = roc_auc_score(y_test, y_prob)
print("ROC-AUC:", roc_auc)
```

6. **Business Value in Real-World Healthcare**

- **Early detection:** Helps identify high-risk patients before severe symptoms appear.
- **Resource allocation:** Hospitals can prioritize testing, treatments, or specialist referrals.
- **Decision support:** Assists doctors with evidence-based risk assessment.
- **Cost optimization:** Reduces unnecessary testing and interventions while improving patient outcomes.
- **Scalability:** Can integrate with electronic health records for continuous monitoring and predictive analytics.