ML Lab Report

Name : Pooja Srinivasan

USN:1BM18CS069

Class & Sec : 6 ‘B’

Batch : B2

Prg 1: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

```
import numpy as np
import pandas as pd
```

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
data = pd.read_csv("/content/drive/MyDrive/finddata.csv")
print(data,"\n")
```

```
d = np.array(data)[:,:-1]
print("\n The attributes are: ",d)
target = np.array(data)[:,-1]
print("\n The target is: ",target)
```

```
 The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]

 The target is:  ['Yes' 'No' 'Yes' 'Yes']
```

```
def findS(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

print("\n The final hypothesis is:",findS(d,target))
```

Dataset :

	Time	Weather	Temperature	Company	Humidity	Wind	Goes
0	Morning	Sunny	Warm	Yes	Mild	Strong	Yes
1	Evening	Rainy	Cold	No	Mild	Normal	No
2	Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
3	Evening	Sunny	Cold	Yes	High	Strong	Yes

Output :

```
The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

Prg 2: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np
import pandas as pd

#to read the data in the csv file
data = pd.DataFrame(data=pd.read_csv('/content/drive/MyDrive/enjoysport.csv
'))
print(data,"\n")

#making an array of all the attributes
concepts = np.array(data.iloc[:,0:-1])
print("The attributes are: ",concepts)

#segragating the target that has positive and negative examples
target = np.array(data.iloc[:,-1])
print("\n The target is: ",target)

#training function to implement candidate_elimination algorithm
def learn(concepts, target):
 specific_h = concepts[0].copy()
 print("\n Initialization of specific_h and general_h")
 print(specific_h)
 general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
 print(general_h)
 for i, h in enumerate(concepts):
     if target[i] == "yes":
         for x in range(len(specific_h)):
             if h[x]!= specific_h[x]:
                 specific_h[x] ='?'
                 general_h[x][x] ='?'
             print(specific_h)
```

```
        print(specific_h)
        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("\n Steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
    indices = [i for i, val in enumerate(general_h) if val ==
['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)

#obtaining the final hypothesis
print("\nFinal Specific_h:", s_final, sep="\n")
print("\nFinal General_h:", g_final, sep="\n")
```

Dataset :

	sky	temp	humidity	wind	water	forcast	enjoysport
0	sunny	warm	normal	strong	warm	same	yes
1	sunny	warm	high	strong	warm	same	yes
2	rainy	cold	high	strong	warm	change	no
3	sunny	warm	high	strong	cool	change	yes

Output:

```
The attributes are:  [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

 The target is:  ['yes' 'yes' 'no' 'yes']

 Initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
```

```
 Steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']

 Steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']

 Steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' '?' 'same']
['sunny' 'warm' '?' 'strong' '?' '?']
['sunny' 'warm' '?' 'strong' '?' '?']

 Steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Prg3: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
import
math
        import csv
        def load_csv(filename):
            lines=csv.reader(open(filename,"r"))
            dataset = list(lines)
            headers = dataset.pop(0)
            return dataset,headers

        class Node:
            def __init__(self,attribute):
                self.attribute=attribute
                self.children=[]
                self.answer=""

        def subtables(data,col,delete):
            dic={}
            coldata=[row[col] for row in data]
            attr=list(set(coldata))

            counts=[0]*len(attr)
```

```
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic

def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums

def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy
```

```
def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]


    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node

def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)


def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)

'''Main program'''
dataset,features=load_csv("id3.csv")
node1=build_tree(dataset,features)
```

```
print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3_test.csv")

for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:")
    classify(node1,xtest,features)
```

Output:

```
bmsce@bmsce-Precision-T1700:~/Documents/LAB - 3 - DECISION TREE$ python ml3.py
The decision tree for the dataset using ID3 algorithm is
('', 'Outlook')
('   ', 'overcast')
('     ', 'yes')
('   ', 'sunny')
('     ', 'Humidity')
('       ', 'high')
('         ', 'no')
('       ', 'normal')
('         ', 'yes')
('   ', 'rain')
('     ', 'Wind')
('       ', 'strong')
('         ', 'no')
('       ', 'weak')
('         ', 'yes')
('The test instance:', ['rain', 'cool', 'normal', 'strong'])
The label for test instance:
no
('The test instance:', ['sunny', 'mild', 'normal', 'strong'])
The label for test instance:
yes
```

Prog 4: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

from google.colab import drive
drive.mount("/content/drive")


df = pd.read_csv("/content/drive/MyDrive/pima_indian.csv")
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness
', 'insulin', 'bmi', 'diab_pred', 'age']
predicted_class_names = ['diabetes']

X = df[feature_col_names].values
y = df[predicted_class_names].values
```

```
print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)

print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)

clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))

print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))

print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```

Dataset:

```
<bound method NDFrame.head of      num_preg  glucose_conc  diastolic_bp  ...  diab_pred  age  diabetes
0           6           148            72  ...      0.627   50         1
1           1            85            66  ...      0.351   31         0
2           8           183            64  ...      0.672   32         1
3           1            89            66  ...      0.167   21         0
4           0           137            40  ...      2.288   33         1
..        ...           ...           ...  ...        ...  ...       ...
763        10           101            76  ...      0.171   63         0
764         2           122            70  ...      0.340   27         0
765         5           121            72  ...      0.245   30         0
766         1           126            60  ...      0.349   47         1
767         1            93            70  ...      0.315   23         0

[768 rows x 9 columns]>

 the total number of Training Data : (514, 1)

 the total number of Test Data : (254, 1)
```

Output:

```
 Confusion matrix
[[132  29]
 [ 45  48]]

 Accuracy of the classifier is 0.7086614173228346

 The value of Precision 0.6233766233766234

 The value of Recall 0.5161290322580645
Predicted Value for individual Test Data: [1]
```

Prg 5a: Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

Note:Program with built-in functions

```
from google.colab import drive
drive.mount('//content/drive')
```

```
!pip install pgmpy
```

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
```

```
  import pandas.util.testing as tm
#read Cleveland Heart Disease data
heartDisease = pd.read_csv('/content/drive/MyDrive/heart.csv')
heartDisease = heartDisease.replace('?',np.nan)
```

```
#display the data
print('Sample instances from the dataset are given below')
print(heartDisease.head())
```

```
#display the Attributes names and datatyes
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
#Creat Model-Bayesian Network
model =  BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exa
ng','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('hear
tdisease','chol')])


#Learning CPDs using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
```

```
 Learning CPD using Maximum likelihood estimators
#Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
```

```
 Inferencing with Bayesian Network:
#computing the Probability of HeartDisease given restecg
print('\n 1.Probability of HeartDisease given evidence= restecg :1')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'reste
cg':1})
print(q1)
```

```
#computing the Probability of HeartDisease given cp
print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
```

```
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2
})
print(q2)
```

Dataset :

```
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  ...  oldpeak  slope  ca  thal  heartdisease
0   63    1   1       145   233  ...      2.3      3   0     6             0
1   67    1   4       160   286  ...      1.5      2   3     3             2
2   67    1   4       120   229  ...      2.6      2   2     7             1
3   37    1   3       130   250  ...      3.5      3   0     3             0
4   41    0   2       130   204  ...      1.4      1   0     3             0

[5 rows x 14 columns]
```

```
   Attributes and datatypes
age               int64
sex               int64
cp                int64
trestbps          int64
chol              int64
fbs               int64
restecg           int64
thalach           int64
exang             int64
oldpeak         float64
slope             int64
ca               object
thal             object
heartdisease      int64
dtype: object
```

Output:

```
 2.Probability of HeartDisease given evidence= cp:2
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Prg 5b: Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

Note:Program without built-in functions

```
from google.colab import drive
drive.mount('//content/drive')
```

```
!pip install bayespy
```

```
import pandas as pd
```

```
import bayespy as bp
import numpy as np
import csv
!pip3 install colorama
!pip3 install colorama
from colorama import init
from colorama import Fore, Back, Style
init()

# Define Parameter Enum values
# Age
ageEnum = {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1,
           'MiddleAged': 2, 'Youth': 3, 'Teen': 4}
# Gender
genderEnum = {'Male': 0, 'Female': 1}
# FamilyHistory
familyHistoryEnum = {'Yes': 0, 'No': 1}
# Diet(Calorie Intake)
dietEnum = {'High': 0, 'Medium': 1, 'Low': 2}
# LifeStyle
lifeStyleEnum = {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}
# Cholesterol
cholesterolEnum = {'High': 0, 'BorderLine': 1, 'Normal': 2}
# HeartDisease
heartDiseaseEnum = {'Yes': 0, 'No': 1}

data = pd.read_csv("/content/drive/MyDrive/heart_disease_data.csv")

data =np.array(data, dtype='int8')
N = len(data)

# Input data column assignment
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:, 0])

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:, 1])

p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:, 2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
diet = bp.nodes.Categorical(p_diet, plates=(N,))
diet.observe(data[:, 3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
```

```
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:, 4])

p_cholesterol = bp.nodes.Dirichlet(1.0*np.ones(3))
cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))
cholesterol.observe(data[:, 5])
# Prepare nodes and establish edges
# np.ones(2) -> HeartDisease has 2 options Yes/No
# plates(5, 2, 2, 3, 4, 3) -> corresponds to options present for domain val
ues
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.MultiMixture(
    [age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Ca
tegorical, p_heartdisease)
heartdisease.observe(data[:, 6])
p_heartdisease.update()

# Interactive Test
m = 0
while m == 0:
    print("\n")
    res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))),
int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter FamilyHist
ory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum: ' + str(
        dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))),
int(input('Enter Cholesterol: ' + str(cholesterolEnum)))], bp.nodes.Categor
ical, p_heartdisease).get_moments()[0][heartDiseaseEnum['Yes']]
    print("Probability(HeartDisease) = " + str(res))

# print(Style.RESET_ALL)
    m = int(input("Enter for Continue:0, Exit :1 "))
```

Output:

```
Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}0
Enter Gender: {'Male': 0, 'Female': 1}1
Enter FamilyHistory: {'Yes': 0, 'No': 1}1
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}2
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}2
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 0


Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}4
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}1
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}1
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 1
```

Prg 5c :Program for the illustration of Baysian Belief networks using 5nodes using Lung Cancer data.(The conditional probabilities are given)

```
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
!pip install pgmpy
```

```
cancer_model=BayesianModel([('Pollution','Cancer'),('Smoker','Cancer'),
('Cancer','Xray'),('Cancer','Dyspnoea')])
print('Bayesian network models are :')
print('\t',cancer_model.nodes())
print('Bayesian edges are:')
print('\t',cancer_model.edges())
```

```
Bayesian network models are :
        ['Pollution', 'Cancer', 'Smoker', 'Xray', 'Dyspnoea']
Bayesian edges are:
        [('Pollution', 'Cancer'), ('Cancer', 'Xray'), ('Cancer', 'Dyspnoea'), ('Smoker', 'Cancer')]
```

```
cpd_poll = TabularCPD(variable='Pollution', variable_card=2,
                      values=[[0.9], [0.1]])
cpd_smoke = TabularCPD(variable='Smoker', variable_card=2,
                       values=[[0.3], [0.7]])
cpd_cancer = TabularCPD(variable='Cancer', variable_card=2,
                        values=[[0.03, 0.05, 0.001, 0.02],
                                [0.97, 0.95, 0.999, 0.98]],
                        evidence=['Smoker', 'Pollution'],
                        evidence_card=[2, 2])
cpd_xray = TabularCPD(variable='Xray', variable_card=2,
                      values=[[0.9, 0.2], [0.1, 0.8]],
                      evidence=['Cancer'], evidence_card=[2])
cpd_dysp = TabularCPD(variable='Dyspnoea', variable_card=2,
                      values=[[0.65, 0.3], [0.35, 0.7]],
                      evidence=['Cancer'], evidence_card=[2])
```

```
# Associating the parameters with the model structure.
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dy
sp)

# Checking if the cpds are valid for the model.
cancer_model.check_model()
```

Output: True

```
cancer_infer=VariableElimination(cancer_model)
```

```
print('All local independecies are as follows')
cancer_model.get_independencies()
print('Displaying CPDs')
print(cancer_model.get_cpds('Pollution'))
print(cancer_model.get_cpds('Smoker'))
```

```
print(cancer_model.get_cpds('Cancer'))
print(cancer_model.get_cpds('Xray'))
print(cancer_model.get_cpds('Dyspnoea'))
```

```
All local independecies are as follows
Displaying CPDs
+--------------+-----+
| Pollution(0) | 0.9 |
+--------------+-----+
| Pollution(1) | 0.1 |
+--------------+-----+
+-----------+-----+
| Smoker(0) | 0.3 |
+-----------+-----+
| Smoker(1) | 0.7 |
+-----------+-----+
+-----------+--------------+--------------+--------------+--------------+
| Smoker    | Smoker(0)    | Smoker(0)    | Smoker(1)    | Smoker(1)    |
+-----------+--------------+--------------+--------------+--------------+
| Pollution | Pollution(0) | Pollution(1) | Pollution(0) | Pollution(1) |
+-----------+--------------+--------------+--------------+--------------+
| Cancer(0) | 0.03         | 0.05         | 0.001        | 0.02         |
+-----------+--------------+--------------+--------------+--------------+
| Cancer(1) | 0.97         | 0.95         | 0.999        | 0.98         |
+-----------+--------------+--------------+--------------+--------------+
+---------+-----------+-----------+
| Cancer  | Cancer(0) | Cancer(1) |
+---------+-----------+-----------+
| Xray(0) | 0.9       | 0.2       |
+---------+-----------+-----------+
| Xray(1) | 0.1       | 0.8       |
+---------+-----------+-----------+
+-------------+-----------+-----------+
| Cancer      | Cancer(0) | Cancer(1) |
+-------------+-----------+-----------+
| Dyspnoea(0) | 0.65      | 0.3       |
+-------------+-----------+-----------+
| Dyspnoea(1) | 0.35      | 0.7       |
+-------------+-----------+-----------+
```

```
print('\n Probablity of Cancer given smoker')
q=cancer_infer.query(variables=['Cancer'],evidence={'Smoker':1})
print(q)
```

```
 Probablity of Cancer given smoker
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0029 |
+-----------+---------------+
| Cancer(1) |        0.9971 |
+-----------+---------------+
```

```
print('\n Probablity of Cancer given smoker')
q=cancer_infer.query(variables=['Cancer'],evidence={'Smoker':1,'Polluti
on':1})
print(q)
```

Probablity of Cancer given smoker

Cancer	phi(Cancer)
Cancer(0)	0.0200
Cancer(1)	0.9800