Decision Tree Learning

Aim:

1) Implement the Decision Tree Learning algorithm to build a decision tree for a given dataset
2) Evaluate the accuracy and effectiveness of the decision tree on test data
3) Visualize and interpret the generated decision tree.

Theory:

A Decision Tree is a supervised machine learning algorithm used for both classification and regression tasks. It's a tree-like model that makes decisions based on a series of conditions or rules learned from the training data. Each internal node in the tree represents a decision rule based on a feature, and each leaf node represents a class label (in classification) or a predicted value (in regression). Decision trees are characterized by their simplicity, interpretability, and ability to handle both categorical and numerical data.

Decision trees work in the following way
1. Root Node: At the root of the tree, the algorithm selects the feature that best separates the data based on a criterion (e.g., Gini impurity or entropy for classification, mean squared error for regression). This feature becomes the root node.

2. Internal Nodes: The algorithm recursively selects features to split the data into subsets at each internal node. These splits are determined to minimize impurity (for classification) or reduce error (for regression).

3. Leaf Nodes: The process continues until a stopping criterion is met, such as a maximum depth of the tree or a minimum number of data points in a node. The final nodes are called leaf nodes and represent the predicted class or value.

Decision trees have several advantages that make them a popular choice for various machine learning and data analysis tasks. Some of the key advantages of decision trees include:

1. Interpretability: Decision trees provide a clear and intuitive representation of the decision-making process. It's easy to understand and explain the logic of a decision tree to non-technical stakeholders. This makes decision trees valuable in domains where model interpretability is essential, such as healthcare and finance.

2. Handling Mixed Data Types: Decision trees can handle both categorical and numerical data without the need for extensive data pre-processing. Other algorithms may require encoding categorical variables or scaling numerical features.

3. No Assumptions About Data Distribution: Decision trees do not assume that the data follows a particular statistical distribution, making them versatile for various types of data.

4. Non-Linearity: Decision trees can capture non-linear relationships between features and the target variable. They can model complex decision boundaries, which is especially useful when the relationship between variables is not linear.

5. Feature Importance: Decision trees can naturally identify feature importance by evaluating which features are used for splitting nodes higher in the tree. This information can guide feature selection and dimensionality reduction efforts.

6. Robustness to Outliers: Decision trees are relatively robust to outliers in the data. Outliers may affect individual branches of the tree but tend to have a limited impact on the overall structure.

7. Scalability: Decision trees are computationally efficient, and the training time complexity is generally linear in the number of training examples and features. This makes them suitable for both small and large datasets.

8. Ensemble Methods: Decision trees can be combined into ensemble methods like Random Forest and Gradient Boosting, which can significantly improve predictive performance by reducing overfitting and increasing robustness.

9. Handling Missing Data: Decision trees can handle missing data by making decisions based on the available features, reducing the need for imputation or data removal.

10. Wide Range of Applications: Decision trees can be applied to various tasks, including classification, regression, anomaly detection, and recommendation systems.

11. Balance Between Bias and Variance: Decision trees can be controlled and pruned to find an appropriate trade-off between bias and variance, which helps mitigate overfitting.

Despite these advantages, it's important to note that decision trees also have limitations, such as their susceptibility to overfitting, instability with small changes in the data, and potential bias towards features with many categories. To address some of these limitations, ensemble methods like Random Forest and Gradient Boosting are often used, combining multiple decision trees to improve model performance.

Data Set:

We download the Iris data set and use it for the present case. As we are performing the practical on Google Colab, we upload the dataset as Iris.csv

Python Code:

```
# Import necessary libraries

import numpy as np
import pandas as pd  # Import Pandas for data loading
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load your dataset from a local file (e.g., CSV)
# Replace 'your_dataset.csv' with the actual path to your dataset file
data = pd.read_csv('Iris.csv')

# Assuming the target variable is in a column named 'target'
X = data.drop('target', axis=1)
y = data['target']

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

# Create a Decision Tree classifier
clf = DecisionTreeClassifier()
```

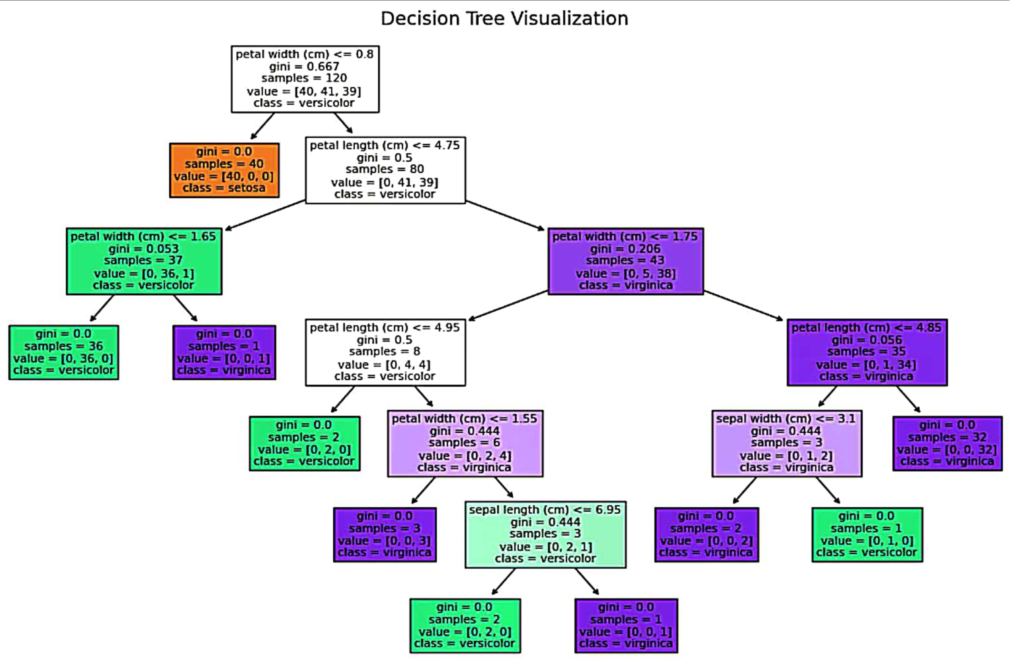
```
# Fit the classifier to the training data
clf.fit(X_train, y_train)

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

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Visualize and interpret the generated decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns,
class_names=y.unique().astype(str))
plt.title("Decision Tree Visualization")
plt.show()
```

Output:


Decision Tree Visualization
petal width (cm) <= 0.8
gini = 0.667
samples = 120
value = [40, 41, 39]
class = versicolor
gini = 0.0
samples = 40
value = [40, 0, 0]
class = setosa
petal length (cm) <= 4.75
gini = 0.5
samples = 80
value = [0, 41, 39]
class = versicolor
petal width (cm) <= 1.65
gini = 0.053
samples = 37
value = [0, 36, 1]
class = versicolor
petal width (cm) <= 1.75
gini = 0.206
samples = 43
value = [0, 5, 38]
class = virginica
gini = 0.0
samples = 36
value = [0, 36, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
petal length (cm) <= 4.95
gini = 0.5
samples = 8
value = [0, 4, 4]
class = versicolor
petal length (cm) <= 4.85
gini = 0.056
samples = 35
value = [0, 1, 34]
class = virginica
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
petal width (cm) <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
class = virginica
sepal width (cm) <= 3.1
gini = 0.444
samples = 3
value = [0, 1, 2]
class = virginica
gini = 0.0
samples = 32
value = [0, 0, 32]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
sepal length (cm) <= 6.95
gini = 0.444
samples = 3
value = [0, 2, 1]
class = versicolor
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

For Video demonstration of the practical click on the link or scan the QR-code

https://youtu.be/fBfRo0to2uA	