Assignment Code: DA-AG-012 — Decision Tree | Completed

Instructions: This document contains each question followed by the answer and example Python code where requested. Save or download this page as a PDF to submit to your LMS.

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 model used for classification and regression that models decisions using a tree-like structure. In classification, a decision tree splits the feature space into regions by asking a sequence of simple questions (tests) about feature values. Each internal node represents a test on a single feature, each branch corresponds to an outcome of the test, and each leaf node represents a class label (or a distribution over classes).

How it works (high-level):

1. Start at the root node with the full dataset.
2. Select the best feature and threshold to split the data into two (or more) groups so that each group is purer (i.e., samples more similar in label) than before.
3. Recursively repeat splitting on child nodes until a stopping condition is met (e.g., maximum depth, minimum samples per leaf, or pure leaf nodes).
4. For prediction, traverse the tree from the root to a leaf following decisions at each node. The leaf's class (or majority class) is the predicted label.

Decision trees are intuitive, produce human-readable rules, and can handle both numerical and categorical features.

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

Answer: Both **Gini Impurity** and **Entropy** (from information theory) quantify the impurity or disorder of a set of class labels in a node. A pure node (all samples same class) has impurity 0.

- **Gini Impurity** for a node with class probabilities p_k : $Gini = 1 - \sum_k p_k^2$ It measures the probability of incorrectly classifying a randomly chosen element if it were labeled according to the node's class distribution.

- **Entropy** (Shannon entropy): $Entropy = -\sum_k p_k \log_2 p_k$ Entropy measures the average information (in bits) needed to identify the class of a randomly drawn example from the node.

Impact on splits: During tree building, algorithms evaluate candidate splits by computing the impurity reduction. The split that yields the greatest reduction in impurity (weighted by child node sizes) is chosen.

- **Information Gain** uses entropy: `IG = Entropy(parent) - weighted_avg(Entropy(children))`.
- **Gini gain** or impurity reduction with Gini uses analogous formula.

In practice, Gini and Entropy often choose similar splits. Gini is slightly faster to compute and may prefer larger, purer partitions; entropy is more sensitive to distribution changes and has an information-theoretic interpretation. Differences in final trees are usually minor.

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):** Stop the tree growth early using constraints such as `max_depth`, `min_samples_split`, `min_samples_leaf`, or minimum impurity decrease. The tree is never allowed to grow beyond these constraints.

- *Advantage:* Reduces overfitting and training time by keeping the tree smaller and simpler from the start.

- **Post-Pruning (Tree Pruning after Growth):** Grow a full (or large) tree first, then prune back nodes (replace subtrees with leaves) based on validation set performance or cost-complexity criteria (e.g., cost-complexity pruning — `ccp_alpha` in scikit-learn).

- *Advantage:* Often yields a more accurate model because pruning decisions are based on actual validation performance and allow complex interactions to be discovered before removing those that don't generalize.

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

Answer: Information Gain (IG) is the reduction in entropy (uncertainty) achieved by partitioning the data according to a particular feature. Formally:

$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$

where S is the parent node's samples and S_v are child subsets after splitting on feature A.

Importance: IG measures how well a feature separates the classes. A higher IG indicates that the feature provides more information about the class label and is therefore a better choice for splitting. Decision tree algorithms typically pick the feature (and threshold) that maximizes impurity reduction (IG for entropy or Gini reduction using the Gini index).

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

Answer: Applications:

- Medical diagnosis (classifying disease vs. no disease)
- Credit scoring and loan approval
- Customer churn prediction
- Fraud detection (as part of ensemble models)
- Marketing segmentation and rule-based targeting
- Manufacturing (fault detection)

Advantages:

- Easy to interpret and visualize (business-friendly rules).
- Handles both numerical and categorical data.
- Requires little data preprocessing (no need for feature scaling; can handle missing values in some implementations).
- Fast for prediction and can model non-linear relationships.

Limitations:

- Prone to overfitting if not pruned or regularized.
- Can be unstable: small changes in data can produce different trees.
- Greedy splitting can miss globally optimal trees.
- Typically outperformed by ensembles (Random Forest, Gradient Boosting) on prediction accuracy in many tasks.

Dataset Info

- **Iris** dataset for classification: `sklearn.datasets.load_iris()`.
- **Boston Housing** dataset for regression: historically `sklearn.datasets.load_boston()` (deprecated in newer sklearn versions). The assignment expects Boston; code below includes fallbacks/comments for environments where `load_boston` is unavailable.

Question 6: Python program — Load Iris, Train Decision Tree (Gini), Print accuracy and feature importances

Answer (code):

```
# Q6: Decision Tree Classifier on Iris (Gini)
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

# Load data
iris = load_iris()
X, y = iris.data, iris.target

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

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

# Predict and evaluate
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print(f"Test accuracy: {acc:.4f}")

# Feature importances
for name, importance in zip(iris.feature_names, clf.feature_importances_):
    print(f"{name}: {importance:.4f}")

# Sample output (will vary slightly with random_state and scikit-learn version):
# Test accuracy: 1.0000
# sepal length (cm): 0.0000
# sepal width (cm): 0.0000
# petal length (cm): 0.5300
# petal width (cm): 0.4700
```

(The printed sample output above is illustrative; actual values depend on the split and sklearn version.)

Question 7: Python program — Iris, Decision Tree with max_depth=3, compare accuracy to fully-grown tree

Answer (code):

```
# Q7: Compare max_depth=3 vs fully-grown tree
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, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

# Fully-grown (default) tree
clf_full = DecisionTreeClassifier(criterion='gini', random_state=42)
clf_full.fit(X_train, y_train)
acc_full = accuracy_score(y_test, clf_full.predict(X_test))

# Limited-depth tree
clf_depth3 = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=42)
clf_depth3.fit(X_train, y_train)
acc_depth3 = accuracy_score(y_test, clf_depth3.predict(X_test))

print(f"Fully-grown test accuracy: {acc_full:.4f}")
print(f"max_depth=3 test accuracy: {acc_depth3:.4f}")

# Sample possible output:
# Fully-grown test accuracy: 1.0000
# max_depth=3 test accuracy: 1.0000

# Note: On Iris dataset, both may perform similarly because the problem is
relatively simple.
```

Question 8: Python program — Load Boston Housing, Train Decision Tree Regressor, Print MSE and feature importances

Answer (code):

```python
# Q8: Decision Tree Regressor on Boston Housing
import warnings
warnings.filterwarnings('ignore')

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

# Loading Boston dataset with fallback
try:
    # Older sklearn versions
    from sklearn.datasets import load_boston
    data = load_boston()
    X, y = data.data, data.target
    feature_names = data.feature_names
except Exception:
    # If load_boston unavailable (deprecated), try fetch_openml (may not work
offline)
    try:
        from sklearn.datasets import fetch_openml
        boston = fetch_openml(name='boston', version=1, as_frame=False)
        X, y = boston.data, boston.target
        feature_names = boston.feature_names
    except Exception:
        raise RuntimeError("Boston dataset not available in this environment.
Please provide CSV or use an alternative dataset (e.g., California housing).")

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

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

y_pred = reg.predict(X_test)
mse = mean_squared_error(y_test, y_pred)

print(f"Test MSE: {mse:.4f}")

# Feature importances
for name, imp in zip(feature_names, reg.feature_importances_):
    print(f"{name}: {imp:.4f}")

# Sample output (illustrative):
# Test MSE: 24.1203
# CRIM: 0.0456
# ZN: 0.0000
# ...
```

(If the environment lacks `** and cannot fetch online, substitute with another regression dataset such as **` *.)*

Question 9: Python program — Tune Decision Tree's max_depth and min_samples_split using GridSearchCV on Iris

Answer (code):

```
# Q9: GridSearchCV for Decision Tree hyperparameters on Iris
from sklearn.datasets import load_iris
from sklearn.model_s
```