| **Name:** | **James Lewis** |
| --- | --- |
| **Roll No:** | **32** |
| **Class/Sem:** | TE/V |
| **Experiment No.:** | 7 |
| **Title:** | Implementation of Decision Tree using languages like JAVA/ python. |
| **Date of Performance:** |  |
| **Date of Submission:** |  |
| **Marks:** |  |
| **Sign of Faculty:** |  |

**Aim:** To implement a 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:

//add dataset

* 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.tree import DecisionTreeClassifier, plot\_tree

import matplotlib.pyplot as plt

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast',

'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool',

'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal',

'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong',

'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',

'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

}

df = pd.DataFrame(data)

X = pd.get\_dummies(df[['Outlook', 'Temperature', 'Humidity', 'Wind']])

y = df['PlayTennis']

clf = DecisionTreeClassifier(criterion="entropy", max\_depth=4)

clf.fit(X, y)

plt.figure(figsize=(20,12))

plot\_tree(clf, feature\_names=X.columns, class\_names=clf.classes\_,

filled=True, rounded=True, fontsize=12)

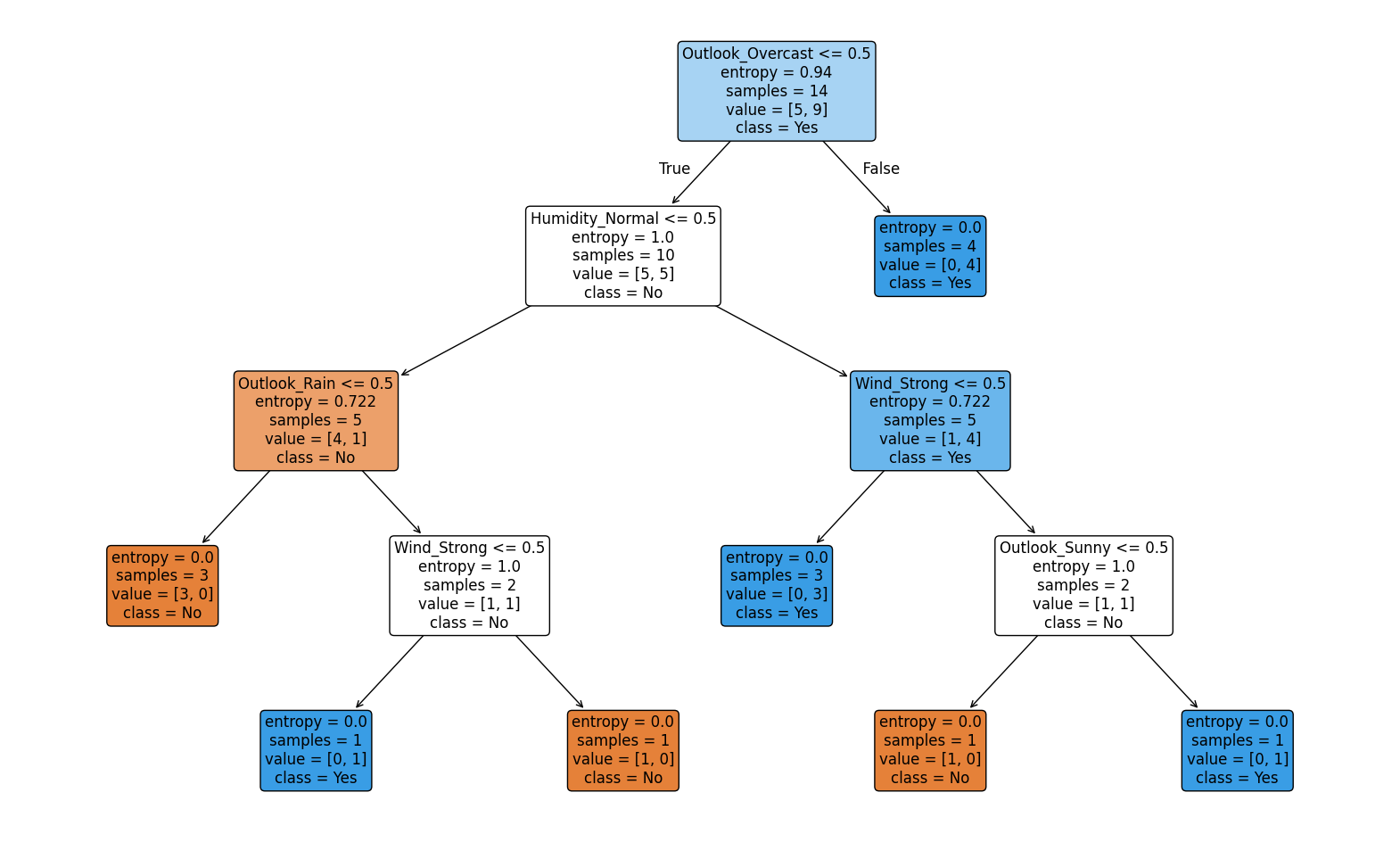
plt.show()

test = pd.DataFrame([[1,0,0, 0,0,1, 0,1, 0,1]], columns=X.columns)

prediction = clf.predict(test)

print("Prediction for (Sunny, Mild, Normal, Weak):", prediction[0])

**Output:**

* Predict the class label for new instances based on the constructed decision tree.  
    
  Prediction for (Sunny, Mild, Normal, Weak): Yes

#### Conclusion

**Describe techniques or modifications to decision tree algorithms that can address issues caused by class imbalance in datasets.**  
Techniques to Address Class Imbalance in Decision Trees

1. Class Weights (Cost-Sensitive Learning)  
   * Assign higher weights to minority class samples during training.
   * In scikit-learn, this can be done using DecisionTreeClassifier(class\_weight="balanced").
   * Ensures that the tree does not always favor the majority class.
2. Resampling the Dataset  
   * Oversampling minority class: Duplicate or synthetically generate more minority samples (e.g., SMOTE – Synthetic Minority Oversampling Technique).
   * Undersampling majority class: Reduce the number of samples in the majority class.
   * This balances the class distribution before training.
3. Pruning the Tree  
   * Deep decision trees tend to overfit majority classes.
   * Using pre-pruning (max\_depth, min\_samples\_split) or post-pruning can reduce bias.
4. Ensemble Methods  
   * Use algorithms like Random Forest or Gradient Boosted Trees with class balancing.
   * Balanced Random Forest trains each tree on a balanced bootstrap sample.
5. Evaluation Metrics Adjustment  
   * Instead of accuracy (which can be misleading with imbalance), use F1-score, Precision, Recall, ROC-AUC to guide tree construction and evaluation.