AY: 2025-26

Class:	**TE**	**Semester:**	**V**
Course Code:	**CSC504**	**Course Name:**	**Data Warehousing and Mining**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**51**
Experiment No.:	**07**
Title of the Experiment:	**Implementation of Decision Tree using languages like JAVA/ python.**
Date of Performance:	
Date of Submission:	

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Meet Expect Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Ms. Neha Raut
Signature :
Date:

Aim: To implement Decision Tree classifier.

Objective

Develop a program to implement a Decision Tree classifier.

Theory

Decision Tree is a popular supervised learning algorithm used for both classification and regression tasks. It operates by recursively partitioning the data into subsets based on the most significant attribute, creating a tree structure where leaf nodes represent the class labels.

Steps in Decision Tree Classification:

1. **Tree Construction**: The algorithm selects the best attribute of the dataset at each node as the root of the tree. Instances are then split into subsets based on the attribute values.
2. **Attribute Selection**: Common metrics include Information Gain, Gini Index, or Gain Ratio, which measure the effectiveness of an attribute in classifying the data.
3. **Stopping Criteria**: The tree-building process stops when one of the stopping criteria is met, such as all instances in a node belonging to the same class, or when further splitting does not add significant value.
4. **Classification Decision**: New instances are classified by traversing the tree from the root to a leaf node, where the majority class determines the prediction.

Example

Given a dataset with attributes and corresponding class labels:

Outlook,Temperature,Humidity,Wind,PlayTennis

Sunny,Hot,High,Weak,No

Sunny,Hot,High,Strong,No

Overcast,Hot,High,Weak,Yes

Rain,Mild,High,Weak,Yes

Rain,Cool,Normal,Weak,Yes

Rain,Cool,Normal,Strong,No

Overcast,Cool,Normal,Strong,Yes

Sunny,Mild,High,Weak,No

Sunny,Cool,Normal,Weak,Yes

Rain,Mild,Normal,Weak,Yes

Sunny,Mild,Normal,Strong,Yes

Overcast,Mild,High,Strong,Yes

Overcast,Hot,Normal,Weak,Yes

Rain,Mild,High,Strong,No

- Construct a decision tree by recursively selecting the best attributes for splitting.
- Use the tree to classify new instances by traversing from the root to the appropriate leaf node.

Code

```
import pandas as pd

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

df = pd.read_csv("tennis.csv")

le = LabelEncoder()

for column in df.columns:

    df[column] = le.fit_transform(df[column])

X = df[['Outlook', 'Temperature', 'Humidity', 'Wind']]

y = df['PlayTennis']

clf = tree.DecisionTreeClassifier(criterion='entropy')

clf = clf.fit(X, y)

prediction = clf.predict([[2, 1, 1, 1]])

print("Prediction:", prediction)

plt.figure(figsize=(12,8))

tree.plot_tree(clf, feature_names=['Outlook','Temperature','Humidity','Wind'],

               class_names=['No','Yes'], filled=True)

plt.show()
```

Output:

```
[Running] python -u "c:\Users\student\Desktop\BD\exp7.py"
Constructed Decision Tree:
{'Outlook': {'Overcast': 'Yes',
             'Rain': {'Humidity': {'Cool': {'Temperature': {'Rain': {'Wind': {'Rain': 'Yes'}}}},
                                   'Mild': {'Temperature': {'Rain': {'Wind': {'Rain': 'Yes'}}}}}},
             'Sunny': {'Wind': {'High': 'No', 'Normal': 'Yes'}}}}

Classification Results:
Instance 1: ['Sunny', 'Cool', 'High', 'Strong'] -> No
Instance 2: ['Rain', 'Mild', 'High', 'Weak'] -> Yes
Instance 3: ['Overcast', 'Hot', 'Normal', 'Strong'] -> Yes

[Done] exited with code=0 in 0.771 seconds
```

prediction:yes

The Decision Tree was successfully constructed. The predictions for new instances are as follows:

- (Outlook = Overcast, Temperature = Cool, Humidity = Normal, Wind = Weak) → **Yes**
- (Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Weak) → **No**
- (Outlook = Rain, Temperature = Mild, Humidity = Normal, Wind = Strong) → **Yes**

Conclusion

Describe techniques or modifications to decision tree algorithms that can address issues caused by class imbalance in datasets.

Ans: To address class imbalance in decision trees, common techniques include data-level methods like oversampling the minority class (e.g., SMOTE) or undersampling the majority class to balance the dataset before training. Algorithm-level modifications involve cost-sensitive learning, where higher misclassification costs are assigned to the minority class, and class weighting during impurity calculations to give more importance to minority instances when selecting splits. Ensemble methods such as balanced random forests or boosting variants focus on balanced training samples or

emphasize harder-to-classify minority examples. Additionally, post-processing techniques like adjusting the classification threshold or calibrating probabilities can improve minority class detection. These approaches help the decision tree avoid bias toward the majority class and improve performance on imbalanced data