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 to classify data into categories.
It works like a flowchart where each node represents a decision based on a feature, each branch shows the outcome of that decision, and each leaf node represents a final class label.

How it works:

1. **The algorithm selects the feature that best splits the data (using Gini Index or Information Gain).**
2. **It divides the data into branches based on that feature's values.**
3. **This process repeats for each branch until all data is classified or a stopping condition is reached.**
4. **For a new sample, the tree is followed from root to leaf to predict the 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:

In a Decision Tree, we need to decide which feature to split on at each step.
To do that, the algorithm measures how "pure" or "impure" a node is — that means how mixed the classes are.
Two common impurity measures are Gini Impurity and Entropy.

---

1. Gini Impurity

- **It measures how often a randomly chosen element would be incorrectly classified if it was labeled according to the class distribution in the node.**
- **Formula:**
  $Gini = 1 - \sum(pi)2$
- **where pi = probability of class *i* in that node.**
- **Range: 0 to 0.5 (for binary classes)**
  - **0 → completely pure (only one class)**
  - **0.5 → maximum impurity (classes evenly mixed)**

---

2. Entropy

- **Entropy measures the amount of uncertainty or randomness in the data.**
- **Formula:**
  $Entropy = -\sum(pilog2pi)$
- **Range: 0 to 1**
  - **0 → pure node (one class only)**
  - **1 → maximum impurity (equal mix of classes)**

How They Impact Splits:

- **When building the tree, the algorithm tries to reduce impurity at each split.**
- **It calculates impurity before and after splitting a node.**

- **The feature that gives the maximum reduction in impurity (called Information Gain) is chosen for the split.**

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

Answer:

Difference between Pre-Pruning and Post-Pruning in Decision Trees

Aspect	Pre-Pruning	Post-Pruning
Meaning	**Stops the tree from growing too large during training.**	**Grows the full tree first, then removes unnecessary branches.**
When applied	**While building the tree.**	**After the complete tree is built.**
How it works	**Uses conditions like maximum depth, minimum samples per split, or minimum information gain.**	**Evaluates each branch on validation data and prunes if it doesn't improve accuracy.**
Goal	**Prevent overfitting early.**	**Simplify an already complex model.**

Practical Advantages:

- **Pre-Pruning: Saves time and computation, as the tree doesn't grow unnecessarily deep.**
- **Post-Pruning: Gives better accuracy because it first learns all patterns, then removes only the unhelpful ones.**

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

Answer:

Information Gain in Decision Trees

Information Gain (IG) is a measure used to decide which feature to split on when building a Decision Tree.
It shows how much "information" or reduction in impurity (uncertainty) is achieved after splitting the data on a particular feature.

Formula:

Information Gain=Entropy (Parent)−∑(ni/ni×Entropy (Child_i))
where nin_ini is the number of samples in each child node.

Why It's Important:

- **It helps the algorithm choose the best feature that gives the most pure (least mixed) child nodes.**
- **A higher Information Gain means a better split, leading to more accurate and efficient classification.**

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

Answer:

Real-World Applications of Decision Trees

1. **Medical Diagnosis:**
   Used to predict diseases or treatment outcomes based on patient data.
2. **Finance and Banking:**
   Used for credit scoring, loan approval, and fraud detection.
3. **Marketing:**
   Helps identify potential customers and predict buying behavior.

4. **Manufacturing:**
   Used for quality control and fault detection in production lines.

5. **Education:**
   Predicts student performance or dropout risk based on academic data.

Main Advantages

- **Easy to understand and interpret (no complex math needed).**
- **Handles both categorical and numerical data.**
- **No need for data scaling or normalization.**
- **Can capture non-linear relationships.**

Main Limitations

- **Prone to overfitting (especially with deep trees).**
- **Small data changes can alter the structure (unstable).**
- **Less accurate compared to ensemble models like Random Forests or Gradient Boosted Trees.**

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:

```
#Ans:-
# Question 6: Decision Tree Classifier on Iris Dataset

# Step 1: Import required libraries
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

# Step 2: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 3: Split the data 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
)

# Step 4: Create and train the Decision Tree Classifier using Gini
criterion
clf = DecisionTreeClassifier(criterion='gini', random_state=42)
clf.fit(X_train, y_train)

# Step 5: Make predictions
y_pred = clf.predict(X_test)

# Step 6: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)

# Step 7: Display feature importances
```

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

#output:-

Model 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
```

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:

```
#Ans:-
```

```
# Step 1: Import required libraries

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



# Step 2: Load the Iris dataset

iris = load_iris()
```

```
X = iris.data

y = iris.target



# Step 3: Split the 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

)



# Step 4: Train a Decision Tree with max_depth=3

tree_limited = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=42)

tree_limited.fit(X_train, y_train)

y_pred_limited = tree_limited.predict(X_test)

accuracy_limited = accuracy_score(y_test, y_pred_limited)



# Step 5: Train a fully-grown Decision Tree (no depth limit)

tree_full = DecisionTreeClassifier(criterion='gini', random_state=42)

tree_full.fit(X_train, y_train)

y_pred_full = tree_full.predict(X_test)

accuracy_full = accuracy_score(y_test, y_pred_full)



# Step 6: Compare results

print("Accuracy with max_depth=3:", accuracy_limited)
```

```
print("Accuracy with fully-grown tree:", accuracy_full)



#output:-

Accuracy with max_depth=3: 1.0

Accuracy with fully-grown tree: 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:

```
#Ans:-

# Step 1: Import required libraries
from sklearn.datasets import fetch_openml
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Step 2: Load the Boston Housing dataset
# (Note: load_boston() is deprecated, so we use fetch_openml instead)
boston = fetch_openml(name="boston", version=1, as_frame=True)
X = boston.data
y = boston.target

# Step 3: Split the 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
)

# Step 4: Train a Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)

# Step 5: Make predictions
y_pred = regressor.predict(X_test)

# Step 6: Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# Step 7: Display feature importances
print("\nFeature Importances:")
for feature, importance in zip(X.columns,
regressor.feature_importances_):
    print(f"{feature}: {importance:.4f}")
#output
Mean Squared Error (MSE): 11.588026315789474

Feature Importances:
CRIM: 0.0585
ZN: 0.0010
INDUS: 0.0099
CHAS: 0.0003
NOX: 0.0071
RM: 0.5758
AGE: 0.0072
DIS: 0.1096
RAD: 0.0016
TAX: 0.0022
PTRATIO: 0.0250
B: 0.0119
LSTAT: 0.1900
```

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:

```
#Ans:-
# Step 1: Import required libraries
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score

# Step 2: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 3: Split the 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
)

# Step 4: Define the Decision Tree model
dt = DecisionTreeClassifier(random_state=42)

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

# Step 6: Apply GridSearchCV
grid_search = GridSearchCV(
    estimator=dt,
    param_grid=param_grid,
    cv=5,              # 5-fold cross-validation
```

```
    scoring='accuracy'
)

grid_search.fit(X_train, y_train)

# Step 7: Print best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

# Step 8: Evaluate the best model on the test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Test Set Accuracy:", accuracy)


#output:-
Best Parameters: {'max_depth': 4, 'min_samples_split': 6}
Best Cross-Validation Accuracy: 0.9428571428571428
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:

```
# #Ans:-
# 1. Handle Missing Values:

# For numerical data → fill with mean/median.

# For categorical data → fill with most frequent or a new "missing"
category.

# 2. Encode Categorical Features:

# Use LabelEncoder or OneHotEncoder to convert text data into numbers.

# 3. Train Decision Tree Model:

# Split data into train and test sets.

# Train using DecisionTreeClassifier() and fit it on the training data.

# 4. Tune Hyperparameters:

# Use GridSearchCV to find the best max_depth, min_samples_split, etc.

# Choose the model with the highest cross-validation accuracy.

# 5. Evaluate Performance:

# Use metrics like accuracy, precision, recall, F1-score, and confusion
matrix on the test data.
```