**Practical no -5**

import math

class DataSet:

def \_\_init\_\_(self,ds):

self.dataset = ds

def print(self):

print(self.dataset)

def uniqueAns(self) : # calculates how many unique answers are there

ans\_set = set(self.dataset["Ans"])

return len(ans\_set)

def getMaxOccur(self): # returns a value with maximum occurances of answer

d = {self.dataset["Ans"].count(x):x for x in self.dataset["Ans"]}

keys = list(d.keys())

keys.sort(reverse=True)

if len(keys) >1 and keys[0] != keys[1]:

return d[keys[0]]

else:

return None

def copy(self): # creates a copy of dataset

newdataset = dict()

keys = list(self.dataset.keys())

newdataset = dict.fromkeys(keys,0)

for key in keys :

newdataset[key] = self.dataset[key].copy()

return DataSet(newdataset)

def maxInfoGain(self): #finds a feature with max info gain

features = self.dataset["Features"]

if len(features) >=0 :

maxinfogain = self.infoGain(features[0])

maxfeat = features[0]

i=0

while i<len(features):

ig = self.infoGain(features[i])

if ig>=maxinfogain :

maxinfogain = ig

maxfeat = features[i]

i = i + 1

return maxfeat

**Roll no 17,19**

else:

return None

def infoGain(self, feat) : # finds info gain of a particular feature

if feat in self.dataset.keys() :

featlist1 = self.dataset[feat]

total = len(featlist1)

#print(featlist1)

featset = set(featlist1)

#print(featset)

d = dict.fromkeys(featset, 0)

#print(d)

for item in featlist1 :

d[item] = d[item]+1

#print("d = ", d)

branches = self.splitOnFeature(feat)

#print( " info gain :" )

gain = 0

for key in branches :

dsobj = branches[key]

#print(feat , " having value : ", key )

#print(feat , " having value : ", key , " has dataset : " )

#dsobj.print()

#print(feat , " having value : ", key , " has entropy : " , dsobj.getEntropy())

gain = gain + ( (d[key] / total) \*( dsobj.getEntropy()) )

#print("gain = ", gain)

#print("entropy = ", self.getEntropy() )

#print("info gain = ", self.getEntropy() - gain)

return self.getEntropy() - gain

else :

print("feature does not exists")

return None

def getEntropy(self) : # calculates entropy

list1 = self.dataset["Ans"]

total = len(list1)

#print(list1)

aset = set(list1)

#print(aset)

d = dict.fromkeys(aset, 0)

#print(d)

for item in list1 :

d[item] = d[item]+1

#print(d)

ent = 0

for k in aset :

x = d[k]/total

ent = ent + (x \* math.log(x,2))

return -ent

# this function splits original dataset on a feature and creates dictionary whose key values are

#values of the feature with a value as new dataset with that feature removed.

def splitOnFeature(self, feat):

if feat in self.dataset["Features"] :

#print(feat, " exists as a feature")

ans\_set = set(self.dataset[feat])

#print("unique answers for " , feat ," in data set :", ans\_set)

newfeatures = self.dataset["Features"].copy()

newfeatures.remove(feat)

#print("newfeatures :" , newfeatures)

#create empty replica of dataset without the feature

keys = list(self.dataset.keys())

newdataset = dict()

for akey in keys :

newdataset[akey] = list()

newdataset["Features"] = newfeatures.copy()

newdataset.pop(feat)

#print("new empty data set :", newdataset)

branches = dict()

for akey in ans\_set:

branches[akey] = dict()

for key in list(newdataset.keys()) :

branches[akey][key] = newdataset[key].copy()

#print("new empty Branches :" , branches)

#copy data from original dataset to new datasets

i=-1

for featval in self.dataset[feat]:

i=i+1

#print("i=", i)

if featval in list(branches.keys()) :

branches[featval]["Ans"].append(self.dataset["Ans"][i])

for nfeat in newfeatures :

branches[featval][nfeat].append(self.dataset[nfeat][i])

#print("branches : ", branches)

for key in branches :

#print(key , ":", branches[key])

branches[key] = DataSet(branches[key])

return branches

else:

print(feat , " feature is not available")

return None

# main function that calculates the answer

def calculateAns(dsobj, feature, maxoccur, descr ):

#print("Feature :", feature, ", Feature Val :", dsobj.dataset[feature], ", Ans :", dsobj.dataset['Ans'] )

branches = dsobj.splitOnFeature(feature)

for key in list(branches.keys()):

newdsobj = branches[key]

#print("Splitting Feature :", descr+"-" + feature, ", value :", key)

#newdsobj.print()

for key in list(branches.keys()):

newdsobj = branches[key]

#input("Continue-->")

#print("For ans :", key, ", ans in dataset :", newdsobj.dataset['Ans'])

if (newdsobj.uniqueAns() == 1):

print("Answer for " , descr+"-" +feature , " with value =", key , " is :", newdsobj.dataset['Ans'][0])

elif(newdsobj.uniqueAns() == 0):

print("in zero")

print("Answer for " , descr+"-" +feature , " with value =", key , " is :", maxoccur)

elif(newdsobj.uniqueAns() >1 and len(newdsobj.dataset["Features"]) ==0 ) :

print("Answer for " , descr+"-" +feature , " with value =", key , " is :", maxoccur)

else:

newfeat = newdsobj.maxInfoGain()

#print("Recursive call")

newmaxoccur = newdsobj.getMaxOccur()

if(newmaxoccur == None) :

newmaxoccur = maxoccur

calculateAns(newdsobj, newfeat, newmaxoccur , descr + ":" + feature +":->" + key +" " )

dataset = {

"Ans" :["Wait", "Wait", "Leave", "Wait", "Wait", "Wait", "Leave", "Leave", "Wait", "Leave"],

"Features":["Reservation", "Raining", "BadService"],

"Reservation" : ["T", "T", "T", "F", "T", "T", "T", "T", "T", "F"],

"Raining" : ["T", "F", "T", "T", "T","T", "F", "T", "T", "F"],

"BadService":["F", "F", "T", "F", "F","F", "T", "T", "F", "F"]

}

d1 = DataSet(dataset)

if d1.uniqueAns() != 1 :

feat = d1.maxInfoGain()

calculateAns(d1, feat, d1.getMaxOccur(), "")

**Output:**

**Answer for -BadService with value = T is : Leave**

**Answer for :BadService:->F -Raining with value = T is : Wait**

**Answer for :BadService:->F :Raining:->F -Reservation with value = T is : Wait**

**Answer for :BadService:->F :Raining:->F -Reservation with value = F is : Leave**