ML LAB REPORT

1BM18CS061

NIKHIL A S

Week 1:
i)Find s algorithm

```
# This Python 3 environment comes with many helpful analytics libraries ins
talled
# It is defined by the kaggle/python Docker image: https://github.com/kaggl
e/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will
list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be sa
ved outside of the current session
```

```
/kaggle/input/datasetcsv/data.csv
```

```
data = pd.read_csv("/kaggle/input/datasetcsv/data.csv")
print("The entered data is \n")
print(data,"\n")
d = np.array(data)[:,:-1]
print("\n The attributes are: \n",d)
target = np.array(data)[:,-1]
print("\n The target is: ",target)
def training(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)):In
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass
    return specific_hypothesis
print("\n The final hypothesis is:",training(d,target))
```

Output:

```
The entered data is

  Weather Temperature Humidity Goes
0  Sunny        Warm     Mild  Yes
1   Rainy        Cold     Mild   No
2  Sunny     Moderate    Nomal  Yes
3  Sunny         Cold    High   Yes


 The attributes are:
 [['Sunny ' 'Warm ' 'Mild']
 ['Rainy' 'Cold' 'Mild']
 ['Sunny ' 'Moderate' 'Nomal']
 ['Sunny ' 'Cold' 'High ']]

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

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

Week 2:

ii) Candidate elimination algorithm:

```
# This Python 3 environment comes with many helpful analytics libraries ins
talled
# It is defined by the kaggle/python Docker image: https://github.com/kaggl
e/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will
list all files under the input directory

import os
```

```
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be sa
ved outside of the current session
/kaggle/input/candidatecsv/candidate.csv

data = pd.read_csv("/kaggle/input/candidatecsv/candidate.csv")
print("Entered data is")
print(data)
concepts = np.array(data)[:,:-1]
print("\n The attributes are: \n",d)
target = np.array(data)[:,-1]
print("\n The target is: ",target)

Entered data is
     sky airtemp humidity     wind water forecast 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

 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']

#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")
```

Output:

```
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' '?' '?' 'warm' 'same']
['sunny' 'warm' '?' '?' 'warm' 'same']
['sunny' 'warm' '?' '?' 'warm' 'same']
['sunny' 'warm' '?' '?' 'warm' 'same']


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

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

```
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '
?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
['sunny' 'warm' '?' '?' 'warm' 'same']
['sunny' 'warm' '?' '?' 'warm' 'same']
['sunny' 'warm' '?' '?' 'warm' 'same']
['sunny' 'warm' '?' '?' 'warm' 'same']
['sunny' 'warm' '?' '?' '?' 'same']
['sunny' 'warm' '?' '?' '?' '?']
['sunny' 'warm' '?' '?' '?' '?']

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

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

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

Week 3:
ID3 algorithm:

```
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
```

Week 4:

iv)Naïve bayes classifier:

```
import pandas as pd

data = pd.read_csv('PlayTennis.csv')
data.head()
```

```
y =
```

```
list(data['PlayTennis'].values)
```

```
X = data.iloc[:,1:].values
```

```
print(f'Target Values: {y}')
```

```
print(f'Features: \n{X}')
```

```
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes',
'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
```

```
y_train = y[:8]
y_val = y[8:]

X_train = X[:8]
X_val = X[8:]

print(f"Number of instances in training set: {len(X_train)}")
print(f"Number of instances in testing set: {len(X_val)}")
```

```
Number of instances in training set: 8
Number of instances in testing set: 6
```

```
class NaiveBayesClassifier:


    def __init__(self, X, y):

        self.X, self.y = X, y

        self.N = len(self.X)
```

```
        self.dim = len(self.X[0])

        self.attrs = [[] for _ in range(self.dim)]

        self.output_dom = {}

        self.data = []

        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])

            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1

            else:
                self.output_dom[self.y[i]] += 1

            self.data.append([self.X[i], self.y[i]])
        def classify(self, entry):

        solve = None
        max_arg = -1

        for y in self.output_dom.keys():

            prob = self.output_dom[y]/self.N

            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] an
d x[1] == y]
                n = len(cases)
                prob *= n/self.N

            if prob > max_arg:
                max_arg = prob
                solve = y

        return solve

nbc = NaiveBayesClassifier(X_train, y_train)

total_cases = len(y_val)

good = 0
bad = 0
predictions = []

for i in  range(total_cases):
    predict = nbc.classify(X_val[i])
    predictions.append(predict)

    if y_val[i] == predict:
        good += 1
    else:
        bad += 1
```

```
print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', good)
print('Number of wrong predictions:', bad)
print()
print('Accuracy of Bayes Classifier:', good/total_cases)
```

Output:

```
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```

Week 5:

v) Bayesian network:

```
# This Python 3 environment comes with many helpful analytics libraries installed # It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python # For example, here's several helpful packages to load import numpy as np

# linear algebra

import pandas as pd

import pgmpy as pgmpy from pgmpy.estimators

import MaximumLikelihoodEstimator from pgmpy.models

import BayesianModel from pgmpy.inference

import VariableElimination

import os for dirname, _, filenames in os.walk('/kaggle/input'):

for filename in filenames: print(os.path.join(dirname, filename)) # You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All" # You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
#read Cleveland Heart Disease data
heartDisease = pd.read_csv("/kaggle/input/bayesiannetwork/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'),('
exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg')
,('heartdisease','chol')])
#Learning CPDs using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
# Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
#computing the Probability of HeartDisease given restecg
print('\n 1.Probability of HeartDisease given evidence= restecg :1')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'r
estecg':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={'c
p':2})
print(q2)
```

Output:

```
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak
slope  \
0   63    1   1       145   233    1        2      150      0      2.3
3
1   67    1   4       160   286    0        2      108      1      1.5
2
2   67    1   4       120   229    0        2      129      1      2.6
2
3   37    1   3       130   250    0        0      187      0      3.5
3
4   41    0   2       130   204    0        2      172      0      1.4
1

  ca thal  heartdisease
0  0    6             0
1  3    3             2
2  2    7             1
3  0    3             0
4  0    3             0

 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

 Learning CPD using Maximum likelihood estimators
Finding Elimination Order: :   0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: age:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: chol:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: cp:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: sex:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: exang: 100%|██████████| 5/5 [00:00<00:00, 189.65it/s]
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 132.81it/s]
Finding Elimination Order: :   0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: age:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: chol:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: restecg:0%|             | 0/5 [00:00<?, ?it/s]
Eliminating: sex:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: exang: 100%|██████████| 5/5 [00:00<00:00, 230.00it/s]
 Inferencing with Bayesian Network:
1.Probability of HeartDisease given evidence= restecg :1
+-----------------------------+-----------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=======================+
| heartdisease(0) |               0.1012 |
+-----------------------------+-----------------------+
| heartdisease(1) |               0.0000 |
+-----------------------------+-----------------------+
| heartdisease(2) |               0.2392 |
+-----------------------------+-----------------------+
| heartdisease(3) |               0.2015 |
+-----------------------------+-----------------------+
| heartdisease(4) |               0.4581 |
+-----------------------------+-----------------------+

 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
```

```
+----------------------+----------------------------+
```

WEEK 6:

vi) Inferring from Bayesian model:

```
from pgmpy.models import BayesianModel
```

```
from pgmpy.factors.discrete import TabularCPD
```

```
from pgmpy.inference import VariableElimination
```

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

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])
```

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

```
cancer_model.add_cpds(cpd_poll,cpd_smoke,cpd_cancer,cpd_xray,cpd_dysp)
print('Model generated by adding cpts(cpds)')
print('Checking correctness of model:',end='')
print(cancer_model.check_model())
```

```
Model generated by adding cpts(cpds)
Checking correctness of model:True
```

```
print('All local depencies are as follows')
```

```
cancer_model.get_independencies()
```

All local depencies are as follows

Out[10]:

```
(Pollution ⊥ Smoker)
(Pollution ⊥ Dyspnoea, Xray | Cancer)
(Pollution ⊥ Xray | Dyspnoea, Cancer)
(Pollution ⊥ Dyspnoea | Cancer, Xray)
(Pollution ⊥ Dyspnoea, Xray | Cancer, Smoker)
(Pollution ⊥ Xray | Dyspnoea, Cancer, Smoker)
(Pollution ⊥ Dyspnoea | Cancer, Xray, Smoker)
(Smoker ⊥ Pollution)
(Smoker ⊥ Dyspnoea, Xray | Cancer)
(Smoker ⊥  Xray | Dyspnoea, Cancer)
(Smoker ⊥ Dyspnoea, Xray | Pollution, Cancer)
(Smoker ⊥ Dyspnoea | Cancer, Xray)
(Smoker ⊥ Xray | Dyspnoea, Pollution, Cancer)
(Smoker ⊥ Dyspnoea | Pollution, Cancer, Xray)
(Xray ⊥ Dyspnoea, Pollution, Smoker | Cancer)
(Xray ⊥ Pollution, Smoker | Dyspnoea, Cancer)
(Xray ⊥ Dyspnoea, Smoker | Pollution, Cancer)
(Xray ⊥ Dyspnoea, Pollution | Cancer, Smoker)
(Xray ⊥ Smoker | Dyspnoea, Pollution, Cancer)
(Xray ⊥ Pollution | Dyspnoea, Cancer, Smoker)
(Xray ⊥ Dyspnoea | Pollution, Cancer, Smoker)
(Dyspnoea ⊥ Pollution, Xray, Smoker | Cancer)
(Dyspnoea ⊥ Xray, Smoker | Pollution, Cancer)
(Dyspnoea ⊥ Pollution, Smoker | Cancer, Xray)
(Dyspnoea ⊥ Pollution, Xray | Cancer, Smoker)
(Dyspnoea ⊥ Smoker | Pollution, Cancer, Xray)
(Dyspnoea ⊥ Xray | Pollution, Cancer, Smoker)
(Dyspnoea ⊥ Pollution | Cancer, Xray, Smoker)
```

```
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'))
```

```
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           |
+--------------------+---------------+---------------+
```

```
cancer_infer=VariableElimination(cancer_model)
print('\n Inferencing with bayesian network')
print("\n Probability of Cancer given smoker")
q=cancer_infer.query(variables=['Cancer'],evidence={'Smoker':1})
print(q)

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

Output:

```
Finding Elimination Order: :   0%|          | 0/3 [00:00<?, ?it/s]
  0%|          | 0/3 [00:00<?, ?it/s]
Eliminating: Dyspnoea:   0%|          | 0/3 [00:00<?, ?it/s]
Eliminating: Pollution:   0%|          | 0/3 [00:00<?, ?it/s]
Eliminating: Xray: 100%|██████████| 3/3 [00:00<00:00, 359.52it/s]

  0%|          | 0/2 [00:00<?, ?it/s]
Finding Elimination Order: :   0%|          | 0/2 [00:00<?, ?it/s]

  0%|          | 0/2 [00:00<?, ?it/s]
```

```
Eliminating: Dyspnoea:   0%|          | 0/2 [00:00<?, ?it/s]

Eliminating: Xray: 100%|██████████| 2/2 [00:00<00:00, 333.49it/s]A
 Inferencing with bayesian network

 Probability of Cancer given smoker
+--------------+--------------------+
| Cancer       |   phi(Cancer)      |
+==============+====================+
| Cancer(0)    |           0.0029   |
+--------------+--------------------+
| Cancer(1)    |           0.9971   |
+--------------+--------------------+
Probability of Cancer given smoker,pollution
+--------------+--------------------+
| Cancer       |   phi(Cancer)      |
+==============+====================+
| Cancer(0)    |           0.0200   |
+--------------+--------------------+
| Cancer(1)    |           0.9800   |
+--------------+--------------------+
```