Name:	Pranita Kumbhar
Roll No:	70
Class/Sem:	TE/V
Experiment No.:	7
Title:	Implementation of Decision Tree using languages like JAVA/ python.
Date of Performance:	4/9/25
Date of Submission:	11/9/25
Marks:	
Sign of Faculty:	

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:

//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.

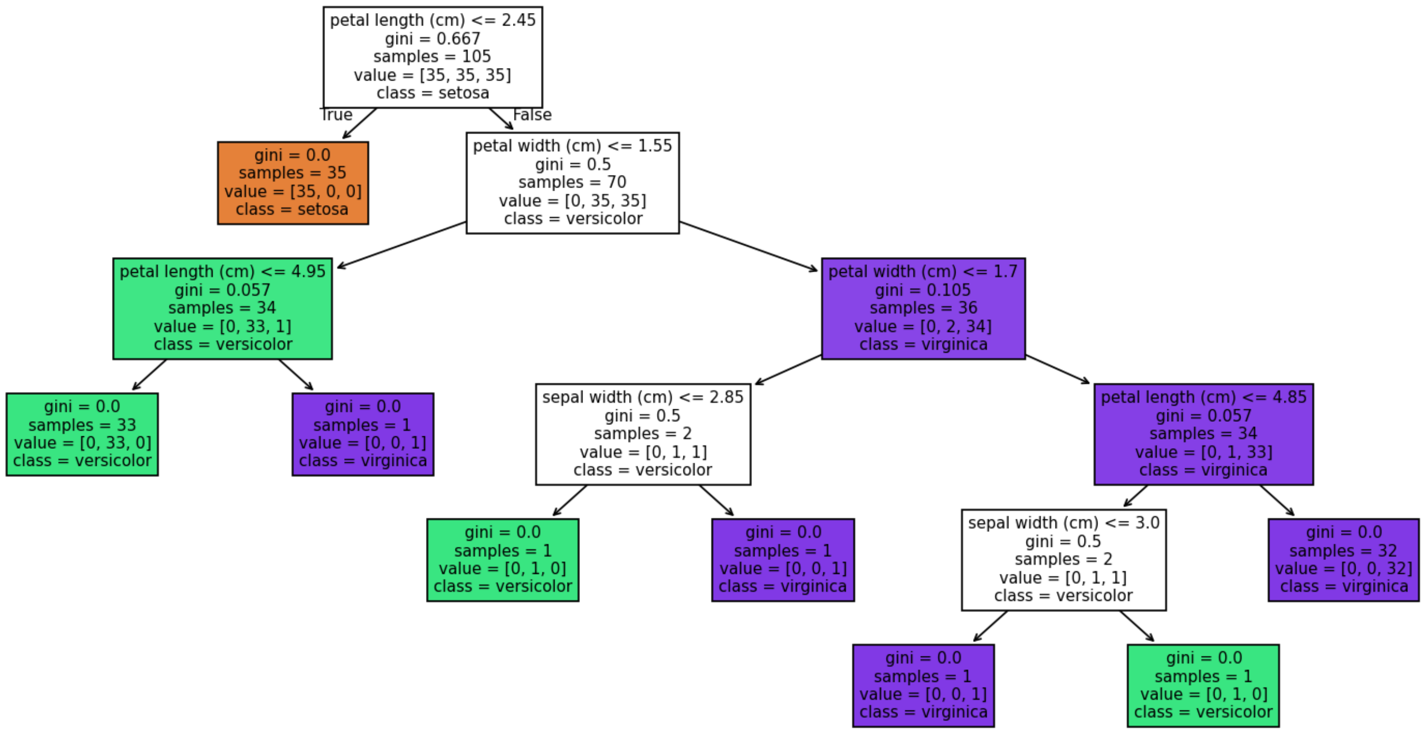
Code

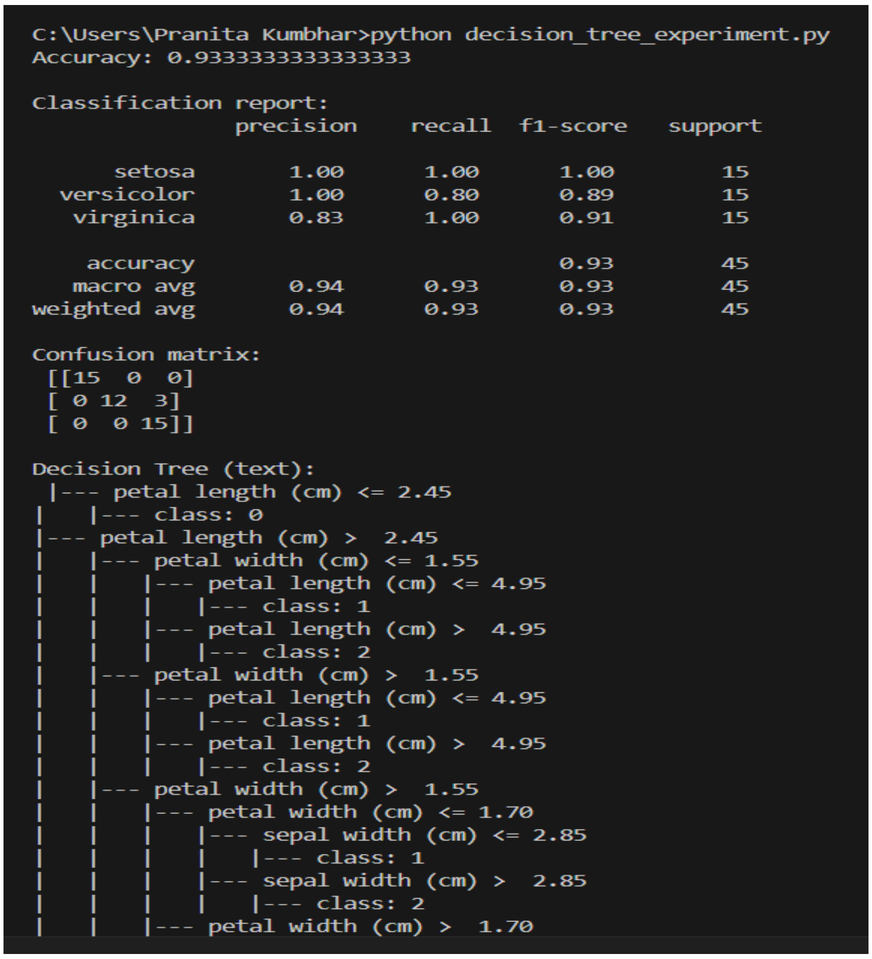
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree, export_graphviz
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt

# 1. Load dataset
```

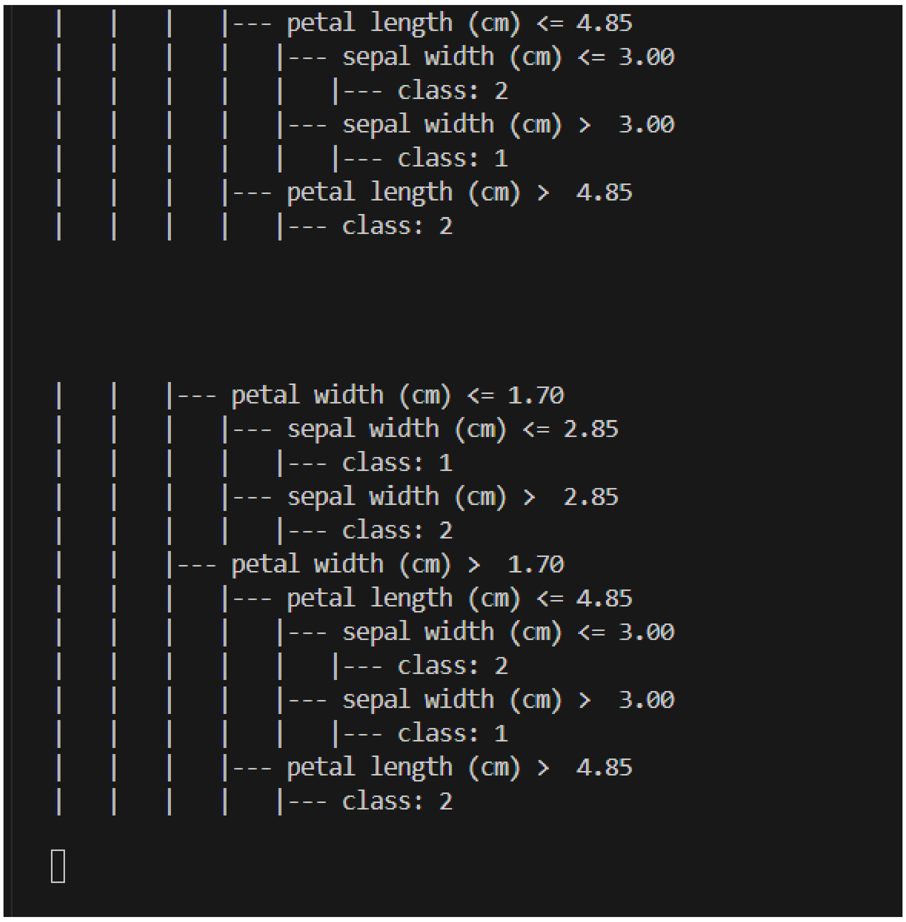
```
data = load_iris()
X = data.data
y = data.target
feature_names = data.feature_names
target_names = data.target_names

# 2. Split
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.3, random_state=42, stratify=y
)

# 3. Create classifier
clf = DecisionTreeClassifier(criterion='gini', max_depth=None, random_state=42)
clf.fit(X_train, y_train)

# 4. Evaluate
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification report:\n", classification_report(y_test, y_pred,
target_names=target_names))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))

# 5. Print text form of the tree
print("\nDecision Tree (text):\n", export_text(clf, feature_names=feature_names))

# 6. Plot tree
plt.figure(figsize=(12,8))
plot_tree(clf, feature_names=feature_names, class_names=target_names, filled=True)
plt.title("Decision Tree")
plt.savefig("decision_tree_plot.png")   # saves image
plt.show()

# 7. (Optional) cross-validation
scores = cross_val_score(clf, X, y, cv=5)
print("5-fold CV accuracy: mean={:.3f}, std={:.3f}".format(scores.mean(), scores.std()))
```

Output:

- Predict the class label for new instances based on the constructed decision tree.




```
C:\Users\Pranita Kumbhar>python decision_tree_experiment.py
Accuracy: 0.9333333333333333

Classification report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       1.00      0.80      0.89        15
   virginica       0.83      1.00      0.91        15

    accuracy                           0.93        45
   macro avg       0.94      0.93      0.93        45
weighted avg       0.94      0.93      0.93        45

Confusion matrix:
 [[15  0  0]
 [ 0 12  3]
 [ 0  0 15]]

Decision Tree (text):
 |--- petal length (cm) <= 2.45
|   |--- class: 0
|--- petal length (cm) >  2.45
|   |--- petal width (cm) <= 1.55
|   |   |--- petal length (cm) <= 4.95
|   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.95
|   |   |   |--- class: 2
|   |--- petal width (cm) >  1.55
|   |   |--- petal length (cm) <= 4.95
|   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.95
|   |   |   |--- class: 2
|   |--- petal width (cm) >  1.55
|   |   |--- petal width (cm) <= 1.70
|   |   |   |--- sepal width (cm) <= 2.85
|   |   |   |   |--- class: 1
|   |   |   |--- sepal width (cm) >  2.85
|   |   |   |   |--- class: 2
|   |   |--- petal width (cm) >  1.70
```



```
|   |   |   |--- petal length (cm) <= 4.85
|   |   |   |   |--- sepal width (cm) <= 3.00
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- sepal width (cm) >  3.00
|   |   |   |   |   |--- class: 1
|   |   |   |--- petal length (cm) >  4.85
|   |   |   |   |--- class: 2




|   |   |--- petal width (cm) <= 1.70
|   |   |   |--- sepal width (cm) <= 2.85
|   |   |   |   |--- class: 1
|   |   |   |--- sepal width (cm) >  2.85
|   |   |   |   |--- class: 2
|   |   |--- petal width (cm) >  1.70
|   |   |   |--- petal length (cm) <= 4.85
|   |   |   |   |--- sepal width (cm) <= 3.00
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- sepal width (cm) >  3.00
|   |   |   |   |   |--- class: 1
|   |   |   |--- petal length (cm) >  4.85
|   |   |   |   |--- class: 2
```

Conclusion

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

➢

class imbalance (where one class has many more samples than others) can make a Decision Tree biased toward predicting the majority class. Here are the **main techniques and modifications** you can describe in your experiment report:

1. Resampling Methods

- **Oversampling the minority class**: Duplicate existing minority samples or generate synthetic ones (e.g., **SMOTE – Synthetic Minority Oversampling Technique**).
- **Undersampling the majority class**: Randomly remove some majority class samples so the dataset becomes balanced.
- **Hybrid methods**: Combine oversampling and undersampling for better results.

2. Class Weights in Decision Trees

- Most implementations (e.g., DecisionTreeClassifier in scikit-learn) allow class_weight='balanced'.
- This tells the algorithm to **assign higher penalties for misclassifying minority samples**, making the tree pay more attention to them.

3. Pruning & Minimum Samples

- Adjusting parameters like min_samples_split or min_samples_leaf can **prevent the tree from overfitting to the majority class**.
- Pruning (removing weak branches) avoids splits that only capture noise from the majority.

4. Cost-Sensitive Learning

- Instead of resampling, you can **assign higher misclassification costs to minority classes**.
- This is like saying: "Predicting a minority class instance incorrectly is more serious than predicting a majority one incorrectly."

5. Ensemble Methods

- **Random Forests** and **Boosting (AdaBoost, XGBoost, LightGBM)** work better with imbalanced datasets.
- They often allow built-in imbalance handling (scale_pos_weight in XGBoost, class_weight in RandomForest).
- Boosting in particular focuses on **hard-to-classify (often minority) samples**.

6. Threshold Moving / Probability Calibration

- After training, you can adjust the **decision threshold** (default = 0.5) to favor recall of the minority class.
- For example, lower the threshold for predicting the minority class so the model catches more of them.