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, but it's most commonly used in classification.

A Decision Tree mimics human decision-making. It's a flowchart-like structure where:

- Each internal node represents a feature (attribute).
- Each branch represents a decision rule.
- Each leaf node represents an outcome (class label).

Work in Classification

1. Start with the entire dataset.
2. Choose the best feature to split the data. This is typically based on a metric like:
   - Gini Impurity
   - Entropy / Information Gain
   - Chi-square, etc.
3. Split the dataset into subsets based on this feature's possible values.
4. Repeat recursively:
   - Choose the best feature for each subset.
   - Continue splitting until:
     - All samples at a node belong to the same class, or
     - You reach a stopping condition (like max depth or minimum samples per node).
5. Classify new data:
   - Start at the root and follow the decisions down the tree according to the feature values of the input.

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

Answer:

Decision Tree Assignment

In Decision Trees, Gini Impurity and Entropy are two common measures used to evaluate the "impurity" or disorder of a dataset. These measures help determine the best feature and threshold to split the data at each node, aiming to create the purest possible child nodes.

1. Gini Impurity

Definition: Gini Impurity measures the probability of incorrectly classifying a randomly chosen element if it were randomly labeled according to the distribution of labels in the subset.

2. Entropy (Information Gain)

Definition: Entropy is a measure from information theory. It quantifies the amount of uncertainty or surprise associated with a random variable.

Impact on Decision Tree Splits

When building a decision tree, at each node:

1. Calculate the impurity (Gini or Entropy) of the parent node.
2. Evaluate each possible split, and compute the weighted average impurity of the child nodes.
3. Select the split that leads to the largest impurity reduction (also called Information Gain when using entropy).

   Information Gain = Impurity (Parent) - Weighted Impurity (Children)

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 used in Decision Trees to prevent overfitting by controlling the tree's growth.

Pre-Pruning (Early Stopping)

Definition: Pre-pruning stops the tree from growing once a certain condition is met before it becomes too complex.

Common criteria for stopping:

- Maximum depth of the tree
- Minimum number of samples required to split a node

- Minimum information gain or impurity reduction

Advantage: Faster training — because the tree doesn't grow unnecessarily deep, saving time and memory.

Example: A tree stops splitting when a node has fewer than 10 samples.

Post-Pruning (Pruning After Full Growth)

Definition: Post-pruning allows the tree to grow fully, then removes branches that have little importance, based on performance on a validation set.

Common techniques:

- Reduced error pruning
- Cost complexity pruning (used in CART)
- Minimal error pruning

Advantage: Better generalization — because it evaluates the impact of subtrees and only removes what actually hurts validation performance.

Example: After building the full tree, we remove branches that don't improve accuracy on the validation set.

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

Answer:

Information Gain in Decision Trees

Information Gain (IG) is a key metric used to choose the best feature and split at each node of a Decision Tree. It measures how much uncertainty (entropy) is reduced after splitting the dataset based on a particular feature.

Definition : Information Gain is defined as the reduction in entropy after a dataset is split on an attribute.

Information Gain = Entropy (parent) − Weighted Average

Why is it Important?

- Goal of a Decision Tree: Split the data so that each group becomes as pure as possible (i.e., contains mostly one class).
- Information Gain tells us which feature gives us the most reduction in impurity.
- The feature with the highest Information Gain is selected to split the data at that node.

Decision Tree Assignment

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

Answer:

Real-World Applications of Decision Trees

Decision Trees are a versatile machine learning model that can be used in a variety of fields. They are particularly useful for classification and regression tasks due to their interpretability and flexibility. Here are some common applications:

1. Healthcare: Disease Diagnosis

- Use case: Predicting the likelihood of a patient having a disease based on features such as age, gender, test results, and medical history.
- Example: Classifying whether a patient has diabetes based on glucose levels, blood pressure, etc.
- Why Decision Trees: Easy to interpret and visualize, which helps medical professionals understand why a diagnosis is made.

2. Finance: Credit Scoring

- Use case: Determining whether a customer is likely to default on a loan or credit card based on their financial history, income, and spending behavior.
- Example: A decision tree can classify whether a loan application is approved or rejected.
- Why Decision Trees: Transparent and interpretable, which is essential for explaining credit decisions.

3. Retail: Customer Segmentation & Churn Prediction

- Use case: Predicting whether a customer will churn (leave) or remain based on purchase history, customer support interactions, and other behavioral data.
- Example: Identifying high-risk customers who are likely to stop using a service, allowing businesses to take preventive actions.
- Why Decision Trees: Can handle both categorical and continuous data and provide insights into factors leading to churn.

Advantages of Decision Trees

1. Interpretability: Decision trees are easy to understand and visualize, which makes them highly interpretable even for non-technical users.
2. No Feature Scaling Required: Unlike algorithms like k-NN or SVM, decision trees don’t require normalization or scaling of features.
3. Handles Mixed Data Types: They can handle both numerical and categorical data well.
4. Non-Linear Relationships: They can capture non-linear relationships between

features without needing explicit transformation.
5. Handles Missing Data: Decision trees can handle missing data in the features, using methods like surrogate splits.
6. Can Handle Outliers: Less sensitive to outliers compared to other algorithms like linear regression.
7. Fast Prediction: Once trained, decision trees can make predictions very quickly.

Limitations of Decision Trees

1. Overfitting: Decision trees tend to overfit, especially when they are too deep or when there are too many branches. This can be mitigated by pruning, cross-validation, or using ensemble methods like Random Forests.
2. Instability: A small change in the data can result in a completely different tree. This is why decision trees can sometimes lack robustness compared to other models.
3. Bias Toward Features with More Categories: Decision trees might favor features with more categories or continuous features, which can lead to biased splits.
4. Complex Trees: A deep tree with many nodes can be computationally expensive, difficult to interpret, and less efficient.
5. Poor Performance with Complex Patterns: Decision trees might struggle to capture complex relationships in data compared to other models like neural networks.
6. Greedy Algorithm: Decision trees use a greedy approach (choosing the best split at each node) which may not always lead to the globally optimal tree.

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 pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Decision Tree Assignment

```
import warnings
warnings.filterwarnings('ignore')
```

```
# Load the Iris dataset
from sklearn.datasets import load_iris
data = load_iris()
```

```
#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 = 0.3, random_state=1)
```

```
# Train the Decision Tree using Gini criterion
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='gini',
random_state = 42)
classifier.fit(X_train, y_train)
```

```
y_pred = classifier.predict(X_test)
y_pred
```

```
#Evaluate the model
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)

# Feature Importances
importances = model.feature_importances_
feature_importance_df = pd.DataFrame({'Feature':
feature_names, 'Importance':
importances}).sort_values(by='Importance',
ascending=False)
```

Question 7: Write a Python program to:

- Load the Iris Dataset

<u>Decision Tree Assignment</u>

- 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 required libraries

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

# Load the Iris dataset

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

# Split data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train Decision Tree with max_depth=3

tree_limited = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=42)
tree_limited.fit(X_train, y_train)
y_pred_limited = tree_limited.predict(X_test)
acc_limited = accuracy_score(y_test, y_pred_limited)

# Train fully-grown Decision Tree

tree_full = DecisionTreeClassifier(criterion='gini', random_state=42) # no
max_depth
tree_full.fit(X_train, y_train)
y_pred_full = tree_full.predict(X_test)
acc_full = accuracy_score(y_test, y_pred_full)
```

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:

```
# Import required libraries

import numpy as np
import pandas as pd
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
import matplotlib.pyplot as plt

# Load the California Housing dataset (Boston dataset is deprecated)

data = fetch_california_housing()
X = data.data
y = data.target
feature_names = data.feature_names

# Train-Test Split

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

# Train Decision Tree Regressor

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

# Predict and Evaluate

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

# Feature Importances

importances = model.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance' :
importances}).sort_values(by='Importance', ascending=False)

# Bar Plot of Feature Importances

feature_importance_df.plot(kind='barh', x='Feature', y='Importance', legend=False,
figsize=(8, 5))
plt.gca().invert_yaxis()
plt.title('Feature Importances (Decision Tree Regressor)')
plt.xlabel('Importance Score')
plt.tight_layout()
plt.show()
```

Question 9: Write a Python program to:

Decision Tree Assignment

- 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:

```
# Import required libraries

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 = iris.data
y = iris.target

# Split into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Define parameter grid for GridSearchCV

param_grid = {'max_depth': [2, 3, 4, 5, 6], 'min_samples_split': [2, 3, 4, 5]}

# Create and run GridSearchCV

grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
param_grid=param_grid, cv=5, scoring='accuracy')
# cv=5,  5-fold cross-validation
grid_search.fit(X_train, y_train)

# Best Parameters and Accuracy

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

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:

Decision Tree Assignment

- 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:

 # Handle the missing values

```
from sklearn.impute import SimpleImputer
num_imputer = SimpleImputer(strategy='mean') # or 'median'
cat_imputer = SimpleImputer(strategy='most_frequent')
```

 # Encode Categorical Features Use One-Hot Encoding for nominal data

```
from sklearn.preprocessing import OneHotEncoder
```

 # Train a Decision Tree Model

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

 # Split data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

 # Train model

```
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

 # Tune Hyperparameters with GridSearchCV Search for best values

```
from sklearn.model_selection import GridSearchCV
param_grid = {'max_depth': [3, 5, 10, None], 'min_samples_split': [2, 5, 10], 'criterion':
['gini', 'entropy']}
grid = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)
```

 # Evaluate Model Performance Use metrics

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
y_pred = grid.predict(X_test)
```