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** used for both **classification** and **regression** tasks. In the context of **classification**, it is a flowchart-like structure where:

- **Each internal node** represents a test on an attribute (feature).
- **Each branch** represents the outcome of that test.
- **Each leaf node** represents a class label (decision taken after computing all attributes).

It is called a "tree" because it starts with a **root node** and branches out to **leaves**, resembling a tree structure.

Start with the whole dataset.

1. At each node, the algorithm selects the **best feature** to split the data based on some **impurity measure** (e.g., Gini impurity, entropy for information gain).
2. The dataset is divided into subsets based on the selected feature and its values.
3. This process is repeated **recursively** on each subset (subtree), until:

   All data in a node belongs to the same class, or

   A stopping criterion is met (e.g., maximum depth, minimum samples per leaf).
4. The final result is a **tree of decisions**, and to classify a new example, it is passed down the tree following the feature tests until it reaches a leaf node (the predicted 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 decision trees, choosing the **best feature** to split the data at each step is critical. To do this, the algorithm uses **impurity measures** to evaluate how well a feature separates the data. Two of the most common impurity measures are:

- **Gini Impurity**
- **Entropy (used in Information Gain)**

1. Gini Impurity

Definition:

Gini Impurity measures the probability of **incorrectly classifying a randomly chosen element** from the dataset if it were labeled randomly according to the class distribution in that node.

2. Entropy

Definition:
Entropy is a concept from information theory. It measures the **amount of randomness or disorder** in the data at a node.

When building a decision tree:

- The algorithm **evaluates each possible split** of the data using an impurity measure.
- It chooses the split that **reduces impurity the most** — i.e., gives the **highest Information Gain** (for Entropy) or **largest reduction in Gini**.

Information Gain (used with Entropy):

Information Gain=Entropy (parent)−Weighted sum of Entropy (children)

Gini Reduction (used similarly):

Gini Gain=Gini (parent)−Weighted sum of Gini (children)

The **greater the reduction in impurity**, the better 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 :

1. Pre-Pruning (Early Stopping)

Definition:
Pre-pruning stops the decision tree **before it fully grows** by applying certain **stopping criteria** during the tree-building process.

Common Pre-Pruning Conditions:

- Maximum tree depth (max_depth)
- Minimum number of samples required to split a node (min_samples_split)
- Minimum number of samples per leaf (min_samples_leaf)
- Minimum impurity decrease (min_impurity_decrease)

Practical Advantage:
Faster training and lower memory usage
Since the tree doesn't grow unnecessarily large, it's quicker to train and uses less computational resources.

2. Post-Pruning (Pruning After Full Growth)

Definition:
Post-pruning allows the decision tree to **grow to its full depth**, then **removes branches** that do not provide meaningful predictive power. This is usually done using a **validation set** or **cost-complexity pruning** (e.g., in CART).

How It Works:

- The full tree is built.
- Subtrees are removed if they do not significantly improve accuracy on a validation set or cause overfitting.

Practical Advantage:
Improves generalization by reducing overfitting
By trimming overfitted branches, the model becomes more robust and performs better 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 (IG) is a metric used in decision trees to measure how well a given feature **separates** or **classifies** the data. It quantifies the **reduction in entropy** (i.e., disorder or impurity) after splitting a dataset based on a specific feature.

In each step of building a decision tree, the algorithm must decide **which feature to split on** to best separate the data into pure (or less impure) subsets. Information Gain helps in this decision:

- **High Information Gain** → the feature provides a good split (i.e., results in purer child nodes).
- **Low Information Gain** → the feature is not very useful for classification.

The algorithm selects the feature with the **highest Information Gain** to split at each node, ensuring that the tree makes the **most informative decisions** as early as possible.

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. **Medical Diagnosis**

   Predict diseases based on symptoms, age, test results, etc.

   Example: Classifying whether a tumor is benign or malignant.

2. **Credit Scoring / Loan Approval**

   Assessing a customer’s risk based on income, credit history, employment, etc.

Helps banks decide whether to approve or reject a loan application.

3. **Fraud Detection**

   Detect suspicious transactions or behaviors in banking and e-commerce.

4. **Customer Churn Prediction**

   Identify which customers are likely to stop using a service or subscription.

5. **Marketing and Recommendation Systems**

   Segmenting customers to target the right audience with personalized offers.

6. **Manufacturing and Quality Control**

   Classify products as pass/fail based on test results or measurements.

7. **Retail and Inventory Management**

   Predicting product demand or classifying sales patterns.

Advantages of Decision Trees (in Points):

1. **Easy to understand and interpret**

   Decision trees are visual and intuitive — even non-experts can follow the logic.

2. **Requires little data preprocessing**

   No need for feature scaling, normalization, or extensive cleaning.

3. **Handles both numerical and categorical data**

   Can work with a mix of data types without transformation.

4. **Fast to train and predict**

   Especially efficient on small to medium-sized datasets.

5. **Handles missing values well**

   Some implementations can handle missing data during training and prediction.

6. **Works well with large feature sets**

   Automatically selects the most important features during splitting.

7. **Supports non-linear relationships**

   Can model complex patterns without needing a specific mathematical form.

8. **Useful for feature importance analysis**

   Can provide insights into which features are most important for predictions.

9. **No assumptions about data distribution**

   Unlike linear models, decision trees do not assume a specific underlying distribution.

10. **Can be used in ensemble methods**

    Forms the base of powerful models like Random Forest and Gradient Boosting 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

Answer :

Code :

```
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


# 1. Load the Iris dataset

iris = load_iris()

X = iris.data

y = iris.target

feature_names = iris.feature_names

target_names = iris.target_names


# 2. Split into training and test 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)


# 3. Create and train the Decision Tree Classifier (using Gini)

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

clf.fit(X_train, y_train)
```

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

# 5. Print model accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")

# 6. Print feature importances
print("\nFeature Importances:")
for name, importance in zip(feature_names, clf.feature_importances_):
    print(f"{name}: {importance:.4f}")
```

OUTPUT :

```
Model Accuracy: 1.00
Feature Importances:
sepal length (cm): 0.0000
sepal width (cm): 0.0167
petal length (cm): 0.9061
petal width (cm): 0.0772
```

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.

Answer :

Code :

```
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

# 1. Load the Iris dataset
```

```
iris = load_iris()
X = iris.data
y = iris.target

# 2. Split the dataset into train and test sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Train Decision Tree with max_depth=3
clf_limited = DecisionTreeClassifier(max_depth=3, random_state=42)
clf_limited.fit(X_train, y_train)
y_pred_limited = clf_limited.predict(X_test)
accuracy_limited = accuracy_score(y_test, y_pred_limited)

# 4. Train fully-grown Decision Tree (no max_depth)
clf_full = DecisionTreeClassifier(random_state=42)
clf_full.fit(X_train, y_train)
y_pred_full = clf_full.predict(X_test)
accuracy_full = accuracy_score(y_test, y_pred_full)

# 5. Print both accuracies
print(f"Accuracy with max_depth=3: {accuracy_limited:.2f}")
print(f"Accuracy with fully-grown tree: {accuracy_full:.2f}")
```

OUTPUT :

```
Accuracy with max_depth=3: 1.00
Accuracy with fully-grown tree: 1.00
```

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

Answer :

Code :

```
from sklearn.datasets import fetch_california_housing
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# 1. Load the California Housing dataset
data = fetch_california_housing()
X = data.data
y = data.target
feature_names = data.feature_names

# 2. Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# 4. Predict and calculate MSE
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")

# 5. Print feature importances
print("\nFeature Importances:")
for name, importance in zip(feature_names, regressor.feature_importances_):
  print(f"{name}: {importance:.4f}")
```

OUTPUT :

```
Mean Squared Error: 0.50

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
```

Click to add a cell.

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

Answer :

Code :

```python
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

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

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

# 3. Define the Decision Tree Classifier
dtree = DecisionTreeClassifier(random_state=42)

# 4. Define the parameter grid
param_grid = {
    'max_depth': [2, 3, 4, 5, None],
    'min_samples_split': [2, 3, 4, 5]
}

# 5. Setup GridSearchCV
grid_search = GridSearchCV(
    estimator=dtree,
    param_grid=param_grid,
    cv=5,  # 5-fold cross-validation
    scoring='accuracy'
)

# 6. Fit the model with GridSearch
grid_search.fit(X_train, y_train)

# 7. Get the best parameters and best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# 8. Evaluate on the test set
y_pred = best_model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)


# 9. Print the results

print(f"Best Parameters: {best_params}")

print(f"Test Set Accuracy: {accuracy:.2f}")
```

OUTPUT :

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

Test Set Accuracy: 1.00
```

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. Handling Missing Values

Start by cleaning the data. For **numerical features**, fill missing values using the **median** or **mean**. For **categorical features**, fill missing values with the **most common value** (called the mode) or a placeholder like "Unknown". This ensures the model doesn't crash or misinterpret missing data.

2. Encoding Categorical Features

Machine learning models can’t understand text directly, so you need to convert categorical data into numbers. Use **One-Hot Encoding**, which turns categories into binary columns (e.g., "Male" → [1, 0], "Female" → [0, 1]). This helps the decision tree understand the differences between categories without assuming any order.

3. Training the Decision Tree Model

Once your data is clean and encoded, you can train the **Decision Tree Classifier** using your training data. This model will learn patterns in the data to predict whether a patient has the disease.

4. Tuning Hyperparameters

To improve accuracy, tune the model using **GridSearchCV**. This method tests different values for settings like:

- max_depth: How deep the tree can go
- min_samples_split: The minimum number of samples to split a node

It finds the best combination using cross-validation, which checks the model's performance on different parts of the training data.

5. Evaluating the Model

After training, test the model on unseen data. Measure its accuracy (how many predictions were correct) and use tools like the **confusion matrix** or **classification report** to see how well it detects both positive and negative cases. This helps you know if the model is reliable.

Real-World Business Value in Healthcare

- **Early Detection**: Helps doctors identify diseases sooner, improving patient outcomes.
- **Cost Reduction**: Avoids unnecessary tests by focusing on high-risk patients.
- **Better Resource Use**: Helps hospitals prioritize patients who need urgent care.
- **Transparency**: Since decision trees are easy to interpret, doctors can understand why a prediction was made.
- **Supports Decision Making**: Gives healthcare providers data-driven insights alongside medical judgment.