Machine Learning Lab-7

Bayesian Networks

Bayesian networks are a powerful inference tool, in which nodes represent some random variable we care about, edges represent dependencies and a lack of an edge between two nodes represents a conditional independence. A powerful algorithm called the sum-product or forward-backward algorithm allows for inference to be done on this network, calculating posteriors on unobserved (“hidden”) variables when limited information is given.

The Monty Hall Gameshow

The Monty Hall problem arose from the gameshow Let's Make a Deal, where a guest had to choose which one of three doors had a prize behind it. The twist was that after the guest chose, the host, originally Monty Hall, would then open one of the doors the guest did not pick and ask if the guest wanted to switch which door they had picked. Initial inspection may lead you to believe that if there are only two doors left, there is a 50-50 chance of you picking the right one, and so there is no advantage one way or the other. However, it has been proven both through simulations and analytically that there is in fact a 66% chance of getting the prize if the guest switches their door, regardless of the door they initially went with.

We can reproduce this result using Bayesian networks with three nodes, one for the guest, one for the prize, and one for the door Monty chooses to open. The door the guest initially chooses and the door the prize is behind are completely random processes across the three doors, but the door which Monty opens is dependent on both the door the guest chooses (it cannot be the door the guest chooses), and the door the prize is behind (it cannot be the door with the prize behind it).

To create the Bayesian network in pomegranate, we first create the distributions which live in each node in the graph. For a discrete (aka categorical) bayesian network we use DiscreteDistribution objects for the root nodes and ConditionalProbabilityTable objects for the inner and leaf nodes. The columns in a ConditionalProbabilityTable correspond to the order in which the parents (the second argument) are specified, and the last column is the value the ConditionalProbabilityTable itself takes. In the case below, the first column corresponds to the value 'guest' takes, then the value 'prize' takes, and then the value that 'monty' takes. 'B', 'C', 'A' refers then to the probability that Monty reveals door 'A' given that the guest has chosen door 'B' and that the prize is actually behind door 'C', or P(Monty='A'|Guest='B', Prize='C').

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set_style('whitegrid')
import numpy

from pomegranate import *

numpy.random.seed(0)
numpy.set_printoptions(suppress=True)

%load_ext watermark
%watermark -m -n -p numpy,scipy,pomegranate
```

```
numpy      : 1.21.4
scipy      : 1.7.0
pomegranate: 0.14.7

Compiler   : MSC v.1916 64 bit (AMD64)
OS         : Windows
Release    : 10
Machine    : AMD64
Processor  : Intel64 Family 6 Model 142 Stepping 10, GenuineIntel
CPU cores  : 8
Architecture: 64bit
```

In [2]:
```
# The guests initial door selection is completely random
guest = DiscreteDistribution({'A': 1./3, 'B': 1./3, 'C': 1./3})

# The door the prize is behind is also completely random
prize = DiscreteDistribution({'A': 1./3, 'B': 1./3, 'C': 1./3})

   # Monty is dependent on both the guest and the prize.
monty = ConditionalProbabilityTable(
      [[ 'A', 'A', 'A', 0.0 ],
       [ 'A', 'A', 'B', 0.5 ],
       [ 'A', 'A', 'C', 0.5 ],
       [ 'A', 'B', 'A', 0.0 ],
       [ 'A', 'B', 'B', 0.0 ],
       [ 'A', 'B', 'C', 1.0 ],
       [ 'A', 'C', 'A', 0.0 ],
       [ 'A', 'C', 'B', 1.0 ],
       [ 'A', 'C', 'C', 0.0 ],
       [ 'B', 'A', 'A', 0.0 ],
       [ 'B', 'A', 'B', 0.0 ],
       [ 'B', 'A', 'C', 1.0 ],
       [ 'B', 'B', 'A', 0.5 ],
       [ 'B', 'B', 'B', 0.0 ],
       [ 'B', 'B', 'C', 0.5 ],
       [ 'B', 'C', 'A', 1.0 ],
       [ 'B', 'C', 'B', 0.0 ],
       [ 'B', 'C', 'C', 0.0 ],
       [ 'C', 'A', 'A', 0.0 ],
       [ 'C', 'A', 'B', 1.0 ],
       [ 'C', 'A', 'C', 0.0 ],
       [ 'C', 'B', 'A', 1.0 ],
       [ 'C', 'B', 'B', 0.0 ],
       [ 'C', 'B', 'C', 0.0 ],
       [ 'C', 'C', 'A', 0.5 ],
       [ 'C', 'C', 'B', 0.5 ],
       [ 'C', 'C', 'C', 0.0 ]], [guest, prize])
```

In [3]:
```
# State objects hold both the distribution, and a high level name.
s1 = State(guest, name="guest")
s2 = State(prize, name="prize")
s3 = State(monty, name="monty")
```

In [4]:
```
# Create the Bayesian network object with a useful name
model = BayesianNetwork("Monty Hall Problem")

# Add the three states to the network
model.add_states(s1, s2, s3)
```

In [5]:
```
# Add edges which represent conditional dependencies, where the second node is
# conditionally dependent on the first node (Monty is dependent on both guest and prize)
model.add_edge(s1, s3)
model.add_edge(s2, s3)
```

In [6]:
```
model.bake()
```

Predicting Probabilities

In [7]:
```
model.probability([['A', 'B', 'C']])
```

Out[7]: 0.11111111111111109

In [8]:
```
model.probability([['A', 'B', 'B']])
```

Out[8]: 0.0

Performing Inference

In [9]:
```
model.predict_proba({})
```

Out[9]:
```
array([{
        "class" : "Distribution",
        "dtype" : "str",
        "name" : "DiscreteDistribution",
        "parameters" : [
          {
            "A" : 0.33333333333333337,
            "B" : 0.33333333333333337,
            "C" : 0.33333333333333337
          }
        ],
        "frozen" : false
      }                                ,
      {
        "class" : "Distribution",
        "dtype" : "str",
        "name" : "DiscreteDistribution",
        "parameters" : [
          {
            "A" : 0.33333333333333337,
            "B" : 0.33333333333333337,
            "C" : 0.33333333333333337
          }
        ],
        "frozen" : false
      }                                ,
      {
        "class" : "Distribution",
        "dtype" : "str",
        "name" : "DiscreteDistribution",
        "parameters" : [
          {
            "C" : 0.3333333333333333,
            "B" : 0.3333333333333333,
            "A" : 0.3333333333333333
          }
        ],
        "frozen" : false
      }                                ], dtype=object)
```

In [10]:
```
model.predict_proba([{'guest': 'A', 'monty': 'C'}])
```

Out[10]:
```
[array(['A', {
            "class" : "Distribution",
            "dtype" : "str",
            "name" : "DiscreteDistribution",
            "parameters" : [
              {
                 "A" : 0.3333333333333334,
                 "B" : 0.6666666666666664,
                 "C" : 0.0
              }
            ],
            "frozen" : false
          }                          , 'C'], dtype=object)]
```

In [ ]:
```

```