Risk and Decision-Making for Data Science and AI

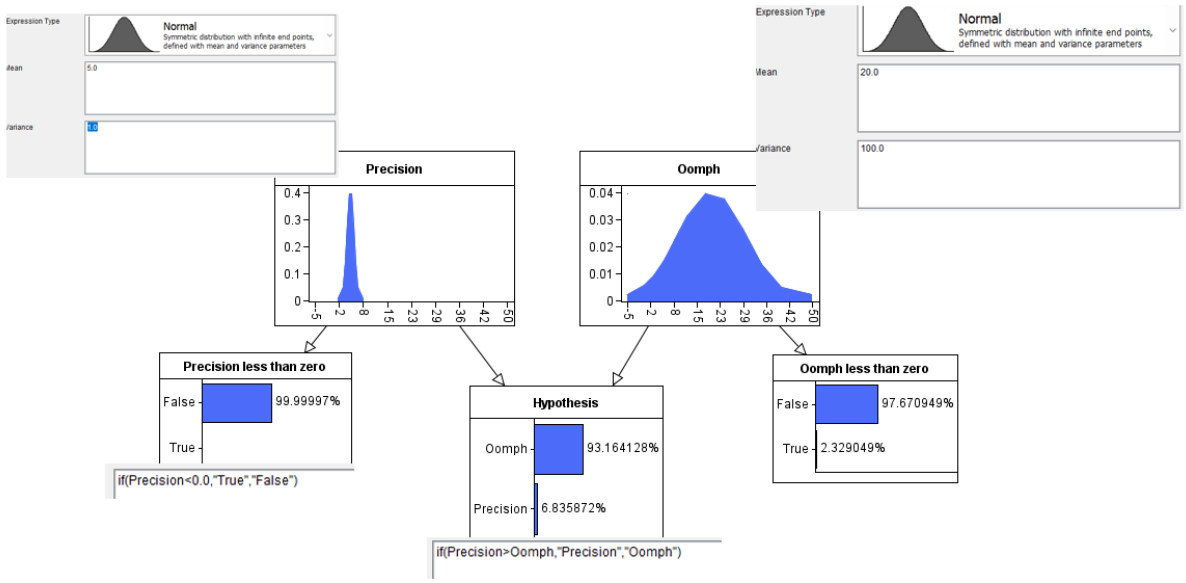
Week 5 Lab 4 Answers

Question 1:

a) Wrong: this interpretation incorrectly suggests a level of certainty about the true parameter within a specific interval. Confidence intervals reflect the uncertainty around the estimation, not the probability of the true parameter being within a specific interval.
b) Correct
c) Correct
d) Wrong: failing to reject the null hypothesis does not necessarily mean that there is absolutely no effect. It simply means that there is not enough evidence to conclude that there is an effect based on the data at hand. So, while the statement is generally correct in the context of statistical testing, it's important to avoid overinterpreting it to mean there is absolutely no effect whatsoever. It's more accurate to say that the evidence for an effect is inconclusive based on the given data and statistical analysis.

Question 2:

Using Agena:



The hypothesis is approximately 93% in favor of Oomph over Precision, since only 7% of time if Precision likely to result in more weight loos than Oomph. Interestingly, we have included here two additional nodes to cover the risk that any of the drugs actually cause negative weight loos, that is, weight gain for the good reason that some may be fearful of putting on weight and might choose the drug that was less effective but also less risky. In this case someone might choose Precision since Oomph has a 2.3% chance of weight gain, where the chance of weight gain with Precision is negligible.

Using Python:

Risk and Decision-Making for Data Science and AI

Week 5 Lab 4 Answers

Importing packages needed to work with Bayesian Networks in pyAgrum.

```
from pylab import *
import matplotlib.pyplot as plt
import pyAgrum as gum
import pyAgrum.lib.notebook as gnb
```

Python

The way pyAgrum handles variables defined with probability distribution is "quasi-continuous" nodes where we define a high number for variable states and it approximates the continuous distribution. The "quasi-continuous" nodes also require a lower and an upper threshold.

Continous nodes are defined using the pyAgrum class `NumericalDiscreteVariable` - remember that discrete nodes use the class `LabelizedVariable`.

Below we define the limits for Precision and Oomph as [-75, 75], and we are using 600 states for discretisation.

Note that the number of states for these nodes increases the computation required for a network exponentially (especially if the continuous nodes have parent/child nodes).

```
min_precision, max_precision = -75, 75
min_oomph, max_oomph = -75, 75
NB=600
```

Python

Now we create the network called `hypothesis-testing`, and add two continuous nodes. First two input parameters in `NumericalDiscreteVariable` are the name and description of the node. Name and description are followed by the lower limit, upper limit, and the number of states here (instead of state names).

```
bn = gum.BayesNet("hypothesis-testing")
bn.add(gum.NumericalDiscreteVariable("Precision","Precision",min_precision, max_precision, NB))
bn.add(gum.NumericalDiscreteVariable("Oomph","Oomph",min_oomph, max_oomph, NB))
```

Python

```
1
```

We create the discrete nodes in the network, similar to how we always created discrete nodes with the name, description, and state names.

```
bn.add(gum.LabelizedVariable("PrecisionLessThanZero", "Precision less than zero",["False","True"]))
bn.add(gum.LabelizedVariable("OomphLessThanZero", "Oomph less than zero", ["False","True"]))
bn.add(gum.LabelizedVariable("Hypothesis","Hypothesis",["Oomph","Precision"]))
```

Python

```
4
```

Adding the arcs between nodes, and displaying the network.

```
bn.addArc("Precision","PrecisionLessThanZero")
bn.addArc("Oomph","OomphLessThanZero")
bn.addArc("Precision", "Hypothesis")
bn.addArc("Oomph", "Hypothesis")
gnb.showBN(bn)
```

Python

Now we need to define the probability distribution of the continuous nodes. As they both use normal distribution, we just import the `norm` function from `scipy.stats`, and we define a normalisation function to be used on the probability distribution. This function will be used when populating the continuous node "tables".

```
from scipy.stats import norm

def normalise(rv,vmin,vmax,size):
    pdf=rv.pdf(np.linspace(vmin,vmax,size))
    return (pdf/sum(pdf))
```

Python

Now we need to define the probability distribution of the continuous nodes. As they both use normal distribution, we just import the `norm` function from `scipy.stats`, and we define a normalisation function to be used on the probability distribution. This function will be used when populating the continuous node "tables".

```
from scipy.stats import norm

def normalise(rv,vmin,vmax,size):
    pdf=rv.pdf(np.linspace(vmin,vmax,size))
    return (pdf/sum(pdf))
```

Python

The Precision node has a normal distribution with the mean value of 5 and the standard deviation of 1, we define its distribution, and fill in its CPT with the normalise function.

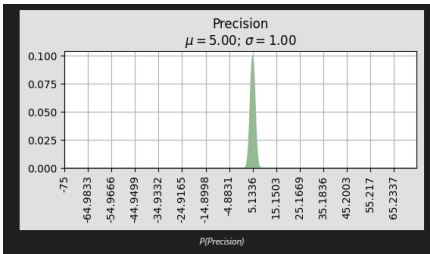
We can use the pyAgrum visualisation functions to display the node prior distribution.

```
# defining the mean and standard deviation of the distribution
mean_precision, std_precision = 5, 1

# creating a normal distribution object
dist_precision = norm(loc=mean_precision, scale=std_precision)

# filling in the CPT
bn.cpt("Precision").fillWith(normalise(dist_precision,min_precision,max_precision,NB).flatten())

# visualisation
gnb.flow.clear()
gnb.flow.add(gnb.getProba(bn.cpt("Precision")),caption="P(Precision)")
gnb.flow.display()
```

Python

Risk and Decision-Making for Data Science and AI

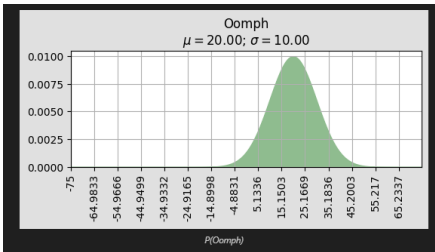
Week 5 Lab 4 Answers



The Oomph node has a normal distribution with the mean value of 20 and a standard deviation of 10. We define the node in the same way:

```
# defining the mean and standard deviation of the distribution
mean_oomph, std_oomph = 20, 10

# creating a normal distribution object
dist_oomph = norm(loc=mean_oomph, scale=std_oomph)

# filling in the CPT
bn.cpt("Oomph").fillWith(normalise(dist_oomph,min_oomph,max_oomph,NB).flatten())

# visualisation
gnb.flow.clear()
gnb.flow.add(gnb.getProba(bn.cpt("Oomph")),caption="P(Oomph)")
gnb.flow.display()
```

Python



Now we need to populate the CPTs of the discrete rule-based nodes. For this, we need to employ the logic of the if statement.

For the `PrecisionLessThanZero` node:

- Where the value of `Precision` is greater than 0, the state probabilities of `PrecisionLessThanZero` will be P(True)=0, P(False)=1.
- Where the value of `Precision` is smaller than 0, the state probabilities of `PrecisionLessThanZero` will be P(True)=1, P(False)=0.

For the `OomphLessThanZero` node:

- Where the value of `Oomph` is greater than 0, the state probabilities of `OomphLessThanZero` will be P(True)=0, P(False)=1.
- Where the value of `Oomph` is smaller than 0, the state probabilities of `OomphLessThanZero` will be P(True)=1, P(False)=0.

For the `Hypothesis` node:

- Where the value of `Precision` is greater than the value of `Oomph` , the state probabilities of `Hypothesis` should be P(Precision)=1, P(Oomoh)=0
- Where the value of `Precision` is smaller than the value of `Oomph` , the state probabilities of `Hypothesis` should be P(Precision)=0, P(Oomoh)=1

```
# creating lists of possible values of the discretised continuous nodes for the if/else rules
precision_values = linspace(min_precision,max_precision,NB)
oomph_values = linspace(min_oomph, max_oomph, NB)
```

Python

We populate the CPT of `PrecisionLessThanZero` following the above logic:

```
for i, pr in enumerate(precision_values):
    if pr>0: bn.cpt("PrecisionLessThanZero")[i] = [1, 0]
    else:    bn.cpt("PrecisionLessThanZero")[i] = [0, 1]
```

Python

The CPT of `OomphLessThanZero`:

```
for i, om in enumerate(oomph_values):
    if om>0: bn.cpt("OomphLessThanZero")[i] = [1, 0]
    else:    bn.cpt("OomphLessThanZero")[i] = [0, 1]
```

Python

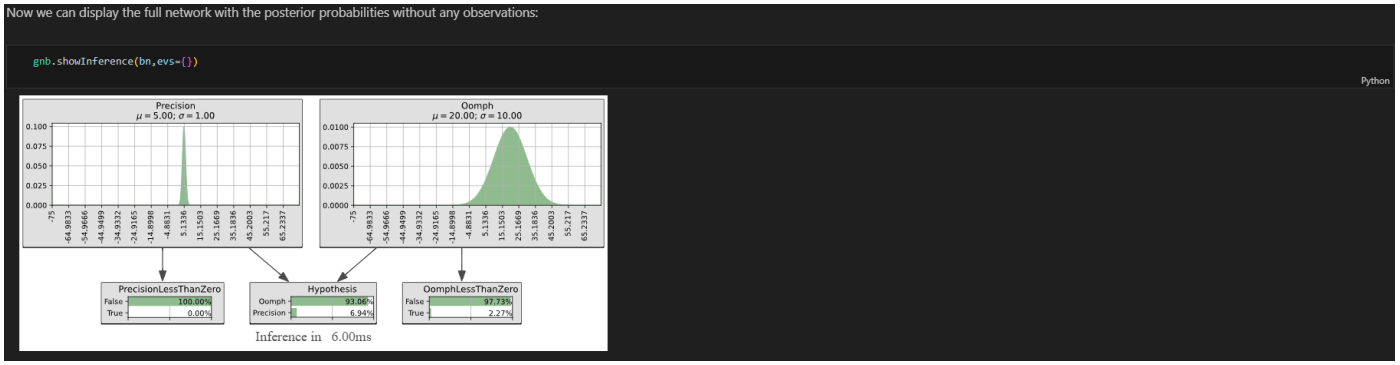
And finally, the CPT of the node `Hypothesis`:

```
for i, pr in enumerate(precision_values):
    for k, om in enumerate(oomph_values):
        if pr>om:   bn.cpt("Hypothesis")[i, k, :] = [1, 0]
        else:   bn.cpt("Hypothesis")[i, k, :] = [0, 1]
```

Python

Risk and Decision-Making for Data Science and AI

Week 5 Lab 4 Answers



Question 3:

a) MCAR. The missingness is completely random and unrelated to the participants' responses.
b) MAR. In other words, the probability of missingness depends on the values observed in the dataset.
c) MAR. The missingness is related to SES (observed variable) but not to academic performance (unobserved variable).
d) MNAR. The missingness is related to the unobserved variable (severity of depression) even after considering the observed data (other survey responses).
e) MCAR. The missingness is unrelated to the temperature values recorded at the stations.