Bayesian

In [1]:
```
import sorobn as hh

bn = hh.BayesNet(
     ('Burglary', 'Alarm'),
     ('Earthquake', 'Alarm'),
     ('Alarm', 'John calls'),
     ('Alarm', 'Mary calls')
 )
```

In [2]:
```
import pandas as pd

# P(Burglary)
bn.P['Burglary'] = pd.Series({False: .999, True: .001})
z
# P(Earthquake)
bn.P['Earthquake'] = pd.Series({False: .998, True: .002})

# P(Alarm | Burglary, Earthquake)
bn.P['Alarm'] = pd.Series({
      (True, True, True): .95,
      (True, True, False): .05,

      (True, False, True): .94,
      (True, False, False): .06,

      (False, True, True): .29,
      (False, True, False): .71,

      (False, False, True): .001,
      (False, False, False): .999
  })

# P(John calls | Alarm)
bn.P['John calls'] = pd.Series({
      (True, True): .9,
      (True, False): .1,
      (False, True): .05,
      (False, False): .95
  })

# P(Mary calls | Alarm)
bn.P['Mary calls'] = pd.Series({
      (True, True): .7,
      (True, False): .3,
      (False, True): .01,
      (False, False): .99
  })
```

In [3]:
```
bn.prepare()
```

In [4]:
```
# Second example for bayesian network

# _ = hh.BayesNet(
#      ('Cloud', 'Rain'),
```

```
#      (['Rain', 'Cold'], 'Snow'),
#      'Wind speed'  # has no dependencies
#  )
```

In [5]:
```
bn.query('Burglary', event={'Mary calls': True, 'John calls': True})
```

Out[5]:
```
Burglary
False    0.715828
True     0.284172
Name: P(Burglary), dtype: float64
```

In [6]:
```
bn.query('John calls', 'Mary calls', event={'Earthquake': True})
```

Out[6]:
```
John calls  Mary calls
False       False         0.675854
            True          0.027085
True        False         0.113591
            True          0.183470
Name: P(John calls, Mary calls), dtype: float64
```

In [7]:
```
import numpy as np
np.random.seed(42)

bn.query(
     'Burglary',
     event={'Mary calls': True, 'John calls': True},
     algorithm='gibbs',
     n_iterations=1000
 )
```

```
C:\Users\dell\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kf
ra8p0\LocalCache\local-packages\Python310\site-packages\sorobn\bayes_net.py:690: F
utureWarning: Not prepending group keys to the result index of transform-like appl
y. In the future, the group keys will be included in the index, regardless of whet
her the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  post = post.groupby(boundary).apply(lambda g: g / g.sum())
C:\Users\dell\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kf
ra8p0\LocalCache\local-packages\Python310\site-packages\sorobn\bayes_net.py:690: F
utureWarning: Not prepending group keys to the result index of transform-like appl
y. In the future, the group keys will be included in the index, regardless of whet
her the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  post = post.groupby(boundary).apply(lambda g: g / g.sum())
C:\Users\dell\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kf
ra8p0\LocalCache\local-packages\Python310\site-packages\sorobn\bayes_net.py:690: F
utureWarning: Not prepending group keys to the result index of transform-like appl
y. In the future, the group keys will be included in the index, regardless of whet
her the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  post = post.groupby(boundary).apply(lambda g: g / g.sum())
```

Out[7]:
```
Burglary
False    0.739
True     0.261
Name: P(Burglary), dtype: float64
```

Missing value imputation

In [8]:
```python
from pprint import pprint

sample = {
     'Alarm': True,
     'Burglary': True,
     'Earthquake': False,
     'John calls': None,  # missing
     'Mary calls': None   # missing
 }

sample = bn.impute(sample)
pprint(sample)
```

```
{'Alarm': True,
 'Burglary': True,
 'Earthquake': False,
 'John calls': True,
 'Mary calls': True}
```

Likelihood estimation

In [9]:
```
event = {
     'Alarm': False,
     'Burglary': False,
     'Earthquake': False,
     'John calls': False,
     'Mary calls': False
 }

bn.predict_proba(event)
```

Out[9]:
```
0.9367427006190001
```

In [10]:
```
event = {'Alarm': True, 'Burglary': False}
bn.predict_proba(event)
```

Out[10]:
```
0.001576422
```

In [11]:
```
event = {'Alarm': False}
bn.predict_proba(event)
```

Out[11]:
```
0.9974835580000001
```

In [12]:
```
events = pd.DataFrame([
     {'Alarm': False, 'Burglary': False, 'Earthquake': False,
      'John calls': False, 'Mary calls': False},

     {'Alarm': False, 'Burglary': False, 'Earthquake': False,
      'John calls': True, 'Mary calls': False},

     {'Alarm': True, 'Burglary': True, 'Earthquake': True,
      'John calls': True, 'Mary calls': True}
 ])

bn.predict_proba(events)
```

Out[12]:
```
Alarm  Burglary  Earthquake  John calls  Mary calls
False  False     False       False       False         0.936743
                             True        False         0.049302
True   True      True        True        True          0.000001
Name: P(Alarm, Burglary, Earthquake, John calls, Mary calls), dtype: float64
```

In [ ]: