Assignment Code: DA-AG-012

Decision Tree | Assignment

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 that predicts an output by following a series of decision rules based on the values of input features. In **classification**, it's used to assign data points into predefined categories.

How it works in classification:

1. **Root node creation** – The algorithm starts with the entire dataset at the root node.
2. **Best split selection** – At each node, it evaluates all features and possible split points using an impurity measure such as **Gini Impurity** or **Entropy**. The split that results in the most "pure" child nodes is chosen.
3. **Recursive splitting** – The dataset is divided into smaller subsets, and this process is repeated for each subset, creating branches of the tree.
4. **Stopping criteria** – Splitting stops when a node becomes pure (all samples from one class) or when other conditions are met (e.g., maximum depth, minimum samples per node).
5. **Prediction** – To classify a new observation, the model starts at the root and follows the decision rules until it reaches a leaf node. The majority class in that leaf is the prediction.

Example: If `petal length ≤ 2.45 cm` → classify as *Setosa*. Else if `petal width ≤ 1.75 cm` → classify as *Versicolor*. Otherwise → classify as *Virginica*.

Advantages: simple to understand, interpretable, handles numeric & categorical data. **Limitation:** prone to overfitting if not properly controlled.

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

Answer:

1. Gini Impurity

- Measures how often a randomly chosen element from the node would be incorrectly classified if it was randomly labeled according to the distribution of labels in the node.
- Formula:

$$ Gini = 1 - \sum_{k=1}^{K} p_k^2 $$

where p_k is the proportion of samples belonging to class k in the node.

- **Range:** 0 (pure node) to a maximum value when classes are evenly distributed (e.g., 0.5 for 2 classes at 50%-50%).

2. Entropy

- Comes from information theory; measures the amount of disorder or uncertainty in the node.
- Formula:

$$ Entropy = - \sum_{k=1}^{K} p_k \log_2(p_k) $$

where p_k is the proportion of samples in class k.

- **Range:** 0 (pure node) to $\log_2(K)$ for perfectly mixed classes.

Impact on Decision Tree splits:

- At each node, the algorithm tests possible splits and calculates the weighted average impurity (Gini or Entropy) of the resulting child nodes.
- The **best split** is the one with the **largest impurity reduction**:

$$ \text{Impurity Reduction} = Impurity_{parent} - \left(\frac{n_L}{n_{total}} Impurity_L + \frac{n_R}{n_{total}} Impurity_R \right) $$

- Gini and Entropy usually select similar splits, but:
    - **Gini** is slightly faster to compute and tends to favor the most frequent class.
    - **Entropy** is more sensitive to class imbalance and can result in slightly different splits in some cases.

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)

- The tree's growth is **stopped early** based on certain conditions **before** it becomes overly complex.
- Common stopping criteria:
    - Maximum depth (`max_depth`)
    - Minimum samples to split (`min_samples_split`)
    - Minimum samples in a leaf (`min_samples_leaf`)
    - Minimum impurity decrease
- **Practical advantage:** Saves computation time and prevents overfitting by keeping the model simpler from the start.

Post-Pruning (Prune After Full Growth)

- The tree is **grown to its maximum size** (or nearly so), then branches that provide little predictive power are **removed**.
- Techniques:
    - Reduced Error Pruning (evaluate on validation set and remove unhelpful branches)
    - Cost Complexity Pruning (`ccp_alpha` in scikit-learn)
- **Practical advantage:** Allows the model to initially capture complex patterns and then simplifies it for better generalization, often leading to

improved accuracy on unseen data.

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

Answer:

Information Gain measures how much **uncertainty** (impurity) is reduced in the target variable after splitting a dataset based on a particular feature. It is most often calculated using **entropy** as the impurity measure.

Formula: For a split S that divides a parent node P into left (L) and right (R) child nodes:

$$ \text{Information Gain} = Entropy(P) - \left(\frac{n_L}{n_P} \cdot Entropy(L) + \frac{n_R}{n_P} \cdot Entropy(R) \right) $$

Where:

- $Entropy(X)$ = impurity of node X
- n_L, n_R, n_P = number of samples in left child, right child, and parent node.

Why it’s important for choosing the best split:

- At each node, the Decision Tree algorithm evaluates all possible splits and computes their Information Gain.
- **Higher Information Gain** means the split makes the child nodes **purer** (more homogeneous) compared to the parent.
- The split with the **maximum Information Gain** is chosen because it best separates the classes, leading to a more accurate and efficient tree.

Example: If splitting on “petal length ≤ 2.45” reduces entropy from **1.0** to an average of **0.2** in the child nodes, the Information Gain is **0.8**, meaning the split significantly improves class purity.

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:

1. **Medical diagnosis** – Predicting if a patient has a disease based on symptoms and test results.
2. **Credit scoring** – Classifying loan applicants as “low risk” or “high risk” based on financial history.
3. **Fraud detection** – Identifying suspicious transactions.
4. **Customer churn prediction** – Predicting if a customer will stop using a service.
5. **Product recommendation** – Suggesting products based on user behavior and preferences.
6. **Regression tasks** – Predicting house prices, sales forecasting, etc.

Main advantages:

- **Easy to understand & interpret** – Produces clear, human-readable rules.
- **Handles both numerical and categorical data** – Works with mixed data types without complex preprocessing.
- **No need for feature scaling** – Normalization or standardization is not required.
- **Captures non-linear relationships** – Can model complex decision boundaries.
- **Feature importance** – Identifies which variables are most influential.

Main limitations:

- **Overfitting** – Fully grown trees can memorize the training data and perform poorly on unseen data.
- **High variance** – Small changes in data can produce very different trees.
- **Bias toward features with many levels** – Features with more unique values can dominate splits.

- **Less accurate than ensembles** – Often outperformed by Random Forests or Gradient Boosting on complex problems.

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

In [1]:
```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

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

# Split into training and test sets (stratified to maintain class proportions)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Train 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
importances = clf.feature_importances_

# Output
print(f"Accuracy: {accuracy:.4f}")
print("Feature importances:")
for name, importance in zip(feature_names, importances):
    print(f"  {name}: {importance:.4f}")
```

```
Accuracy: 0.9333
Feature importances:
  sepal length (cm): 0.0000
  sepal width (cm): 0.0286
  petal length (cm): 0.5412
  petal width (cm): 0.4303
```

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.

In [2]:
```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

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

# Split into train/test sets (stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Fully-grown tree (no depth limit)
clf_full = DecisionTreeClassifier(random_state=42)
clf_full.fit(X_train, y_train)
acc_full = accuracy_score(y_test, clf_full.predict(X_test))

# Tree with max_depth=3
clf_md3 = DecisionTreeClassifier(max_depth=3, random_state=42)
clf_md3.fit(X_train, y_train)
acc_md3 = accuracy_score(y_test, clf_md3.predict(X_test))

# Output
print(f"Fully-grown tree accuracy: {acc_full:.4f}")
print(f"Max depth=3 tree accuracy: {acc_md3:.4f}")
```

```
Fully-grown tree accuracy: 0.9333
Max depth=3 tree accuracy: 0.9778
```

Question 8: Write a Python program to:

● Load the California Housing dataset from sklearn

● Train a Decision Tree Regressor

● Print the Mean Squared Error (MSE) and feature importances

In [3]:
```
from sklearn.datasets import fetch_california_housing
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Load California Housing dataset
housing = fetch_california_housing()
X, y = housing.data, housing.target
feature_names = housing.feature_names

# Split dataset into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Initialize and train Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)

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

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)

# Print results
print(f"Mean Squared Error on test data: {mse:.4f}\n")

print("Feature Importances:")
for name, importance in zip(feature_names, regressor.feature_importances_):
    print(f" - {name}: {importance:.4f}")
```

```
Mean Squared Error on test data: 0.4952

Feature Importances:
 - MedInc: 0.5285
 - HouseAge: 0.0519
 - AveRooms: 0.0530
 - AveBedrms: 0.0287
 - Population: 0.0305
 - AveOccup: 0.1308
 - Latitude: 0.0937
 - Longitude: 0.0829
```

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

In [4]:
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score

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

# Split into train and test sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Define the Decision Tree classifier
dt = DecisionTreeClassifier(random_state=42)

# Set up the grid of parameters to search
param_grid = {
    'max_depth': [2, 3, 4, 5, 6, None],
    'min_samples_split': [2, 5, 10]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(dt, param_grid, cv=5, n_jobs=-1)

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

# Best parameters found
best_params = grid_search.best_params_

# Evaluate the best estimator on test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Print the results
print(f"Best parameters: {best_params}")
print(f"Model accuracy on test set: {accuracy:.4f}")
```

```
Best parameters: {'max_depth': 4, 'min_samples_split': 2}
Model accuracy on test set: 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**

- **Understand the missingness:** First, analyze *how* data is missing. Is it random, or does it follow a pattern? This affects how you handle it.
- **Imputation:**
    - For **numerical features**, you could fill missing values with mean, median, or use more advanced methods like K-Nearest Neighbors imputation.
    - For **categorical features**, you might fill missing with the mode (most frequent value) or create a special category like `"Unknown"`.
    - If missing values are too prevalent or critical, consider dropping those features or samples carefully.
- **Why it matters:** Models can't handle missing data directly, so cleaning this up ensures your model sees complete, reliable inputs.

2. **Encode Categorical Features**

- **Identify categorical variables:** This could be patient gender, blood type, or any non-numeric info.
- **Encoding methods:**
    - For **nominal categories** without order (e.g., blood type), use **One-Hot Encoding**.

        - For **ordinal categories** (e.g., disease severity: mild, moderate, severe), use **Label Encoding** or map them to meaningful numeric scales.
    - **Why it matters:** Machine learning models, including Decision Trees, require numeric input, so encoding transforms your data into a digestible form.

3. **Train a Decision Tree Model**

- **Split the data:** Use an 80-20 or 70-30 split between training and test sets, or use cross-validation to ensure your results generalize.
- **Initialize the model:** Start with a default Decision Tree classifier.
- **Train on processed data:** Fit the model on your training data.
- **Why Decision Trees:** They handle mixed data types well, are interpretable (important in healthcare), and can capture nonlinear patterns.

4. **Tune Hyperparameters**

- **Key hyperparameters to tune:**
    - `max_depth`: controls tree complexity, balancing underfitting and overfitting.
    - `min_samples_split` and `min_samples_leaf`: control how many samples needed to split or be a leaf node, affecting generalization.
    - `max_features`: number of features to consider at each split.
- **Use GridSearchCV or RandomizedSearchCV:** Explore combinations systematically with cross-validation to find the sweet spot.
- **Why tuning matters:** Proper tuning prevents overfitting or underfitting, improving the model's predictive power on unseen data.

5. **Evaluate Performance**

- **Metrics:**
    - For classification, consider **Accuracy**, **Precision**, **Recall**, **F1-score**, and **ROC-AUC**.

        - In healthcare, **Recall** (sensitivity) is often critical — you want to catch as many patients with the disease as possible, even at the cost of some false positives.
- **Validation:** Use a separate test set or cross-validation to ensure your model performs reliably.
- **Interpretability:** Use feature importance and decision tree visualization to explain model decisions to clinicians and stakeholders.

Business Value of This Model

- **Early detection:** Predicting disease early means patients can receive timely treatment, improving outcomes and reducing healthcare costs.
- **Resource optimization:** Helps healthcare providers prioritize high-risk patients for screening or intervention, making better use of limited resources.
- **Personalized care:** Tailors monitoring and care plans based on individual risk, improving patient satisfaction and effectiveness.
- **Data-driven decisions:** Provides actionable insights backed by data, enabling the company to develop better products, policies, or outreach programs.
- **Trust and transparency:** Decision Trees' interpretability supports building trust with clinicians, regulators, and patients, crucial in healthcare.
