Name : Sagar Sidhu

Email : sagarsidhu33321@gmail.com

Assignment Name : Decision Tree

Drive Link : N/A

Github Link : N/A

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 also regression) that makes predictions by splitting data into subsets based on feature values.

Here's how it works in **classification**:

1. **Root Node** – The tree starts with the entire dataset at the top (root).
2. **Splitting** – At each node, the algorithm chooses the feature and threshold that best separate the classes, often using metrics like:
    - **Gini Impurity**
    - **Entropy / Information Gain**
3. **Branches** – Each split creates branches leading to new subsets of data.
4. **Leaf Nodes** – Splitting continues until a stopping condition is met (e.g., max depth, minimum samples per node, or pure class labels). Each leaf node represents a **predicted class**.
5. **Prediction** – To classify a new data point, the model follows the path from the root to a leaf based on its feature values.

Example:
If you were classifying whether a person will buy a product:

- First split: "Age ≤ 30?"
- If **yes**, check "Income ≤ 50k?"
- If **no**, check "Has Credit Card?"
  …and so on, until a final **Yes/No** decision.

Advantages: Easy to interpret, handles both numerical and categorical data, no need for feature scaling.
Limitations: Can overfit if not pruned or regularized.

If you want, I can also include a **simple diagram** showing the classification process—it makes the idea much clearer.

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 measures of how "mixed" the classes are in a node of a Decision Tree. They help the tree decide the **best feature and threshold** for splitting the data.

---

1. Gini Impurity

- **Definition:** Probability of incorrectly classifying a randomly chosen sample from the node if it were labeled randomly according to the class distribution in that node.
- **Formula:**

$$Gini=1-\sum_{i=1}^{C} pi^2$$

where pip_ipi is the proportion of samples of class iii in the node, and CCC is the number of classes.

- **Interpretation:**
    - **0** → Node is *pure* (only one class).
    - **Max** (≈ 0.5 for binary classes) → Classes are evenly mixed.

Example:
If a node has 80% Class A and 20% Class B:

$$Gini=1-(0.8^2+0.2^2)=1-(0.64+0.04)=0.32$$

---

2. Entropy (Information Gain)

- **Definition:** Measures the amount of disorder or uncertainty in a node.
- **Formula:**

$$Entropy=-\sum_{i=1}^{C} pi \log_2(pi)$$

- **Interpretation:**
    - **0** → Node is pure.

- **Max** (1 for binary classes) → Classes are evenly mixed.

Example:
For the same node (80% A, 20% B):

$$\text{Entropy} = -[0.8\log_2(0.8) + 0.2\log_2(0.2)] \approx 0.72$$

Impact on Decision Tree Splits

- At each split, the Decision Tree algorithm:
    1. Calculates Gini or Entropy for the parent node.
    2. Splits the data on each possible feature & threshold.
    3. Computes the weighted average impurity of child nodes.
    4. Chooses the split that **maximizes impurity reduction**:
        - **For Gini:** called **Gini Gain**.
        - **For Entropy:** called **Information Gain**.

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 techniques to prevent a Decision Tree from **overfitting** by controlling its growth.

---

1. Pre-Pruning (Early Stopping)

- **Definition:** Stop the tree from growing beyond a certain point during training.
- **How it works:** Apply constraints such as:
    - Maximum depth (max_depth)
    - Minimum samples to split (min_samples_split)
    - Minimum samples in a leaf (min_samples_leaf)
    - Minimum impurity decrease
- **Advantage:**
  ✅ **Faster training & simpler models** — saves computation time and produces a smaller, more interpretable tree.

---

2. Post-Pruning (Prune After Growing)

- **Definition:** First grow the full tree, then remove branches that do not provide significant predictive power.
- **How it works:**
  - Build the full tree until pure or until all leaves meet stopping criteria.
  - Use **cost complexity pruning** (e.g., CART's ccp_alpha) or validation set performance to cut unnecessary branches.
- **Advantage:**
  ✅ **Better generalization** — starts with a complete model and removes only the parts that harm performance on unseen data.

Quick Analogy:

- **Pre-Pruning** → "Don't overpack for the trip in the first place."
- **Post-Pruning** → "Pack everything, then remove what you don't need."

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

Information Gain (IG) is a metric used in Decision Trees (especially when using **Entropy** as the impurity measure) to decide **which feature and threshold to split on**. It measures how much **uncertainty** (entropy) is reduced after a split.

Formula

IG=Entropyparent−Weighted Average EntropychildrenIG

Where:

- **Entropy_parent** → impurity before splitting.
- **Weighted Average Entropy_children** → impurity after splitting, weighted by the proportion of samples in each child node.

Why It's Important

- Decision Trees aim to make nodes **pure** (samples in a node belong mostly to one class).
- A **high Information Gain** means the split reduces uncertainty a lot → children nodes are more homogeneous.
- The algorithm chooses the split with **maximum IG** at each step, ensuring the most informative features are used early in the tree.

Example:
Suppose a node has 50% Class A and 50% Class B (Entropy = 1).
After splitting:

- Left node: 80% A, 20% B (Entropy ≈ 0.72)
- Right node: 90% B, 10% A (Entropy ≈ 0.47)

Weighted average after split: ≈ 0.63

IG=1−0.63=0.37IG = 1 - 0.63 = 0.37IG=1−0.63=0.37

This split reduces uncertainty by **0.37**, so it's a strong candidate.

In short:

- **High IG → Better split** (more separation between classes).
- **Low IG → Poor split** (classes still mixed).

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

Answer:

Common Real-World Applications of Decision Trees

---

1. Finance & Banking

- **Applications:**
    - Credit scoring (approve/deny loans)
    - Fraud detection
- **Example:** Classifying customers as "High Risk" or "Low Risk" based on income, payment history, and credit score.

---

2. Healthcare

- **Applications:**
    - Disease diagnosis
    - Predicting patient outcomes
- **Example:** Classifying if a patient is at risk for diabetes based on blood sugar levels, BMI, and age.

---

3. Marketing & Sales

- **Applications:**
    - Customer segmentation
    - Predicting purchase likelihood
- **Example:** Deciding which customers are most likely to respond to a marketing campaign.

---

4. Manufacturing & Quality Control

- **Applications:**
    - Detecting defects in production
    - Predictive maintenance
- **Example:** Predicting machine failure from sensor data.

---

5. Education

- **Applications:**
    - Predicting student performance
    - Identifying students at dropout risk

Main Advantages

1. **Easy to Understand & Interpret** – Looks like a flowchart, no complex math for end users.

2. **Handles Numerical & Categorical Data** – Can split on both types without preprocessing.
3. **No Need for Feature Scaling** – Works without normalization or standardization.
4. **Captures Nonlinear Relationships** – Can model complex decision boundaries.

Main Limitations

1. **Overfitting** – Can create overly complex trees if not pruned.
2. **Instability** – Small data changes can lead to a very different tree.
3. **Bias Toward Features with Many Levels** – Features with many unique values might get chosen more often.
4. **Lower Predictive Accuracy (Alone)** – Often less accurate than ensemble methods like Random Forests or Gradient Boosting.

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=

```python
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
import pandas as pd

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

# Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Create Decision Tree Classifier with Gini criterion
clf = DecisionTreeClassifier(criterion="gini", random_state=42)
clf.fit(X_train, y_train)

# Predict on test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Feature importances
feature_importances = pd.Series(
    clf.feature_importances_, index=iris.feature_names
)

# Output results
print("Decision Tree Classifier (Gini Criterion)")
print("Accuracy on test set:", accuracy)
print("\nFeature Importances:")
```

```
Decision Tree Classifier (Gini Criterion)
Accuracy on test set: 1.0

Feature Importances:
sepal length (cm)    0.000000
sepal width (cm)     0.016670
petal length (cm)    0.906143
petal width (cm)     0.077186
dtype: float64
```

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
# Import libraries
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 the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Fully-grown Decision Tree
full_tree = DecisionTreeClassifier(random_state=42)
full_tree.fit(X_train, y_train)
y_pred_full = full_tree.predict(X_test)
acc_full = accuracy_score(y_test, y_pred_full)

# Decision Tree with max_depth=3
pruned_tree = DecisionTreeClassifier(max_depth=3, random_state=42)
pruned_tree.fit(X_train, y_train)
y_pred_pruned = pruned_tree.predict(X_test)
acc_pruned = accuracy_score(y_test, y_pred_pruned)

# Output results
print("Decision Tree Accuracy Comparison:")
print(f"Fully-grown Tree Accuracy: {acc_full:.4f}")
print(f"Max Depth = 3 Tree Accuracy: {acc_pruned:.4f}")
```

```
Decision Tree Accuracy Comparison:
Fully-grown Tree Accuracy: 1.0000
Max Depth = 3 Tree Accuracy: 1.0000
```

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
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import pandas as pd

# Load data
boston = fetch_openml(name="boston", version=1, as_frame=True)
X, y = boston.data, boston.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Results
mse = mean_squared_error(y_test, reg.predict(X_test))
print(f"MSE: {mse:.4f}")
print("Feature Importances:\n", pd.Series(reg.feature_importances_, index=X.columns))
```

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

# Load data
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=0.2, random_state=42)

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

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

# Best parameters & accuracy
best_params = grid.best_params_
best_model = grid.best_estimator_
accuracy = accuracy_score(y_test, best_model.predict(X_test))

print("Best Parameters:", best_params)
print(f"Accuracy: {accuracy:.4f}")
```

```
Best Parameters: {'max_depth': 4, 'min_samples_split': 2}
Accuracy: 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=

1. Handle Missing Values

- **Numerical Features:** Impute using **mean** or **median** (median is safer for skewed data).
- **Categorical Features:** Impute using **most frequent category** or create a new category like "Unknown".

- Use SimpleImputer in sklearn to automate this process.

---

2. Encode Categorical Features

- **Ordinal Data:** Use **Ordinal Encoding** if categories have a natural order.
- **Nominal Data:** Use **One-Hot Encoding** to avoid implying an order.
- Implement with OneHotEncoder or ColumnTransformer to process only categorical columns.

---

3. Train a Decision Tree Model

- Split data into **train** and **test** sets (e.g., 80/20).
- Use DecisionTreeClassifier from sklearn with initial default parameters.
- Fit the model on the training set.

---

4. Tune Hyperparameters

- Use **GridSearchCV** or **RandomizedSearchCV** to search over:
    - max_depth – to control tree complexity.
    - min_samples_split / min_samples_leaf – to prevent overfitting.
    - criterion – choose between "gini" and "entropy".
- Use cross-validation to select the best parameters.

---

5. Evaluate Performance

- Metrics for classification:
    - **Accuracy** for balanced datasets.
    - **Precision, Recall, F1-score** for imbalanced datasets.
    - **ROC-AUC** for overall discriminatory power.
- Check **confusion matrix** to see misclassification patterns.

---

6. Business Value in Healthcare

- **Early Detection:** Helps identify patients at high risk before symptoms worsen.
- **Resource Allocation:** Prioritize testing or treatment for likely positive cases.
- **Cost Reduction:** Avoid unnecessary expensive diagnostic procedures for low-risk patients.
- **Personalized Care:** Tailor preventive measures and monitoring to patient risk levels.