Experiment – 1.4

Name: Samuel **UID :24MAI10018**

Branch: AI&ML **Section/Group: MAI – 1**

Subject: Machine Learning Lab **Subject Code: 24CSH – 667**

Semester - 2 **Date of Performance:**

Aim – Implement Decision Tree algorithm using Python.

Software Required: Python, IDE like Visual Studio Code or jupyter notebook, Python libraries like pandas, matplotlib, scikit-learn

Theory

A Decision Tree is a popular supervised learning algorithm used for both classification and regression tasks. Decision Trees are a powerful and interpretable machine learning technique for classification and regression. Their ability to handle diverse data types, visualize decisions, and model non-linear relationships makes them widely used across various domains. It models data by recursively splitting it into smaller subsets based on feature values, creating a hierarchical tree structure. Each internal node represents a decision based on a feature, branches indicate possible outcomes, and leaf nodes represent final predictions or class labels. Decision Trees work similarly to a flowchart, where each decision leads to further sub-decisions, ultimately arriving at a final output. The primary objective of the Decision Tree algorithm is to create a model that accurately predicts the target variable by learning decision rules inferred from training data.

The Decision Tree algorithm follows these steps:

Select the Best Feature: The dataset is split based on the feature that provides the highest information gain or lowest Gini impurity.

Recursive Splitting: The dataset is divided into subsets based on feature values until a stopping criterion is met. This process is repeated for each subset, forming deeper levels of the tree.

Assign Class Labels: Once splitting is completed, the leaf nodes are assigned the majority class (in classification) or the average value (in regression).

Tree Pruning (Optional): To prevent overfitting, the tree may be pruned by removing branches that do not contribute significantly.

Pruning Methods

Pre-pruning: Stops tree growth early based on criteria like maximum depth.

Post-pruning: Removes branches after the tree is fully grown, using cross-validation.

Algorithm

Step 1: Importing Libraries

Import necessary libraries such as pandas for data manipulation, numpy for numerical computations, and scikit-learn for implementing the Naïve Bayes classifier.

Step 2: Loading the Dataset

The load_iris() function loads the Iris dataset.

The dataset is converted into a Pandas DataFrame for better manipulation.

Step 3: Data Pre-processing

The species labels (0, 1, 2) are mapped to their corresponding names.

Independent variables (x) contain the four feature columns.

The dependent variable (y) contains the species labels.

Step 4: Splitting the Data

The dataset is split into a training set (70%) and a testing set (30%).

Step 5: Building the Decision Tree Model

An instance of DecisionTreeClassifier is created with a maximum depth of 3.

The model is trained using the training data.

Step 5: Visualizing the Decision Tree

The plot_tree() function generates a graphical representation of the decision tree.

It displays feature splits and decision nodes.

Step 6: Making Predictions

The trained model is used to predict species for the test data.

Step 7: Evaluating the Model

A confusion matrix compares predicted vs. actual labels.

A heatmap visually represents the confusion matrix using seaborn.heatmap().

Code:

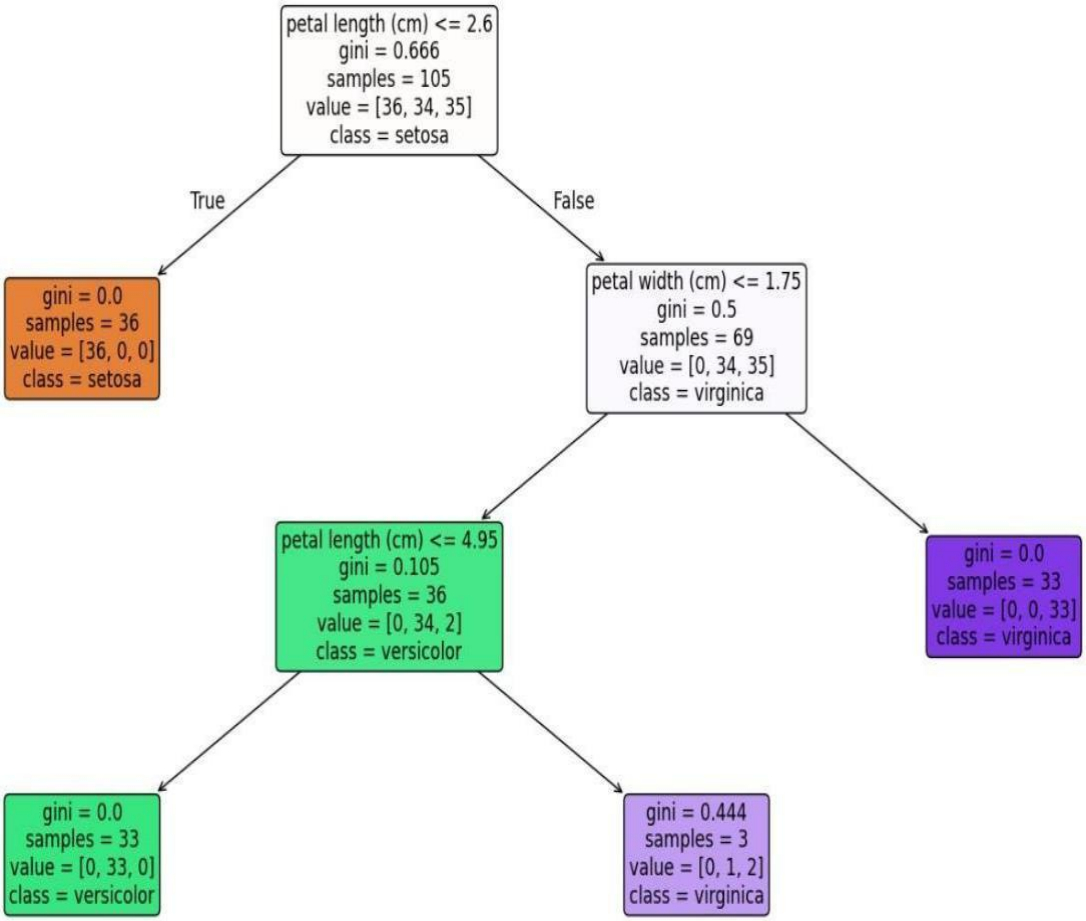
```
# Import necessary libraries

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

```
from sklearn import metrics
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Loading the dataset
iris = load_iris()

# Converting the data to a pandas dataframe
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Creating a separate column for the target
variable data['Species'] = iris.target

# Replacing target variable categories with actual species names
target_dict = dict(zip(np.unique(iris.target), np.unique(iris.target_names)))

data['Species'] = data['Species'].replace(target_dict)

# Separating the independent and dependent
variables x = data.drop(columns="Species")
y = data["Species"]

# Splitting the dataset into training and testing datasets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=93)

# Creating an instance of the Decision Tree classifier
dtc = DecisionTreeClassifier(max_depth=3, random_state=93)

y_pred = classifier.predict(X_test)

# Fitting the training dataset to the
model dtc.fit(x_train, y_train)

# Plotting the Decision Tree
plt.figure(figsize=(20, 10))
plot_tree(dtc, feature_names=x.columns, class_names=y.unique(), rounded=True,
filled=True, fontsize=12) plt.show()

# Making predictions on the test set
y_pred = dtc.predict(x_test)

# Computing the confusion matrix
# Compute the confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)

# Get unique class labels dynamically
class_labels = np.unique(y_test) # Extracts unique class labels
```

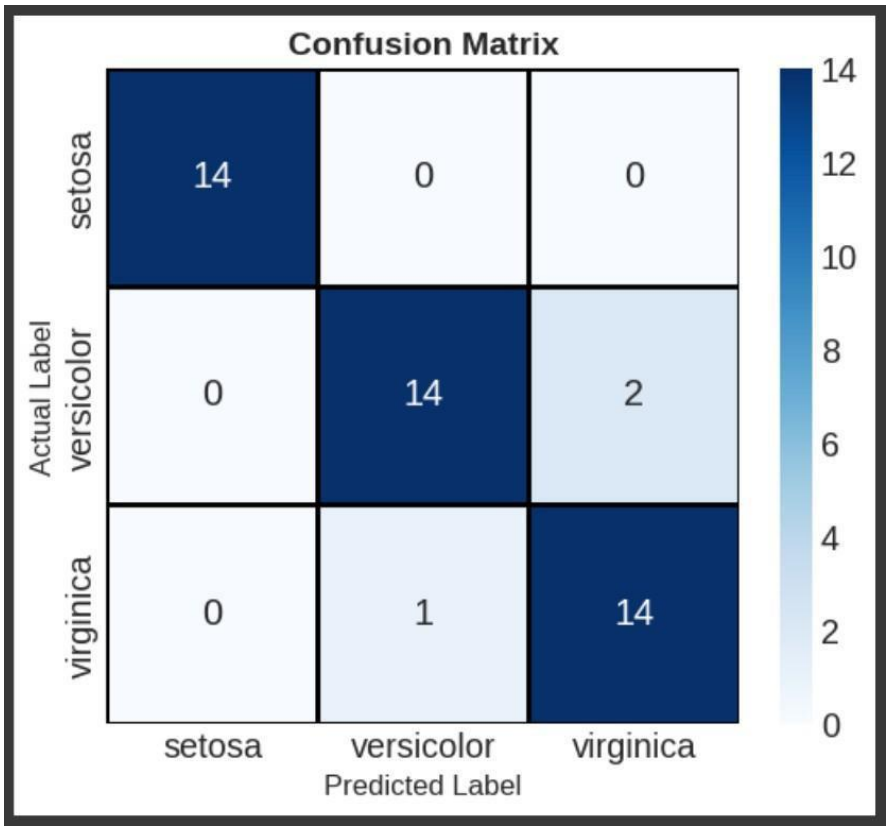
```
# Convert matrix to a labeled DataFrame
matrix = pd.DataFrame(conf_matrix, index=class_labels, columns=class_labels)

# Plot confusion matrix using a heatmap
plt.figure(figsize=(6, 5), dpi=100)
sns.heatmap(matrix, annot=True, fmt="d", cmap="Blues", linewidths=1, linecolor="black")

# Add labels and title
plt.title("Confusion Matrix", fontsize=14, fontweight='bold')
plt.xlabel("Predicted Label", fontsize=12)
plt.ylabel("Actual Label", fontsize=12)

# Show the plot
plt.show()
```

Output:

Decision Tree



Confusion Matrix



Learning Outcomes:

1. I understood the working of the Decision Tree.
2. I visualized the Decision Tree using the plot_tree() function to interpret decision-making.
3. I implemented the decision tree for iris dataset.
4. I learned how to compute and analyze a confusion matrix to evaluate classification performance.
5. I used seaborn to create a heatmap for better visualization of the confusion matrix.