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 type of supervised machine learning algorithm that is commonly used for **classification** and **regression** tasks. In the context of classification, it works by breaking down a dataset into smaller and smaller subsets based on feature values, forming a tree-like structure.

Here's how it works step by step:

1. **Root Node:** The process starts with the entire dataset at the root. The algorithm looks for the feature that best splits the data into different classes.
2. **Splitting:** Based on some criteria like **Gini Impurity**, **Entropy (Information Gain)**, or **Chi-square**, the dataset is divided into branches. Each branch represents a possible outcome of a decision based on a particular feature.
3. **Decision Nodes and Leaf Nodes:** As the tree grows, internal nodes represent decisions (tests on attributes), and the final nodes, called **leaf nodes**, represent the predicted class labels.
4. **Prediction:** When a new data point is introduced, it travels down the tree by following the decisions at each node until it reaches a leaf node, where the class label is assigned.

Decision Trees are easy to interpret and visualize, but they can sometimes overfitting the training data if not properly pruned or regularized.

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 two common measures used to determine how "pure" or "impure" a node is in a Decision Tree — that is, how mixed the classes are within that node. Both help the algorithm decide which feature and threshold to use when splitting the data.

Here's what they mean:

1. Gini Impurity

- **Definition:** Gini Impurity measures how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the class distribution in that node.
- **Formula:**

$$Gini = 1 - \Sigma(p_i)^2$$

where pi is the probability of a sample belonging to class i,

Interpretation:

  - Gini = 0 → the node is *pure* (all samples belong to one class).
  - Higher Gini → the node is more *impure* (mixed classes).

2. Entropy (Information Gain)

- **Definition:** Entropy measures the amount of randomness or disorder in the data. It comes from information theory and quantifies the uncertainty of a random variable.
- **Formula:**

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

- **Interpretation:**
  - Entropy = 0 → completely pure node.
  - Higher entropy → higher uncertainty or class mixture.

Impact on Splits

When building a Decision Tree:

- The algorithm evaluates all possible splits based on these impurity measures.
- For each possible split, it calculates the **reduction in impurity** — known as **Information Gain** (for Entropy) or **Gini Gain** (for Gini).
- The split that results in the **largest reduction in impurity** is chosen as the best split.

In simple terms, both Gini Impurity and Entropy help the tree figure out where to split so that each branch becomes as "pure" as possible — meaning, each branch mostly contains data from a single class.

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

Answer:

1. Pre-Pruning (Early Stopping)

Definition:
Pre-pruning stops the tree from growing too deep by setting certain conditions during the construction phase. These conditions might include:

- A maximum depth for the tree
- A minimum number of samples required to split a node
- A minimum information gain required for a split

Example:
If the information gain from a potential split is very small, the algorithm won't perform the split and will stop at that point.

Practical Advantage:
Faster training and simpler trees.
Because it limits growth early on, pre-pruning saves computational time and produces more interpretable models.

---

2. Post-Pruning (Pruning After Training)

Definition:
Post-pruning allows the tree to grow fully first (possibly overfitting), and then it removes branches that do not contribute significantly to the model's performance — usually based on validation set performance or cost-complexity pruning.

Example:
After building the full tree, branches that add little improvement to accuracy are trimmed back.

Practical Advantage:
Better generalization.
Post-pruning usually results in a model that performs better on unseen data, as it carefully removes only the unnecessary complexity after evaluating the entire structure.

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

Answer:

Information Gain is the **difference** between the impurity of the parent node and the **weighted average impurity** of the child nodes after a split.

Information Gain=Impurity (Parent)−∑ni/ni×Impurity (Child)

where:

- ni = number of samples in the i^th child node
- n = total number of samples in the parent node
- Impurity can be measured using **Entropy** or **Gini Impurity**

Intuition:

- A **high Information Gain** means the feature split leads to **purer** subsets — meaning the data in each child node is more homogeneous.
- A **low Information Gain** means the feature doesn't help much in distinguishing between classes.

Intuition:

- A **high Information Gain** means the feature split leads to **purer** subsets — meaning the data in each child node is more homogeneous.
- A **low Information Gain** means the feature doesn't help much in distinguishing between classes.

Importance for Choosing the Best Split:

Information Gain is used as the **splitting criterion** in Decision Trees (especially in algorithms like ID3 and C4.5).

- The feature that provides the **highest Information Gain** is chosen for splitting at that node.
- This ensures the tree is built in a way that **maximizes class separation** at each step.

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

Answer:

Common Applications:

- **Healthcare:** Predicting diseases or patient outcomes.
- **Finance:** Credit scoring, loan approval, fraud detection.
- **Marketing:** Customer segmentation and churn prediction.
- **Manufacturing:** Quality control and fault detection.
- **HR:** Predicting employee turnover or performance.

Advantages:

- Simple and easy to interpret.
- Works with both numerical and categorical data.
- No need for data scaling.
- Captures non-linear relationships.

Limitations:

- Can **overfit**ting easily.
- **Sensitive** to small data changes.
- May be **biased** toward features with many categories.

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:

```
import numpy as np
import pandas as pd

from sklearn.datasets import load_iris
data = load_iris()
df = pd.DataFrame(data = data.data, columns=data.feature_names)

df.columns

df['target'] = data.target

df.columns

# spliting X and y
X = df.iloc[:,:-1]
y = df.iloc[:,-1]

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y
,test_size=0.30,random_state=1)

# decision Tree
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(criterion="gini")
dt_gini_model = dt_model.fit(X_train,y_train)
dt_gini_model

# prediction
```

```
y_pred = dt_gini_model.predict(X_test)

# accuracy score
from sklearn.metrics import accuracy_score
print("1) The Accuracy Score is ",accuracy_score(y_test,y_pred))

# Print feature importances
feature_importances = pd.DataFrame({'feature': X.columns, 'importance':
dt_gini_model.feature_importances_})
feature_importances = feature_importances.sort_values('importance',
ascending=False)
print('\n')
print("2) Feature Importances:")
print(feature_importances)

# output
1) The Accuracy Score is  0.9555555555555556


2) Feature Importances:
             feature  importance
2  petal length (cm)    0.571965
3   petal width (cm)    0.385096
1   sepal width (cm)    0.021469
0  sepal length (cm)    0.021469
```

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:

```
import numpy as np
import pandas as pd

# loading dataset
from sklearn.datasets import load_iris
data = load_iris()
```

```
data.target
df = pd.DataFrame(data= data.data, columns=data.feature_names)
df['target'] = data.target
df.columns

# spliting
X = df.iloc[:,:-1]
y = df.iloc[:,-1]

# train_test_split
from sklearn.model_selection import train_test_split
X_train,X_test, y_train,y_test =
train_test_split(X,y,test_size=0.30,random_state=1)

# import model
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='gini',max_depth=3)
model = model.fit(X_train,y_train)
model

# prediction
y_pred = model.predict(X_test)

# metrics
from sklearn.metrics import accuracy_score
print(f"The Accuracy score of Pre-Pruning Model
(max_depth={model.max_depth}) -",accuracy_score(y_pred,y_test))

# full grown decision tree
full_model = DecisionTreeClassifier(criterion='gini')
full_grown_model = full_model.fit(X_train,y_train)

# prediction of full grown model
y_pred_full = full_grown_model.predict(X_test)

# accuracy score without any pruning
print(f"The Accuracy score of full grown model
(max_depth={full_grown_model.max_depth}) -
",accuracy_score(y_pred_full,y_test))

#output
```

```
The Accuracy score of Pre-Pruning Model (max_depth=3) - 0.9555555555555556
The Accuracy score of full grown model (max_depth=None) - 0.9555555555555556
```

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:

```python
import numpy as np
import pandas as pd

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

# Load the California Housing Dataset (as Boston Housing is deprecated)
housing = fetch_california_housing(as_frame=True)
X = housing.data
y = housing.target

# Split 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)

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

# Make predictions
y_pred = dt_regressor.predict(X_test)

# Print the Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
```

```
print(f"Mean Squared Error: {mse:.2f}")

# Print feature importances
feature_importances = pd.DataFrame({
    'feature': X.columns,
    'importance': dt_regressor.feature_importances_
})
feature_importances = feature_importances.sort_values('importance',
ascending=False)
print("\nFeature Importances:")
print(feature_importances)

#output
Mean Squared Error: 0.53

Feature Importances:
      feature  importance
0      MedInc    0.523456
5    AveOccup    0.139012
6    Latitude    0.089992
7   Longitude    0.088806
1    HouseAge    0.052135
2    AveRooms    0.049418
4  Population    0.032206
3   AveBedrms    0.024974
```

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:

```python
import pandas as pd
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

# Load the Iris Dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)

# 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)

# Define the parameter grid for GridSearchCV
param_grid = {
    'max_depth': [2, 3, 4, 5, None],  # None means no limit on depth
    'min_samples_split': [2, 5, 10, 15]
}

# Initialize a Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=dt_classifier,
param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit GridSearchCV to the training data
grid_search.fit(X_train, y_train)

# Print the best parameters
print("Best Parameters found by GridSearchCV:")
print(grid_search.best_params_)

# Get the best model
best_dt_model = grid_search.best_estimator_

# Make predictions on the test set with the best model
y_pred = best_dt_model.predict(X_test)
```

```python
# Print the accuracy of the best model
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy of the best Decision Tree Model: {accuracy:.4f}")

#output
Best Parameters found by GridSearchCV:
{'max_depth': 4, 'min_samples_split': 10}

Accuracy of the best Decision Tree Model: 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

- Check how much and why data is missing.
- **Numerical:** fill with median or mean values.
- **Categorical:** fill with the mode or add a "Missing" category.
- Add a "missing indicator" column if missingness might be meaningful.

Step 2: Encode Categorical Features

- **Low-cardinality:** one-hot encoding.
- **Ordinal:** label encoding (preserve order).
- **High-cardinality:** frequency or target encoding (done carefully to avoid leakage).

Step 3: Train a Decision Tree Model

- Split data into **train/test** (stratified if classes are imbalanced).
- Use `DecisionTreeClassifier()` with `class_weight='balanced'` if needed.
- Fit the model on the training data using a pipeline (imputation → encoding → model).

Step 4: Tune Hyperparameters

- Use **cross-validation** with **GridSearchCV** or **RandomizedSearchCV**.
- Key parameters:
  - `max_depth`, `min_samples_split`, `min_samples_leaf`, `max_features`, `ccp_alpha`.
- Aim for the best trade-off between accuracy and simplicity.

Step 5: Evaluate Performance

- Use metrics like **Accuracy**, **Precision**, **Recall**, **F1-score**, **ROC-AUC**.
- Focus on **Recall** if missing a disease case is costly.
- Check **confusion matrix** and **feature importance** for interpretability.

Business Value

- **Early disease detection:** helps doctors identify high-risk patients faster.
- **Better resource allocation:** reduces unnecessary tests and costs.
- **Improved decision support:** gives interpretable, evidence-based insights for clinicians.