**Assignment Code: DA-AG-012**

# Decision Tree | **Assignment**

**Instructions:** Carefully read each question. Use Google Docs, Microsoft Word, or a similar tool to create a document where you type out each question along with its answer. Save the document as a PDF, and then upload it to the LMS. Please do not zip or archive the files before uploading them. Each question carries 20 marks.

**Total Marks**: 100

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

**Answer:**

A Decision Tree is a supervised learning algorithm used for classification and regression tasks. It works by recursively splitting the dataset into subsets based on feature values, forming a tree-like structure. Each internal node represents a decision rule (e.g., "Petal length ≤ 2.5"), branches represent outcomes, and leaf nodes represent final predictions (class labels). In classification, the tree assigns a class to each input by following the path from root to leaf.

**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: Measures the probability of incorrectly classifying a randomly chosen element.
Formula:
Gini=1-\sum _{i=1}^np_i^2

- where p_i is the probability of class i.
- Entropy: Measures the amount of disorder or uncertainty. Formula:

Entropy=-\sum _{i=1}^np_i\log _2(p_i)

- Impact: Both metrics guide the tree to choose splits that reduce impurity. Lower impurity means better separation of classes. Entropy is more sensitive to class distribution, while Gini is computationally simpler.

**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: Stops tree growth early using constraints (e.g., max depth, min samples per split).
*Advantage:* Prevents overfitting and reduces computation.

Post-Pruning: Grows the full tree first, then removes branches that don't improve performance.
*Advantage:* Produces a simpler, more generalizable model after evaluating actual performance.

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

**Answer:**

Information Gain measures the reduction in entropy after a dataset is split on a feature.

$$IG=Entropy(parent)-\sum \frac{|child|}{|parent|}\cdot Entropy(child)$$

It is important because it helps select the feature that best separates the data, ensuring the tree makes the most informative splits.

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

**Answer:**

Applications:

- Medical diagnosis
- Fraud detection
- Customer churn prediction
- Credit risk assessment
- Recommendation systems

- Advantages: Easy to interpret, handles mixed data types, requires little preprocessing.
- Limitations: Prone to overfitting, unstable with small changes in data, less effective for very complex patterns compared to ensemble methods.

***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.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

# Load dataset

iris = load_iris()

X, y = iris.data, iris.target

# Split data

X_train, X_test, y_train, y_test = train_test_split(X, y,

test_size=0.3, random_state=42

# Train Decision Tree

clf = DecisionTreeClassifier(criterion="gini", random_state=42)
```

```
clf.fit(X_train, y_train)

# Accuracy

y_pred = clf.predict(X_test)

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

# Feature Importances

print("Feature Importances:", clf.feature_importances_)
```

**Output (approx):**
Accuracy: ~0.95
Feature Importances: [0.02, 0.01, 0.45, 0.52]

**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:**

```
# Full tree
clf_full = DecisionTreeClassifier(random_state=42)
clf_full.fit(X_train, y_train)
acc_full = accuracy_score(y_test, clf_full.predict(X_test))

# Limited depth
clf_limited = DecisionTreeClassifier(max_depth=3, random_state=42)
clf_limited.fit(X_train, y_train)
acc_limited = accuracy_score(y_test, clf_limited.predict(X_test))

print("Full Tree Accuracy:", acc_full)
print("Max Depth=3 Accuracy:", acc_limited)
```

**Output (approx):**
Full Tree Accuracy: ~0.95
Max Depth=3 Accuracy: ~0.92

**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 load_boston
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Load dataset
boston = load_boston()
X, y = boston.data, boston.target

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

# Train regressor
reg = DecisionTreeRegressor(random_state=42)
reg.fit(X_train, y_train)

# Predictions
y_pred = reg.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred))
print("Feature Importances:", reg.feature_importances_)
```

Output (approx):
MSE: ~20–25
Feature Importances: varies, often "RM" (rooms) and "LSTAT" (lower status population) dominate.

**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.model_selection import GridSearchCV

param_grid = {
    "max_depth": [2, 3, 4, None],
    "min_samples_split": [2, 5, 10]
}

grid = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid, cv=5)
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best Accuracy:", grid.best_score_)
```

**Output (approx):**
Best Parameters: {'max_depth': 3, 'min_samples_split': 2}
Best Accuracy: ~0.95

**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:**

Handle Missing Values:

    - Numerical: impute with mean/median.
    - Categorical: impute with mode or "Unknown".
  - Encode Categorical Features:
    - Use one-hot encoding or label encoding.
  - Train Decision Tree Model:
    - Fit on processed dataset.
  - Tune Hyperparameters:
    - Use GridSearchCV for max_depth, min_samples_split, criterion.
  - Evaluate Performance:
    - Accuracy, Precision, Recall, F1-score, ROC-AUC.
  - Business Value:
- Helps doctors predict disease risk quickly.
- Supports preventive healthcare.
- Reduces costs by early detection.
- Improves patient outcomes and decision-making.