Practical 05

Aim- To implement Decision Tree Algorithms

Theory-

A **Decision Tree** is one of the most widely used supervised machine learning algorithms, applicable to both classification and regression tasks. It works by learning simple decision rules from input features and representing them in a tree-like structure.

Structure:

- Each internal node represents a test or condition on a feature.
- Each branch represents the outcome of that condition.
- Each leaf node provides a final prediction or decision.

This hierarchical structure closely mimics human decision-making, making decision trees easy to interpret and visualize. For instance, in a fraud detection system, a decision tree may split transactions based on features such as transaction amount, frequency, or source.

How Decision Trees Work
The algorithm recursively splits the dataset into subsets using the most informative features. Splitting continues until a stopping condition is met (like maximum depth or all data classified).

Splitting Criteria (for classification):

- **Gini Index**
- **Information Gain (Entropy-based)**
- Chi-Square

Regression Trees: Instead of purity measures, they minimize variance within subsets.

A trained decision tree predicts new data by traversing from the root node to a leaf node according to feature values.

Advantages

1. High interpretability – decisions are transparent.
2. Can handle both numerical and categorical data.

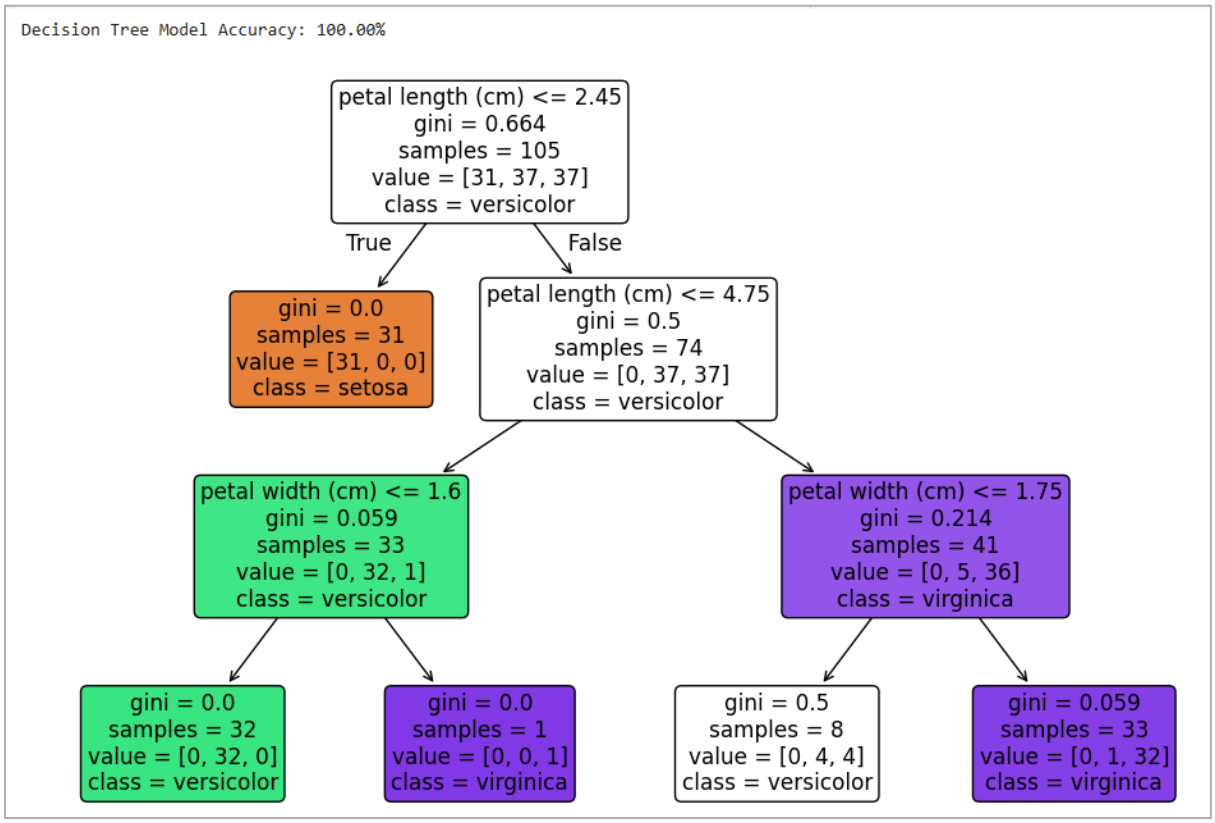
Limitations:

1. Prone to overfitting if the tree becomes too deep.
2. Sensitive to small changes in data.

Code-

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# Load dataset (Iris flower dataset)
iris = load_iris()
X = iris.data        # Features
y = iris.target      # Target labels

# Split data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Initialize Decision Tree Classifier
# criterion can be "gini" or "entropy"
clf = DecisionTreeClassifier(criterion="gini", max_depth=3,
random_state=42)

# Train the model
clf.fit(X_train, y_train)

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

# Evaluate accuracy
accuracy = clf.score(X_test, y_test)
print("Decision Tree Model Accuracy: {:.2f}%".format(accuracy * 100))

# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(
    clf,
    feature_names=iris.feature_names,
    class_names=iris.target_names,
    filled=True,
    rounded=True
)
plt.show()
```

Output-



Conclusion- In this practical, we successfully implemented the Decision Tree Algorithm and demonstrated its ability to classify blockchain transactions as legitimate or fraudulent based on features like transaction amount, frequency, and reputation scores. The experiment highlighted the strengths of decision trees in terms of simplicity, accuracy, and interpretability, making them highly suitable for applications such as fraud detection and anomaly detection in blockchain systems.