Data Mining lab

Assignment 6

Name: Ayush Kumar

Regno: 20214284

Group: CSE-6B1

1. Compute the split point for each attribute in the dataset using the following strategies:

a. Information Gain

# b. Gini Indexes

c. Gain Ratio

import pandas as pd import numpy as np

from sklearn.tree import DecisionTreeClassifier # Load the dataset

df = pd.read\_csv("vehicle.csv")

# Preprocessing (if required) # Check for missing values print(df.isnull().sum())

# Handle missing values (if any)

# For simplicity, let's drop rows with missing values df.dropna(inplace=True)

# Separate features (X) and target variable (y)

X = df.drop(columns=['class']) # Assuming 'class' is the target variable y = df['class']

# Initialize the decision tree classifier clf = DecisionTreeClassifier()

# Fit the classifier to the data clf.fit(X, y)

# Function to compute Information Gain def information\_gain(y, y\_splits):

entropy\_parent = entropy(y) total\_instances = len(y)

weighted\_entropy\_children = sum((len(y\_split) / total\_instances) \* entropy(y\_split) for y\_split in y\_splits)

return entropy\_parent - weighted\_entropy\_children # Function to compute Gini Index

def gini\_index(y, y\_splits): gini\_parent = gini\_impurity(y) total\_instances = len(y)

weighted\_gini\_children = sum((len(y\_split) / total\_instances) \* gini\_impurity(y\_split) for y\_split in y\_splits)

return gini\_parent - weighted\_gini\_children

# Function to compute entropy def entropy(y):

classes, counts = np.unique(y, return\_counts=True) probabilities = counts / len(y)

return -sum(p \* np.log2(p) for p in probabilities)

# Function to compute Gini impurity def gini\_impurity(y):

classes, counts = np.unique(y, return\_counts=True) probabilities = counts / len(y)

return 1 - sum(p\*\*2 for p in probabilities)

# Function to compute split points using Information Gain, Gini Index, and Gain Ratio

def compute\_split\_points(X, y): split\_points = {}

for col in X.columns:

# Assuming each attribute is numeric values = X[col].unique()

for value in values:

# Split the dataset based on the attribute value left\_indices = X[col] < value

right\_indices = ~left\_indices

y\_splits = [y[left\_indices], y[right\_indices]] # Compute metrics for split points

info\_gain = information\_gain(y, y\_splits) gini\_idx = gini\_index(y, y\_splits)

gain\_ratio = info\_gain / (entropy(X[col]) + 1e-10) # Add small value to avoid division by zero

split\_points[(col, value)] = {'Information Gain': info\_gain, 'Gini Index': gini\_idx, 'Gain Ratio': gain\_ratio}

return split\_points

# Compute split points for each attribute split\_points = compute\_split\_points(X, y)

# Print split points

for key, value in split\_points.items(): print(f"Split Point for {key}: {value}")

# Design module for creating the decision tree and its representation in graphical format for the following cases:

* 1. Binary Tree (each node split into exactly two branches).

# General Tree (each node may split into more than two branches depending on count nominal labels corresponding attributes).

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report from sklearn.tree import export\_graphviz

import graphviz

# Load the dataset

df = pd.read\_csv("vehicle.csv")

# Check for missing values before preprocessing print("Missing values before preprocessing:") print(df.isnull().sum())

# Drop rows with missing values df.dropna(inplace=True)

# Check for missing values after preprocessing print("\nMissing values after preprocessing:") print(df.isnull().sum())

# Split the dataset into features and target variable X = df.drop(columns=['class'])

y = df['class']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the decision tree classifier clf = DecisionTreeClassifier()

# Fit the classifier to the training data clf.fit(X\_train, y\_train)

# Predict the labels of the test set y\_pred = clf.predict(X\_test)

# Evaluate the model

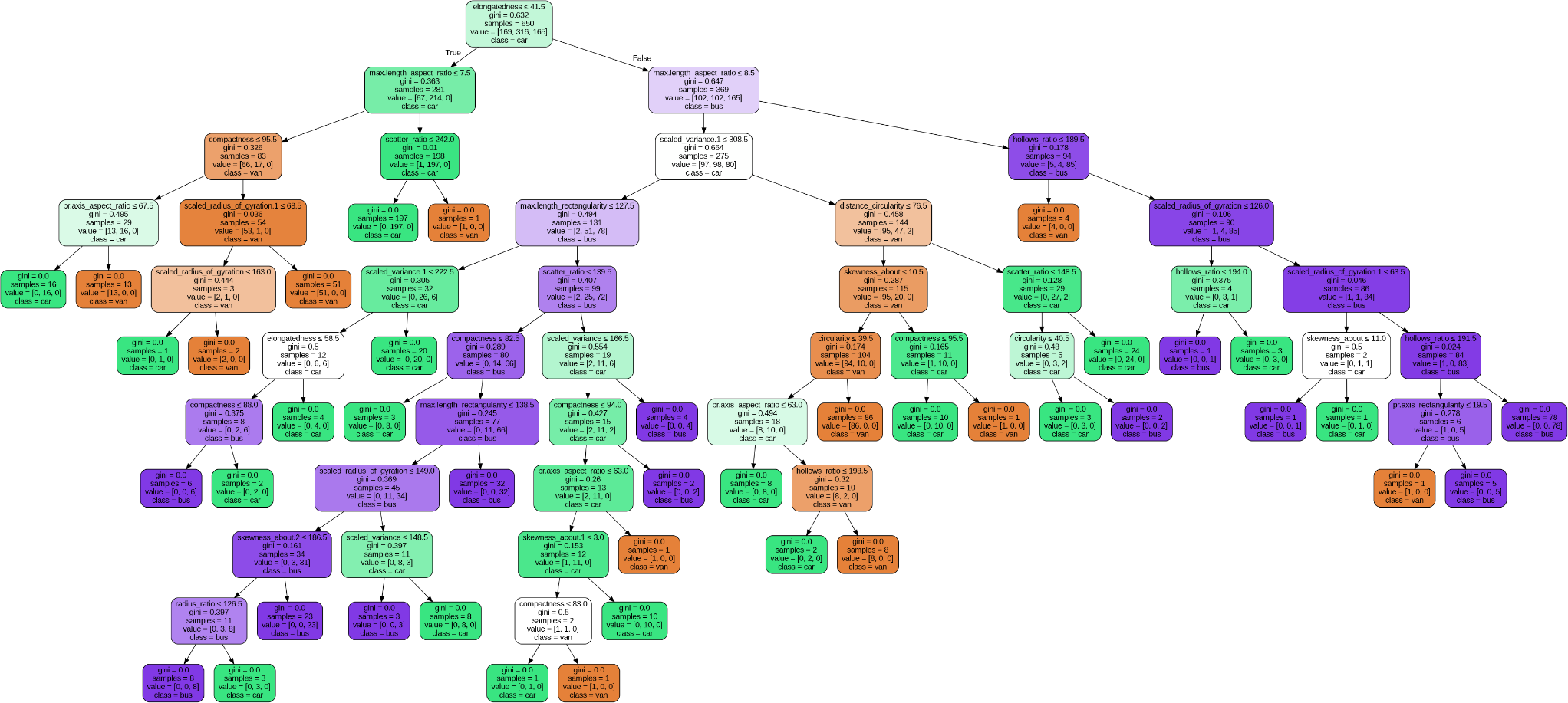
accuracy = accuracy\_score(y\_test, y\_pred) print("\nAccuracy:", accuracy) print("\nClassification Report:") print(classification\_report(y\_test, y\_pred))

# Visualize the decision tree

dot\_data = export\_graphviz(clf, out\_file=None,

feature\_names=X.columns, class\_names=y.unique(), filled=True, rounded=True, special\_characters=True)

graph = graphviz.Source(dot\_data) graph.render("vehicle\_decision\_tree", format='png', cleanup=True)



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

import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Load the dataset

df = pd.read\_csv('vehicle.csv')

# Drop rows with missing values df.dropna(inplace=True)

# Split dataset into features and target variable X = df.drop(columns=['class'])

y = df['class']

# Split data into training and testing subsets (80% training, 20% testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a decision tree classifier clf = DecisionTreeClassifier()

# Train the classifier on the training data clf.fit(X\_train, y\_train)

# Predict the classes of testing data y\_pred = clf.predict(X\_test)

# Evaluate accuracy

accuracy = accuracy\_score(y\_test, y\_pred) print("Accuracy:", accuracy)