Assignment Decision Tree (ML-7)

Theoratical Questions

Q1. What is a Decision Tree, and how does it work?

Ans-> A decision tree is a flowchart-like structure used in machine learning and decision analysis to model possible outcomes and decisions. It's a supervised learning algorithm that breaks down complex problems into a series of simpler decisions based on attributes or features. Each node in the tree represents a test or a decision, branches represent the possible outcomes of the test, and leaf nodes represent the final outcome or prediction.

- How it works:
    1. Training Data: The algorithm learns from a dataset containing past outcomes and related variables (features).
    2. Root Node: The process starts with the root node, representing the entire dataset.
    3. Splitting: The algorithm identifies the most important feature to split the data based on, creating subtrees.
    4. Decision Rules: Each internal node represents a decision based on a specific feature, and branches represent the different possible values of that feature.
    5. Leaf Nodes: When the data is split into homogeneous groups (e.g., all outcomes of one type), the process reaches the leaf nodes, which represent the final decisions or predictions.
    6. Predicting: To use the tree for prediction, you start at the root and follow the branches based on the input data, eventually reaching a leaf node that provides the prediction.

Q2. What are impurity measures in Decision Trees?

Ans-> In Decision Trees, impurity measures quantify the "mixed-up-ness" or disorder within a node, indicating how well the data within that node belongs to a single class. The goal of building a decision tree is to create nodes that are as pure as possible, meaning they contain mostly (or ideally all) instances of a single class. Common impurity measures include Gini impurity and entropy.

- Gini Impurity:
- Measures the probability of a randomly chosen element being incorrectly classified.
- Ranges from 0 to 0.5, where 0 represents a pure node (all data points belong to the same class) and 0.5 represents maximum impurity (equal distribution of classes).
- In binary classification, it's calculated as 1 - (p^2 + (1-p)^2), where 'p' is the proportion of one class.
- Entropy:
- Measures the amount of uncertainty or randomness in a dataset.
- Ranges from 0 to 1, where 0 represents a pure node and 1 represents maximum impurity.
- Calculated using logarithms (usually base 2) of probabilities of different classes within a node.
- Higher entropy means more uncertainty and a greater mix of classes.

Q3. What is the mathematical formula for Gini Impurity?

Ans-> The mathematical formula for Gini Impurity, used in decision tree algorithms, is: $Gini(D) = 1 - \Sigma [p_i^2]$.

- Where:
- Gini(D): represents the Gini Impurity of a dataset D.
- p_i: is the proportion of instances belonging to class 'i' within that dataset.
- The summation (Σ) is taken over all 'C' classes in the dataset. Decision Trees. Part 4: Gini Index | by om pramod | Medium
- In simpler terms, for a dataset with 'C' classes, the Gini Impurity is calculated by subtracting the sum of the squares of the proportions of each class from 1.
- For a binary classification problem (where there are only two classes), the formula simplifies to: $Gini(D) = 1 - (p^2 + (1-p)^2)$, where 'p' is the proportion of one class.
- ML | Gini Impurity and Entropy in Decision Tree - GeeksforGeeks A Gini Impurity of 0 indicates perfect purity (all instances belong to the same class), while a value of 0.5 represents maximum impurity (an equal distribution of classes).

Q4. What is the mathematical formula for Entropy?

Ans-> The most common mathematical formula for entropy (often denoted as 'S') is S = k * ln(W), where 'k' is the Boltzmann constant and 'W' is the number of microstates of a system. Another related formula, particularly useful for calculating entropy change (ΔS) in thermodynamics, is ΔS = Q/T, where 'Q' is the heat transferred and 'T' is the absolute temperature.

- Here's a breakdown:
- S = k * ln(W): This is the statistical definition of entropy, relating it to the number of possible microstates (W) of a system. Boltzmann constant (k) is approximately 1.38 x 10^-23 J/K.

- ΔS = Q/T: This formula applies to reversible processes, where a small amount of heat (Q) is transferred at a constant temperature (T). It calculates the change in entropy (ΔS).
- ΔS = Sf - Si: This is a more general formula for entropy change, where Sf is the final entropy and Si is the initial entropy.
- ΔS = ∫(dQ/T): For processes with varying temperature, the entropy change is calculated using this integral, where dQ is an infinitesimal amount of heat transfer and T is the absolute temperature.

Q5. What is Information Gain, and how is it used in Decision Trees?

Ans-> Information gain is a metric used in decision tree algorithms to determine the effectiveness of a feature in classifying data. It quantifies the reduction in entropy (uncertainty) achieved by splitting the data based on that feature. Features with higher information gain are preferred for creating decision nodes in the tree because they provide more clarity and reduce uncertainty in predictions.

- Elaboration:
- Entropy: Entropy is a measure of impurity or randomness in a dataset. A dataset with high entropy has a mix of different classes, while a dataset with low entropy is dominated by a single class.
- Information Gain Calculation: Information gain is calculated by subtracting the weighted average entropy of the child nodes (resulting from a split on a feature) from the entropy of the parent node (the dataset before the split).
- Decision Tree Construction:
- In decision tree construction, the algorithm iterates through all possible features and calculates the information gain for each. The feature with the highest information gain is selected to create a split at the current node, effectively dividing the data into subsets based on the feature's values.
- Example: Imagine a decision tree trying to classify emails as spam or not spam. If a feature like "contains the word 'free'" has a high information gain (meaning it separates spam and non-spam emails well), it would be chosen to create a split at that node.

Q6. What is the difference between Gini Impurity and Entropy?

Ans-> Gini impurity and entropy are both used in decision tree algorithms to determine the best split at each node. They measure the impurity of a dataset, with lower values indicating a purer split (i.e., a split where data points belong to fewer classes). While both aim to minimize impurity, they differ in their calculation and sensitivity to class distribution. Gini impurity is generally faster to compute, while entropy can be more sensitive to class imbalance.

- Key Differences:
- Calculation: Gini impurity uses a squared function, while entropy uses a logarithmic function.
- Computational Cost: Gini impurity is generally faster to compute than entropy.
- Sensitivity to Class Imbalance: Entropy can be more sensitive to class imbalance than Gini impurity.
- Interpretation: Gini impurity can be interpreted as the expected error rate, while entropy can be interpreted as the average amount of information needed to specify the class of an instance.
- In practice: Both Gini impurity and entropy generally produce similar results and are effective splitting criteria in decision trees.
- The choice between them often comes down to computational efficiency and the specific characteristics of the dataset.
- For large datasets, Gini impurity is often preferred due to its faster computation.
- For datasets with high class imbalance, entropy might be a better choice due to its sensitivity to imbalance.

Q7. What is the mathematical explanation behind Decision Trees?

Ans-> Decision trees, in their mathematical essence, use splitting criteria based on concepts like information gain or Gini impurity to recursively partition data into increasingly homogeneous subsets. These criteria quantify the "purity" or "impurity" of a node, and the algorithm selects the split that maximizes information gain or minimizes Gini impurity, effectively reducing uncertainty at each step.

- Here's a breakdown of the key mathematical concepts:

    1. Purity and Impurity Measures: Entropy: Measures the impurity or randomness of a dataset. A dataset with all instances belonging to the same class has an entropy of 0 (perfectly pure), while a dataset with an equal distribution of classes has a maximum entropy. Gini Index: Another measure of impurity, it represents the probability of misclassifying a randomly chosen element if it were randomly classified according to the distribution of classes in the node. A Gini index of 0 means all elements belong to the same class. A Gini index of 1 means maximal inequality.
    2. Splitting Criteria: Information Gain: Calculates the reduction in entropy achieved by splitting a dataset based on a particular attribute. The attribute that results in the highest information gain is chosen for the split. Gini Impurity Reduction: Similar to information gain, but uses the Gini index as the impurity measure. The split that results in the lowest Gini impurity is selected.
    3. Mathematical Representation (Simplified): Entropy: H(T) = - Σ [p(i) * log2(p(i))], where p(i) is the probability of class i in node T. Gini Index: Gini(T) = 1 - Σ [p(i)^2], where p(i) is the probability of class i. Information Gain: IG(T, a) = H(T) - Σ [p(v) * H(T|v)], where H(T) is the entropy of the parent node, a is the attribute, v are the values of attribute a, and H(T|v) is the weighted average entropy of the child nodes after splitting on attribute a.
    4. Tree Construction: The decision tree algorithm starts with the root node (the entire dataset) and recursively applies the chosen splitting criterion to find the best attribute to split on at each node. This process continues until a stopping criterion is met (e.g., reaching a maximum depth, achieving a minimum number of samples per leaf, or reaching a certain level of purity).

5. Regression Trees: For regression tasks (predicting continuous values), decision trees use different splitting criteria, often based on minimizing the variance within each node. The most common approach is to minimize the sum of squared errors (SSE) or the mean squared error (MSE).

Q8. What is Pre-Pruning in Decision Trees?

Ans-> Pre-pruning, also known as early stopping, is a method used in decision tree algorithms to prevent overfitting by limiting the growth of the tree before it reaches its full complexity. This is achieved by setting constraints on the tree's structure, such as maximum depth or minimum samples required for a split.

- Here's a more detailed explanation:
- Early Stopping: Pre-pruning halts the tree-building process before it becomes overly complex, preventing it from fitting the training data too closely.
- Hyperparameter Tuning: It involves setting hyperparameters like maximum depth, minimum samples per leaf, or minimum samples per split to control the tree's growth.
- Benefits:
- Pre-pruning helps to: Avoid overfitting, leading to better generalization on unseen data.
- Create a more compact and easier-to-interpret decision tree. Improve computational efficiency by preventing the creation of overly complex subtrees.

Q9. What is Post-Pruning in Decision Trees?

Ans-> Post-pruning in decision trees involves removing branches from a fully grown tree to improve generalization and reduce overfitting. This is done after the tree has been constructed, allowing it to first learn the training data thoroughly. By then removing branches that contribute little to overall accuracy, the model becomes simpler and more robust, leading to better performance on unseen data.

- Here's a more detailed explanation:

1. Full Tree Growth: The decision tree algorithm is allowed to grow to its full potential, potentially overfitting the training data.
2. Post-pruning Techniques: Cost Complexity Pruning (CCP): This method assigns a cost (or complexity) to each branch based on its performance and complexity. Branches with the lowest cost are removed. Reduced Error Pruning: This technique removes branches that do not significantly affect the overall accuracy of the model.

- Minimum Impurity Decrease: Prunes nodes if the decrease in impurity (e.g., Gini impurity) is below a certain threshold.
- Minimum Leaf Size: Removes leaf nodes with fewer samples than a specified threshold.

3. Evaluating Performance: The performance of the pruned tree is evaluated on a separate validation set to ensure it generalizes well to unseen data.
4. Benefits of Post-pruning: Improved Generalization: By removing unnecessary branches, the pruned tree is less likely to overfit the training data and performs better on new, unseen examples. Simplified Model: The pruned tree is easier to interpret and understand. Reduced Complexity: A simpler model requires less memory and computational resources.
5. Cost-Complexity Pruning (CCP) in Scikit-learn: In the Scikit-learn library, the ccp_alpha parameter can be used to control the amount of pruning during post-pruning. A higher ccp_alpha value leads to more pruning and a simpler tree.

Q10. What is the difference between Pre-Pruning and Post-Pruning?

Ans-> Pre-pruning and post-pruning are two strategies used to optimize decision trees by reducing their complexity and preventing overfitting. Pre-pruning, also known as early stopping, stops the tree's growth during construction based on certain criteria, while post-pruning removes branches from a fully grown tree.

- Here's a more detailed comparison:
- Pre-Pruning (Early Stopping):
- Mechanism: During the tree construction phase, a decision is made at each node whether to split further or not. This decision is based on predefined criteria like minimum samples per node, maximum tree depth, or minimum information gain. If the criteria are not met, the node is not split, and it becomes a leaf node.
- Pros: More computationally efficient as it avoids growing unnecessary branches. Can prevent overfitting by stopping the tree from growing too deep.
- Cons: May be too aggressive and prematurely stop the tree from learning important patterns, potentially underfitting the data.
- The optimal stopping criteria can be difficult to determine.
- Post-Pruning (Reduced Error Pruning):
- Mechanism: The decision tree is first grown to its full depth, potentially overfitting the training data. Then, parts of the tree are removed (pruned) based on metrics like validation error or cross-validation error.
- Pros: More flexible as it allows the tree to explore all possible patterns before pruning. Can lead to a more optimal tree structure that generalizes better to unseen data.
- Cons: Computationally more expensive than pre-pruning because it requires growing the full tree first. Pruning criteria need to be carefully chosen to avoid removing important branches

Q11. What is a Decision Tree Regressor?

Ans-> A Decision Tree Regressor is a machine learning algorithm that uses a tree-like model to predict continuous numerical values, unlike classification which predicts categorical values. It works by recursively splitting data based on features, creating a tree structure where each internal node represents a decision rule, each branch represents an outcome of a decision, and each leaf node provides the predicted value.

- Key Concepts:
- Continuous Target Variable: Decision tree regressors are used when the output you are trying to predict is a continuous numerical value, such as house price, temperature, or stock price.
- Tree Structure: The model builds a tree-like structure by splitting the data based on the features that best separate the target values.
- Splitting Criteria: The algorithm uses a metric like Mean Squared Error (MSE) to determine the best split at each node, minimizing the difference between the predicted and actual values within each subset.
- Recursive Partitioning: The process of splitting is repeated recursively on each subset, creating branches and leaf nodes until a stopping criterion is met.
- Leaf Nodes: The leaf nodes of the tree contain the final predicted values, often the average or median of the target values in that subset.
- Prediction: To make a prediction for a new data point, the algorithm traverses the tree, following the branches that match the features of the data point, until it reaches a leaf node, where the predicted value is obtained.

Q12. What are the advantages and disadvantages of Decision Trees?

Ans-> Decision trees offer advantages like interpretability, versatility, and ease of understanding, but they also come with disadvantages such as overfitting, instability, and potential bias towards dominant classes.

- Advantages:
- Interpretability: Decision trees are easy to understand and visualize, resembling a flowchart, which makes it easy to follow the logic and decision-making process.
- Versatility: They can be used for both classification and regression tasks, making them adaptable to different types of data and problems.
- No Feature Scaling: Decision trees do not require normalization or scaling of the data, simplifying the data preparation process.
- Handles Non-Linear Relationships: They can capture non-linear relationships between features and target variables, unlike some linear models.
- Handles Missing Values: Decision trees can handle missing data points by using surrogate splits to continue making decisions.
- Robust to Outliers: Decision trees are less sensitive to outliers compared to some other algorithms.
- Easy to Implement: Decision tree algorithms are relatively simple to implement and require less computational power compared to some complex models.
- Disadvantages:
- Overfitting: Decision trees are prone to overfitting, especially when they are deep, which can lead to poor performance on unseen data.
- Instability: Small changes in the training data can lead to significantly different tree structures, making them unstable.
- Bias Towards Dominant Classes: In imbalanced datasets, decision trees may be biased towards the majority class, leading to poor performance on minority classes.
- Sensitivity to Data Variability: Small variations in the data can lead to different splits and different trees.
- Not Ideal for High-Dimensional Data: Decision trees can struggle with high-dimensional data, where the number of features is large.
- Limited Expressiveness: Certain concepts like XOR, parity, or multiplexer problems can be difficult to learn with decision trees.

Q13. How does a Decision Tree handle missing values?

Ans-> To handle missing values effectively, decision trees use surrogate splits. These surrogate splits act as backup choices when the primary attribute for a split has missing values. The algorithm identifies the next best attribute that can provide a similar separation as the primary attribute.

Decision Trees can handle missing values in several clever ways, depending on how the algorithm is implemented. Here are the most common strategies:

1. **Surrogate Splits** (used in CART and some other libraries):
   When the best feature for a split has missing values, the tree finds an alternative (surrogate) feature that closely mimics the original split. If the main feature is missing, the surrogate is used instead to decide the branch.
2. **Assignment by Distribution**:
   When building the tree, instances with missing values are split *proportionally* based on the distribution of known values. For example, if 70% of known samples go left and 30% go right, the missing-value instances are split accordingly.
3. **Mean/Mode Imputation (Preprocessing Step)**:
   Before training, missing values in numerical features can be replaced with the mean or median, and in categorical features with the mode. This is more of a data prep strategy than something the tree inherently does.

4. **Using Missing as a Category**:
   For categorical features, some implementations treat “missing” as just another category. This can preserve information that might otherwise be lost through imputation.

In libraries like scikit-learn, Decision Trees **do not handle missing values natively**, so you'd typically preprocess your data first (imputation or encoding). But frameworks like XGBoost, LightGBM, and CatBoost have built-in strategies to handle them internally and elegantly.

Since you're exploring machine learning for practical use, especially with small datasets, knowing how to manage missing values is crucial—it can make or break your model’s performance. Want to walk through an example using the Wine Quality dataset?

Q14. How does a Decision Tree handle categorical features?

Ans-> Decision trees naturally handle categorical features, but how they do so can depend on the specific implementation. Some implementations, like those in scikit-learn, require categorical features to be numerically encoded (e.g., using one-hot or ordinal encoding) before being passed to the model. Other implementations, like some found in CatBoost, LightGBM, and XGBoost, can handle categorical features directly without explicit encoding.

- Here's a more detailed breakdown:

  1. Numerical Encoding (Preprocessing): One-Hot Encoding: This method creates a new binary column for each category in a feature. For example, if a feature "color" has categories "red", "blue", and "green", one-hot encoding would create three new columns: "color_red", "color_blue", and "color_green", with values of 1 or 0 indicating the presence or absence of each color. Ordinal Encoding: This method assigns a numerical value to each category based on their inherent order (if any). For example, if "size" has categories "small", "medium", and "large", ordinal encoding might assign 1 to "small", 2 to "medium", and 3 to "large".
  2. Direct Handling (No Preprocessing): Some decision tree implementations, particularly those found in gradient boosting libraries like CatBoost and LightGBM, can directly handle categorical features. They achieve this by finding the best split points based on the categories themselves, rather than relying on numerical representations. This can be more efficient and potentially more accurate in some cases, as it avoids the potential loss of information or introduction of bias during encoding.
  3. How Decision Trees Split on Categorical Features: When a decision tree encounters a categorical feature, it evaluates different splits to find the one that best separates the data based on the target variable. For example, if a decision tree is trying to predict whether a customer will buy a product, it might split based on "color" and group all "red" items together, all "blue" items together, etc., to see which split results in the most accurate predictions. The algorithm continues splitting the data until it reaches a stopping point (e.g., a maximum depth, a minimum number of samples per leaf).
  4. Important Considerations: Cardinality: Decision trees can struggle with categorical features that have a very high number of unique categories (high cardinality) because it can lead to excessive branching and make the tree overly complex.

- Feature Importance: Decision trees can be used to assess the importance of categorical features in predicting the target variable.
- Interpretability: One advantage of decision trees is their interpretability. Even when handling categorical features, the resulting tree structure can be relatively easy to understand and explain.

Q15. What are some real-world applications of Decision Trees?

Ans-> Decision trees are versatile tools with applications across numerous fields. They are used for classification and regression tasks, helping to predict outcomes based on input data. In healthcare, they aid in medical diagnosis and predicting patient outcomes. Businesses leverage them for customer segmentation, churn prediction, and fraud detection. Finance utilizes decision trees for credit scoring and risk assessment. They are also employed in manufacturing for quality control and in engineering for various predictive modeling tasks.

- Here's a more detailed look at some real-world applications:

  1. Healthcare: Medical Diagnosis: Decision trees can help doctors diagnose diseases by analyzing patient symptoms and test results, guiding them through a series of questions to arrive at a likely diagnosis.

- Patient Risk Assessment: They can predict the likelihood of a patient developing a specific condition or the severity of a disease, enabling timely intervention and resource allocation.
- Treatment Planning: Decision trees can help determine the most appropriate treatment plan for a patient based on their individual characteristics and medical history.

  2. Business: Customer Segmentation: Businesses use decision trees to group customers based on their demographics, behavior, and purchasing patterns, enabling targeted marketing campaigns.

- Customer Churn Prediction: They help identify customers at risk of leaving a service, allowing businesses to implement proactive retention strategies.
- Fraud Detection: Decision trees can analyze financial transactions to identify patterns indicative of fraudulent activity, helping to minimize financial losses.
- Credit Scoring: Financial institutions use decision trees to assess the creditworthiness of loan applicants, determining their likelihood of defaulting on payments.
- Sales Forecasting: Decision trees can be used to predict future sales based on historical data and various influencing factors.

  3. Finance:

- Risk Management: Decision trees are used to assess and manage financial risks, helping institutions make informed decisions about investments and lending.
- Option Pricing: They are used in financial modeling to determine the fair price of options contracts.

4. Manufacturing:

- Quality Control: Decision trees can analyze data from sensors and production processes to identify potential defects in products, improving overall quality control.

5. Other Applications:

- Spam Filtering: Decision trees can be used to classify emails as spam or not spam based on various email features.
- Image Recognition: Decision trees can be used as part of more complex image recognition models to classify objects within images.
- Predicting Crime Risk: They can be used to analyze crime data and identify areas with a higher risk of criminal activity.

Practical

Q16.Write a Python program to train a Decision Tree Classifier on the Iris dataset and print the model accuracy.

```
# prompt: Write a Python program to train a Decision Tree Classifier on the Iris dataset and print the model accuracy.

# Train a Decision Tree Classifier on the Iris dataset and print the model accuracy.
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 the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Decision Tree Classifier
clf = DecisionTreeClassifier()

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = clf.predict(X_test)

# Calculate and print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy}")
```

```
Model Accuracy: 1.0
```

Q17. Write a Python program to Train a Decision Tree Classifier using Gini Impurity as the Criterion and print the feature importances.

```
# prompt: Write a Python program to Train a Decision Tree Classifier using Gini Impurity as the Criterion and print the feature importar

# Create a Decision Tree Classifier using Gini Impurity
clf_gini = DecisionTreeClassifier(criterion='gini', random_state=42)

# Train the classifier on the training data
clf_gini.fit(X_train, y_train)

# Print the feature importances
print("Feature Importances:", clf_gini.feature_importances_)
```

```
Feature Importances: [0.         0.01911002 0.89326355 0.08762643]
```

Q18. Write a Python program to train a Decision Tree Classifier using Entropy as the splitting criterion and print the model accuracy.

```
# prompt: Write a Python program to train a Decision Tree Classifier using Entropy as the splitting criterion and print the model accura
```

```
# Create a Decision Tree Classifier using Entropy
clf_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42)

# Train the classifier on the training data
clf_entropy.fit(X_train, y_train)

# Make predictions on the testing data
y_pred_entropy = clf_entropy.predict(X_test)

# Calculate and print the accuracy
accuracy_entropy = accuracy_score(y_test, y_pred_entropy)
print(f"Model Accuracy (Entropy): {accuracy_entropy}")
```

```
Model Accuracy (Entropy): 0.9777777777777777
```

Q19. Write a Python program to train a Decision Tree Regressor on a housing dataset and evaluate using Mean Squared Error (MSE).

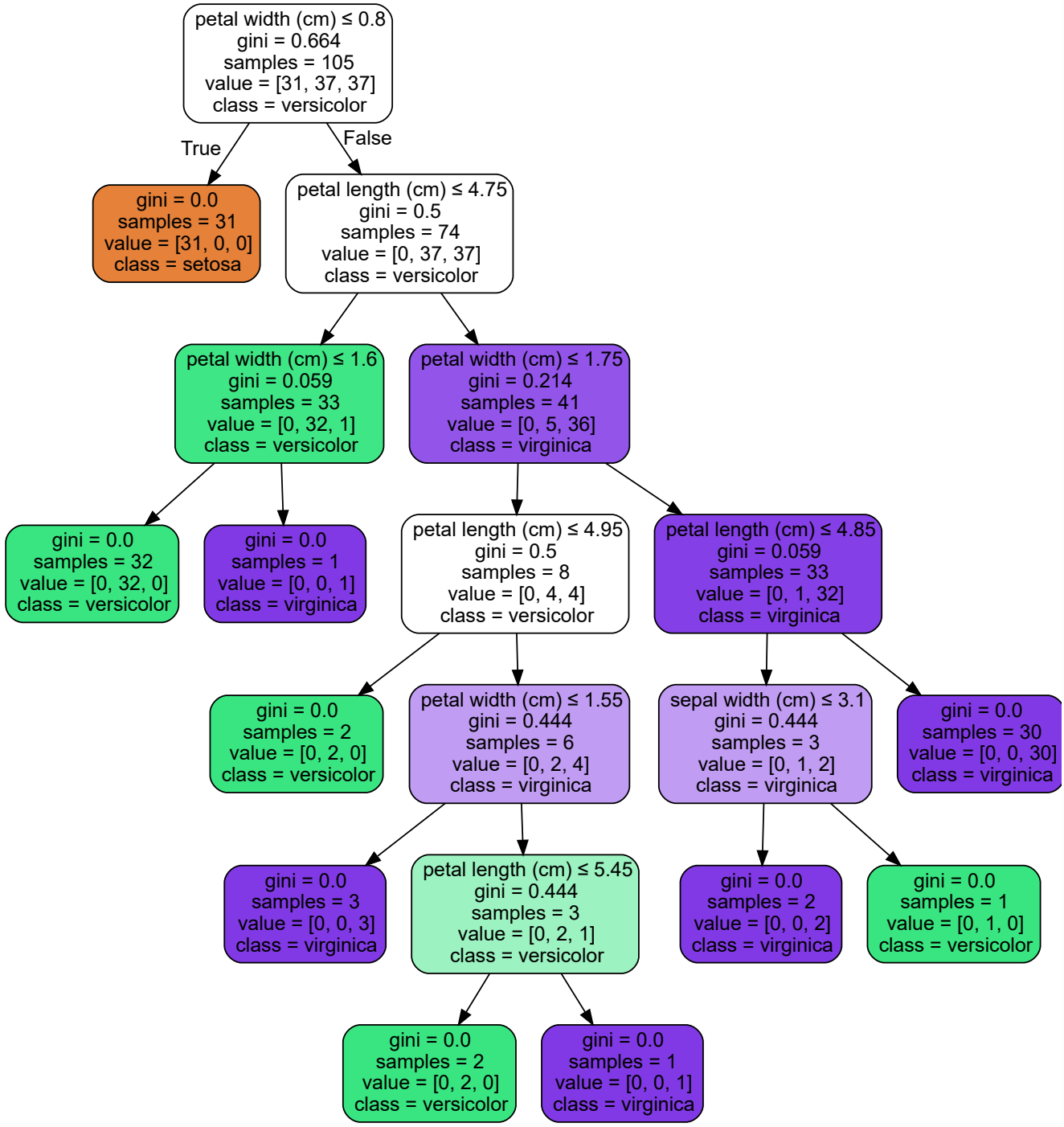
```
# prompt: Write a Python program to train a Decision Tree Regressor on a housing dataset and evaluate using Mean Squared Error (MSE).

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

# Load the California Housing dataset
housing = fetch_california_housing()
X_housing = housing.data
y_housing = housing.target

# Split the dataset into training and testing sets
X_train_housing, X_test_housing, y_train_housing, y_test_housing = train_test_split(X_housing, y_housing, test_size=0.3, random_state=42)

# Create a Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)

# Train the regressor on the training data
regressor.fit(X_train_housing, y_train_housing)

# Make predictions on the testing data
y_pred_housing = regressor.predict(X_test_housing)

# Calculate and print the Mean Squared Error
mse = mean_squared_error(y_test_housing, y_pred_housing)
print(f"Mean Squared Error (MSE): {mse}")
```

```
Mean Squared Error (MSE): 0.5280096503174904
```

Q20. Write a Python Program to Train Decision Tree Classifier and visualize the tree using graphviz.

```
# prompt: Write a Python Program to Train Decision Tree Classifier and visualize the tree using graphviz.

!pip install graphviz

from sklearn.tree import export_graphviz
import graphviz

# Assume clf is your trained Decision Tree Classifier from Q16 or Q18
# Using the classifier trained with the default gini criterion from Q16
dot_data = export_graphviz(clf, out_file=None,
                           feature_names=iris.feature_names,
                           class_names=iris.target_names,
                           filled=True, rounded=True,
                           special_characters=True)
graph = graphviz.Source(dot_data)
# Render the graph (will save a file like Source.gv.pdf or display in notebook)
graph
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (0.21)
```



Q21. Write a Python program to Train a Decision Tree Classifier with a maximum depth of 3 and compare its accuracy with a fully grown tree.

```
# prompt: Write a Python program to Train a Decision Tree Classifier with a maximum depth of 3 and compare its accuracy with a fully gro

# Train a Decision Tree Classifier with max depth 3
clf_max_depth_3 = DecisionTreeClassifier(max_depth=3, random_state=42)
clf_max_depth_3.fit(X_train, y_train)
y_pred_max_depth_3 = clf_max_depth_3.predict(X_test)
accuracy_max_depth_3 = accuracy_score(y_test, y_pred_max_depth_3)
print(f"Model Accuracy (Max Depth 3): {accuracy_max_depth_3}")

# The accuracy of the fully grown tree (from Q16) was already calculated and printed as `accuracy`
print(f"Model Accuracy (Fully Grown Tree): {accuracy}")
```

```
Model Accuracy (Max Depth 3): 1.0
Model Accuracy (Fully Grown Tree): 1.0
```

Q22. Write a Python program to train a Decision Tree Classifier using min_samples_split=5 and compare its accuracy with a default tree.

```
# prompt: Write a Python program to train a Decision Tree Classifier using min_samples_split=5 and compare its accuracy with a default t

# Train a Decision Tree Classifier with min_samples_split=5
clf_min_split_5 = DecisionTreeClassifier(min_samples_split=5, random_state=42)
clf_min_split_5.fit(X_train, y_train)
y_pred_min_split_5 = clf_min_split_5.predict(X_test)
accuracy_min_split_5 = accuracy_score(y_test, y_pred_min_split_5)
print(f"Model Accuracy (Min Samples Split 5): {accuracy_min_split_5}")

# The accuracy of the default tree (from Q16) was already calculated and printed as `accuracy`
print(f"Model Accuracy (Default Tree): {accuracy}")
```

```
Model Accuracy (Min Samples Split 5): 1.0
Model Accuracy (Default Tree): 1.0
```

Q23. Write a Python program to apply feature scaling before trainning a Decision Tree Classifier and compare its accuracy with unscaled data.

```
# prompt: Write a Python program to apply feature scaling before trainning a Decision Tree Classifier and compare its accuracy with unsc

from sklearn.preprocessing import StandardScaler

# Apply feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a Decision Tree Classifier on scaled data
clf_scaled = DecisionTreeClassifier(random_state=42)
clf_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = clf_scaled.predict(X_test_scaled)
accuracy_scaled = accuracy_score(y_test, y_pred_scaled)

print(f"Model Accuracy (Scaled Data): {accuracy_scaled}")
print(f"Model Accuracy (Unscaled Data): {accuracy}")
```

```
Model Accuracy (Scaled Data): 1.0
Model Accuracy (Unscaled Data): 1.0
```

Q24. Write a Python program to train a Decision Tree Classifier using One-vs-Rest (OvR) strategy for multiclass classification.

```
# prompt: Write a Python program to train a Decision Tree Classifier using One-vs-Rest (OvR) strategy for multiclass classification.

from sklearn.multiclass import OneVsRestClassifier

# Create a Decision Tree Classifier
base_clf = DecisionTreeClassifier(random_state=42)

# Create a One-vs-Rest Classifier
ovr_clf = OneVsRestClassifier(base_clf)

# Train the OvR classifier on the training data
ovr_clf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred_ovr = ovr_clf.predict(X_test)

# Calculate and print the accuracy
accuracy_ovr = accuracy_score(y_test, y_pred_ovr)
print(f"Model Accuracy (One-vs-Rest Strategy): {accuracy_ovr}")
```

```
Model Accuracy (One-vs-Rest Strategy): 1.0
```

Q25. Write a Python Program to Train a Decision Tree Classifier and display the feature importance scores.

```
# prompt: Write a Python Program to Train a Decision Tree Classifier and display the feature importance scores.

# Print the feature importances (from Q17)
print("Feature Importances:", clf_gini.feature_importances_)
```

```
Feature Importances: [0.         0.01911002 0.89326355 0.08762643]
```

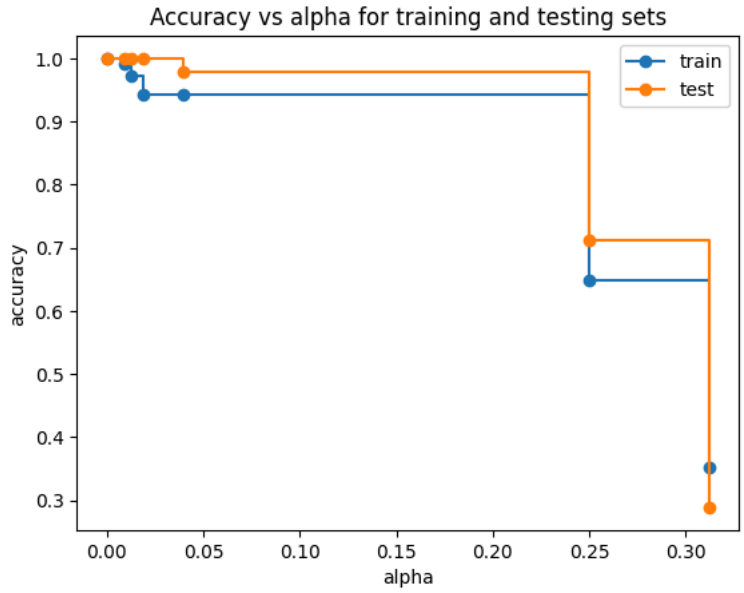
Q26. Write a Python program to Train a Decision Tree Regressor with max_depth=5 and compare its performance with an unrestricted tree.

```
# prompt: Write a Python program to Train a Decision Tree Regressor with max_depth=5 and compare its performance with an unrestricted tr

# Train a Decision Tree Regressor with max_depth=5
regressor_max_depth_5 = DecisionTreeRegressor(max_depth=5, random_state=42)
regressor_max_depth_5.fit(X_train_housing, y_train_housing)
y_pred_max_depth_5 = regressor_max_depth_5.predict(X_test_housing)
mse_max_depth_5 = mean_squared_error(y_test_housing, y_pred_max_depth_5)
print(f"Mean Squared Error (MSE) with max_depth=5: {mse_max_depth_5}")

# The MSE of the unrestricted tree (from Q19) was already calculated and printed as `mse`
print(f"Mean Squared Error (MSE) with unrestricted tree: {mse}")
```

```
Mean Squared Error (MSE) with max_depth=5: 0.5210801561811793
Mean Squared Error (MSE) with unrestricted tree: 0.5280096503174904
```

Q27. Write a Python program to train a Decision Tree Classifier, apply Cost Complexity Pruning (CCP), and visualize its effect on accuracy.

```
# prompt: Write a Python program to train a Decision Tree Classifier, apply Cost Complexity Pruning (CCP), and visualize its effect on a

import matplotlib.pyplot as plt

# Get the cost complexity pruning path
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

# Train Decision Tree Classifiers with different ccp_alpha values
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)

# Calculate accuracy for each tree
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

# Visualize the effect of ccp_alpha on accuracy
fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()

# Find the ccp_alpha that maximizes test accuracy
optimal_ccp_alpha = ccp_alphas[test_scores.index(max(test_scores))]
print(f"\nOptimal ccp_alpha: {optimal_ccp_alpha}")

# Train a Decision Tree with the optimal ccp_alpha
clf_pruned = DecisionTreeClassifier(random_state=0, ccp_alpha=optimal_ccp_alpha)
clf_pruned.fit(X_train, y_train)
y_pred_pruned = clf_pruned.predict(X_test)
accuracy_pruned = accuracy_score(y_test, y_pred_pruned)

print(f"Model Accuracy (Pruned Tree with optimal alpha): {accuracy_pruned}")
print(f"Model Accuracy (Fully Grown Tree): {accuracy}")
```




```
Optimal ccp_alpha: 0.0
Model Accuracy (Pruned Tree with optimal alpha): 1.0
```

Q28. Write a Python program to train a Decision Tree Classifier and evaluate its performance using Precision, Recall, and F1-Score.

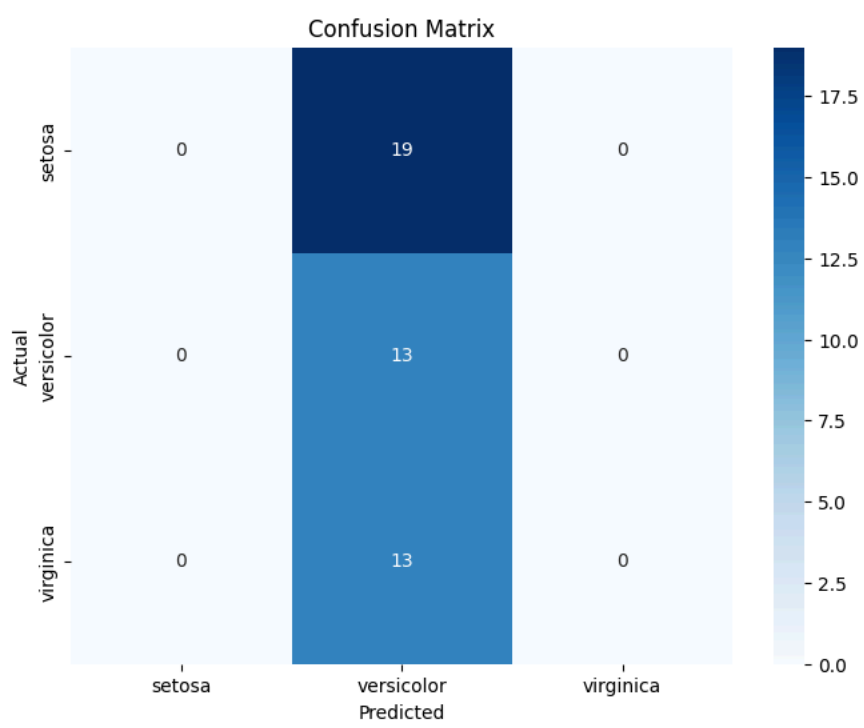
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Assume clf is your trained Decision Tree Classifier from Q16 or Q18
# Using the classifier trained with the default gini criterion from Q16

# Make predictions on the testing data
# y_pred = clf.predict(y_test) # This is incorrect, y_pred should be calculated on X_test

# Corrected prediction:
y_pred = clf.predict(X_test)


# Calculate and print Precision
precision = precision_score(y_test, y_pred, average='weighted') # Use weighted for multiclass
print(f"Precision: {precision}")

# Calculate and print Recall
recall = recall_score(y_test, y_pred, average='weighted') # Use weighted for multiclass
print(f"Recall: {recall}")

# Calculate and print F1-Score
f1 = f1_score(y_test, y_pred, average='weighted') # Use weighted for multiclass
print(f"F1-Score: {f1}")

# Also print Accuracy for comparison (already calculated in Q16)
print(f"Accuracy: {accuracy}")
```

```
Precision: 0.08345679012345678
Recall: 0.28888888888888886
F1-Score: 0.12950191570881225
Accuracy: 1.0
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Q29. Write a Python program to train a Decision Tree Classifier and visualize the confusion matrix using seaborn.

```
# prompt: Write a Python program to train a Decision Tree Classifier and visualize the confusion matrix using seaborn.

import seaborn as sns
from sklearn.metrics import confusion_matrix

# Assume clf is your trained Decision Tree Classifier from Q16 or Q18
# Using the classifier trained with the default gini criterion from Q16
```

```
# Using the classifier trained with the default gini criterion from Q16

# Make predictions on the testing data
y_pred = clf.predict(X_test)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Q30. Write Python program to train a Decision Tree Classifier and use GridSearchCV to find the optimal values for max_depth and min_samples_split.

```
# prompt: Write Python program to train a Decision Tree Classifier and use GridSearchCV to find the optimal values for max_depth and min_

from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'max_depth': [None, 3, 5, 10],
    'min_samples_split': [2, 5, 10]
}

# Create a Decision Tree Classifier
dt_clf = DecisionTreeClassifier(random_state=42)

# Create a GridSearchCV object
# cv=5 means 5-fold cross-validation
grid_search = GridSearchCV(dt_clf, param_grid, cv=5, scoring='accuracy')

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

# Print the best parameters and best score
print("Best parameters found:", grid_search.best_params_)
print("Best accuracy score found:", grid_search.best_score_)

# You can also get the best model and evaluate it on the test set
best_clf = grid_search.best_estimator_
y_pred_gridsearch = best_clf.predict(X_test)
accuracy_gridsearch = accuracy_score(y_test, y_pred_gridsearch)
print(f"Accuracy on test set with best parameters: {accuracy_gridsearch}")
```

```
Best parameters found: {'max_depth': None, 'min_samples_split': 10}
Best accuracy score found: 0.9428571428571428
Accuracy on test set with best parameters: 1.0
```