Expt. No. 9 Date

Page No. 17.

Aim :- To implement Bayesian network for monty hall.

description: A Bayesian network falls under the category of probabilities graphical modelling technique that is used to compute uncertainties by using concept of probabilities. It is used in directed acyclic graphs.

code :- input math.

```
from Domegranate import*
g = Discrete Distribution ({'A': 1/3 ; 'B': 1/3, 'C': 1/3})
D = Discrete Distribution ({'A': 1/3 ; 'B': 1/3, 'C': 1/3})
m = Conditional probability Table ([['A', 'A', 'A', 0],
['A', 'A', 'B', 0.5] ['A', 'A', 'C', 0.5], ['A', 'B', 'A', 0]
['A', 'B', 'B', 0], ['A', 'C', 'C', 1], ['A', 'C', 'A', 0].
['A', 'C', 'B', 1]. ['A', 'C', 'C', 0], ['B', 'A', 'B', '0'] ['B', 'A', 'C'],
['B', 'B', 'A', 0.5'], ['B', 'B', 'B', 0], ['B', 'B', 'C', 0.5). ['B', 'C', 'A', 1)
['B', 'C', 'B', 0], ['B', 'C', 'C', 0), ['C', 'A', 'A', 0)
['C', 'A', 'B', 1], ['C', 'A', 'C', 0], ['C', 'B', 'A', 1]
['C', 'B', 'B', 0], ['C', 'B', 'C', 0], ['C', 'C', 'A', 0.5)
['C', 'C', 'B', 0.5], ['C', 'C', 'C', 0], [g, F))
d1 = State (g, name = "guest")
d2 = State (P, name = "plize")
d3 = State (m, name = "monty")
n = BayesianNetwork ("solving the monty hall problem")
n. add_State (d1, d2, d3)
n. add_edge (d1, d3)
n. add_edge (d2, d3)
n. bake ()
```

Teacher's Signature

<u>output:-</u>

guest A .
Prize {
"class": "Distribution",
"dtype" : "str"
"name" : "Discrete distribution"
"parameters": [
{
'A' : 0.333,
'B' : 0.333,
'C' : 0.33, }}].

```
b=n.predict_proba({'g':'A'})
b= map ( str, b)
Print ("n", join ("{ }t{ }". format (state.name, belief) for state , belief in zip (n.states, beliefs))
```

Expt. No. 10 Date 18/11/22

Page No. 18.

Aim:- To implement burglary and earthquake alarm problem

description: A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using directed cyclic graph.

problem:- Calculate the probability that alarm has sounded but their is neither a burglary nor an earthquake occured and John & mary both called the alarm.

Solution :- The network structure shows that burglary and earthquake is parent node of alarm and directly affecting probability of alarm's going off but John and mary calls depend on alarm probability.

In CPT, boolean variable with k boolean parents contain 2^k probabilities.

Events - Burglary (B), Earthquake (E), Alarm(A), Johncalls (J), marycalls (M).

Code:-

```
Pip install pomegranate.
from collections import defaultdict, Counter
import itertools.
import math.
import random.
class BayesNet(Object)
    def __init__(self)
        self.variables = [], self.backup = {}.
```

Teacher's Signature

```
...n . opt .
def add (self.name, parent names, cpt):
        parents = [self.lookup [name] for name in parentnames]
        var = variables(name, cpt, parents)
        self . variables.append(var)
        self . lookup.[name]= var
        return self.
class variable (object):
    def __init__(self, name, cpt, parent =[]):
            self.__name__= name
            self.__parents = parents.
            self.cpt = CPTable (cpt, parents)
```

Teacher's Signature ____________

Date _____

Expt. No. _____ Page No. 21

```
        self.domain = set(itertools.chain(*self.cpt.values()))
    def __repr__(self):
        return self.__name__
class Factor(dict):
class Probdist(Factor):
    def __init__(self, mapping=(), **kwargs):
        if isinstance(mapping, float):
            mapping = {T: mapping, F: 1-mapping}
        self.update(mapping, **kwargs)
        normalize(self)
class Evidence(dict):
class Ctable(dict):
    def __init__(self, mapping, parent=()):
```

```
if len(parents)==0 and not(isinstance(mapping,dict)
and set(mapping, key[]) == {[]}):
        mapping = {() mapping}
for (row, dict) in mapping.items():
    if (len(parents)==1 and not(isinstance(row,
                                tuple)):
        row = (row).
        self[row] = Probdist(dist).
class Boot(int):
    __str__ = __repr__ = lambda self: 'T' if
self else 'F'
T = Bool(True)
F = Bool(False)
def P(var, evidence={}):
    row = tuple(evidence[parent] for
        parent in var.parents)
    return var.cpt[row]
def normalize(dist):
    total = sum(dist.values())
    for key in dist:
        dist[key] = dist[key]/total.
        assert 0<= dist[key]<=1,
    return dist.
def sample(Probdist):
    r = random.random()
    c = 0.0
for outcome in Probdist:
    ct = probdist[outcome]
    if r<= c:
        return outcome.
```

Date

Expt. No. _____ Page No. 20

```
def globalize(mapping):
    globals().update(mapping)
alarm_net = BayesNet()
    .add('Burglary', [], 0.001)
    .add('Earthquake', [], 0.002)
    .add('Alarm', ['Burglary', 'Earthquake'],
        {(T,T): 0.95, (T,F): 0.94, (F,T): 0.29,
        (F,F): 0.01})
    .add('JohnCalls', ['Alarm'], {T: 0.90, F: 0.03})
    .add('MaryCalls', ['Alarm'], {T: 0.70, F: 0.01}))

globalize(alarm_net.lookup)
alarm_net.variables, [Burglary, Earthquake, Alarm, JohnCalls,
    MaryCalls].
P(Burglary) {P: 0.999, T: 0.001}.
P{Alarm, {Burglary: T, Earthquake: F}):
    {F: 0.06, T: 0.94}
Alarm.cpt.
```

output :- {(T,T) {T : 0.95, F : 0.05},
(T, F) : {T: 0.94, F : 0.06},
(F,T) : {T: 0.29, F : 0.71},
(F, F) : {T: 0.001, F : 0.999}}.