import pandas as pd

import math

# function to calculate the entropy of entire dataset

# -----------------------------------------------------------------------

def base\_entropy(dataset):

p = 0

n = 0

target = dataset.iloc[:, -1]

targets = list(set(target))

for i in target:

if i == targets[0]:

p = p + 1

else:

n = n + 1

if p == 0 or n == 0:

return 0

elif p == n:

return 1

else:

entropy = 0 - (

((p / (p + n)) \* (math.log2(p / (p + n))) + (n / (p + n)) \* (math.log2(n/ (p + n)))))

return entropy

# -----------------------------------------------------------------------

# function to calculate the entropy of attributes

# -----------------------------------------------------------------------

def entropy(dataset, feature, attribute):

p = 0

n = 0

target = dataset.iloc[:, -1]

targets = list(set(target))

for i, j in zip(feature, target):

if i == attribute and j == targets[0]:

p = p + 1

elif i == attribute and j == targets[1]:

n = n + 1

if p == 0 or n == 0:

return 0

elif p == n:

return 1

else:

entropy = 0 - (

((p / (p + n)) \* (math.log2(p / (p + n))) + (n / (p + n)) \* (math.log2(n/ (p + n)))))

return entropy

# -----------------------------------------------------------------------

# a utility function for checking purity and impurity of a child

# -----------------------------------------------------------------------

def counter(target, attribute, i):

p = 0

n = 0

targets = list(set(target))

for j, k in zip(target, attribute):

if j == targets[0] and k == i:

p = p + 1

elif j == targets[1] and k == i:

n = n + 1

return p, n

# -----------------------------------------------------------------------

# function that calculates the information gain

# -----------------------------------------------------------------------

def Information\_Gain(dataset, feature):

Distinct = list(set(feature))

Info\_Gain = 0

for i in Distinct:

Info\_Gain = Info\_Gain + feature.count(i) / len(feature) \* entropy(dataset,feature, i)

Info\_Gain = base\_entropy(dataset) - Info\_Gain

return Info\_Gain

# -----------------------------------------------------------------------

# function that generates the childs of selected Attribute

# -----------------------------------------------------------------------

def generate\_childs(dataset, attribute\_index):

distinct = list(dataset.iloc[:, attribute\_index])

childs = dict()

for i in distinct:

childs[i] = counter(dataset.iloc[:, -1], dataset.iloc[:, attribute\_index], i)

return childs

# -----------------------------------------------------------------------

# function that modifies the dataset according to the impure childs

# -----------------------------------------------------------------------

def modify\_data\_set(dataset,index, feature, impurity):

size = len(dataset)

subdata = dataset[dataset[feature] == impurity]

del (subdata[subdata.columns[index]])

return subdata

# -----------------------------------------------------------------------

# function that return attribute with the greatest Information Gain

# -----------------------------------------------------------------------

def greatest\_information\_gain(dataset):

max = -1

attribute\_index = 0

size = len(dataset.columns) - 1

for i in range(0, size):

feature = list(dataset.iloc[:, i])

i\_g = Information\_Gain(dataset, feature)

if max < i\_g:

max = i\_g

attribute\_index = i

return attribute\_index

# -----------------------------------------------------------------------

# function to construct the decision tree

# -----------------------------------------------------------------------

def construct\_tree(dataset, tree):

target = dataset.iloc[:, -1]

impure\_childs = []

attribute\_index = greatest\_information\_gain(dataset)

childs = generate\_childs(dataset, attribute\_index)

tree[dataset.columns[attribute\_index]] = childs

targets = list(set(dataset.iloc[:, -1]))

for k, v in childs.items():

if v[0] == 0:

tree[k] = targets[1]

elif v[1] == 0:

tree[k] = targets[0]

elif v[0] != 0 or v[1] != 0:

impure\_childs.append(k)

for i in impure\_childs:

sub = modify\_data\_set(dataset,attribute\_index,

dataset.columns[attribute\_index], i)

tree = construct\_tree(sub, tree)

return tree

# ---------------------------------------------------------------------------

# main function

# -----------------------------------------------------------------------

def main():

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

tree = dict()

result = construct\_tree(df, tree)

for key, value in result.items():

print(key, " => ", value)

# -----------------------------------------------------------------------

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**

outlook => {'sunny': (3, 2), 'overcast': (0, 4), 'rainy': (2, 3)}

overcast => yes

temp => {'mild': (1, 2), 'cool': (1, 1)}

hot => no

cool => yes

humidity => {'normal': (1, 1)}

high => no

normal => yes

windy => {'Weak': (0, 1), 'Strong': (1, 0)}

Weak => yes

Strong => no