post-operative

July 11, 2022

1 Post-operative prediction using Bayesian Networks

FAIKR module 3 project, done by Michele Calvanese, Samuele Marino

1.1 Introduction

Postoperative patient care has several components: - surveillance, — prevention of complications associated with surgical disease or other preexisting comorbidities, — specific postoperative treatment of the surgical disease and its complications. While these distinctions are purely didactic, the postoperative care merges into an active surveillance with a higher level of standardization than it would seem at first glance.

The goal of this project is to determine where patients in a postoperative recovery area should be sent to next. Because hypothermia is a significant concern after surgery, the attributes correspond roughly to body temperature measurements.

Number of Instances: 90

Number of Attributes: 9 including the decision (class attribute)

Attribute Information:

1. L-CORE (patient's internal temperature in C):

   `high (> 37), mid (>= 36 and <= 37), low (< 36)`

2. L-SURF (patient's surface temperature in C):

   `high (> 36.5), mid (>= 36.5 and <= 35), low (< 35)`

3. L-O2 (oxygen saturation in %):

   ```
   excellent (>= 98), good (>= 90 and < 98),
   fair (>= 80 and < 90), poor (< 80)
   ```

4. L-BP (last measurement of blood pressure):

   `high (> 130/90), mid (<= 130/90 and >= 90/70), low (< 90/70)`

5. SURF-STBL (stability of patient's surface temperature):

   `stable, mod-stable, unstable`

6. CORE-STBL (stability of patient's core temperature)

   `stable, mod-stable, unstable`

7. BP-STBL (stability of patient's blood pressure)

```
stable, mod-stable, unstable
```

8. COMFORT (patient's perceived comfort at discharge, measured as

```
an integer between 0 and 20)
```

9. ADM-DECS (discharge decision):

```
I (patient sent to Intensive Care Unit),
S (patient prepared to go home),
A (patient sent to general hospital floor)
```

Dataset link.

[ ]:
```
import pandas as pd
import networkx as nx
import pylab as plt
from pgmpy.models import BayesianNetwork, BayesianModel
from pgmpy.inference import VariableElimination, ApproxInference
from pgmpy.sampling import BayesianModelSampling
from pgmpy.estimators import HillClimbSearch, BDsScore, K2Score, BicScore,␣
↪BDeuScore, MaximumLikelihoodEstimator, BayesianEstimator
import matplotlib.pyplot as plt
from networkx.drawing.nx_pydot import graphviz_layout
```

1.2 Data cleaning and preparation

All attributes were used for making the Bayesian Network model. Before creating the model, all the attributes need to be discretized, as the pgmpy library only works with discrete variables.

[ ]:
```
path = "./Dataset/post-operative.data"
attributes = ['L-CORE', 'L-SURF', 'L-O2', 'L-BP', 'SURF-STBL', 'CORE-STBL',␣
↪'BP-STBL', 'COMFORT', 'ADM-DECS']
data = pd.read_csv(path, sep=',', header=None, names=attributes)
data.head()
```

[ ]:

	L-CORE	L-SURF	L-O2	L-BP	SURF-STBL	CORE-STBL	BP-STBL	COMFORT	ADM-DECS
0	mid	low	excellent	mid	stable	stable	stable	15	A
1	mid	high	excellent	high	stable	stable	stable	10	S
2	high	low	excellent	high	stable	stable	mod-stable	10	A
3	mid	low	good	high	stable	unstable	mod-stable	15	A
4	mid	mid	excellent	high	stable	stable	stable	10	A

Verify the value of all attributes

[ ]:
```
for attribute in data.columns:
    print(attribute, ':', data[attribute].unique())
```

```
L-CORE : ['mid' 'high' 'low']
L-SURF : ['low' 'high' 'mid']
L-O2 : ['excellent' 'good']
L-BP : ['mid' 'high' 'low']
SURF-STBL : ['stable' 'unstable']
CORE-STBL : ['stable' 'unstable' 'mod-stable']
BP-STBL : ['stable' 'mod-stable' 'unstable']
COMFORT : ['15' '10' '05' '07' '?']
ADM-DECS : ['A' 'S' 'A ' 'I']
```

In attribute `ADM-DECS` there are same incorrect values. So i replace the 'A' values with 'A'

[ ]:
```
data = data.replace({'A ': 'A'})
data['ADM-DECS'].unique()
```

[ ]:
```
array(['A', 'S', 'I'], dtype=object)
```

Also in `COMFORT` there are inconsistent value, for thise reason I remove all the row where '?' is present

[ ]:
```
data = data.drop(data[data.COMFORT == '?'].index)

data['COMFORT'] = pd.cut(x=data['COMFORT'].astype(int),
                         bins=[0, 5, 10, 15, 20],
                         labels=["low_range", "normal_range", "medium_range",␣
 ↪"high_range"])

data['COMFORT'].unique()
```

[ ]:
```
['medium_range', 'normal_range', 'low_range']
Categories (4, object): ['low_range' < 'normal_range' < 'medium_range' <
'high_range']
```

Before modifying the variables, dataset is checked for the presence of N/A values.

[ ]:
```
data.isnull().sum()
```

[ ]:
```
L-CORE       0
L-SURF       0
L-O2         0
L-BP         0
SURF-STBL    0
CORE-STBL    0
BP-STBL      0
```

```
COMFORT      0
ADM-DECS     0
dtype: int64
```

So there is no N/A values

Show the number of samples after the cline up

[ ]:
```
data.shape
```

[ ]:
```
(87, 9)
```

[ ]:
```
data.head()
```

[ ]:
```
  L-CORE L-SURF       L-O2  L-BP SURF-STBL CORE-STBL     BP-STBL  \
0    mid    low  excellent   mid    stable    stable      stable
1    mid   high  excellent  high    stable    stable      stable
2   high    low  excellent  high    stable    stable  mod-stable
3    mid    low       good  high    stable  unstable  mod-stable
4    mid    mid  excellent  high    stable    stable      stable

         COMFORT ADM-DECS
0  medium_range        A
1  normal_range        S
2  normal_range        A
3  medium_range        A
4  normal_range        A
```

1.3 Bayesian Network

1.3.1 Learning the structure of the network

One option for defining the structure of the network would be to manually add the nodes and connections between them. Knowing the connections between nodes requires having a sufficient knowledge about the field of application in order to be able to define conditional independence assertions.

Luckily, pgmpy library offers many ways for learning a structure for discrete, fully observed networks. Given a set of data samples, the algorithm estimates a directed acyclic graph that captures dependencies between the variables. We will use Score-based structure estimation that learns the model as an optimization task, and which lets the programmer choose a search strategy and a scoring function which will be used.

Some of the available search strategies are:

- Exhaustive search
- Hill Climb search

The search space is super-exponential in the number of variables, which is the reason why Exhaustive search wasn't chosen for this particular project. However, for a very small number of nodes, it is guaranteed to find the best-scoring graph. When networks are bigger it is better to use Hill

Climb search, which implements a greedy local search starting usually from a disconnected graph. In every iteration it makes one change on edges of the graph that maximally increases the score.

However, this approach often leads to inconsistent results: the node ADM-DECS may have exiting edges (which we do not want, because this is the attribute we want to predict), or multiple disconnected graphs may be found. We therefore remove all the edges that have ADM-DECS as parent and the ones that have Age as child. In addition, according to scientific medical sources [link1, link2, link3], we force the learning algorithm to include the following edges in the network: ('BP-STBL', 'ADM-DECS'), ('L-O2', 'ADM-DECS'), ('SURF-STBL', 'ADM-DECS').

[ ]:
```
black_list = [('ADM-DECS', attribute) for attribute in data.columns[:-1]]
fixed_attributes = ['BP-STBL', 'L-O2', 'SURF-STBL']
fixed_edges = [(attribute, 'ADM-DECS') for attribute in fixed_attributes]
# fixed_edges += [('L-SURF', 'CORE-STBL')]

#learning the stucutre of the network
est = HillClimbSearch(data, use_cache=False)
best_model = est.estimate(scoring_method=K2Score(data),
                          black_list=black_list,
                          fixed_edges=fixed_edges
                          )
```

```
  0%|          | 6/1000000 [00:01<56:15:17,  4.94it/s]
```

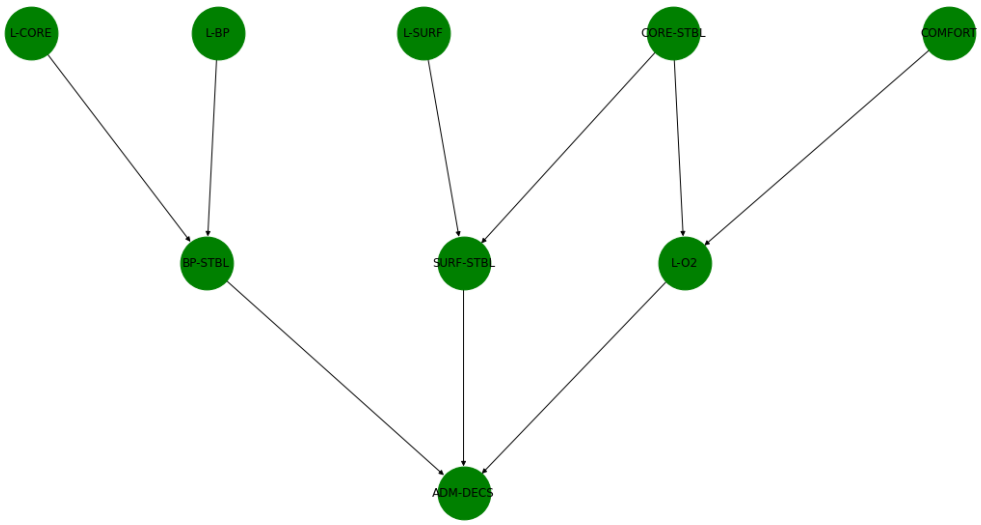
[ ]:
```
#the estimated model returns a DAG at a (local) score maximum
edges = list(best_model.edges())
model = BayesianNetwork(edges)

#nodes of the model
model.nodes
```

[ ]:
```
NodeView(('L-CORE', 'BP-STBL', 'L-SURF', 'SURF-STBL', 'L-O2', 'ADM-DECS',
'L-BP', 'CORE-STBL', 'COMFORT'))
```

[ ]:
```
#edges of the model
model.edges
```

[ ]:
```
OutEdgeView([('L-CORE', 'BP-STBL'), ('BP-STBL', 'ADM-DECS'), ('L-SURF', 'SURF-
STBL'), ('SURF-STBL', 'ADM-DECS'), ('L-O2', 'ADM-DECS'), ('L-BP', 'BP-STBL'),
('CORE-STBL', 'SURF-STBL'), ('CORE-STBL', 'L-O2'), ('COMFORT', 'L-O2')])
```

[ ]:
```
pos = graphviz_layout(model, prog="dot")
plt.figure(figsize=(16, 8))
nx.draw(model, with_labels=True, pos=pos, node_size=3000, node_color="green")
plt.savefig('model.png')
plt.show()
```



1.3.2 Learning the parameters and analyzing of the network

As mentioned before, pgmpy currently supports parameter learning only for networks with discrete nodes. There are 2 available methods for determining the values of the conditional probability distributions:

- Maximum Likelihood estimation.

  `This method uses relative frequencies for estimating conditional probabilities. However, in case of small datasets it is prone to overfitting.`

- Bayesian estimation.

  `On the other hand, Bayesian estimator assumes prior CPDs on variables and then updates them using state counts from observed data.`

Because our dataset has only 87 samples, after cleaning, we use **Bayesian estimation**

[ ]:
```
model.fit(data=data, estimator=BayesianEstimator, prior_type="BDeu")

for cpd in model.get_cpds():
    print(f'CPT of {cpd.variable}:')
    print(cpd, 2*'\n', 80*"=", '\n')
```

```
CPT of L-CORE:
+--------------+----------+
| L-CORE(high) | 0.148551 |
+--------------+----------+
| L-CORE(low)  | 0.213768 |
+--------------+----------+
| L-CORE(mid)  | 0.637681 |
```

```
+-------------+----------+
```

```
================================================================================
```

CPT of BP-STBL:

```
+---------------------+-----+--------------------+
| L-BP                | … | L-BP(mid)          |
+---------------------+-----+--------------------+
| L-CORE              | … | L-CORE(mid)        |
+---------------------+-----+--------------------+
| BP-STBL(mod-stable) | … | 0.19132149901380668 |
+---------------------+-----+--------------------+
| BP-STBL(stable)     | … | 0.5374753451676528  |
+---------------------+-----+--------------------+
| BP-STBL(unstable)   | … | 0.27120315581854043 |
+---------------------+-----+--------------------+
```

```
================================================================================
```

CPT of L-SURF:

```
+-------------+----------+
| L-SURF(high) | 0.192029 |
+-------------+----------+
| L-SURF(low)  | 0.278986 |
+-------------+----------+
| L-SURF(mid)  | 0.528986 |
+-------------+----------+
```

```
================================================================================
```

CPT of SURF-STBL:

```
+---------------------+-----+--------------------+
| CORE-STBL           | … | CORE-STBL(unstable) |
+---------------------+-----+--------------------+
| L-SURF              | … | L-SURF(mid)        |
+---------------------+-----+--------------------+
| SURF-STBL(stable)   | … | 0.07812500000000001 |
+---------------------+-----+--------------------+
| SURF-STBL(unstable) | … | 0.921875           |
+---------------------+-----+--------------------+
```

```
================================================================================
```

CPT of L-O2:

```
+-----------------+-----+----------------------+
| COMFORT         | … | COMFORT(normal_range) |
+-----------------+-----+----------------------+
| CORE-STBL       | … | CORE-STBL(unstable)   |
```

L-O2(excellent)	…	0.921875
L-O2(good)	…	0.07812500000000001

==========

CPT of ADM-DECS:

BP-STBL	…	BP-STBL(unstable)
L-O2	…	L-O2(good)
SURF-STBL	…	SURF-STBL(unstable)
ADM-DECS(A)	…	0.6450216450216449
ADM-DECS(I)	…	0.02164502164502164
ADM-DECS(S)	…	0.33333333333333326

==========

CPT of L-BP:

L-BP(high)	0.322464
L-BP(low)	0.0507246
L-BP(mid)	0.626812

==========

CPT of CORE-STBL:

CORE-STBL(mod-stable)	0.0289855
CORE-STBL(stable)	0.898551
CORE-STBL(unstable)	0.0724638

==========

CPT of COMFORT:

```
+----------------------+-----------+
| COMFORT(low_range)    | 0.0398551 |
+----------------------+-----------+
| COMFORT(medium_range) | 0.224638  |
+----------------------+-----------+
| COMFORT(normal_range) | 0.735507  |
+----------------------+-----------+

================================================================================
```

Checking if it has errors

[ ]:
```
model.check_model()
```

[ ]:
```
True
```

Cardinality of all model nodes

[ ]:
```
model.get_cardinality()
```

[ ]:
```
defaultdict(int,
            {'L-CORE': 3,
             'BP-STBL': 3,
             'L-SURF': 3,
             'SURF-STBL': 2,
             'L-O2': 2,
             'ADM-DECS': 3,
             'L-BP': 3,
             'CORE-STBL': 3,
             'COMFORT': 3})
```

Local independecies of a signle node, namely "SURF-STBL".

In this simplification of a real case structure stability of patient's blood pressure(BP-STBL), last measureof blod pressure(L-BP), oxygen saturation(L-O2), patient's perceived comfort(COMFORT) and patient's internal temperature(L-CORE) don't influence directly the stability of patient's surface temperature(SURF-STBL) given their influence on other factors such as stability core temperature(CORE-STBL) and patient's surface temperature(L-SURF).

[ ]:
```
model.local_independencies("SURF-STBL")
```

[ ]:
```
(SURF-STBL  BP-STBL, L-BP, L-O2, COMFORT, L-CORE | CORE-STBL, L-SURF)
```

Checking d-separation between variables with and without evidence

Two sets of nodes X, Y are d-separated given Z if there is no active trail between any X X and Y Y given Z

[ ]:
```
print(model.is_dconnected("COMFORT", "L-O2"))
print(model.is_dconnected("BP-STBL", "SURF-STBL", observed=["L-O2"]))
```

```
True
False
```

Checking the markov blanket for ADM-DECS and L-O2

[ ]:
```
print("Markov blanket for variable:")
print("ADM-DECS: ", model.get_markov_blanket("ADM-DECS"))
print("L-O2: ", model.get_markov_blanket("L-O2"))
```

```
Markov blanket for variable:
ADM-DECS:  ['L-O2', 'SURF-STBL', 'BP-STBL']
L-O2:  ['BP-STBL', 'SURF-STBL', 'CORE-STBL', 'COMFORT', 'ADM-DECS']
```

1.4 Inference

1.4.1 Exact inference

Pgmpy library offers two ways of doing exact inference, *Variable Elimination method*, presented bellow, and *Belief Propagation method*. The basic concept of Variable Elimination is in a way similar to doing marginalization over Joint Distribution, except that it doesn't compute the complete Joint Distribution but marginalizes just over factors that involve the variable that is being eliminated.

[ ]:
```
infer = VariableElimination(model)
```

[ ]:
```
print("Probability of discharge decision:\n")
print(infer.query(["ADM-DECS"]))
```

```
Probability of discharge decision:


Finding Elimination Order: : 100%|      | 8/8 [00:00<00:00, 2068.71it/s]
Eliminating: COMFORT: 100%|      | 8/8 [00:00<00:00, 397.47it/s]
+-------------+-----------------+
| ADM-DECS    |   phi(ADM-DECS) |
+=============+=================+
| ADM-DECS(A) |          0.6878 |
+-------------+-----------------+
| ADM-DECS(I) |          0.0311 |
+-------------+-----------------+
| ADM-DECS(S) |          0.2811 |
+-------------+-----------------+
```

Causal inference (prediction):

What is the discharge decision given patient's internal temperature high and oxygen saturation excellent?

[ ]:
```
print(infer.query(["ADM-DECS"], {'L-CORE': 'high', 'L-O2': 'excellent'}))
```

```
Finding Elimination Order: : 100%|      | 6/6 [00:00<00:00, 1661.00it/s]
Eliminating: L-SURF: 100%|      | 6/6 [00:00<00:00, 383.74it/s]
```

```
+-------------+-----------------+
| ADM-DECS    |   phi(ADM-DECS) |
+=============+=================+
| ADM-DECS(A) |          0.7137 |
+-------------+-----------------+
| ADM-DECS(I) |          0.0250 |
+-------------+-----------------+
| ADM-DECS(S) |          0.2613 |
+-------------+-----------------+
```

Evidential inference (explaination):

Explaining the prob of stability of patient's blood pressure given that the patient was sent to Intensive Care Unit

[ ]:
```
print(infer.query(["BP-STBL"], evidence={"ADM-DECS": 'I'}))
```

```
Finding Elimination Order: : 100%|      | 7/7 [00:00<00:00, 2149.67it/s]
Eliminating: L-SURF: 100%|      | 7/7 [00:00<00:00, 490.67it/s]
```

```
+--------------------+----------------+
| BP-STBL            |   phi(BP-STBL) |
+====================+================+
| BP-STBL(mod-stable) |         0.2140 |
+--------------------+----------------+
| BP-STBL(stable)    |         0.1953 |
+--------------------+----------------+
| BP-STBL(unstable)  |         0.5907 |
+--------------------+----------------+
```

Intercausal inference (explaining away):

Why the patient was sent to general hospital floor given the fact that he had surface temperature unstable and blood pressure on average stable ?

[ ]:
```
print(infer.query(["L-O2"], evidence={"ADM-DECS": 'A', "SURF-STBL": 'unstable',␣
↪"BP-STBL": 'mod-stable'}))
```

```
Finding Elimination Order: : 100%|      | 3/3 [00:00<00:00, 1157.05it/s]
Eliminating: CORE-STBL: 100%|      | 3/3 [00:00<00:00, 469.42it/s]
```

```
+----------------+-------------+
| L-O2           |   phi(L-O2) |
+================+=============+
```

```
| L-O2(excellent) |      0.4297 |
+-----------------+-------------+
| L-O2(good)      |      0.5703 |
+-----------------+-------------+
```

Inferences using hard evidence

[ ]:
```
print(infer.query(["ADM-DECS"], evidence={"L-O2": 'excellent', "SURF-STBL":␣
↪'stable', "BP-STBL": 'stable'}))
```

```
Finding Elimination Order: : : 0it [00:00, ?it/s]
0it [00:00, ?it/s]
```

```
+-------------+-----------------+
| ADM-DECS    |   phi(ADM-DECS) |
+=============+=================+
| ADM-DECS(A) |          0.7360 |
+-------------+-----------------+
| ADM-DECS(I) |          0.0112 |
+-------------+-----------------+
| ADM-DECS(S) |          0.2528 |
+-------------+-----------------+
```

The joint probability:

What is the probability of patient's oxygen saturation and surface temperature given that his core temperature is unstable?

[ ]:
```
print(infer.query(['L-O2', 'SURF-STBL'], {'CORE-STBL': 'unstable'}))
```

```
Finding Elimination Order: : 100%|      | 2/2 [00:00<00:00, 1396.24it/s]
Eliminating: L-SURF: 100%|      | 2/2 [00:00<00:00, 448.83it/s]
```

```
+-----------------+--------------------+-----------------------+
| L-O2            | SURF-STBL          |   phi(L-O2,SURF-STBL) |
+=================+====================+=======================+
| L-O2(excellent) | SURF-STBL(stable)  |                0.2000 |
+-----------------+--------------------+-----------------------+
| L-O2(excellent) | SURF-STBL(unstable) |                0.5224 |
+-----------------+--------------------+-----------------------+
| L-O2(good)      | SURF-STBL(stable)  |                0.0769 |
+-----------------+--------------------+-----------------------+
| L-O2(good)      | SURF-STBL(unstable) |                0.2008 |
+-----------------+--------------------+-----------------------+
```

1.4.2 Approximate Inference

Approximate Inference in pgmpy is implemented with a number of different sampling methods, in particular in this section the main focus is on the ones seen during the course. We will be applying approximate inference to show convergence to a certain probability as we increase the number of samples.

[ ]:
```
infer = ApproxInference(model)
inference = BayesianModelSampling(model)
```

Showing the convergence increasing number of samples Approximate probability with a low number of samples

[ ]:
```
print(infer.query(variables=["ADM-DECS"], n_samples=10))
```

```
Generating for node: CORE-STBL:   0%|          | 0/9 [00:00<?,
?it/s]/home/sam/anaconda3/envs/FAIKR3/lib/python3.9/site-
packages/pgmpy/utils/mathext.py:83: UserWarning: Probability values don't
exactly sum to 1. Differ by: 1.1102230246251565e-16. Adjusting values.
  warn(
Generating for node: ADM-DECS: 100%|     | 9/9 [00:00<00:00, 182.24it/s]
```

```
+-------------+-----------------+
| ADM-DECS    |   phi(ADM-DECS) |
+=============+=================+
| ADM-DECS(A) |          0.8000 |
+-------------+-----------------+
| ADM-DECS(S) |          0.2000 |
+-------------+-----------------+
```

Approximate probability with an high number of samples

[ ]:
```
print(infer.query(variables=["ADM-DECS"], n_samples=50))
```

```
Generating for node: CORE-STBL:   0%|          | 0/9 [00:00<?,
?it/s]/home/sam/anaconda3/envs/FAIKR3/lib/python3.9/site-
packages/pgmpy/utils/mathext.py:83: UserWarning: Probability values don't
exactly sum to 1. Differ by: 1.1102230246251565e-16. Adjusting values.
  warn(
Generating for node: ADM-DECS: 100%|     | 9/9 [00:00<00:00, 223.56it/s]
```

```
+-------------+-----------------+
| ADM-DECS    |   phi(ADM-DECS) |
+=============+=================+
| ADM-DECS(A) |          0.6200 |
+-------------+-----------------+
| ADM-DECS(I) |          0.0200 |
+-------------+-----------------+
```

```
| ADM-DECS(S) |          0.3600 |
+-------------+-----------------+
```

True probability

[ ]:
```
print(VariableElimination(model).query(["ADM-DECS"]))
```

```
Finding Elimination Order: : 100%|      | 8/8 [00:00<00:00, 4145.59it/s]
Eliminating: COMFORT: 100%|      | 8/8 [00:00<00:00, 482.74it/s]
```

```
+-------------+-----------------+
| ADM-DECS    |   phi(ADM-DECS) |
+=============+=================+
| ADM-DECS(A) |          0.6878 |
+-------------+-----------------+
| ADM-DECS(I) |          0.0311 |
+-------------+-----------------+
| ADM-DECS(S) |          0.2811 |
+-------------+-----------------+
```

The convergence of discharge decision given patient's perceived confort 'medium_range' and internal temperature 'low'

Approximate probability with a low number of samples

[ ]:
```
print(infer.query(variables=["ADM-DECS"], evidence={'COMFORT': 'normal_range',␣
↪'L-CORE': 'low'}, n_samples=10))
```

```
  0%|          | 0/10 [00:00<?,
?it/s]/home/sam/anaconda3/envs/FAIKR3/lib/python3.9/site-
packages/pgmpy/utils/mathext.py:83: UserWarning: Probability values don't
exactly sum to 1. Differ by: 1.1102230246251565e-16. Adjusting values.
  warn(
/home/sam/anaconda3/envs/FAIKR3/lib/python3.9/site-
packages/pgmpy/utils/mathext.py:83: UserWarning: Probability values don't
exactly sum to 1. Differ by: 1.1102230246251565e-16. Adjusting values.
  warn(
 90%|      | 9/10 [00:00<00:00,
84.23it/s]/home/sam/anaconda3/envs/FAIKR3/lib/python3.9/site-
packages/pgmpy/utils/mathext.py:83: UserWarning: Probability values don't
exactly sum to 1. Differ by: 1.1102230246251565e-16. Adjusting values.
  warn(
100%|      | 10/10 [00:00<00:00, 67.28it/s]
```

```
+-------------+-----------------+
| ADM-DECS    |   phi(ADM-DECS) |
+=============+=================+
| ADM-DECS(A) |          0.8000 |
```

```
+-------------+-----------------+
| ADM-DECS(S) |          0.2000 |
+-------------+-----------------+
```

Approximate probability with an high number of samples

[ ]:
```
print(infer.query(variables=["ADM-DECS"], evidence={'COMFORT': 'normal_range',␣
↪'L-CORE': 'low'}, n_samples=70))
```

```
  0%|          | 0/70 [00:00<?,
?it/s]/home/sam/anaconda3/envs/FAIKR3/lib/python3.9/site-
packages/pgmpy/utils/mathext.py:83: UserWarning: Probability values don't
exactly sum to 1. Differ by: 1.1102230246251565e-16. Adjusting values.
  warn(
/home/sam/anaconda3/envs/FAIKR3/lib/python3.9/site-
packages/pgmpy/utils/mathext.py:83: UserWarning: Probability values don't
exactly sum to 1. Differ by: 1.1102230246251565e-16. Adjusting values.
  warn(
100%|     | 70/70 [00:00<00:00, 663.78it/s]
```

```
+-------------+-----------------+
| ADM-DECS    |   phi(ADM-DECS) |
+=============+=================+
| ADM-DECS(A) |          0.6714 |
+-------------+-----------------+
| ADM-DECS(I) |          0.0429 |
+-------------+-----------------+
| ADM-DECS(S) |          0.2857 |
+-------------+-----------------+
```

The get_distribution method computes distribution of variables from given data samples. Likelihood sampling

[ ]:
```
print(infer.get_distribution(inference.likelihood_weighted_sample(size=50),␣
↪["ADM-DECS"], joint=True))
```

```
Generating for node: CORE-STBL:   0%|          | 0/9 [00:00<?,
?it/s]/home/sam/anaconda3/envs/FAIKR3/lib/python3.9/site-
packages/pgmpy/utils/mathext.py:83: UserWarning: Probability values don't
exactly sum to 1. Differ by: 1.1102230246251565e-16. Adjusting values.
  warn(
Generating for node: ADM-DECS: 100%|     | 9/9 [00:00<00:00, 248.22it/s]
```

```
+-------------+-----------------+
| ADM-DECS    |   phi(ADM-DECS) |
+=============+=================+
| ADM-DECS(A) |          0.7800 |
```

```
+-------------+-----------------+
| ADM-DECS(I) |          0.0200 |
+-------------+-----------------+
| ADM-DECS(S) |          0.2000 |
+-------------+-----------------+
```

Rejection sampling

[ ]:
```
print(infer.get_distribution(inference.likelihood_weighted_sample(size=50),␣
↪["ADM-DECS"]))
```

```
Generating for node: CORE-STBL:   0%|          | 0/9 [00:00<?,
?it/s]/home/sam/anaconda3/envs/FAIKR3/lib/python3.9/site-
packages/pgmpy/utils/mathext.py:83: UserWarning: Probability values don't
exactly sum to 1. Differ by: 1.1102230246251565e-16. Adjusting values.
  warn(
Generating for node: ADM-DECS: 100%|     | 9/9 [00:00<00:00, 229.52it/s]
```

```
+-------------+-----------------+
| ADM-DECS    |   phi(ADM-DECS) |
+=============+=================+
| ADM-DECS(A) |          0.7400 |
+-------------+-----------------+
| ADM-DECS(I) |          0.0200 |
+-------------+-----------------+
| ADM-DECS(S) |          0.2400 |
+-------------+-----------------+
```

1.5 Conclusion