Name: Srishti Pandey
Class-Roll No.: TY9-40
Batch: B
PRN: 22UF17054CM100

Experiment No. 3

Aim : Implementation of Classification Algorithm (Decision Tree / Naïve Bayes) using Python.

Introduction :
• Classification is a supervised machine learning task to predict the class label of data.
• Decision Trees: Tree-like models using a series of rules based on data features for
predictions. Easy to interpret and visualize.
• Naïve Bayes: Probabilistic algorithm based on Bayes' theorem. Assumes feature
independence. Calculates probabilities of a data point belonging to each class and
predicts the class with the highest probability.

Procedure :
1. Import Necessary Libraries
2. Load and Prepare the Dataset
3. Split Data into Training and Testing Sets
4. Create and Train the Decision Tree Classifier
5. Make Predictions
6. Evaluate the Model
7. Visualize the Decision Tree

Source Code :

Name: Srishti Pandey

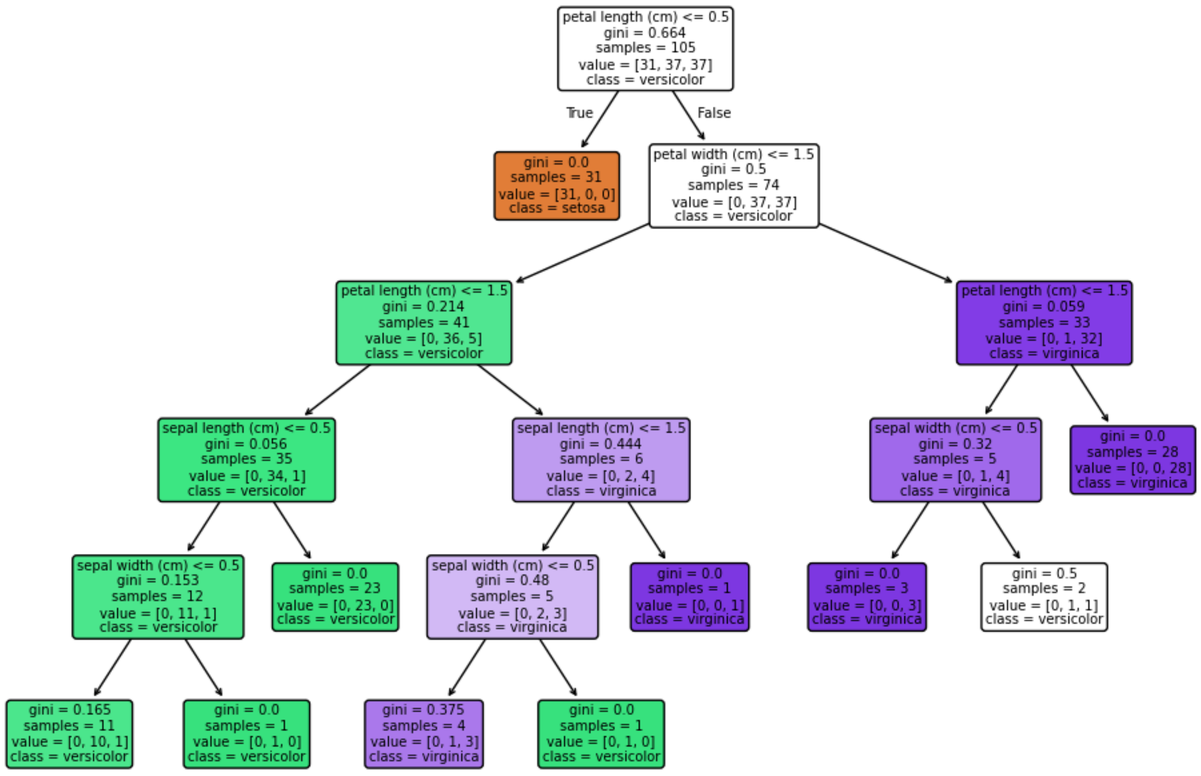
Class-Roll No.: TY9-40

Batch: B

PRN: 22UF17054CM100

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


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


df = pd.DataFrame(X, columns=columns)
df['species'] = y


def equal_width_binning(df, num_bins):
    discretized_data = df.copy()
    for column in df.columns:
        if column != 'species':  # Skip target column
            discretized_data[column] = pd.cut(df[column], bins=num_bins, labels=False)
    return discretized_data

# Apply discretization to the feature columns
discretized_df = equal_width_binning(df, num_bins=3)  # Discretize into 3 bins
print("Discretized Data:\n", discretized_df.head())

# Step 3: Split the data into training and testing sets
X_discretized = discretized_df.drop('species', axis=1)
y_discretized = discretized_df['species']
X_train, X_test, y_train, y_test = train_test_split(X_discretized, y_discretized, test_size=0.3, random_state=42)

# Step 4: Train a Decision Tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Step 5: Make predictions and evaluate the model
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of Decision Tree on Discretized Data: {accuracy * 100:.2f}%")

# Step 6: Visualize the Decision Tree
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
plot_tree(clf, feature_names=X_discretized.columns, class_names=iris.target_names, filled=True, rounded=True)
plt.show()
```

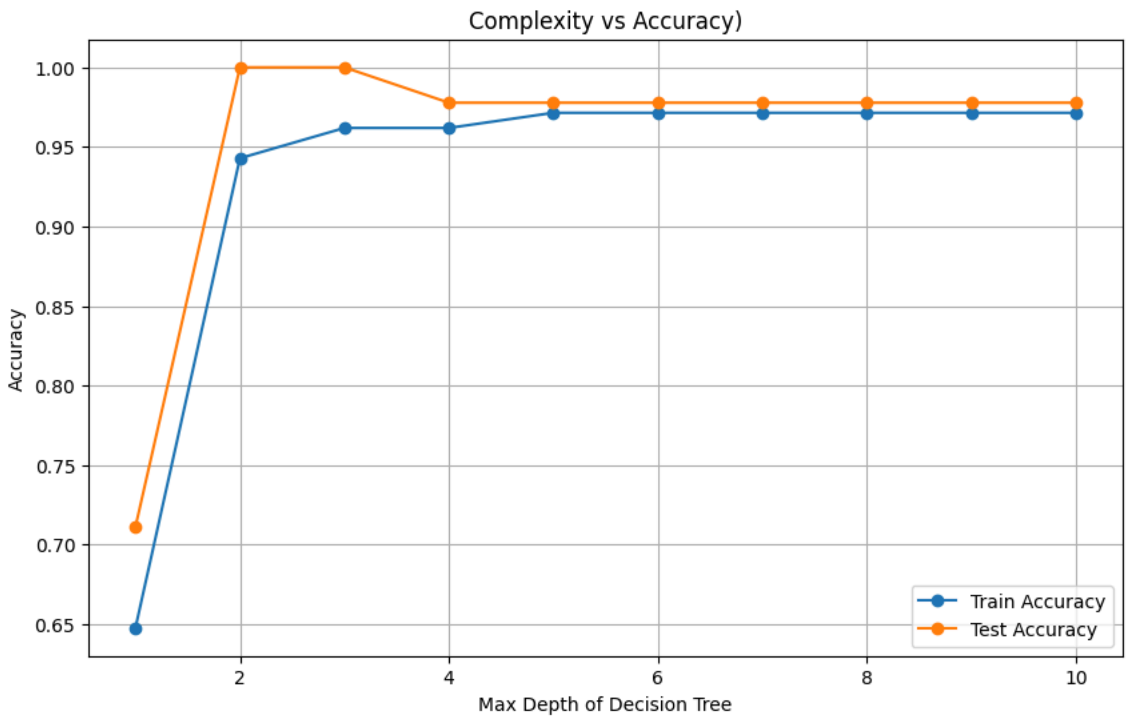
```
Discretized Data:
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  0                 1                  0                 0   
1                  0                 1                  0                 0   
2                  0                 1                  0                 0   
3                  0                 1                  0                 0   
4                  0                 1                  0                 0   

   species  
0        0  
1        0  
2        0  
3        0  
4        0  
Accuracy of Decision Tree on Discretized Data: 97.78%
```



```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
columns = iris.feature_names

# Convert to DataFrame for easier manipulation
df = pd.DataFrame(X, columns=columns)
df['species'] = y

# Step 2: Apply Discretization (Equal-Width Binning)
def equal_width_binning(df, num_bins):
    discretized_data = df.copy()
    for column in df.columns:
        if column != 'species':  # Skip target column
            discretized_data[column] = pd.cut(df[column], bins=num_bins, labels=False)
    return discretized_data

# Apply discretization to the feature columns
discretized_df = equal_width_binning(df, num_bins=3)  # Discretize into 3 bins
print("Discretized Data:\n", discretized_df.head())

# Step 3: Split the data into training and testing sets
X_discretized = discretized_df.drop('species', axis=1)
y_discretized = discretized_df['species']
X_train, X_test, y_train, y_test = train_test_split(X_discretized, y_discretized, test_size=0.3, random_state=42)

# Step 4: Train Decision Tree with varying max_depth and evaluate accuracy
train_accuracies = []
```

```
test_accuracies = []

# Vary max_depth from 1 to 10
for max_depth in range(1, 11):
    clf = DecisionTreeClassifier(max_depth=max_depth, random_state=42)
    clf.fit(X_train, y_train)

    # Make predictions on train and test sets
    y_train_pred = clf.predict(X_train)
    y_test_pred = clf.predict(X_test)

    # Calculate accuracies
    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)

# Step 5: Plot the Model Complexity (max_depth) vs Accuracy
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), train_accuracies, label='Train Accuracy', marker='o')
plt.plot(range(1, 11), test_accuracies, label='Test Accuracy', marker='o')
plt.xlabel('Max Depth of Decision Tree')
plt.ylabel('Accuracy')
plt.title('Complexity vs Accuracy)')
plt.legend()
plt.grid(True)
plt.show()
```

```
Discretized Data:
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  0                 1                  0                 0
1                  0                 1                  0                 0
2                  0                 1                  0                 0
3                  0                 1                  0                 0
4                  0                 1                  0                 0

   species
0        0
1        0
2        0
3        0
4        0
```



Conclusion: In this experiment, we implemented Decision Tree and Naïve Bayes for classification using Python. The Decision Tree offered an intuitive and interpretable model but was prone to overfitting, while Naïve Bayes, based on probabilistic principles, performed efficiently, particularly when feature independence was assumed. Our analysis highlighted that both algorithms have distinct advantages depending on the dataset characteristics. Overall, both classifiers proved to be valuable tools for data classification, underscoring their relevance in Data Warehousing and Mining (DWM) applications.

Start coding or generate with AI.

Review Questions :

1. **What is a Decision Tree classifier, and how does it work?**

Ans: A Decision Tree classifier is a supervised learning algorithm used for classification tasks. It structures decisions in a tree-like format by recursively splitting data based on feature values.

- Nodes represent attributes or decision points.
- Branches indicate possible outcomes of decisions.
- Leaf nodes contain the final class labels.

The algorithm selects the most relevant attribute using a splitting criterion (e.g., Gini Index, Information Gain) and continues until a stopping condition is met. While highly interpretable, Decision Trees can overfit the data, which can be controlled using pruning techniques.

2. **Explain the Naïve Bayes algorithm and its underlying assumptions.**

Ans: Naïve Bayes is a probabilistic classifier based on Bayes' Theorem, which calculates the probability of a class given input features:

$$P(X/Y) \ = \frac{P(X/Y)\ P(Y)}{P(X)}$$

Assumptions:

1. Feature Independence: Each feature contributes independently to the probability of a class.
2. Equal Importance of Features: Assumes all features are equally relevant.

Despite its simplicity, Naïve Bayes is efficient and performs well in text classification, spam detection, and sentiment analysis.

3. **Compare the working principles of Decision Tree and Naïve Bayes classifiers.**

Ans:

Feature	**Decision Tree**	**Naïve Bayes**
Type	Rule-based model	Probabilistic model
Working Principle	Splits data based on feature values using criteria like Gini Index or Information Gain	Uses Bayes' theorem to compute class probabilities
Feature Dependence	Considers relationships between features	Assumes feature independence
Interpretability	Highly interpretable, easy to visualize	Less interpretable due to probability calculations
Speed	Slower for large datasets	Fast, even for high-dimensional data

4. **What are the different types of Decision Tree splitting criteria?**

Ans:

- Gini Index – Measures impurity; lower values indicate better splits.

$$Gini(D) = 1 - \Sigma P_i^2$$

- Entropy & Information Gain – Entropy measures disorder, and Information Gain selects attributes that reduce uncertainty.

$$Entropy(D) = -\Sigma P_i \, log_2 P_i$$

$$Entropy(D) = Entropy(D) - \Sigma \left(\frac{|D_i|}{|D|} \times Entropy(D_j) \right)$$

- Gain Ratio – Normalized version of Information Gain to avoid bias toward attributes with many categories.
- Chi-Square – Measures statistical significance of attribute splits.
- Reduction in Variance – Used in regression trees to minimize variance in target values.

Github Link: https://github.com/SrishtiPandey15/DWM-Batch-B-Exps