**University of Missouri-Columbia / College of Engineering**

**CS 8750: Artificial Intelligence II**

****

**Programming Assignment #2**

Chanmann Lim  
Adil A. Al-Azzawi  
Fernando Torre

# Introduction

In our previous assignment, we implemented two discrete Bayesian networks to tackle the problem of determining a person’s state of drunkenness from already established conditional probabilities tables (CPTs). We found that the conditional probability differed appreciably from the prior probability for all networks, suggesting an imperfect model. Furthermore, new data is more likely to come as a measured value as opposed to three discrete options.

In dealing with real-value data we cannot assign the probability for each possible value: not only would the probabilities for each measurement be unusably small, listing the probabilities for all real value data-points is simply impossible. However, it is possible to learn CPTs from measured, real-value data. In this regard, two solutions appear to be reasonable:

1. Discretization of group the real value data into ranges (for Bayesian network 1 and 2).
2. Assume that each independent observation comes from Gaussian (for Bayesian network 3 and 4) then construct the CPTs or parameters of the Gaussian distribution from the given datasets.

In this assignment, we will implement four Bayes networks and measure their confusion matrix. The first two will be our discrete networks from assignment#1 while the last two will be continuous. For each network we give:

1. the Learning Program that learns the CPTs for the Bayesian network, giving the mathematical form first
2. the program code, implemented in Matlab,
3. and the parameters learned from the datasets.

# BN#1

Our first network, from our previous assignment, is a discrete naïve Bayes network where breathing rate, heart rate, and skin temperature are given a single cause: the state of drunkenness. This network takes the continuous values measured and discretizes them into High, Medium, or Low.

To perform the conversion, values are discretized using the following formula:

where is the specific value being converted; is the multiset of all the values the variable takes on in the dataset; , , represent High, Medium, or Low respectively; is the Maximum Likelihood estimator for the mean, given by ; and ; is the Maximum Likelihood Estimator for the standard deviation, given by . Note that here, the vertical bars denote the cardinality of the set.

## Learning program

Our program learns the parameters by computing the conditional probabilities through Maximum Likelihood.

### Formulas of Maximum Likelihood

Let be the set of all values of drunkenness and be the th value of in the dataset (either or ). Let , , and be the set of all values for breathing rate, heart rate, and skin temperature, respectively, in the dataset; with , , and the th value of each respective set in the dataset (H,M, or L).

With this formulation,

specifies a state of the Bayesian network completely. Each of these probabilities can be specified through Maximum Likelihood:

with , and being the desired from the dataset. Once again, the vertical bars denote the cardinality of the set.

Thus,

We can similarly specify as

### Program code

#### BN1.m

%% Learn parameters of the CPDs

P\_Pd = [sum(Pd == 1)/length(Pd); sum(Pd == 0)/length(Pd)];

P\_Xb\_given\_Pd = BN1.CPT(Xb, Pd);

P\_Xh\_given\_Pd = BN1.CPT(Xh, Pd);

P\_Xt\_given\_Pd = BN1.CPT(Xt, Pd);

#### BN1.CPT.m

function [ cpt ] = CPT ( x , Pd )

% BN1.CPT - construct CPT for a random variable in BN 1

% learn CPT parameters using Maximum Likelihood estimator

%

% Input:

% x - discrete random variable. Domain = {H, M, L}

% Pd - evidence variable 'drink'

%

% Output:

% cpt - conditional probability table of the random variable given the

% evidence drink.

% eg.

% Xb

% Pd | H | M | L |

% 1 | | | |

% 0 | | | |

% Note:

% The elements/rows of evidence must correspond to x's

%

domain = ['H', 'M', 'L'];

cpt = zeros(2, length(domain));

Pd\_is\_1 = sum(Pd == 1);

Pd\_is\_0 = sum(Pd == 0);

% count(x = {H, M, L} , pd = {1, 0})

% likelihood = -----------------------------------

% count(pd = {1, 0})

for i = 1:length(domain)

x\_i = Pd(x == domain(i));

cpt(1, i) = sum(x\_i == 1) / Pd\_is\_1;

cpt(2, i) = sum(x\_i == 0) / Pd\_is\_0;

end

end

### Parameters learned

#### Dataset 1001

***P(Pd)***

|  |  |
| --- | --- |
| **+** | 0.1848 |
| **-** | 0.8152 |

***P(Xb|Pd)***

| **Drink** | **H** | **M** | **L** |
| --- | --- | --- | --- |
| **1** | 0 | 0.8824 | 0.1176 |
| **0** | 0.16 | 0.6933 | 0.1467 |

***P(Xh|Pd)***

| **Drink** | **H** | **M** | **L** |
| --- | --- | --- | --- |
| **1** | 0.0588 | 0.8235 | 0.1176 |
| **0** | 0.1067 | 0.8933 | 0 |

***P(Xt|Pd)***

| **Drink** | **H** | **M** | **L** |
| --- | --- | --- | --- |
| **1** | 0 | 0.9412 | 0.0588 |
| **0** | 0.16 | 0.7467 | 0.0933 |

#### Dataset 1004

***P(Pd)***

|  |  |
| --- | --- |
| **+** | 0.2083 |
| **-** | 0.7917 |

***P(Xb|Pd)***

| **Drink** | **H** | **M** | **L** |
| --- | --- | --- | --- |
| **1** | 0.2 | 0.4 | 0.4 |
| **0** | 0.1053 | 0.8070 | 0.0877 |

***P(Xh|Pd)***

| **Drink** | **H** | **M** | **L** |
| --- | --- | --- | --- |
| **1** | 0 | 0.8 | 0.2 |
| **0** | 0.1754 | 0.7368 | 0.0877 |

***P(Xt|Pd)***

| **Drink** | **H** | **M** | **L** |
| --- | --- | --- | --- |
| **1** | 0.0667 | 0.4667 | 0.4667 |
| **0** | 0.1404 | 0.7544 | 0.1053 |

## Prediction program

### Formulas

To determine from , , and , we have,

We will call the terms of this denominator P\_normalized.

### Program code

#### BN1.m

evidence = [Xb Xh Xt];

data\_size = size(evidence, 1);

prediction = zeros(data\_size, 1);

% prediction

model = BN1.model(P\_Pd, P\_Xb\_given\_Pd, P\_Xh\_given\_Pd, P\_Xt\_given\_Pd);

for i = 1:data\_size

prediction(i, 1) = model.predict(1, evidence(i, :));

end

#### BN1.model.m

classdef model

% MODEL - BN1 prediction

% Compute probability of 'Pd' given the evidences

properties

P\_Pd;

P\_Xb\_given\_Pd;

P\_Xh\_given\_Pd;

P\_Xt\_given\_Pd;

end

methods

%

% Initializer

%

function obj = model( ...

P\_Pd, P\_Xb\_given\_Pd, P\_Xh\_given\_Pd, P\_Xt\_given\_Pd)

obj.P\_Pd = P\_Pd;

% Add total of probability as the last column

obj.P\_Xb\_given\_Pd = [P\_Xb\_given\_Pd ones(2,1)];

obj.P\_Xh\_given\_Pd = [P\_Xh\_given\_Pd ones(2,1)];

obj.P\_Xt\_given\_Pd = [P\_Xt\_given\_Pd ones(2,1)];

end

%

% Compute P(pd|e)

%

function P = predict(obj, pd, e)

p\_pd\_is\_1 = obj.p\_pd(1);

p\_xb\_given\_pd\_is\_1 = obj.p\_x\_given\_pd(e(1), 1, obj.P\_Xb\_given\_Pd);

p\_xh\_given\_pd\_is\_1 = obj.p\_x\_given\_pd(e(2), 1, obj.P\_Xh\_given\_Pd);

p\_xt\_given\_pd\_is\_1 = obj.p\_x\_given\_pd(e(3), 1, obj.P\_Xt\_given\_Pd);

p\_pd\_is\_0 = obj.p\_pd(0);

p\_xb\_given\_pd\_is\_0 = obj.p\_x\_given\_pd(e(1), 0, obj.P\_Xb\_given\_Pd);

p\_xh\_given\_pd\_is\_0 = obj.p\_x\_given\_pd(e(2), 0, obj.P\_Xh\_given\_Pd);

p\_xt\_given\_pd\_is\_0 = obj.p\_x\_given\_pd(e(3), 0, obj.P\_Xt\_given\_Pd);

P\_normalized = [ ...

p\_pd\_is\_1 \* p\_xb\_given\_pd\_is\_1 \* p\_xh\_given\_pd\_is\_1 \* p\_xt\_given\_pd\_is\_1 ...

p\_pd\_is\_0 \* p\_xb\_given\_pd\_is\_0 \* p\_xh\_given\_pd\_is\_0 \* p\_xt\_given\_pd\_is\_0

];

P = P\_normalized(obj.pd\_row(pd)) / sum(P\_normalized);

end

end

%

% private functions

%

methods(Access = private)

%

% Lookup P(x|pd) probability from CPT

%

function P = p\_x\_given\_pd(obj, x, pd, cpt)

P = cpt(obj.pd\_row(pd), obj.x\_column(x));

end

%

% Lookup P(pd) from CPT

%

function P = p\_pd(obj, pd)

P = obj.P\_Pd(obj.pd\_row(pd));

end

%

% Map Pd values {1 0} to row [1 2] in the CPT

%

function r = pd\_row(~, pd)

pds = [1 0];

rows = [1 2];

r = rows(pds == pd);

end

%

% Map the values {H, M, L , -} to column [1 2 3 4]

%

function c = x\_column(~, x)

xs = ['H' 'M' 'L' '-'];

columns = [1 2 3 4];

c = columns(xs == x);

end

end

end

## Confusion Matrices

### Dataset 1001

|  |  |  |
| --- | --- | --- |
|  | True + | True − |
| Predicted + | 2/17 | 0/75 |
| Predicted − | 15/17 | 75/75 |

Prediction accuracy = 83.7%

### Dataset 1004

|  |  |  |
| --- | --- | --- |
|  | True + | True − |
| Predicted + | 7/15 | 4/57 |
| Predicted − | 8/15 | 53/57 |

Prediction accuracy = 83.3%

# BN#2

Our second network, from our previous assignment, is a discrete complex Bayes network where breathing rate, heart rate, and skin temperature are given two possible causes: the state of drunkenness and the state of ambulation. This network takes the continuous values measured and discretizes them into High, Medium, or Low using the mechanism described in section 1.

## Learning program

Our program learns the parameters by computing the conditional probabilities through Maximum Likelihood.

### Formulas of Maximum Likelihood

As in section 1, let , , , and represent the drunkenness, breathing rate, heart rate, and skin temperature, respectively. We introduce the set of states of ambulation, denoted by and the values (Fast, Slow, or Stationary) denoted by .

With this formulation,

specifies a state of the Bayesian network completely. Again, each of these probabilities can be specified through Maximum Likelihood:

with , , and being the desired from the dataset. Once again, the vertical bars denote the cardinality of the set.

Thus,

We can similarly specify and as

### Program code

#### BN2.m

%% Learn parameters of the CPDs

P\_Pd = [sum(Pd == 1)/length(Pd); sum(Pd == 0)/length(Pd)];

P\_Xa = [sum(Xa == 1)/length(Xa); sum(Xa == 2)/length(Xa); sum(Xa == 3)/length(Xa)];

P\_Xb\_given\_Pd\_and\_Xa = BN2.CPT(Xb, Pd, Xa);

P\_Xh\_given\_Pd\_and\_Xa = BN2.CPT(Xh, Pd, Xa);

P\_Xt\_given\_Pd\_and\_Xa = BN2.CPT(Xt, Pd, Xa);

#### BN2.CPT.m

function [ cpt ] = CPT ( x , Pd, Xa )

% BN2.CPT - construct CPT for a random variable in BN 2

% learn CPT parameters using Maximum Likelihood estimator

%

% Input:

% x - discrete random variable. Domain = {H, M, L}

% Pd - evidence variable 'drink'. Domain = {1 0}

% Xa - evidence variable 'Ambulation status'. Domain = {1 2 3}

%

% Output:

% cpt - conditional probability table of the random variable given the

% evidence drink and ambulation status.

% eg.

%

% Xb

% |Stationary | Slow | Fast |

% Pd | H | M | L | H | M | L | H | M | L |

% 1 | | | | | | | | | |

% 0 | | | | | | | | | |

% Note:

% The elements/rows of evidences must correspond to x's

%

domain = ['H', 'M', 'L'];

xa\_domain = [1 2 3];

cpt = zeros(2, length(domain) \* length(xa\_domain));

% count(x = {H, M, L} , pd = {1, 0}, xa = {1 2 3})

% likelihood = --------------------------------------------------

% count(pd = {1, 0}, xa = {1 2 3})

for k = 1:length(xa\_domain)

Pd\_is\_1\_and\_xa = sum(Pd == 1 & Xa == xa\_domain(k));

Pd\_is\_0\_and\_xa = sum(Pd == 0 & Xa == xa\_domain(k));

for i = 1:length(domain)

x\_i = Pd(x == domain(i) & Xa == xa\_domain(k));

col = i + (k-1) \* length(domain);

cpt(1, col) = sum(x\_i == 1) / Pd\_is\_1\_and\_xa;

cpt(2, col) = sum(x\_i == 0) / Pd\_is\_0\_and\_xa;

end

end

% normalize lack of data samples

cpt(isnan(cpt)) = 0;

end

### Parameters learned

#### Dataset 1001

***P(Pd)***

|  |  |
| --- | --- |
| **+** | 0.1848 |
| **-** | 0.8152 |

***P(Xa)***

|  |  |
| --- | --- |
| **Stationary** | 0.8152 |
| **Slow** | 0.1087 |
| **Fast** | 0.0761 |

***P(Xb|Pd, Xa)***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Stationary** | | | **Slow** | | | **Fast** | | |
| **Drink** | **H** | **M** | **L** | **H** | **M** | **L** | **H** | **M** | **L** |
| **1** | 0 | 0.875 | 0.125 | 0 | 1 | 0 | 0 | 0 | 0 |
| **0** | 0.1017 | 0.7458 | 0.1525 | 0.2222 | 0.6667 | 0.1111 | 0.5714 | 0.2857 | 0.1429 |

***P(Xh|Pd, Xa)***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Stationary** | | | **Slow** | | | **Fast** | | |
| **Drink** | **H** | **M** | **L** | **H** | **M** | **L** | **H** | **M** | **L** |
| **1** | 0.0625 | 0.8125 | 0.125 | 0 | 1 | 0 | 0 | 0 | 0 |
| **0** | 0.0678 | 0.9322 | 0 | 0.2222 | 0.7778 | 0 | 0.2857 | 0.7143 | 0 |

***P(Xt|Pd, Xa)***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Stationary** | | | **Slow** | | | **Fast** | | |
| **Drink** | **H** | **M** | **L** | **H** | **M** | **L** | **H** | **M** | **L** |
| **1** | 0 | 0.9375 | 0.0625 | 0 | 1 | 0 | 0 | 0 | 0 |
| **0** | 0.1356 | 0.7797 | 0.0847 | 0.1111 | 0.6667 | 0.2222 | 0.4286 | 0.5714 | 0 |

#### Dataset 1004

***P(Pd)***

|  |  |
| --- | --- |
| **+** | 0.2083 |
| **-** | 0.7917 |

***P(Xa)***

|  |  |
| --- | --- |
| **Stationary** | 0.8194 |
| **Slow** | 0.1389 |
| **Fast** | 0.0417 |

***P(Xb|Pd, Xa)***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Stationary** | | | **Slow** | | | **Fast** | | |
| **Drink** | **H** | **M** | **L** | **H** | **M** | **L** | **H** | **M** | **L** |
| **1** | 0.2 | 0.4 | 0.4 | 0 | 0 | 0 | 0 | 0 | 0 |
| **0** | 0.1136 | 0.7727 | 0.1136 | 0.1 | 0.9 | 0 | 0 | 1 | 0 |

***P(Xh|Pd, Xa)***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Stationary** | | | **Slow** | | | **Fast** | | |
| **Drink** | **H** | **M** | **L** | **H** | **M** | **L** | **H** | **M** | **L** |
| **1** | 0 | 0.8 | 0.2 | 0 | 0 | 0 | 0 | 0 | 0 |
| **0** | 0.0909 | 0.8182 | 0.0909 | 0.5 | 0.4 | 0.1 | 0.3333 | 0.6667 | 0 |

***P(Xt|Pd, Xa)***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Stationary** | | | **Slow** | | | **Fast** | | |
| **Drink** | **H** | **M** | **L** | **H** | **M** | **L** | **H** | **M** | **L** |
| **1** | 0.0667 | 0.4667 | 0.4667 | 0 | 0 | 0 | 0 | 0 | 0 |
| **0** | 0.1591 | 0.7045 | 0.1364 | 0.1 | 0.9 | 0 | 0 | 1 | 0 |

## Prediction program

### Formulas

As in section 1, let , , , and represent the drunkenness, breathing rate, heart rate, and skin temperature, respectively, and let represent ambulation as in section 2.1.1.

To determine from , , , we have,

We derive separately and and later plug them into the expression. We note that the former is the joint probability of the entire network in this state and the latter must be expanded by total probability.

Thus, the probability can be expressed as

As before, we will call the denominator P\_normalized.

### Code

#### BN2.m

evidence = [Xb Xh Xt Xa];

data\_size = size(evidence, 1);

prediction = zeros(data\_size, 1);

% prediction

model = BN2.model(P\_Pd, P\_Xa, P\_Xb\_given\_Pd\_and\_Xa, P\_Xh\_given\_Pd\_and\_Xa, P\_Xt\_given\_Pd\_and\_Xa);

for i = 1:data\_size

prediction(i, 1) = model.predict(1, evidence(i, :));

end

#### BN2.model.m

classdef model

% MODEL - BN2 prediction

% Compute probability of 'Pd' given the evidences

properties

P\_Pd;

P\_Xa;

P\_Xb\_given\_Pd\_and\_Xa;

P\_Xh\_given\_Pd\_and\_Xa;

P\_Xt\_given\_Pd\_and\_Xa;

end

methods

%

% Initializer

%

function obj = model( ...

P\_Pd, P\_Xa, P\_Xb\_given\_Pd\_and\_Xa, P\_Xh\_given\_Pd\_and\_Xa, P\_Xt\_given\_Pd\_and\_Xa)

obj.P\_Pd = P\_Pd;

obj.P\_Xa = P\_Xa;

% Add total probabilities

obj.P\_Xb\_given\_Pd\_and\_Xa = [

P\_Xb\_given\_Pd\_and\_Xa(:,1:3) ones(2,1) ...

P\_Xb\_given\_Pd\_and\_Xa(:,4:6) ones(2,1) ...

P\_Xb\_given\_Pd\_and\_Xa(:,7:9) ones(2,1)];

obj.P\_Xh\_given\_Pd\_and\_Xa = [

P\_Xh\_given\_Pd\_and\_Xa(:,1:3) ones(2,1) ...

P\_Xh\_given\_Pd\_and\_Xa(:,4:6) ones(2,1) ...

P\_Xh\_given\_Pd\_and\_Xa(:,7:9) ones(2,1)];

obj.P\_Xt\_given\_Pd\_and\_Xa = [

P\_Xt\_given\_Pd\_and\_Xa(:,1:3) ones(2,1) ...

P\_Xt\_given\_Pd\_and\_Xa(:,4:6) ones(2,1) ...

P\_Xt\_given\_Pd\_and\_Xa(:,7:9) ones(2,1)];

end

%

% Compute P(pd|e)

%

function P = predict(obj, pd, e)

p\_pd\_is\_1 = obj.p\_pd(1);

p\_e\_given\_pd\_is\_1 = obj.p\_e\_given\_pd(1, e);

p\_pd\_is\_0 = obj.p\_pd(0);

p\_e\_given\_pd\_is\_0 = obj.p\_e\_given\_pd(0, e);

P\_normalized = [

p\_pd\_is\_1 \* p\_e\_given\_pd\_is\_1

p\_pd\_is\_0 \* p\_e\_given\_pd\_is\_0

];

P = P\_normalized(obj.pd\_row(pd)) / sum(P\_normalized);

end

end

%

% private functions

%

methods(Access = private)

%

% Compute P(e|pd)

%

function P = p\_e\_given\_pd(obj, pd, e)

xa = e(4);

% xa = 0 (represent xa is not given)

if xa == 0

xa = [1 2 3];

end

p = zeros(1, length(xa));

% sum over all xa

for i = 1:length(xa)

p(i) = obj.P\_Xa(xa) \* obj.p\_x\_given\_pd\_and\_xa(e(1), pd, xa, obj.P\_Xb\_given\_Pd\_and\_Xa) \* obj.p\_x\_given\_pd\_and\_xa(e(2), pd, xa, obj.P\_Xh\_given\_Pd\_and\_Xa) \* obj.p\_x\_given\_pd\_and\_xa(e(3), pd, xa, obj.P\_Xt\_given\_Pd\_and\_Xa);

end

P = sum(p);

end

%

% Lookup P(x|pd,xa) probability from CPT

%

function P = p\_x\_given\_pd\_and\_xa(obj, x, pd, xa, cpt)

P = cpt(obj.pd\_row(pd), obj.x\_column(xa, x));

end

%

% Lookup P(pd) from CPT

%

function P = p\_pd(obj, pd)

P = obj.P\_Pd(obj.pd\_row(pd));

end

%

% Map Pd values {1 0} to row [1 2] in the CPT

%

function r = pd\_row(~, pd)

pds = [1 0];

rows = [1 2];

r = rows(pds == pd);

end

%

% Map the values

% xa = {1, 2, 3} and

% x = {H, M, L , -} to

% column 1:12

%

function c = x\_column(~, xa, x)

xs = ['H' 'M' 'L' '-'];

columns = [1 2 3 4] + (xa - 1) \* 4;

c = columns(xs == x);

end

end

end

## Confusion Matrices

### Dataset 1001

|  |  |  |
| --- | --- | --- |
|  | True + | True − |
| Predicted + | 2/17 | 0/75 |
| Predicted − | 15/17 | 75/75 |

Prediction accuracy = 83.7%

### Dataset 1004

|  |  |  |
| --- | --- | --- |
|  | True + | True − |
| Predicted + | 7/15 | 4/57 |
| Predicted − | 8/15 | 53/57 |

Prediction accuracy = 83.3%

# BN#3

Our third network is a continuous Bayes network where breathing rate, heart rate, and skin temperature are given a single cause: the state of drunkenness.

## Learning program

Our program learns the parameters by computing the conditional probabilities through Maximum Likelihood.

### Formulas of Maximum Likelihood

As in section 1.1.1, the network can be completely specified with

Because , each of these probabilities can be completely specified by using a maximum likelihood estimator over the set of all values of . We will use the and maximum likelihood estimators from section 1.

We assume each conditional probability is also distributed ; however, for the conditional probability maximum likelihood estimators must operate over the set conditioned on the right value in the corresponding , a set which we will denote . More formally,

with

Therefore, each conditional probability can be specified as:

Or more specifically,

Because is still discrete, we estimate it as was done in section 1

### Program code

#### BN3.m

%% Learn parameters of the CPDs

P\_Pd = [sum(Pd == 1)/length(Pd); sum(Pd == 0)/length(Pd)];

P\_Xb\_given\_Pd = mle(Xb, Pd);

P\_Xh\_given\_Pd = mle(Xh, Pd);

P\_Xt\_given\_Pd = mle(Xt, Pd);

#### mle.m

function [ params ] = mle( x, Pd )

% MLE - Maximum likelihood estimator

% Learn parameters (mean and variance) for a 1-D Gaussian dataset

%

% Input:

% x - 1-D Gaussian data sample of the random variable

% Pd - evidence variable 'drink'

%

% Output:

% params - parameters of the distritution over x|pd

% eg.

% Params

% Pd | Mean | Variance |

% 1 | | |

% 0 | | |

%

x\_given\_Pd\_is\_1 = x(Pd == 1);

x\_given\_Pd\_is\_0 = x(Pd == 0);

% empirical mean and variance of x

% mean = sum(x) / length(x);

% var = sum((x - mean) .^ 2) / length(x)

params = [

mean(x\_given\_Pd\_is\_1) var(x\_given\_Pd\_is\_1, 1)

mean(x\_given\_Pd\_is\_0) var(x\_given\_Pd\_is\_0, 1)

];

% normalize lack of data samples

params(isnan(params)) = 0;

end

### Parameters learned

#### Dataset 1001

***P(Pd)***

|  |  |
| --- | --- |
| **+** | 0.1848 |
| **-** | 0.8152 |

***P(Xb|Pd)***

|  |  |  |
| --- | --- | --- |
| **Drink** | **Mean** | **Variance** |
| **1** | 9.3235 | 6.2924 |
| **0** | 11.1467 | 19.3852 |

***P(Xh|Pd)***

|  |  |  |
| --- | --- | --- |
| **Drink** | **Mean** | **Variance** |
| 1 | 78.4412 | 7.8495 |
| 0 | 80.9333 | 31.4089 |

***P(Xt|Pd)***

|  |  |  |
| --- | --- | --- |
| **Drink** | **Mean** | **Variance** |
| 1 | 36.3353 | 0.0352 |
| 0 | 36.2987 | 0.1110 |

#### Dataset 1004

***P(Pd)***

|  |  |
| --- | --- |
| + | 0.2083 |
| - | 0.7917 |

***P(Xb|Pd)***

|  |  |  |
| --- | --- | --- |
| **Drink** | **Mean** | **Variance** |
| 1 | 14.3667 | 22.2156 |
| 0 | 15.8947 | 6.9714 |

***P(Xh|Pd)***

|  |  |  |
| --- | --- | --- |
| **Drink** | **Mean** | **Variance** |
| 1 | 127.9000 | 41.6733 |
| 0 | 135.2544 | 175.0098 |

***P(Xt|Pd)***

|  |  |  |
| --- | --- | --- |
| **Drink** | **Mean** | **Variance** |
| 1 | 35.6867 | 0.2585 |
| 0 | 35.9886 | 0.1025 |

## Prediction program

### Formulas

We will use the same representation as before. Because we are assuming , The probability of any is given by , where is the probability density function for a Gaussian distribution with mean and standard deviation . Given that and have been estimated for all conditional probabilities, can be calculated for any .

To determine from , , and , we again have,

Each probability can be calculated with its respective , function. Again, the denominator will be termed P\_normalized.

### Program code

#### BN3.m

evidence = [Xb Xh Xt];

data\_size = size(evidence, 1);

prediction = zeros(data\_size, 1);

model = BN3.model(P\_Pd, P\_Xb\_given\_Pd, P\_Xh\_given\_Pd, P\_Xt\_given\_Pd);

for i = 1:data\_size

prediction(i, 1) = model.predict(1, evidence(i, :));

end

#### BN3.model.m

classdef model

% MODEL - BN3 prediction

% Compute probability of 'Pd' given the evidences

properties

P\_Pd;

P\_Xb\_given\_Pd;

P\_Xh\_given\_Pd;

P\_Xt\_given\_Pd;

end

methods

%

% Initializer

%

function obj = model( ...

P\_Pd, P\_Xb\_given\_Pd, P\_Xh\_given\_Pd, P\_Xt\_given\_Pd)

obj.P\_Pd = P\_Pd;

obj.P\_Xb\_given\_Pd = P\_Xb\_given\_Pd;

obj.P\_Xh\_given\_Pd = P\_Xh\_given\_Pd;

obj.P\_Xt\_given\_Pd = P\_Xt\_given\_Pd;

end

%

% Compute P(pd|e)

%

function P = predict(obj, pd, e)

p\_pd\_is\_1 = obj.p\_pd(1);

p\_xb\_given\_pd\_is\_1 = obj.p\_x\_given\_pd(e(1), 1, obj.P\_Xb\_given\_Pd);

p\_xh\_given\_pd\_is\_1 = obj.p\_x\_given\_pd(e(2), 1, obj.P\_Xh\_given\_Pd);

p\_xt\_given\_pd\_is\_1 = obj.p\_x\_given\_pd(e(3), 1, obj.P\_Xt\_given\_Pd);

p\_pd\_is\_0 = obj.p\_pd(0);

p\_xb\_given\_pd\_is\_0 = obj.p\_x\_given\_pd(e(1), 0, obj.P\_Xb\_given\_Pd);

p\_xh\_given\_pd\_is\_0 = obj.p\_x\_given\_pd(e(2), 0, obj.P\_Xh\_given\_Pd);

p\_xt\_given\_pd\_is\_0 = obj.p\_x\_given\_pd(e(3), 0, obj.P\_Xt\_given\_Pd);

P\_normalized = [ ...

p\_pd\_is\_1 \* p\_xb\_given\_pd\_is\_1 \* p\_xh\_given\_pd\_is\_1 \* p\_xt\_given\_pd\_is\_1 ...

p\_pd\_is\_0 \* p\_xb\_given\_pd\_is\_0 \* p\_xh\_given\_pd\_is\_0 \* p\_xt\_given\_pd\_is\_0

];

P = P\_normalized(obj.pd\_row(pd)) / sum(P\_normalized);

end

end

%

% private functions

%

methods(Access = private)

%

% Compute P(x|pd) probability from 1-D Gaussian

%

function P = p\_x\_given\_pd(obj, x, pd, cpd)

parameters = cpd(obj.pd\_row(pd),:);

if x ~= 0

m = parameters(1);

v = parameters(2);

P = (1/(sqrt(2\*pi\*v))) \* exp(-(1/(2\*v))\*(x-m)^2);

else

% when x is not given (x == 0)

% the network imply P(x|pd) will sum up to 1

P = 1;

end

end

%

%

%

%

% Lookup P(pd) from CPT

%

function P = p\_pd(obj, pd)

P = obj.P\_Pd(obj.pd\_row(pd));

end

%

% Map Pd values {1 0} to row [1 2] in the CPT

%

function r = pd\_row(~, pd)

pds = [1 0];

rows = [1 2];

r = rows(pds == pd);

end

end

end

## Confusion Matrices

### Dataset 1001

|  |  |  |
| --- | --- | --- |
|  | True + | True − |
| Predicted + | 10/17 | 11/75 |
| Predicted − | 7/17 | 64/75 |

Prediction accuracy = 80.4%

### Dataset 1004

|  |  |  |
| --- | --- | --- |
|  | True + | True − |
| Predicted + | 6/15 | 6/57 |
| Predicted − | 9/15 | 51/57 |

Prediction accuracy = 79.1%

# BN#4

Our final network is a continuous Bayes network where breathing rate, heart rate, and skin temperature are given two possible causes: the state of drunkenness and the state of ambulation.

## Learning program

Our program learns the parameters by computing the conditional probabilities through Maximum Likelihood.

### Formulas of Maximum Likelihood

As in section 2.1.1, we will use the same formulation as for the naïve Bayes network, changing only to introduce the state of ambulation.

As in section 0, we will assume the conditional probabilities are distributed and will estimate and using the and maximum likelihood estimators from section 1.

Once again, we must define a conditioned set:

Finally, each conditional probability can be specified as:

Or more specifically,

Because and are still discrete, like in section 3.1.1, we estimate it as was done in section 1

### Program code

#### BN4.m

%% Learn parameters of the CPDs

P\_Pd = [sum(Pd == 1)/length(Pd); sum(Pd == 0)/length(Pd)];

P\_Xa = [sum(Xa == 1)/length(Xa); sum(Xa == 2)/length(Xa); sum(Xa == 3)/length(Xa)];

P\_Xb\_given\_Pd\_and\_Xa = BN4.CPD(Xb, Pd, Xa);

P\_Xh\_given\_Pd\_and\_Xa = BN4.CPD(Xh, Pd, Xa);

P\_Xt\_given\_Pd\_and\_Xa = BN4.CPD(Xt, Pd, Xa);

#### BN4.CPD.m

function [ cpd ] = CPD ( x , Pd, Xa )

% BN4.CPD - construct CPD for a random variable in BN 4

% learn CPD parameters using Maximum Likelihood estimator

%

% Input:

% x - random variable.

% Pd - evidence variable 'drink'. Domain = {1 0}

% Xa - evidence variable 'Ambulation status'. Domain = {1 2 3}

%

% Output:

% cpd - conditional probability distribution of the random variable given the

% evidence drink and ambulation status.

% eg.

%

% Xb

% |Stationary | Slow | Fast |

% Pd -------------------------------------

% 1 | m1 | var1 | .. | .. | .. | .. |

% 0 | m2 | var2 | .. | .. | .. | .. |

% Note:

% The elements/rows of evidences must correspond to x's

%

xa\_domain = [1 2 3];

xa\_domain\_size = length(xa\_domain);

cpd = zeros(2, 2 \* xa\_domain\_size);

for k = 1:xa\_domain\_size

xa\_is\_k = Xa == k;

col = [1 2] + (k - 1) \* 2;

cpd(:, col) = mle( x(xa\_is\_k), Pd(xa\_is\_k) );

end

end

### Parameters learned

#### Dataset 1001

***P(Pd)***

|  |  |
| --- | --- |
| **+** | 0.8152 |
| **-** | 0.8152 |

***P(Xa)***

|  |  |
| --- | --- |
| **Stationary** | 0.8152 |
| **Slow** | 0.1087 |
| **Fast** | 0.0761 |

***P(Xb|Pd, Xa)***

|  | **Stationary** | | **Slow** | | **Fast** | |
| --- | --- | --- | --- | --- | --- | --- |
| **Drink** | **m** | **var** | **m** | **var** | **m** | **var** |
| **1** | 9.3438 | 6.6787 | 9 | 0 | 0 | 0 |
| **0** | 10.4237 | 10.8628 | 11.8889 | 17.6543 | 16.2857 | 61.9184 |

***P(Xh|Pd, Xa)***

|  | **Stationary** | | **Slow** | | **Fast** | |
| --- | --- | --- | --- | --- | --- | --- |
| **Drink** | **m** | **var** | **m** | **var** | **m** | **var** |
| **1** | 9.3438 | 6.6787 | 9 | 0 | 0 | 0 |
| **0** | 10.4237 | 10.8628 | 11.8889 | 17.6543 | 16.2857 | 61.9184 |

***P(Xt|Pd, Xa)***

|  | **Stationary** | | **Slow** | | **Fast** | |
| --- | --- | --- | --- | --- | --- | --- |
| **Drink** | **m** | **var** | **m** | **var** | **m** | **var** |
| **1** | 78.3438 | 8.1787 | 80 | 0 | 0 | 0 |
| **0** | 80.0678 | 10.1649 | 81.8889 | 16.2654 | 87 | 185.6429 |

#### Dataset 1004

***P(Pd)***

|  |  |
| --- | --- |
| **+** | 0.2083 |
| **-** | 0.7917 |

***P(Xa)***

|  |  |
| --- | --- |
| **Stationary** | 0.8194 |
| **Slow** | 0.1389 |
| **Fast** | 0.0417 |

***P(Xb|Pd, Xa)***

|  | **Stationary** | | **Slow** | | **Fast** | |
| --- | --- | --- | --- | --- | --- | --- |
| **Drink** | **m** | **var** | **m** | **var** | **m** | **var** |
| **1** | 14.3667 | 22.2156 | 0 | 0 | 0 | 0 |
| **0** | 15.6136 | 7.6121 | 17.25 | 3.3625 | 15.5 | 2.1667 |

***P(Xh|Pd, Xa)***

|  | **Stationary** | | **Slow** | | **Fast** | |
| --- | --- | --- | --- | --- | --- | --- |
| **Drink** | **m** | **var** | **m** | **var** | **m** | **var** |
| **1** | 127.9 | 41.6733 | 0 | 0 | 0 | 0 |
| **0** | 133.3068 | 91.4684 | 143.4 | 454.59 | 136.6667 | 189.5556 |

***P(Xt|Pd, Xa)***

|  | **Stationary** | | **Slow** | | **Fast** | |
| --- | --- | --- | --- | --- | --- | --- |
| **Drink** | **m** | **var** | **m** | **var** | **m** | **var** |
| **1** | 35.6867 | 0.2585 | 0 | 0 | 0 | 0 |
| **0** | 35.9795 | 0.1236 | 36.025 | 0.0391 | 36 | 0 |

## Prediction program

### Formulas

Just as section 1 used the formulation from 2.1.1, in this section we will use the formulation derived in section 2.2.1.

As noted in 3.2.1, each one of these probabilities can be calculated with its respective

### Program code

#### BN4.m

evidence = [Xb Xh Xt Xa];

data\_size = size(evidence, 1);

prediction = zeros(data\_size, 1);

% prediction

model = BN4.model(P\_Pd, P\_Xa, P\_Xb\_given\_Pd\_and\_Xa, P\_Xh\_given\_Pd\_and\_Xa, P\_Xt\_given\_Pd\_and\_Xa);

for i = 1:data\_size

prediction(i, 1) = model.predict(1, evidence(i, :));

end

#### BN4.model.m

classdef model

% MODEL - BN4 prediction

% Compute probability of 'Pd' given the evidences

properties

P\_Pd;

P\_Xa;

P\_Xb\_given\_Pd\_and\_Xa;

P\_Xh\_given\_Pd\_and\_Xa;

P\_Xt\_given\_Pd\_and\_Xa;

end

methods

%

% Initializer

%

function obj = model( ...

P\_Pd, P\_Xa, P\_Xb\_given\_Pd\_and\_Xa, P\_Xh\_given\_Pd\_and\_Xa, P\_Xt\_given\_Pd\_and\_Xa)

obj.P\_Pd = P\_Pd;

obj.P\_Xa = P\_Xa;

obj.P\_Xb\_given\_Pd\_and\_Xa = P\_Xb\_given\_Pd\_and\_Xa;

obj.P\_Xh\_given\_Pd\_and\_Xa = P\_Xh\_given\_Pd\_and\_Xa;

obj.P\_Xt\_given\_Pd\_and\_Xa = P\_Xt\_given\_Pd\_and\_Xa;

end

%

% Compute P(pd|e)

%

function P = predict(obj, pd, e)

p\_pd\_is\_1 = obj.p\_pd(1);

p\_e\_given\_pd\_is\_1 = obj.p\_e\_given\_pd(1, e);

p\_pd\_is\_0 = obj.p\_pd(0);

p\_e\_given\_pd\_is\_0 = obj.p\_e\_given\_pd(0, e);

P\_normalized = [

p\_pd\_is\_1 \* p\_e\_given\_pd\_is\_1

p\_pd\_is\_0 \* p\_e\_given\_pd\_is\_0

];

P = P\_normalized(obj.pd\_row(pd)) / sum(P\_normalized);

end

end

%

% private functions

%

methods(Access = private)

%

% Compute P(e|pd)

%

function P = p\_e\_given\_pd(obj, pd, e)

xa = e(4);

% xa = 0 (represent xa is not given)

if xa == 0

xa = [1 2 3];

end

p = zeros(1, length(xa));

% sum over all xa

for i = 1:length(xa)

p(i) = obj.P\_Xa(xa) \* obj.p\_x\_given\_pd\_and\_xa(e(1), pd, xa, obj.P\_Xb\_given\_Pd\_and\_Xa) \* obj.p\_x\_given\_pd\_and\_xa(e(2), pd, xa, obj.P\_Xh\_given\_Pd\_and\_Xa) \* obj.p\_x\_given\_pd\_and\_xa(e(3), pd, xa, obj.P\_Xt\_given\_Pd\_and\_Xa);

end

P = sum(p);

end

%

% Compute P(x|pd,xa) probability from 1-D Gaussian

%

function P = p\_x\_given\_pd\_and\_xa(obj, x, pd, xa, cpd)

parameters = cpd(obj.pd\_row(pd), obj.x\_column(xa));

if x ~= 0

m = parameters(1);

v = parameters(2);

P = (1/(sqrt(2\*pi\*v))) \* exp(-(1/(2\*v))\*(x-m)^2);

else

% when x is not given (x == 0)

% the network imply P(x|pd) will sum up to 1

P = 1;

end;

end

%

% Lookup P(pd) from CPT

%

function P = p\_pd(obj, pd)

P = obj.P\_Pd(obj.pd\_row(pd));

end

%

% Map Pd values {1 0} to row [1 2] in the CPT

%

function r = pd\_row(~, pd)

pds = [1 0];

rows = [1 2];

r = rows(pds == pd);

end

%

% Find the columns that corresponding to

% xa = {1, 2, 3}

% in the CPD

%

function c = x\_column(~, xa)

c = [1 2] + (xa - 1) \* 2;

end

end

end

## Confusion Matrices

### Dataset 1001

|  |  |  |
| --- | --- | --- |
|  | True + | True − |
| Predicted + | 1/17 | 0/75 |
| Predicted − | 16/17 | 75/75 |

Prediction accuracy = 82.6%

### Dataset 1004

|  |  |  |
| --- | --- | --- |
|  | True + | True − |
| Predicted + | 6/15 | 5/57 |
| Predicted − | 9/15 | 52/57 |

Prediction accuracy = 80.5%

# Conclusions

Four Bayesian networks were implemented: two discrete, two continuous. Of each, the first and third were naïve Bayes network and the latter was a more complex two-cause model. The models correctly process the data and all produce similar results. This enables them to all work tougher,

It is of note that the conditional probabilities were calculated for all four networks, and an appreciable number of them were found to be equal to zero.

When comparing the accuracies, we found that the use of more complex models, or the use of continuous data as opposed to discrete data, actually do not improve the accuracy. We suspect this is due to the zero-probabilities. Future work may test this by adding a smoothener to the inputs.