-----------------------------------------------------1-----------------------------------------------------

import csv

a=[]

with open('enjoysport.csv')as trainData:

for row in csv.reader(trainData):

a.append(row)

print(row)

n=len(a[0])-1

S=['0']\*n

print("Initial hypothesis ",S)

print("FIND S ALGORITHM")

S=a[0][:-1]

for i in range(len(a)):

if a[i][n]=="yes":

for j in range(n):

if a[i][j]!=S[j]:

S[j]='?'

print("\nTraining example no {0}, Hypothesis is".format(i+1),S)

print("\nMaximally specific hypothesis is ", S )

-----------------------------------------------------2-----------------------------------------------------

import csv

a = []

with open('enjoySport.csv', 'r') as trainData:

for row in csv.reader(trainData):

a.append(row)

print(row)

n=len(a[0])-1

print("\n The initial value of hypothesis: ")

s = ['0'] \* n

g = ['?'] \* n

print ("\n The most specific hypothesis S0 :",s)

print (" \n The most general hypothesis G0 :",g)

s=a[0][:-1]

temp=[]

print("\n Candidate Elimination algorithm\n")

for i in range(len(a)):

if a[i][n]=="yes": #Use Positive for manufacture.csv

for j in range(n):

if a[i][j]!=s[j]:

s[j]='?'

for j in range(n):

for k in range(len(temp)): #Use len(temp)-1 for manufacture.csv

if temp[k][j]!='?' and temp[k][j]!=s[j]:

del temp[k]

if a[i][n]=="no": #Use Negative for manufacture.csv

for j in range(n):

if s[j]!=a[i][j] and s[j]!='?':

g[j]=s[j]

if g not in temp:

temp.append(g)

g= ['?']\*n

print("\n For Training Example No :{0} the hypothesis is S{0} ".format(i+1),s)

if (len(temp)==0):

print(" For Training Example No :{0} the hypothesis is G{0} ".format(i+1),g)

else:

print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)

-----------------------------------------------------3-----------------------------------------------------

import pandas as pd

from math import log

from pprint import pprint

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

data = df.values.tolist()

attr\_names = df.columns.values.tolist()

print(df)

def entropy(pos, neg):

if pos == 0 or neg == 0:

return 0

tot = pos + neg

return -pos / tot \* log(pos / tot, 2) - neg / tot \* log(neg / tot, 2)

def gain(data, attr, pos, neg):

d, E, acu = {}, entropy(pos, neg), 0

for i in data:

if i[attr] not in d:

d[i[attr]] = {}

d[i[attr]][i[-1]] = 1 + d[i[attr]].get(i[-1], 0)

for i in d:

tot = d[i].get('Yes', 0) + d[i].get('No', 0)

acu += tot / (pos + neg) \* entropy(d[i].get('Yes', 0), d[i].get('No', 0))

return E - acu

def build(data, attr\_names):

pos, sz = len([x for x in data if x[-1] == 'Yes']), len(data[0]) - 1

neg = len(data) - pos

if neg == 0 or pos == 0:

return 'Yes' if neg == 0 else 'No'

root = max([[gain(data, i, pos, neg), i] for i in range(sz)])[1]

fin, res = {}, {}

uniq\_attr = set([x[root] for x in data])

for i in uniq\_attr:

res[i] = build([x[:root] + x[root + 1:] for x in data if x[root] == i], attr\_names[:root] + attr\_names[root+1:])

fin[attr\_names[root]] = res

return fin

tree = build(data, attr\_names)

pprint(tree)

-----------------------------------------------------4-----------------------------------------------------

import numpy as np

inputNeurons=10

hiddenlayerNeurons=5

outputNeurons=2

input = np.random.randint(1,100,inputNeurons)

output = np.array([1.0,0.0])

hidden\_layer=np.random.rand(1,hiddenlayerNeurons)

print(input)

hidden\_biass=np.random.rand(1,hiddenlayerNeurons)

output\_bias=np.random.rand(1,outputNeurons)

hidden\_weights=np.random.rand(inputNeurons,hiddenlayerNeurons)

output\_weights=np.random.rand(hiddenlayerNeurons,outputNeurons)

def sigmoid (layer):

return 1/(1 + np.exp(-layer))

def gradient(layer):

return layer\*(1-layer)

for i in range(1000):

hidden\_layer=np.dot(input,hidden\_weights)

hidden\_layer=sigmoid(hidden\_layer+hidden\_biass)

output\_layer=np.dot(hidden\_layer,output\_weights)

output\_layer=sigmoid(output\_layer+output\_bias)

error = (output-output\_layer)

gradient\_outputLayer=gradient(output\_layer)

error\_terms\_output=gradient\_outputLayer \* error

error\_terms\_hidden=gradient(hidden\_layer)\*np.dot(error\_terms\_output,output\_weights.T)

gradient\_hidden\_weights = np.dot(input.reshape(inputNeurons,1),error\_terms\_hidden.reshape(1,hiddenlayerNeurons))

gradient\_ouput\_weights = np.dot(hidden\_layer.reshape(hiddenlayerNeurons,1),error\_terms\_output.reshape(1,outputNeurons))

hidden\_weights = hidden\_weights + 0.05\*gradient\_hidden\_weights

output\_weights = output\_weights + 0.05\*gradient\_ouput\_weights

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Iteration:",i,"::::",error)

print("Output::::",output\_layer)

-----------------------------------------------------5-----------------------------------------------------

from pprint import pprint

import pandas as pd

df\_golf = pd.read\_csv("golf.csv")

print(df\_golf)

attribute\_names = list(df\_golf.columns)

print("List of Attributes:", attribute\_names)

attribute\_names.remove('label')

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

table=dict()

priorProb=dict()

train=df\_golf.sample(frac=0.6,random\_state=100) #random state is a seed value

test=df\_golf.drop(train.index)

print("---------------------------------")

print(train)

print("---------------------------------")

print(test)

print("---------------------------------")

for attr\_val, data\_subset in train.groupby("label"):

from collections import Counter

valueCount = dict()

count=0

for attr\_value in attribute\_names:

cnt = Counter(x for x in data\_subset[attr\_value])

count=sum(cnt.values())

valueCount[attr\_value]=dict(cnt)

print("value count", valueCount.values())

print("counter:-",cnt)

table[attr\_val]=valueCount

priorProb[attr\_val]=count

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\n\nThe Resultant table is :\n")

pprint(table)

pprint(priorProb)

totalSize=test['label'].count()

correctPridictions=0

for k, row in test.iterrows():

rowTuple=dict(row)

print("print row tuple")

pprint(rowTuple)

postioriList=list()

labelList=list()

for label in table.keys():

posteriori = 1.0

print("RowTuple",rowTuple.keys())

print("RowValues",rowTuple.values())

for key in [x for x in rowTuple.keys() if x != 'label']:

print(key, "label:",label)

attributeValue=rowTuple.get(key)

if attributeValue in table[label][key].keys():

countList=table[label][key].values()

attributeCount=table[label][key][attributeValue]

posteriori=1.0\*attributeCount/sum(countList)\*posteriori

posteriori=posteriori\*priorProb[label]

labelList.append(label)

postioriList.append(posteriori)

print(labelList)

print(postioriList)

maxProbInd = postioriList.index(max(postioriList))

print(rowTuple['label'], "::::", labelList[maxProbInd])

if rowTuple['label'] == labelList[maxProbInd]:

print(rowTuple['label'],"::::",labelList[maxProbInd])

correctPridictions=correctPridictions+1

print("POSTERIORI OF:",label,"is:",posteriori)

print("Number of Correct Predictions : Number of Samples",correctPridictions,":",totalSize)

print("Accuracy:",100.0\*correctPridictions/totalSize)

-----------------------------------------------------6-----------------------------------------------------

import pandas as pd

df\_imdb = pd.read\_csv("imdb\_labelled.txt",sep='\t',index\_col=None)

print(df\_imdb.keys())

print(df\_imdb)

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_imdb['Text'], df\_imdb['Label'],train\_size=0.8,test\_size=0.2,random\_state=100)

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer()

X\_train\_cv = cv.fit\_transform(X\_train)

X\_test\_cv = cv.transform(X\_test)

from sklearn.naive\_bayes import MultinomialNB

naive\_bayes = MultinomialNB()

naive\_bayes.fit(X\_train\_cv, y\_train)

predictions = naive\_bayes.predict(X\_test\_cv)

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score

print('Accuracy score: ', accuracy\_score(y\_test, predictions))

print('Precision score: ', precision\_score(y\_test, predictions))

print('Recall score: ', recall\_score(y\_test, predictions))

-----------------------------------------------------7-----------------------------------------------------

names = "A,B,C,D,E,F,G,H,I,J,K,L,M,RESULT"

names = names.split(",")

print(names)

import pandas as pd

data = pd.read\_csv("datasetheart.csv",names = names)

print(data.head(5))

print(data.tail(5))

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator

model = BayesianModel([("A","B"),("B","C"),("C","D"),("D","RESULT")])

model.fit(data,estimator=MaximumLikelihoodEstimator)

from pgmpy.inference import VariableElimination

infer = VariableElimination(model)

q = infer.query(variables=['RESULT'],evidence={"B":1})

print(q['RESULT'])

-----------------------------------------------------8-----------------------------------------------------

from sklearn import datasets,metrics

iris = datasets.load\_iris()

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

X\_train,X\_test,y\_train,y\_test = train\_test\_split(iris.data,iris.target)

from sklearn.cluster import KMeans

model1 =KMeans(n\_clusters=3)

model1.fit(X\_train,y\_train)

print("K-means: ",metrics.accuracy\_score(y\_test,model1.predict(X\_test)))

from sklearn.mixture import GaussianMixture

model2 = GaussianMixture(n\_components=3)

model2.fit(X\_train,y\_train)

print("EM: ",metrics.accuracy\_score(y\_test,model2.predict(X\_test)))

-----------------------------------------------------9-----------------------------------------------------

from sklearn import datasets

iris = datasets.load\_iris()

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data,iris.target,train\_size=0.8,test\_size=0.2,random\_state=100)

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n\_neighbors=3)

model.fit(X\_train,y\_train)

predicted= model.predict(X\_test)

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, predicted))

print(classification\_report(y\_test, predicted))

-----------------------------------------------------10-----------------------------------------------------

import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 1000)

y = np.log(np.abs((x \*\* 2) - 1) + 0.5)

x = x + np.random.normal(scale=0.05, size=1000)

plt.scatter(x, y, alpha=0.3)

def local\_regression(x0, x, y, tau):

x0 = np.r\_[1, x0]

x = np.c\_[np.ones(len(x)), x]

xw = x.T \* radial\_kernel(x0, x, tau)

beta = np.linalg.pinv(xw @ x) @ xw @ y

return x0 @ beta

def radial\_kernel(x0, x, tau):

return np.exp(np.sum((x - x0) \*\* 2, axis=1) / (-2 \* tau \*\* 2))

def plot\_lr(tau):

domain = np.linspace(-5, 5, num=300)

pred = [local\_regression(x0, x, y, tau) for x0 in domain]

plt.scatter(x, y, alpha=0.3)

plt.plot(domain, pred, color="blue")

return plt

print(plot\_lr(3).show())

-----------------------------------------------------------------------------------------------------------------------------------