**Experiment No. 08**

**Aim:** Implementation of Bayes' Belief Network (Probabilistic Reasoning in an Uncertain Domain)

**Objectives:**

To impart a basic understanding of some of the more advanced topics of AI such as planning and Uncertain Knowledge and Reasoning.

**Outcomes:**

Formulate and solve problems with uncertain information using Bayesian approaches.

**Theory:**

Probabilistic reasoning The aim of a reasoning is to combine the capacity of probability theory to handle uncertainty with the capacity of deductive logic to exploit structure. The result is a richer and more expressive formalism with a broad range of possible application areas. Probabilistic logics attempt to find a natural extension of traditional logic truth tables: the results they define are derived through probabilistic expressions instead. A difficulty with probabilistic logics is that they tend to multiply the computational complexities of their probabilistic and logical components. Other difficulties include the possibility of counter intuitive results, such as those of Dempster-Shafer theory. The need to deal with a broad variety of contexts and issues has led to many different proposals. Probabilistic Reasoning

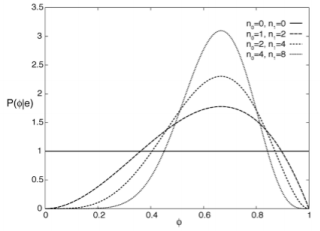
Using Bayesian Learning: The idea of Bayesian learning is to compute the posterior probability distribution of the target features of a new example conditioned on its input features and all of the training examples.

Example: Consider the simplest learning task under uncertainty. Suppose there is a single Boolean random variable, Y. One of two outcomes, a and ¬a, occurs for each example. We want to learn the probability distribution of Y given some examples.

There is a single parameter, φ, that determines the set of all models. Suppose that φ represents the probability of Y=true. We treat this parameter as a real-valued random variable on the interval [0,1]. Thus, by definition of φ, P(a|φ) =φ and P(¬a|φ) =1-φ.

Suppose an agent has no prior information about the probability of Boolean Variable Y and no knowledge beyond the training examples. This ignorance can be modelled by having the prior probability distribution of the variable φ as a uniform distribution over the interval [0,1]. This is the probability density function labeled 0=0, n1=0 in.

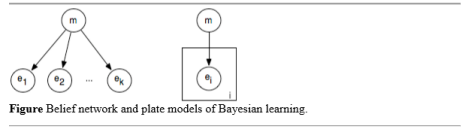
We can update the probability distribution of φ given some examples. Assume that the examples, obtained by running a number of independent experiments, are a particular sequence of outcomes that consists of n0 cases where Y is false and n1 cases where Y is true.



The posterior distribution for φ given the training examples can be derived by Bayes' rule. Let the examples e be the particular sequence of observation that resulted in n1 occurrences of Y=true and n0 occurrences of Y=false. Bayes' rule gives us

P(φ|e) =(P(e|φ) ×P(φ))/(P(e)).

The IID assumption can be represented as a belief network, where each of the ei are independent given model m. This independence assumption can be represented by the belief network.



If m is made into a discrete variable, any of the inference methods of the previous chapter can be used for inference in this network. A standard reasoning technique in such a network is to condition on all of the observed ei and to query the model variable or an unobserved ei variable. The problem with specifying a belief network for a learning problem is that the model grows with the number of observations. Such a network can be specified before any observations have been received by using a plate model. A plate model specifies what variables will be used in the model and what will be repeated in the observations. The plate is drawn as a rectangle that contains some nodes, and an index (drawn on the bottom right of the plate).

The nodes in the plate are indexed by the index. In the plate model, there are multiple copies of the variables in the plate, one for each value of the index. The intuition is that there is a pile of plates, one for each value of the index. The number of plates can be varied depending on the number of observations and what is queried. In this figure, all of the nodes in the plate share a common parent. The probability of each copy of a variable in a plate given the parents is the same for each index.

A plate model lets us specify more complex relationships between the variables. In a hierarchical Bayesian model, the parameters of the model can depend on other parameters. Such a model is hierarchical in the sense that some parameters can depend on other parameters.

**CODE:**

def debugprint(\*args):

pass

def Pr(var, val, e, bn):

parents = bn[var][0]

debugprint('Pr\*\*\*', var, val, e, bn, parents)

if len(parents) == 0:

truePr = bn[var][1][None]

else:

debugprint(' Pr\*\*\*')

parentVals = [e[parent] for parent in parents]

truePr = bn[var][1][tuple(parentVals)]

if val==True: return truePr

else: return 1.0-truePr

def normalize(QX):

total = 0.0

for val in QX.values():

total += val

for key in QX.keys():

QX[key] /= total

return QX

def enumerationAsk(X, e, bn,varss):

QX = {}

for xi in [False,True]:

e[X] = xi

QX[xi] = enumerateAll(varss,e,bn)

del e[X]

#return QX

return normalize(QX)

def enumerateAll(varss, e,bn):

debugprint('EnumerateAll\*\*\*', varss, e, bn)

if len(varss) == 0: return 1.0

Y = varss.pop()

if Y in e:

val = Pr(Y,e[Y],e,bn) \* enumerateAll(varss,e,bn)

varss.append(Y)

return val

else:

total = 0

e[Y] = True

total += Pr(Y,True,e,bn) \* enumerateAll(varss,e,bn)

e[Y] = False

total += Pr(Y,False,e,bn) \* enumerateAll(varss,e,bn)

del e[Y]

varss.append(Y)

return total

bn = {'Burglary':[[],{None:.001}],

'Earthquake':[[],{None:.002}],

'Alarm':[['Burglary','Earthquake'],

{(False,False):.001,(False,True):.29,

(True,False):.94,(True,True):.95}],

'JohnCalls':[['Alarm'],

{(False,):.05,(True,):.90}],

'MaryCalls':[['Alarm'],

{(False,):.01,(True,):.70}]}

varss = ['MaryCalls','JohnCalls','Alarm','Burglary','Earthquake']

print(enumerationAsk('Burglary',{'JohnCalls':True,'MaryCalls':True},bn,varss))

import random

def priorSample(bn, varss):

varss.reverse()

e = {}

for var in varss:

prTrue = Pr(var,True,e,bn)

if random.uniform(0.0,1.0) <= prTrue:

e[var] = True

else:

e[var] = False

varss.reverse()

return e

def consistent(e1, e2):

for k in e1:

if k in e2 and e1[k] != e2[k]: return False

return True

def rejectionSample(X,bn,e,num,varss):

N = {True:0, False:0}

for i in range(0, num):

sample = priorSample(bn,varss)

if consistent(sample,e):

N[sample[X]] += 1

total = float(N[True] + N[False])

if total <= .5:

print('No values...')

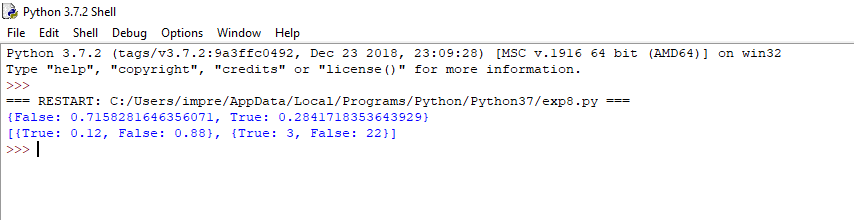
return None

QX = {True: N[True]/total, False: N[False]/total}

return [QX, N]

print(rejectionSample('Burglary',bn,{'JohnCalls':True,'MaryCalls':True},10000,varss))

**OUTPUT:**

**Conclusion:**

From this experiment, the concept of Bayes’ Belief Network was understood and implemented successfully.