LAB ASSESSMENT – 06

Objective:

1. Learn how to calculate Information Gain (ID3) and Gini Impurity (CART) for attribute selection.

2. Construct decision trees manually and compare the ID3 and CART algorithms.

3. Implement ID3 and CART from scratch in Python (without using built-in classifiers).

Dataset:

Consider the following dataset about customers purchasing a laptop based on different attributes:

Exercise

1. Compute Entropy and Information Gain (ID3)

· Calculate the Entropy of the dataset.

· Compute Information Gain for each attribute.

· Identify the best attribute for splitting based on Information Gain.

· Manually construct the first few levels of the ID3 Decision Tree.

2. Compute Gini Impurity (CART)

· Compute the Gini Impurity for each attribute.

· Identify the best attribute for splitting based on the lowest Gini Index.

· Manually construct the first few levels of the CART Decision Tree.

3. Implement ID3 and CART in Python

· Write a Python function to compute Entropy, Information Gain, and select the best attribute for splitting (ID3).

· Write a Python function to compute Gini Impurity and select the best attribute for splitting (CART).

· Construct a Decision Tree from scratch without using DecisionTreeClassifier.

4. Compare ID3 vs. CART

1. Which attribute was chosen first in ID3? Why?

2. Which attribute was chosen first in CART? Why?

3. How do the resulting decision trees differ?

4. Which method is better when dealing with continuous variables

CODE:

import numpy as np

import pandas as pd

import math

file\_path = 'C:/Users/2022503035/Documents/machine\_learning\_2022503035/decisiontree/dataset.xlsx'

df = pd.read\_excel(file\_path)

print(df)

target\_data = df['CLASS:BUYS\_COMPUTER']  
  
def entropy (x) :

class\_count = {}

for val in x:

if val in class\_count:

class\_count[val]+=1

else:

class\_count[val] = 1

total\_instances = len(x)

entropy = 0

for count in class\_count.values():

prob = count/total\_instances

entropy -= prob \* math.log2(prob)

return entropy

dataentropy = entropy (target\_data)

print(f"Entropy of the dataset is :{dataentropy:.4f}")

def info\_gain(df, attribute, target\_column):

total\_entropy = entropy(df[target\_column])

values = df[attribute].unique()

# this helps find the unique values of a given attribute apparently

weight\_entropy = 0

# this is for the sum of weighted entropies of subsets

total\_instances = len(df)

# total number of rows in the dataset

for value in values:

subset = df[df[attribute] == value]

subset\_entropy = entropy(subset[target\_column])

weight = len(subset) / total\_instances

weight\_entropy += weight\* subset\_entropy

return total\_entropy - weight\_entropy

target\_column = 'CLASS:BUYS\_COMPUTER'

attributes = df.columns[:-1] # Exclude target column

info\_gain\_values = {}

for attribute in attributes:

info\_gain\_values[attribute] = info\_gain(df, attribute, target\_column)

# Print Information Gain for each attribute

print("\nInformation Gain for each attribute:")

for attr, ig in info\_gain\_values.items():

print(f"{attr}: {ig:.4f}")

# Find the best attribute for splitting

best\_attribute = max(info\_gain\_values, key=info\_gain\_values.get)

print(f"\nBest attribute for splitting: {best\_attribute}")

def id3(df, target\_column, attributes):

if len(df[target\_column].unique()) == 1:

return df[target\_column].unique()[0]

if len(attributes) == 0:

return df[target\_column].mode()[0]

info\_gain\_values = {attr: info\_gain(df, attr, target\_column) for attr in attributes}

best\_attribute = max(info\_gain\_values, key=info\_gain\_values.get)

tree = {best\_attribute: {}}

remaining\_attributes = [attr for attr in attributes if attr != best\_attribute]

for value in df[best\_attribute].unique():

subset = df[df[best\_attribute] == value]

tree[best\_attribute][value] = id3(subset, target\_column, remaining\_attributes)

return tree

def print\_tree(tree, indent=""):

if not isinstance(tree, dict):

print(indent + "→ " + str(tree))

return

for key, value in tree.items():

print(indent + key)

for sub\_key, sub\_value in value.items():

print(indent + f" ├── {sub\_key}")

print\_tree(sub\_value, indent + " │ ")

target\_column = 'CLASS:BUYS\_COMPUTER'

attributes = [col for col in df.columns if col not in ['RID', target\_column]]

decision\_tree = id3(df, target\_column, attributes)

print\_tree(decision\_tree)

def gini\_split(df, attribute, target\_column):

attribute\_values = df[attribute].value\_counts()

total\_instances = len(df)

weighted\_gini = 0

for value in attribute\_values.keys():

subset = df[df[attribute] == value]

subset\_class\_counts = subset[target\_column].value\_counts()

gini\_A = gini\_impurity(subset\_class\_counts)

weighted\_gini += (len(subset) / total\_instances) \* gini\_A

return weighted\_gini

# this is to compute ggini impurity for each attribute

gini\_attribute = {}

for attr in df.columns:

if attr != 'CLASS:BUYS\_COMPUTER':

gini\_attribute[attr] = gini\_split(df, attr, 'CLASS:BUYS\_COMPUTER')

print(f'Gini for {attr} is {gini\_attribute[attr]:.3f}')

def cart(df, target\_column, attributes):

if len(df[target\_column].unique()) == 1:

return df[target\_column].unique()[0]

if len(attributes) == 0:

return df[target\_column].mode()[0]

gini\_values = {attr: gini\_split(df, attr, target\_column) for attr in attributes}

# the best binary split is calculated based on the most min value

best\_attribute = min(gini\_values, key=gini\_values.get)

# creatingg a subtree

tree = {best\_attribute: {}}

remaining\_attributes = [attr for attr in attributes if attr != best\_attribute]

for value in df[best\_attribute].unique():

subset = df[df[best\_attribute] == value]

tree[best\_attribute][value] = cart(subset, target\_column, remaining\_attributes)

return tree

def print\_tree(tree, indent=""):

if not isinstance(tree, dict):

print(indent + "→ " + str(tree))

return

for key, value in tree.items():

print(indent + key)

for sub\_key, sub\_value in value.items():

print(indent + f" ├── {sub\_key}")

print\_tree(sub\_value, indent + " │ ")

target\_column = 'CLASS:BUYS\_COMPUTER'

attributes = [col for col in df.columns if col not in ['RID', target\_column]]

cart\_tree = cart(df, target\_column, attributes)

print("\nCART Decision Tree")

print\_tree(cart\_tree)

selected\_attribute = min(gini\_attribute, key=gini\_attribute.get)

print(f'The selected attribute is: {selected\_attribute}')