Assignment No. 3

Problem Statement: Implement and analyze the Decision Tree algorithm for classification and regression.

Objective: To understand and implement the **Decision Tree Classification modeling**. The process includes handling missing data, encoding categorical variables, analyzing feature importance, visualizing decision trees, and evaluating model performance..

Prerequisite :

1. A Python environment with essential libraries like pandas, numpy, matplotlib, seaborn, and scikit-learn.
2. Basic knowledge of Python, statistics, and machine learning principles.
3. Understanding of Decision Tree Classification and its concepts, such as Gini Impurity, Entropy, and Overfitting.

Theory :

What is a Decision Tree?

A **Decision Tree** is a supervised learning algorithm used for **classification problems**. It works by **splitting data** into branches based on **feature conditions**, leading to a **tree-like structure** where leaves represent final classifications.

How Decision Trees Work?

Decision trees operate by systematically dividing a dataset into smaller, more homogeneous subsets. This process is carried out using a set of rules that determine how data points are split at each decision node. The aim is to make predictions by traversing from the root of the tree down to a leaf node, where a final classification or numerical value is assigned.

1. Feature Selection

At each step of tree construction, the algorithm selects the most **informative feature** to split the data. This selection is made based on specific mathematical criteria, depending on whether the task is **classification or regression**:

- **Classification:**
    - **Entropy & Information Gain:** Measures how much information is gained by splitting the dataset based on a feature. The feature with the **highest information gain** is selected for the split.

    - **Gini Impurity:** Evaluates the **impurity** of a dataset. A lower Gini value indicates a more homogeneous subset.
- **Regression:**
    - **Mean Squared Error (MSE):** Determines the best split by minimizing the average squared difference between actual and predicted values.

2. Recursive Splitting

Once a feature is chosen, the dataset is **divided into subsets**, and the process is **repeated recursively**. Each subset undergoes the same feature selection and splitting process, progressively refining the groups. This continues until one of the following stopping conditions is met:

- The **maximum depth** of the tree is reached.
- A **minimum number of samples** per node is specified, preventing further splits.
- The subset becomes **pure** (i.e., all samples belong to the same class).

Recursive splitting is what enables decision trees to **model complex patterns** in data, making them effective for both classification and regression tasks.

3. Leaf Nodes

When the recursive splitting process terminates, **leaf nodes** represent the final **classification or predicted value**. Each leaf contains data points that are similar in nature, ensuring accurate predictions.

- For classification, the leaf node holds a **majority class label**.
- For regression, the leaf node contains the **average target value** of the subset.

Key Concepts in Decision Trees

1. Entropy and Information Gain

- **Entropy** is a measure of the **impurity or randomness** in a dataset. A dataset with completely mixed labels has **high entropy**, while a dataset with pure labels has **low entropy**.
- **Information Gain (IG)** quantifies the reduction in entropy achieved by splitting on a feature. A **higher IG** means a feature is more useful for classification.

Mathematically, **Information Gain** is calculated as:

$$IG = \text{Entropy before split} - \sum(\text{Weighted entropy of each subset})$$

A feature with **maximum IG** is chosen for splitting at each node.

2. Gini Impurity

Gini impurity is another measure used for evaluating splits. It quantifies how **frequently** a randomly chosen sample would be **misclassified** if randomly labeled according to the class distribution.

- **A Gini value of 0** represents a **pure** dataset (all samples belong to the same class).
- **A Gini value of 1** indicates **maximum impurity** (completely random labels).

The formula for **Gini Impurity** is:

$$Gini = 1 - \sum(p_i^2)$$

3. Pruning – Controlling Overfitting

One major drawback of decision trees is their tendency to **overfit** if grown too deep. Overfitting happens when the model becomes too **specific** to the training data, leading to poor generalization on new data. **Pruning** helps control overfitting by reducing the size of the tree.

There are two main pruning techniques:

1. **Pre-Pruning (Early Stopping)** – Stops the tree from growing once certain conditions are met, such as:
    - Minimum number of samples per node (e.g., at least 5 samples in a node before splitting).
    - Maximum tree depth (e.g., stopping growth after 10 levels).
2. **Post-Pruning (Prune After Training)** – The tree is fully grown first, and then unnecessary branches are removed based on their contribution to accuracy. This is done by evaluating **validation set performance** and eliminating weak branches.

Pruning ensures that the decision tree remains **generalizable** rather than memorizing the training data.

4. Decision Trees for Regression

While commonly used for classification, decision trees can also be applied to **regression problems**. Instead of predicting **class labels**, they predict **continuous values**. The tree splits are based on minimizing **Mean Squared Error (MSE)**, ensuring that each leaf node represents a value close to actual outcomes.

The regression decision tree algorithm follows these steps:

1. Select the feature that minimizes **MSE**.
2. Perform **recursive binary splits** until a stopping condition is met.
3. The **average target value** in each leaf node is used as the final prediction.

Decision trees for regression work well for datasets with **non-linear relationships** but may struggle with **extrapolation** (predicting beyond observed data).

CODE & OUTPUT

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

```
# Import warnings library
import warnings

# Ignore all warnings to keep output clean
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('D:/ML/seattle-weather.csv')
```

```
df.shape
```

```
(1461, 6)
```

```
# Define column names for better readability
col_names = ['date', 'precipitation', 'temp_max', 'temp_min', 'wind', 'weather']

# Assign the new column names to the dataset
df.columns = col_names
```

```
col_names
```

```
['date', 'precipitation', 'temp_max', 'temp_min', 'wind', 'weather']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1461 entries, 0 to 1460
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           1461 non-null   object 
 1   precipitation  1461 non-null   float64
 2   temp_max       1461 non-null   float64
 3   temp_min       1461 non-null   float64
 4   wind           1461 non-null   float64
 5   weather        1461 non-null   object 
dtypes: float64(4), object(2)
memory usage: 68.6+ KB
```

```
# Display the count of unique values in each column
for col in col_names:
    print(f"Value counts for {col}:\n", df[col].value_counts(), "\n")
```

```
date
2015-12-31    1
2012-01-01    1
2012-01-02    1
2012-01-03    1
2012-01-04    1
             ..
2012-01-10    1
2012-01-09    1
2012-01-08    1
2012-01-07    1
2012-01-06    1
Name: count, Length: 1461, dtype: int64
precipitation
0.0     838
0.3      54
0.5      40
1.0      26
```

```python
# Display the count of unique values in the 'weather' column
df['weather'].value_counts()
```

```
weather
rain       641
sun        640
fog        101
drizzle     53
snow        26
Name: count, dtype: int64
```

```python
df.isnull().sum()
```

```
date             0
precipitation    0
temp_max         0
temp_min         0
wind             0
weather          0
dtype: int64
```

```python
# Define features (X) by dropping the target and date columns
X = df.drop(['weather', 'date'], axis=1)

# Define target variable (y) as the 'weather' column
y = df['weather']
```

```python
# Split the dataset into training and testing sets
# 33% of the data will be used for testing, and 67% for training
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```python
X_train.shape, X_test.shape

# check data types in X_train
X_train.dtypes
```

```
precipitation    float64
temp_max         float64
temp_min         float64
wind             float64
dtype: object
```

```python
# Import Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier

# Initialize the model with Gini impurity as the criterion and a max depth of 3
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)

# Train the model using the training data
clf_gini.fit(X_train, y_train)

# Make predictions on the test data
y_pred_gini = clf_gini.predict(X_test)
```

```python
# Import accuracy_score to evaluate the model
from sklearn.metrics import accuracy_score

# Print the model's accuracy score on the test data
print('Model accuracy score with criterion gini index: {0:0.4f}'.format(accuracy_score(y_test, y_pred_gini)))
```

```
Model accuracy score with criterion gini index: 0.8551
```

```python
# Make predictions on the training data
y_pred_train_gini = clf_gini.predict(X_train)

# Print the accuracy score for the training set
print('Training-set accuracy score: {0:0.4f}'.format(accuracy_score(y_train, y_pred_train_gini)))
```

```
Training-set accuracy score: 0.8548
```

```python
# Print the model's accuracy score on the training set
print('Training set score: {:.4f}'.format(clf_gini.score(X_train, y_train)))

# Print the model's accuracy score on the test set
print('Test set score: {:.4f}'.format(clf_gini.score(X_test, y_test)))
```

```
Training set score: 0.8548
Test set score: 0.8551
```

```python
# Import confusion_matrix to evaluate model performance
from sklearn.metrics import confusion_matrix

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_gini)
```

```
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[  0   0   0   0  15]
 [  0   0   0   0  34]
 [  0   0 195   1  15]
 [  0   0   5   3   0]
 [  0   0   0   0 215]]
```

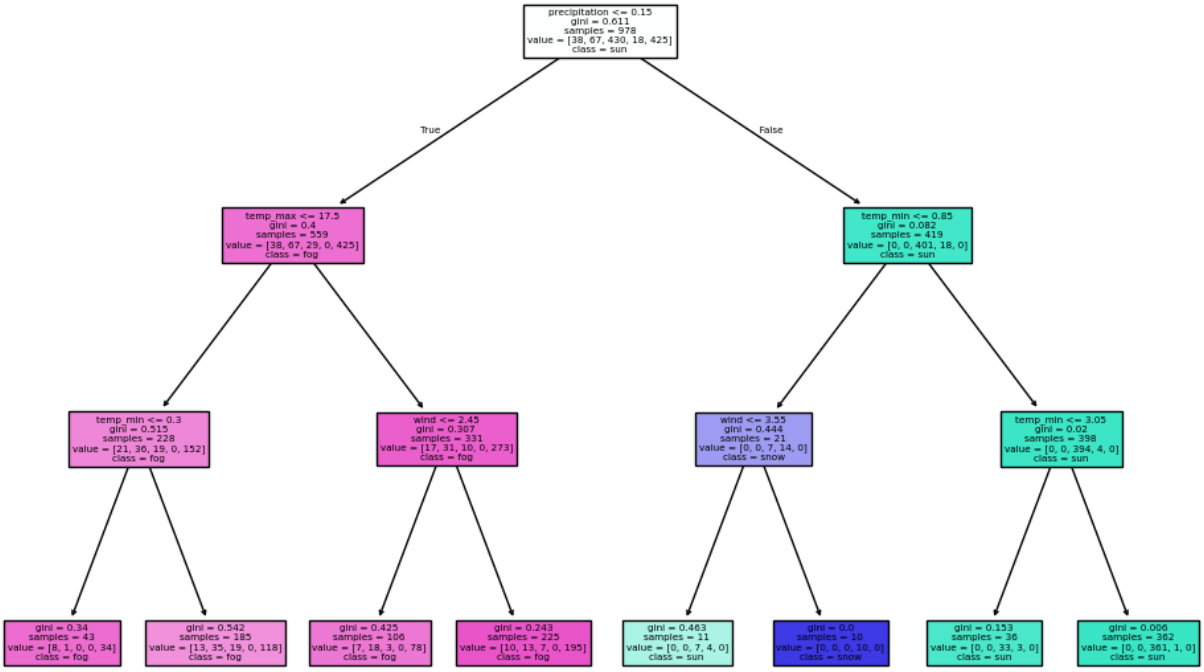
```
# Import classification_report to evaluate model performance
from sklearn.metrics import classification_report

# Print the classification report for precision, recall, and F1-score
print(classification_report(y_test, y_pred_gini))

# Import tree for visualization
from sklearn import tree

# Plot the decision tree
plt.figure(figsize=(12,8))
tree.plot_tree(clf_gini, feature_names=X.columns, class_names=y.unique(), filled=True)
plt.show()
```

	precision	recall	f1-score	support
drizzle	0.00	0.00	0.00	15
fog	0.00	0.00	0.00	34
rain	0.97	0.92	0.95	211
snow	0.75	0.38	0.50	8
sun	0.77	1.00	0.87	215
accuracy			0.86	483
macro avg	0.50	0.46	0.46	483
weighted avg	0.78	0.86	0.81	483



Github : https://github.com/Vishalgodalkar/Machine-Learning

Conclusion:

This **Decision Tree Classification implementation** involved training a model on the **Seattle Weather Dataset**, selecting appropriate parameters, and evaluating performance. The impact of **tree depth, splitting criteria (Gini Impurity vs. Entropy), and pruning techniques** was analyzed. While **deeper trees captured complex patterns**, they also led to **overfitting**, whereas **shallower trees improved generalization but reduced accuracy**. **Pruning** helped balance model complexity and performance. Additionally, **decision boundaries were more interpretable compared to other models** but lacked smoothness. Careful tuning of **max depth and pruning strategies** improved the model's reliability. Overall, the **Decision Tree Classifier** effectively categorized weather conditions, highlighting its advantages and limitations in classification tasks.