**Program 7**

**Write a program to construct aBayesian network considering medical data. Use this**

**model to demonstrate the diagnosis of heart patients using standard Heart Disease**

**Data Set. You can use Java/Python ML library classes/API.**

**Task:** Bayesian networks are a type of **Probabilistic Graphical Model** that can be used to build models from data and/or expert opinion.

They can be used for a wide range of tasks including prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction and decision making under uncertainty. The four major analytics disciplines are , Descriptive analytics, Diagnostic analytics, Predictive analytics and Prescriptive analytics. They are also commonly referred to as Bayes nets, Belief networks and sometimes Causal networks.

**Description:**

The Bayesian belief network is a powerful knowledge representation and reasoning tool under conditions of uncertainty. A Bayesian belief-network is a directed acyclic graph (DAG) with a conditional probability distribution for each node .The DAG structure of such networks contains nodes representing domain variables, and arcs between nodes representing probabilistic dependencies. On constructing Bayesian networks from databases, we use nodes to represent database attributes.

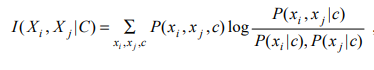
An Approach Based on Information Theory, the Bayesian belief network is a kind of probabilistic models. It uses DAG to represent dependency relationships between variables. Since every independence statement in belief networks satisfies a group of axioms , we can construct belief networks from data by analyzing conditional independence relationships. For any three disjoint node sets X, Y, and Z in a belief network, X is said to be d-separated from Y by Z if there is no active undirected path between X and Y. A path between X and Y is active if:

(1) every node in the path having head-to-head arrows is in Z or has a descendant in Z;

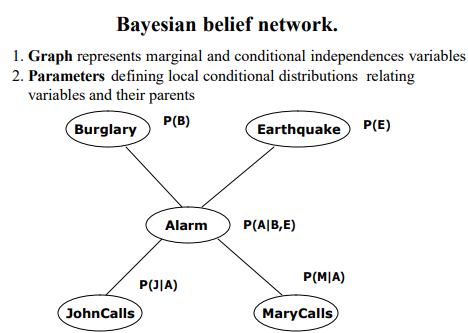
(2) every other node in the path is outside Z. To understand d-Separation, we can use an analogy, which is similar to the one suggested in.

A Belief network is viewed as a network system of information channels, where each node is a valve that is either active or inactive and the valves are connected by noisy information channels. The information flow can pass through an active valve but not an inactive one. When all the valves (nodes) on one undirected path between two nodes are active, we say this path is open. If anyone valve in the path is inactive, we say the path is closed. When all paths between two nodes are closed given the status of a set of valves (nodes), we say the two nodes are d-separated by the set of nodes. The status of valves can be changed through the instantiation of a set of nodes. The amount of information flow between two nodes can be measured by using mutual information, when no nodes are instantiated, or conditional mutual information, when some other nodes are instantiated. In information theory, the mutual information of two nodes xi x j , is defined as





where Xi , X j are two nodes and C is a set of nodes. In our algorithm, we use conditional mutual information as CI tests to measure the average information between two nodes when the statuses of some valves are changed by the condition-set C. When I( Xi , X j |C) is smaller than a certain threshold value ε , we say that Xi , X j are d-separated by the condition-set C, and they are conditionally independent. Below is an example of a bayesian belief network



**BayesPy package:**

The key steps in using BayesPy for variational Bayesian inference by applying BayesPy to a simple problem is as below.

* Construct the model
* Observe some of the variables by providing the data in a proper format
* Run variational Bayesian inference
* Examine the resulting posterior approximation

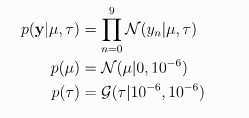
To demonstrate BayesPy, we’ll consider a very simple problem: we have a set of observations from a Gaussian distribution with unknown mean and variance, and we want to learn these parameters. In this case, we do not use any real-world data but generate some artificial data. The dataset consists of ten samples from a Gaussian distribution with mean 5 and standard deviation 10. This dataset can be generated with NumPy as follows:

>>> import numpy as np

>>> data = np.random.normal(5, 10, size=(10,))

**Constructing the model**

Now, given this data we would like to estimate the mean and the standard deviation as if we didn’t know their values. The model can be defined as follows:



where N is the Gaussian distribution parameterized by its mean and precision (i.e., inverse variance), and G is the gamma distribution parameterized by its shape and rate parameters. Note that we have given quite uninformative priors for the variables Mu and T. This simple model can also be shown as a directed factor graph:

% tikzlibrary.code.tex
%
% Copyright 2010-2011 by Laura Dietz
% Copyright 2012 by Jaakko Luttinen
%
% This file may be distributed and/or modified
%
% 1. under the LaTeX Project Public License and/or
% 2. under the GNU General Public License.
%
% See the files LICENSE_LPPL and LICENSE_GPL for more details.

% Load other libraries
\usetikzlibrary{shapes}
\usetikzlibrary{fit}
\usetikzlibrary{chains}
\usetikzlibrary{arrows}

% Latent node
\tikzstyle{latent} = [circle,fill=white,draw=black,inner sep=1pt,
minimum size=20pt, font=\fontsize{10}{10}\selectfont, node distance=1]
% Observed node
\tikzstyle{obs} = [latent,fill=gray!25]
% Constant node
\tikzstyle{const} = [rectangle, inner sep=0pt, node distance=1]
% Factor node
\tikzstyle{factor} = [rectangle, fill=black,minimum size=5pt, inner
sep=0pt, node distance=0.4]
% Deterministic node
\tikzstyle{det} = [latent, diamond]

% Plate node
\tikzstyle{plate} = [draw, rectangle, rounded corners, fit=#1]
% Invisible wrapper node
\tikzstyle{wrap} = [inner sep=0pt, fit=#1]
% Gate
\tikzstyle{gate} = [draw, rectangle, dashed, fit=#1]

% Caption node
\tikzstyle{caption} = [font=\footnotesize, node distance=0] %
\tikzstyle{plate caption} = [caption, node distance=0, inner sep=0pt,
below left=5pt and 0pt of #1.south east] %
\tikzstyle{factor caption} = [caption] %
\tikzstyle{every label} += [caption] %

\tikzset{>={triangle 45}}

%\pgfdeclarelayer{b}
%\pgfdeclarelayer{f}
%\pgfsetlayers{b,main,f}

% \factoredge [options] {inputs} {factors} {outputs}
\newcommand{\factoredge}[4][]{ %
  % Connect all nodes #2 to all nodes #4 via all factors #3.
  \foreach \f in {#3} { %
    \foreach \x in {#2} { %
      \draw[-,#1] (\x) edge[-] (\f) ; %
    } ;
    \foreach \y in {#4} { %
      \draw[->,#1] (\f) -- (\y) ; %
    } ;
  } ;
}

% \edge [options] {inputs} {outputs}
\newcommand{\edge}[3][]{ %
  % Connect all nodes #2 to all nodes #3.
  \foreach \x in {#2} { %
    \foreach \y in {#3} { %
      \draw[->,#1] (\x) -- (\y) ;%
    } ;
  } ;
}

% \factor [options] {name} {caption} {inputs} {outputs}
\newcommand{\factor}[5][]{ %
  % Draw the factor node. Use alias to allow empty names.
  \node[factor, label={[name=#2-caption]#3}, name=#2, #1,
  alias=#2-alias] {} ; %
  % Connect all inputs to outputs via this factor
  \factoredge {#4} {#2-alias} {#5} ; %
}

% \plate [options] {name} {fitlist} {caption}
\newcommand{\plate}[4][]{ %
  \node[wrap=#3] (#2-wrap) {}; %
  \node[plate caption=#2-wrap] (#2-caption) {#4}; %
  \node[plate=(#2-wrap)(#2-caption), #1] (#2) {}; %
}

% \gate [options] {name} {fitlist} {inputs}
\newcommand{\gate}[4][]{ %
  \node[gate=#3, name=#2, #1, alias=#2-alias] {}; %
  \foreach \x in {#4} { %
    \draw [-*,thick] (\x) -- (#2-alias); %
  } ;%
}

% \vgate {name} {fitlist-left} {caption-left} {fitlist-right}
% {caption-right} {inputs}
\newcommand{\vgate}[6]{ %
  % Wrap the left and right parts
  \node[wrap=#2] (#1-left) {}; %
  \node[wrap=#4] (#1-right) {}; %
  % Draw the gate
  \node[gate=(#1-left)(#1-right)] (#1) {}; %
  % Add captions
  \node[caption, below left=of #1.north ] (#1-left-caption)
  {#3}; %
  \node[caption, below right=of #1.north ] (#1-right-caption)
  {#5}; %
  % Draw middle separation
  \draw [-, dashed] (#1.north) -- (#1.south); %
  % Draw inputs
  \foreach \x in {#6} { %
    \draw [-*,thick] (\x) -- (#1); %
  } ;%
}

% \hgate {name} {fitlist-top} {caption-top} {fitlist-bottom}
% {caption-bottom} {inputs}
\newcommand{\hgate}[6]{ %
  % Wrap the left and right parts
  \node[wrap=#2] (#1-top) {}; %
  \node[wrap=#4] (#1-bottom) {}; %
  % Draw the gate
  \node[gate=(#1-top)(#1-bottom)] (#1) {}; %
  % Add captions
  \node[caption, above right=of #1.west ] (#1-top-caption)
  {#3}; %
  \node[caption, below right=of #1.west ] (#1-bottom-caption)
  {#5}; %
  % Draw middle separation
  \draw [-, dashed] (#1.west) -- (#1.east); %
  % Draw inputs
  \foreach \x in {#6} { %
    \draw [-*,thick] (\x) -- (#1); %
  } ;%
}

\node[obs]                                  (y)     {$y_n$} ;
\node[latent, above left=1.5 and 0.5 of y]  (mu)    {$\mu$} ;
\node[latent, above right=1.5 and 0.5 of y] (tau)   {$\tau$} ;
\factor[above=of mu] {mu-f} {left:$\mathcal{N}$} {} {mu} ;
\factor[above=of tau] {tau-f} {left:$\mathcal{G}$} {} {tau} ;

\factor[above=of y] {y-f} {left:$\mathcal{N}$} {mu,tau}     {y};

\plate {} {(y)(y-f)(y-f-caption)} {$n=0,\ldots,9$} ;

Directed factor graph of the example model.This model can be constructed in BayesPy as follows:

>>> from bayespy.nodes import GaussianARD, Gamma

>>> mu = GaussianARD(0, 1e-6)

>>> tau = Gamma(1e-6, 1e-6)

>>> y = GaussianARD(mu, tau, plates=(10,))

This is quite self-explanatory given the model definitions above. We have used two types of nodes [GaussianARD](http://www.bayespy.org/user_api/generated/generated/bayespy.nodes.GaussianARD.html#bayespy.nodes.GaussianARD) and [Gamma](http://www.bayespy.org/user_api/generated/generated/bayespy.nodes.Gamma.html#bayespy.nodes.Gamma) to represent Gaussian and gamma distributions, respectively. There are much more distributions in [bayespy.nodes](http://www.bayespy.org/user_api/generated/bayespy.nodes.html#module-bayespy.nodes) so you can construct quite complex conjugate exponential family models. The node y uses keyword argument plates to define the plates .

**Performing inference**

Now that we have created the model, we can provide our data by setting y as observed:

>>> y.observe(data)

Next we want to estimate the posterior distribution. In principle, we could use different inference engines (e.g., MCMC or EP) but currently only variational Bayesian (VB) engine is implemented. The engine is initialized by giving all the nodes of the model:

>>> from bayespy.inference import VB

>>> Q = VB(mu, tau, y)

The inference algorithm can be run as long as wanted (max. 20 iterations in this case):

>>> Q.update(repeat=20)

Iteration 1: loglike=-6.020956e+01 (... seconds)

Iteration 2: loglike=-5.820527e+01 (... seconds)

Iteration 3: loglike=-5.820290e+01 (... seconds)

Iteration 4: loglike=-5.820288e+01 (... seconds)

Converged at iteration 4.

Now the algorithm converged after four iterations, before the requested 20 iterations.

**Program in Python:**

import bayespy as bp

import numpy as np

import csv

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}

#heart\_disease\_data.csv

with open('heart\_disease\_data.csv') as csvfile:

lines = csv.reader(csvfile)

dataset = list(lines)

data = []

for x in dataset:

data.append([ageEnum[x[0]],genderEnum[x[1]],familyHistoryEnum[x[2]],dietEnum[x[3]],lifeStyleEnum[x[4]],cholesterolEnum[x[5]],heartDiseaseEnum[x[6]]])

# Training data for machine learning todo: should import from csv

data = np.array(data)

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 values

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.Categorical, p\_heartdisease)

heartdisease.observe(data[:,6])

p\_heartdisease.update()

# Sample Test with hardcoded values

#print("Sample Probability")

#print("Probability(HeartDisease|Age=SuperSeniorCitizen, Gender=Female, FamilyHistory=Yes, DietIntake=Medium, LifeStyle=Sedetary, Cholesterol=High)")

#print(bp.nodes.MultiMixture([ageEnum['SuperSeniorCitizen'], genderEnum['Female'], familyHistoryEnum['Yes'], dietEnum['Medium'], lifeStyleEnum['Sedetary'], cholesterolEnum['High']], bp.nodes.Categorical, p\_heartdisease).get\_moments()[0][heartDiseaseEnum['Yes']])

# 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 FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum: ' + str(dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' + str(cholesterolEnum)))], bp.nodes.Categorical, 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 "))

**Input CSV file:**

heart\_disease\_data.csv

**Program Output:**

C:\Users\RAGS\Desktop\BBN>python BBN.py

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

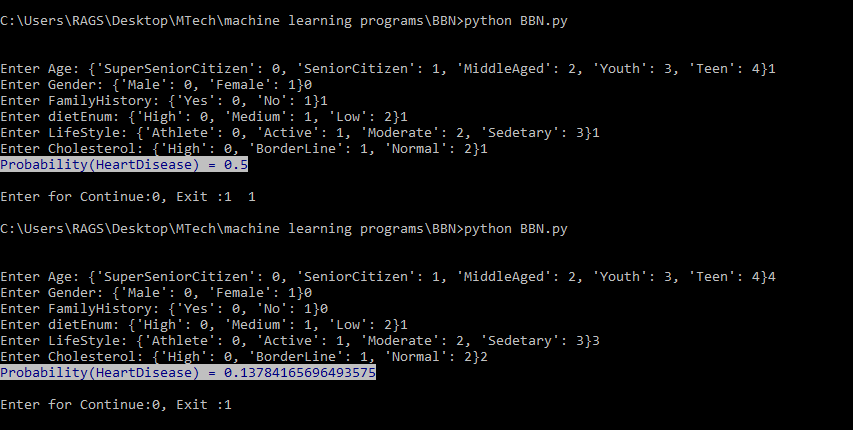
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}3

Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2

Probability(HeartDisease) = 0.13784165696493575

Enter for Continue:0, Exit :1 1

**Screenshot:**



**Advantages:**

Bayesian belief networks

* Give solutions to the space, acquisition bottlenecks
* Significant improvements in the time cost of inferences

**Real time applications:**

Bayesian networks are used for

* [Modelling](https://en.wikipedia.org/wiki/Mathematical_model) beliefs in [computational biology](https://en.wikipedia.org/wiki/Computational_biology) ,
* [Bioinformatics](https://en.wikipedia.org/wiki/Bioinformatics) ([gene regulatory networks](https://en.wikipedia.org/wiki/Gene_regulatory_network), [protein structure](https://en.wikipedia.org/wiki/Protein_structure), [gene expression](https://en.wikipedia.org/wiki/Gene_expression) analysis,
* learning [epistasis](https://en.wikipedia.org/wiki/Epistasis) from [GWAS](https://en.wikipedia.org/wiki/Genome-wide_association_study) data sets,
* [Medicine](https://en.wikipedia.org/wiki/Medicine),
* Forensics,
* [Biomonitoring](https://en.wikipedia.org/wiki/Biomonitoring),
* [Document Classification](https://en.wikipedia.org/wiki/Document_classification),
* [Information retrieval](https://en.wikipedia.org/wiki/Information_retrieval),
* S[emantic search](https://en.wikipedia.org/wiki/Semantic_search),
* Image Processing,
* [Data](https://en.wikipedia.org/wiki/Data_fusion) fusion
* Geophysics and [volcano](https://en.wikipedia.org/wiki/Volcano) monitoring,
* [Law](https://en.wikipedia.org/wiki/Law)