import pandas as pd

df = pd.read\_csv('home/soetcse/Downloads/tennisdata.csv')

print("\n Input Data Set is:\n", df)

t = df.keys()[-1]

print('Target Attribute is: ', t)

# Get the attribute names from input dataset

attribute\_names = list(df.keys())

#Remove the target attribute from the attribute names list

attribute\_names.remove(t)

print('Predicting Attributes: ', attribute\_names)

#Function to calculate the entropy of collection S

import math

def entropy(probs):

return sum( [-prob\*math.log(prob, 2) for prob in probs])

#Function to calulate the entropy of the given Data Sets/List with

#respect to target attributes

def entropy\_of\_list(ls,value):

from collections import Counter

cnt = Counter(x for x in ls)# Counter calculates the propotion of class

print('Target attribute class count(Yes/No)=',dict(cnt))

total\_instances = len(ls)

print("Total no of instances/records associated with {0} is: {1}".format(value,total\_instances ))

probs = [x / total\_instances for x in cnt.values()] # x means no of YES/NO

print("Probability of Class {0} is: {1:.4f}".format(min(cnt),min(probs)))

print("Probability of Class {0} is: {1:.4f}".format(max(cnt),max(probs)))

return entropy(probs) # Call Entropy

def information\_gain(df, split\_attribute, target\_attribute,battr):

print("\n\n-----Information Gain Calculation of ",split\_attribute, " --------")

df\_split = df.groupby(split\_attribute) # group the data based on attribute values

glist=[]

for gname,group in df\_split:

print('Grouped Attribute Values \n',group)

glist.append(gname)

glist.reverse()

nobs = len(df.index) \* 1.0

df\_agg1=df\_split.agg({target\_attribute:lambda x:entropy\_of\_list(x, glist.pop())})

df\_agg2=df\_split.agg({target\_attribute :lambda x:len(x)/nobs})

df\_agg1.columns=['Entropy']

df\_agg2.columns=['Proportion']

# Calculate Information Gain:

new\_entropy = sum( df\_agg1['Entropy'] \* df\_agg2['Proportion'])

if battr !='S':

old\_entropy = entropy\_of\_list(df[target\_attribute],'S-'+df.iloc[0][df.columns.get\_loc(battr)])

else:

old\_entropy = entropy\_of\_list(df[target\_attribute],battr)

return old\_entropy - new\_entropy

def id3(df, target\_attribute, attribute\_names, default\_class=None,default\_attr='S'):

from collections import Counter

cnt = Counter(x for x in df[target\_attribute])# class of YES /NO

## First check: Is this split of the dataset homogeneous?

if len(cnt) == 1:

return next(iter(cnt)) # next input data set, or raises StopIteration when EOF is hit.

## Second check: Is this split of the dataset empty? if yes, return a default value

elif df.empty or (not attribute\_names):

return default\_class # Return None for Empty Data Set

## Otherwise: This dataset is ready to be devied up!

else:

# Get Default Value for next recursive call of this function:

default\_class = max(cnt.keys()) #No of YES and NO Class

# Compute the Information Gain of the attributes:

gainz=[]

for attr in attribute\_names:

ig= information\_gain(df, attr, target\_attribute,default\_attr)

gainz.append(ig)

print('Information gain of ',attr,' is : ',ig)

index\_of\_max = gainz.index(max(gainz)) # Index of Best Attribute

best\_attr = attribute\_names[index\_of\_max] # Choose Best Attribute to split on

print("\nAttribute with the maximum gain is: ", best\_attr)

# Create an empty tree, to be populated in a moment

tree = {best\_attr:{}} # Initiate the tree with best attribute as a node

remaining\_attribute\_names =[i for i in attribute\_names if i != best\_attr]

# Split dataset-On each split, recursively call this algorithm.Populate the empty tree with subtrees, which

# are the result of the recursive call

for attr\_val, data\_subset in df.groupby(best\_attr):

subtree = id3(data\_subset,target\_attribute, remaining\_attribute\_names,default\_class,best\_attr)

tree[best\_attr][attr\_val] = subtree

return tree

from pprint import pprint

tree = id3(df,t,attribute\_names)

print("\nThe Resultant Decision Tree is:")

pprint(tree)