Assignment 07
DM LAB

Abhilasha Jolly
20223008

Objective 1: To design a tree-based machine learning algorithm and preprocess input dataset into a compatible format for the algorithm

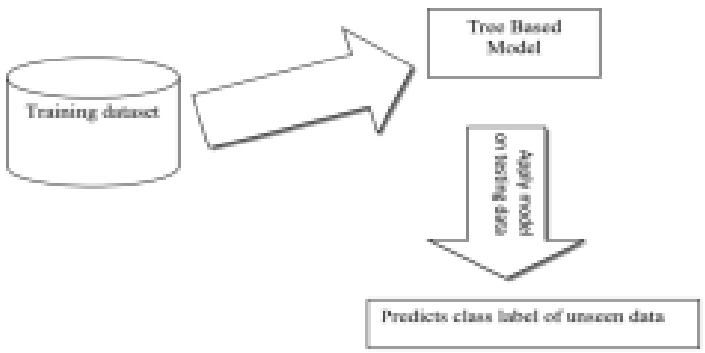
Outcome:
Students will able to learn various preprocessing techniques, design fit functions to train the model on the input dataset, and then apply the trained model for predicting unseen\unknown data into appropriate class label

Load vehicle0.dat file in Data Frame using pandas.read_csv ("vehicle.csv") and prepare a user defined decision tree module. Make suitable preprocessing in the data dataset if required.

1. Compute the split point for each attribute in the dataset using the following strategies: a. Information Gain
   b. Gini Indexs
   c. Gain Ratio
2. Design module for creating the decision tree and its representation in graphical format for the following cases:
   a. Binary Tree (each node split into exactly two branches).
   b. General Tree (each node may split into more than two branches depending on count nominal labels corresponding attributes).

3. Design module which predicts the class label of unknown and unseen data using tree traversal or any other techniques.

(General structure of standard machine learning-based model)



Double-click (or enter) to edit

39s

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn import preprocessing
from collections import Counter

# Load dataset
file_path = "/CAR DETAILS FROM CAR DEKHO.csv"  # Update this path accordingly
df = pd.read_csv(file_path)

# Data Preprocessing (Handling categorical data, missing values, etc.)
# Assuming the last column is the target variable
y = df.iloc[:, -1]
X = df.iloc[:, :-1]

# Convert categorical variables to numerical using Label Encoding
le = preprocessing.LabelEncoder()
y = le.fit_transform(y)

# Handling missing values (if any)
```


✓ 39s completed at 15:01

Commands + Code + Text

RAM Disk

```
# Convert categorical variables to numerical using Label Encoding
le = preprocessing.LabelEncoder()
y = le.fit_transform(y)

# Handling missing values (if any)
X = X.apply(lambda col: col.fillna(col.mode()[0]) if col.dtype == 'O' else col.fillna(col.mean()))

# Convert categorical features to numerical using Label Encoding
for col in X.select_dtypes(include=['object']).columns:
    X[col] = le.fit_transform(X[col])

# Splitting dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Function to compute Entropy
def entropy(y):
    counts = np.bincount(y)
    probabilities = counts / len(y)
    return -np.sum([p * np.log2(p) for p in probabilities if p > 0])

# Function to compute Gini Index
def gini_index(y):
    counts = np.bincount(y)
    probabilities = counts / len(y)
    return 1 - np.sum([p ** 2 for p in probabilities])
```


✓ 39s completed at 15:01

```
# Function to compute Gain Ratio
def gain_ratio(X_feature, y):
    total_entropy = entropy(y)
    values, counts = np.unique(X_feature, return_counts=True)
    weighted_entropy = sum((counts[i] / len(X_feature)) * entropy(y[X_feature == values[i]]) for i in range(len(valu
    info_gain = total_entropy - weighted_entropy
    split_info = -sum((counts[i] / len(X_feature)) * np.log2(counts[i] / len(X_feature)) for i in range(len(values))
    return info_gain / (split_info + 1e-9) if split_info != 0 else 0

# Compute split points using different strategies
for col in X_train.columns:
    print(f"Feature: {col}")
    print(f"  - Gain Ratio: {gain_ratio(X_train[col], y_train)}")
    print(f"  - Gini Index: {gini_index(y_train)}")
    print(f"  - Information Gain: {entropy(y_train) - entropy(X_train[col])}")

# Decision Tree using Gini Index (Binary Tree)
clf_gini = DecisionTreeClassifier(criterion='gini', random_state=42)
clf_gini.fit(X_train, y_train)

# Decision Tree using Entropy (Binary Tree)
clf_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_entropy.fit(X_train, y_train)

# Visualizing the Binary Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(clf_entropy, filled=True, feature_names=X_columns, class_names=[str(cls) for cls in clf_entropy.classes_])
```


✓ 39s completed at 15:01

Commands + Code + Text

RAM Disk

```
clf_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_entropy.fit(X_train, y_train)

# Visualizing the Binary Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(clf_entropy, filled=True, feature_names=X.columns, class_names=[str(cls) for cls in clf_entropy.classes_])

plt.title("Binary Decision Tree")
plt.show()

# General Tree (Multiway split based on unique categorical values)
class GeneralDecisionTree:
    def __init__(self):
        self.tree = {}

    def fit(self, X, y, feature):
        self.tree = self.build_tree(X, y, feature)

    def build_tree(self, X, y, feature):
        tree = {}
        unique_values = np.unique(X[feature])
        for value in unique_values:
            subset_y = y[X[feature] == value]
            most_common_label = Counter(subset_y).most_common(1)[0][0]
            tree[value] = most_common_label
        return tree
```


39s completed at 15:01

39s

```
    def predict(self, X):
        return X.apply(lambda row: self.tree.get(row, "Unknown"))

# Example of General Tree on first categorical feature
general_tree = GeneralDecisionTree()
categorical_feature = X_train.select_dtypes(include=['int64', 'float64']).columns[0]  # Selecting first numerical co
general_tree.fit(X_train, y_train, categorical_feature)
print("General Tree Structure:", general_tree.tree)

# Predicting using the General Tree
y_pred_general = general_tree.predict(X_test[categorical_feature])
print("Predictions using General Tree:", y_pred_general.tolist())

# Predict function for Binary Decision Tree
def predict_binary_tree(model, X):
    return model.predict(X)

# Making predictions on unseen data using the Binary Decision Tree
unseen_data = X_test.iloc[:5, :]  # Example unseen data
binary_predictions = predict_binary_tree(clf_entropy, unseen_data)
print("Binary Decision Tree Predictions:", binary_predictions)

# Making predictions using General Tree
general_predictions = general_tree.predict(X_test[categorical_feature][:5])
print("General Decision Tree Predictions:", general_predictions.tolist())
```


✓ 39s completed at 15:01

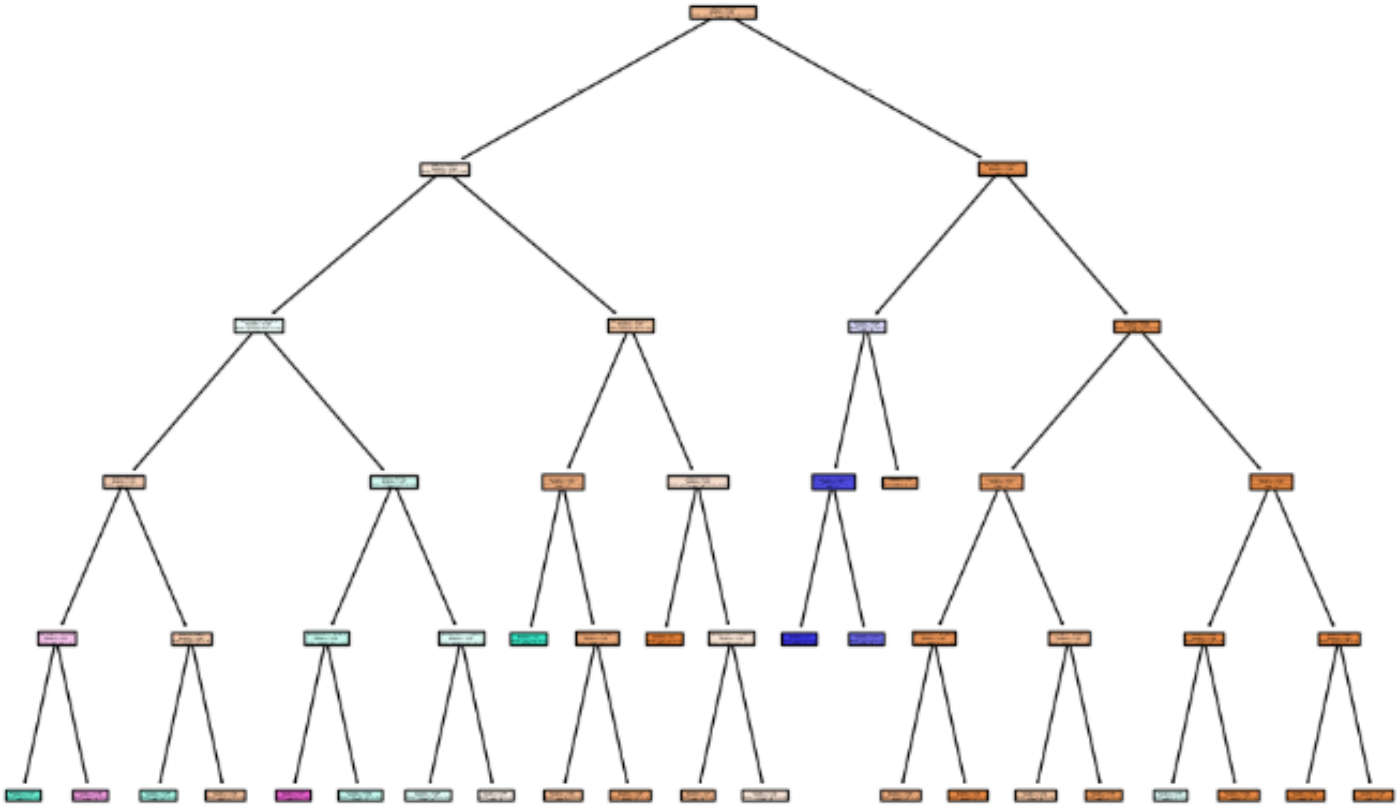
```
Feature: name
  - Gain Ratio: 0.07822694837075536
  - Gini Index: 0.5061743122066724
  - Information Gain: -8.385428871152778
Feature: year
  - Gain Ratio: 0.05929013446966879
  - Gini Index: 0.5061743122066724
  - Information Gain: -2.6501027028505946
Feature: selling_price
  - Gain Ratio: 0.046048005126676846
  - Gini Index: 0.5061743122066724
  - Information Gain: -6.024348224173581
Feature: km_driven
  - Gain Ratio: 0.06016825001278412
```


✓ 39s completed at 15:01

```
Feature: km_driven
  - Gain Ratio: 0.06016825001278412
  - Gini Index: 0.5061743122066724
  - Information Gain: -5.62018614442556
Feature: fuel
  - Gain Ratio: 0.004323368686332767
  - Gini Index: 0.5061743122066724
  - Information Gain: 0.20780283240842934
Feature: seller_type
  - Gain Ratio: 0.07276763031488259
  - Gini Index: 0.5061743122066724
  - Information Gain: 0.399905284808606
Feature: transmission
  - Gain Ratio: 0.013356167434804667
  - Gini Index: 0.5061743122066724
  - Information Gain: 0.8408520297497707
```

```
  warnings.warn(
```

Optimized Binary Decision Tree



```
Binary Decision Tree Predictions: [2 0 0 0 0]
```


✓ 2s completed at 15:23

```
Binary Decision Tree Predictions: [2 0 0 0 0]
General Decision Tree Structure: {np.int64(0): np.int64(2), np.int64(1): np.int64(1), n
General Tree Predictions: [0 0 0 0 2 2 0 0 0 0 0 0 0 2 0 0 0 2 0 0 0 0 0 0 2 0 0 0 0 0
 0 0 0 0 0 0 0 4 4 0 4 0 0 4 0 0 0 2 0 0 1 0 0 4 0 0 0 2 0 0 0 2 0 0 2 2 0
 2 0 0 0 0 0 0 0 0 0 0 2 0 0 0 4 0 0 0 2 0 2 0 0 2 0 0 0 0 0 2 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 4 0 0 4 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 2 2 0
 0 2 0 2 0 0 2 0 0 2 0 0 0 0 0 2 0 2 0 2 2 2 4 0 2 0 0 2 0 0 2 0 0 0 0 0 2
 2 0 0 0 0 0 0 0 0 2 4 4 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 2
 0 0 0 0 0 0 0 0 2 0 0 0 2 3 0 2 0 2 4 0 0 0 2 0 0 0 2 0 0 0 0 0 0 0 2 0 0
 0 0 0 0 0 2 4 0 2 0 2 0 2 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 2 0 2 2 0 0 2 0 0
 0 0 0 0 0 0 0 0 2 0 0 2 0 0 0 0 0 4 0 0 0 4 4 0 2 0 0 0 0 2 0 0 2 0 0 4 0
 0 0 2 0 0 0 0 0 4 0 2 0 0 0 2 4 0 0 4 2 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 2 0 0 0 0 0 0 2 2 0 2 0 2 0 2 0 0 0 0 0 0 0 0 0 0 2 0 0 0 2 0 0 0 2
 0 0 2 0 2 0 0 2 0 0 2 0 0 2 0 0 0 0 0 0 0 4 0 0 2 0 2 0 0 0 4 0 0 0 2 0 0 0
 0 0 0 0 0 0 4 0 2 0 2 0 0 0 0 4 0 0 0 2 2 0 0 0 0 0 0 2 4 0 0 2 0 0 0 0 2 0
 0 0 0 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 2 0 0 2 2 0 0 0 0 2 2 2 0 0 0 2 0 0 0 0
 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 2 0 2 2 4 0 4 2 2 0 2 0 0 2 0 0 0 0
 0 0 0 2 0 0 0 0 0 2 2 0 0 0 0 0 0 2 0 0 4 0 0 0 0 2 2 0 0 2 2 2 0 0 0 0 0
 0 0 0 0 2 0 0 2 0 0 0 0 2 0 0 0 0 0 0 2 0 2 2 0 0 4 0 0 0 0 0 0 0 2 0 0
 0 0 0 0 0 2 0 0 0 2 0 0 0 0 0 2 2 0 0 0 0 2 0 2 2 0 0 0 1 2 2 0 0 0 2 0 0
 0 0 0 0 0 0 0 0 2 0 0 2 0 0 0 0 0 0 0 2 0 0 2 0 0 0 4 0 0 0 2 0 0 0 0 0 2
 0 0 2 0 0 2 0 2 0 0 0 0 0 0 0 0 0 4 2 0 2 0 2 0 2 0 0 2 0 2 0 0 4 4 0 2 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 3 0 0 2 0 2 4 0 0 2 2 2 0 0 2 2 2 0
 0 2 2 2 2 0 0 0 0 2 0 2 2 1 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0 0 0
 0 0 0 0 0 0 0 0 2 0 0 0 2 0 0 0 0 0 0 0 2 4 0 0 0 0 4 0 1 0 0 0 0 0 0 0 2
 2 0 0 0 0 0 0 0 0 0 0 2 0 0 0 2 0 0]
```