Rashmi Dhaduti
1BM18CS080

1. Fins – S algorithm

```
import pandas as pd
import numpy as np
data = pd.read_csv('data.csv')
data
concepts = np.array(data)[:,:-1]
concepts
target = np.array(data)[:,-1]
target
def train(con, tar):
    for i, val in enumerate(tar):
        if val == 'yes':
            specific_h = con[i].copy()
            break

    for i, val in enumerate(con):
        if tar[i] == 'yes':
            for x in range(len(specific_h)):
                if val[x] != specific_h[x]:
                    specific_h[x] = '?'
                else:
                    pass
    return specific_h
print(train(concepts, target))
```

Output

```
In [1]: import pandas as pd
        import numpy as np

In [2]: data = pd.read_csv('data.csv')

In [3]: data
```

Out[3]:

	sky	air temp	humidity	wind	water	forecast	enjoy sport
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

```
In [4]: concepts = np.array(data)[:,:-1]

In [5]: concepts
Out[5]: array([['sunny', 'warm', 'normal', 'strong', 'warm', 'same'],
               ['sunny', 'warm', 'high', 'strong', 'warm', 'same'],
               ['rainy', 'cold', 'high', 'strong', 'warm', 'change'],
               ['sunny', 'warm', 'high', 'strong', 'cool', 'change']],
              dtype=object)

In [6]: target = np.array(data)[:,-1]

In [7]: target
Out[7]: array(['yes', 'yes', 'no', 'yes'], dtype=object)
```

Machine Learning Lab Report

Rashmi Dhaduti
1BM18CS080

In [8]:
```
def train(con, tar):
    for i, val in enumerate(tar):
        if val == 'yes':
            specific_h = con[i].copy()
            break

    for i, val in enumerate(con):
        if tar[i] == 'yes':
            for x in range(len(specific_h)):
                if val[x] != specific_h[x]:
                    specific_h[x] = '?'
                else:
                    pass
    return specific_h
```

In [9]:
```
print(train(concepts, target))
```

```
['sunny' 'warm' '?' 'strong' '?' '?']
```

In [ ]:
```

```

Rashmi Dhaduti
1BM18CS080

2. Candidate elimination algorithm

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("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(" 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)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Rashmi Dhaduti
1BM18CS080

Output

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
```

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
```

In [3]:
```
target = np.array(data.iloc[:,-1])
print(target)
```

```
['yes' 'yes' 'no' 'yes']
```

In [4]:
```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("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(" 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
```

In [5]:
```
s_final, g_final = learn(concepts, target)
```

```
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', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

In [6]:
```
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

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

In [ ]:

Rashmi Dhaduti
1BM18CS080

3. Decision Tree ID3 algorithm

```
import math
import csv
```

In [2]:

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

In [3]:

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

In [4]:

```
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
```

In [5]:

```
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
```

In [6]:

```
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
```

In [7]:

```
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
```

In [8]:

```
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)
```

In [9]:

Rashmi Dhaduti
1BM18CS080

```
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)
```

In [10]:

```
'''Main Program'''
dataset, features = load_csv("data3.csv")
model = build_tree(dataset, features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(model, 0)
testdata, features = load_csv("data3_test.csv")
for xtest in testdata:
    print("The test instance: ", xtest)
    print("The label for test instance: ", end = " ")
    classify(model, xtest, features)
```

Output

In [11]:

```
'''Main Program'''
dataset, features = load_csv("data3.csv")
model = build_tree(dataset, features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(model, 0)
testdata, features = load_csv("data3_test.csv")
for xtest in testdata:
    print("The test instance: ", xtest)
    print("The label for test instance: ", end = "  ")
    classify(model, xtest, features)
```

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
    overcast
       yes
    rain
       Wind
          strong
             no
          weak
             yes
    sunny
       Humidity
          normal
             yes
          high
             no
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
```

In [ ]:

Rashmi Dhaduti
1BM18CS080

4. Naïve Bayesian algorithm

In [1]:

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

In [2]:

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

In [3]:

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

In [7]:

```
print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.30)

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

In [8]:

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

In [9]:

```
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)
```

Rashmi Dhaduti
1BM18CS080

Output

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

In [2]:
```
df = pd.read_csv("pima_indian.csv")
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
predicted_class_names = ['diabetes']
```

In [3]:
```
X = df[feature_col_names].values
y = df[predicted_class_names].values
```

In [7]:
```
print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.30)

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

```
<bound method NDFrame.head of      num_preg  glucose_conc  diastolic_bp  thickness  insulin   bmi  \
0           6           148            72         35        0  33.6
1           1            85            66         29        0  26.6
2           8           183            64          0        0  23.3
3           1            89            66         23       94  28.1
4           0           137            40         35      168  43.1
..        ...           ...           ...        ...      ...   ...
763        10           101            76         48      180  32.9
764         2           122            70         27        0  36.8
765         5           121            72         23      112  26.2
766         1           126            60          0        0  30.1
767         1            93            70         31        0  30.4

     diab_pred  age  diabetes
0        0.627   50         1
1        0.351   31         0
2        0.672   32         1
3        0.167   21         0
4        2.288   33         1
..         ...  ...       ...
763      0.171   63         0
764      0.340   27         0
765      0.245   30         0
766      0.349   47         1
767      0.315   23         0

[768 rows x 9 columns]>

 the total number of Training Data : (537, 1)

 the total number of Test Data : (231, 1)
```

In [8]:
```
clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
```

In [9]:
```
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)
```

```
 Confusion matrix
[[129  28]
 [ 27  47]]

 Accuracy of the classifier is 0.7619047619047619

 The value of Precision 0.6266666666666667

 The value of Recall 0.6351351351351351
```

Rashmi Dhaduti
1BM18CS080

5. Bayesian Network Classifier algorithm

In [1]:

```
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
```

In [3]:

```
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())

print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

In [4]:

```
model= BayesianModel([('age','Heartdisease'),('sex','Heartdisease'),('exang','Hea
rtdisease'),('cp','Heartdisease'),('Heartdisease','restecg'),('Heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
```

In [7]:

```
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['Heartdisease'],evidence={'restecg':
1})
print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['Heartdisease'],evidence={'cp':2})
print(q2)
```

Rashmi Dhaduti
1BM18CS080

Output

In [1]:
```
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
```

In [3]:
```
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())

print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
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
```

In [4]:
```
model= BayesianModel([('age','Heartdisease'),('sex','Heartdisease'),('exang','Heartdisease'),('cp','Heartdisease'),('Heartdisease','restecg'),('Heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
```

```
Learning CPD using Maximum likelihood estimators
```

In [7]:
```
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['Heartdisease'],evidence={'restecg':1})
print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['Heartdisease'],evidence={'cp':2})
print(q2)
```

```
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, 78.41it/s]
Finding Elimination Order: : 100%|████████████████████████████████████████████████████████| 5/5 [00:00<00:00, 56.97it/s]
```

Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence= restecg

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

```
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: restecg:   0%|                                                                      | 0/5 [00:00<?, ?it/s]
Eliminating: chol:   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, 125.34it/s]
```

Heartdisease	phi(Heartdisease)
Heartdisease(0)	0.3610
Heartdisease(1)	0.2159
Heartdisease(2)	0.1373
Heartdisease(3)	0.1537
Heartdisease(4)	0.1321

In [ ]:

Rashmi Dhaduti
1BM18CS080

6. Bayesian Network using cancer dataset

```
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 models are :')
print('\t',cancer_model.nodes())
print('Bayesian 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])


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

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

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'))
```

Rashmi Dhaduti
1BM18CS080

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

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

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

Output

In [1]:
```
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
```

In [2]:
```
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())
```

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

In [3]:
```
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])
```

In [4]:
```
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp)
print('Model generated by adding conditional probability distributions(cpds)')
```

```
Model generated by adding conditional probability distributions(cpds)
```

In [5]:
```
print('Checking for correctness of model: ', end="")
print(cancer_model.check_model())
```

```
Checking for correctness of model: True
```

In [6]:
```
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       |
+-------------+-----------+-----------+
```

In [7]:
```
cancer_infer = VariableElimination(cancer_model)
```

In [8]:
```
print("\nInferencing with Bayesian Network")

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

```
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, 375.83it/s]
Inferencing with Bayesian Network

Probability of Cancer given Smoker
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0029 |
+-----------+---------------+
| Cancer(1) |        0.9971 |
+-----------+---------------+
```

In [9]:
```
print("\nProbability of Cancer given Smoker, Pollution")
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker':1, 'Pollution': 1})
print(q)
```

```
Finding Elimination Order: : 100%|████████████████████████████████████████| 3/3 [00:20<00:00,  6.93s/it]
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, 334.18it/s]
Probability of Cancer given Smoker, Pollution
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0200 |
+-----------+---------------+
| Cancer(1) |        0.9800 |
+-----------+---------------+
```

Rashmi Dhaduti
1BM18CS080

Rashmi Dhaduti
1BM18CS080

Rashmi Dhaduti
1BM18CS080

Rashmi Dhaduti
1BM18CS080