

Shi Xingyue

Dr. Rao Yanghui

Artificial Intelligence

22 November 2018

## Lab 10: Bayesian Network

### [Abstract]

In this lab, I constructed 3 Bayesian networks with given structure using python and pomegranate library. To calculate probability with unknown variables, I in the first-place use nested for loop and sum up all possible combinations of variables. Then I pack this part into function to facilitate using. I also used graphic tools – “graphviz” and “pygraphviz” to plot constructed networks.

## 1 Methods

### 1.1 Algorithm

#### 1.1.1 Probability with Unknown Variables

Take this probability in burglary problem as an example

$$P(\text{JohnCalls} = \text{'True'})$$

where variables “Burglary”, “MaryCalls”, “Earthquake” and “Alarm” are unknown.

I tried using the keyword “None” to represent these unknown variables, but I found that would not work properly. For example, when I used the input as shown in Code 1, the result turned out to be 1.0, which is obviously wrong.

```
print ("Johncalls=")
print model.probability([None, None, None, "t", None])
```

*Code 1 Simply represent unknown variables with keyword "None" does not work properly*

```
ambershek@ubuntu:~/bayesian$ python burglary.py
Johncalls=
1.0
```

*Figure 1 The result is 1.0, which is obviously wrong*

Then I used the expression as below to solve this problem:

$$P(\text{JohnCalls} = 'True', \text{MaryCalls} = 'True') \\ = \sum_{\text{Burglary}} \sum_{\text{Earthquake}} \sum_{\text{Alarm}} P(\text{Burglary}, \text{Earthquake}, \text{Alarm}, \text{JohnCalls} = 'True', \\ \text{MaryCalls} = 'True')$$

where each probability without unknown variables can be got by calling probability function in pomegranate library, and the final result can be summed up to using nested for loop. I also packed this process into a function later.

### 1.1.2 Conditional Probability

After the work done in 1.1.1, I now can easily calculate any probability given some values of variables. Conditional probability can be got using its definition:

$$P(A|BC) = \frac{P(ABC)}{P(BC)}$$

where both numerator and denominator can be got using the method explained in Section 1.1.1.

## 1.2 Pseudo-code

The overview structure of these 3 problems are similar, so I only present the pseudo-code of burglary problem here.

The Bayesian network is constructed as below:

```

1  import * from pomegranate
2
3  define variables of discrete distribution
4  set probability for each result
5
6
7  define variables conditionally dependent on defined variables
8  set conditional probability table
9
10 define each variable as a node network
11
12
13 define edges showing the dependencies between nodes

bake the network

```

*Code 2 Pseudo-code of constructing a Bayesian network*

The function to calculate probability with unknown variables has pseudo-code as below:

```

1  prob (*state)
2      calculate #possible variable sets
3
4      for the variables with given value
5          all variable sets have that value for that variables
6
7      for the variables with unknown value
8          assign all possible value combination to variable sets using tree structure
9
10     calculate probability of each variable set
11
12     sum up all the probability and return it

```

*Code 3 Pseudo-code of the function calculating probability with unknown variables*

Conditional probability is calculated as below:

```

1  p_a=probability of numerator
2  p_b=probability of denominator
3
4  result=p_a/p_b

```

*Code 4 Pseudo-code of calculating conditional probability*

### 1.3 Key Codes with Comments

#### 1.3.1 Calculating Probability with Unknown Variables

The code looks like below:

```

1  for value_b in values:
2      for value_e in values:
3          for value_a in values:
4              p1+=model.probability([value_b, value_e, value_a, 't', 't'])
5  print ("P(JohnCalls, MaryCalls)=")
6  print p1
7  print "\n"

```

*Code 5 Calculating probability with unknown variables with nested for loop*

This process can, and also should, be packed into a function to simplify programming. The function is defined and called as below:

```

1  def prob(*states):
2      variablesets_count=1
3      variablesets=[]
4
5      #get the number of variable sets
6      for state in states:
7          if state==0:
8              variablesets_count*=2
9
10     #construct variable sets using 2-tree structure
11     offset=variablesets_count/2
12
13     for i in range (0,variablesets_count):
14         variablesets.append([])
15
16     proba=0
17
18     #construct variable sets
19     #and calculate their respective probability
20     for j in range (0,5):
21         if states[j]==1:
22             for variableset in variablesets:
23                 variableset+='t'
24         elif states[j]==-1:
25             for variableset in variablesets:
26                 variableset+='f'
27         elif states[j]==0:
28             for i in range (0,variablesets_count):
29                 if (int) (i/offset)%2==0:
30                     variablesets[i]+='t'
31                 else:
32                     variablesets[i]+='f'
33             offset/=2
34
35     for variableset in variablesets:
36         proba+=model.probability(variableset)
37
38     return proba

```

*Code 6 Pack the process of calculating probability with unknown variables into a function*

```

1  state=[0,0,0,1,1]
2  print ("P(JohnCalls, MaryCalls)=")
3  print prob(*state)
4  print "\n"

```

*Code 7 Call the function in Code 3 to calculate probability*

### 1.3.2 Conditional Probability

Quite similar to code 7, I call function prob twice and get the answer.

```

1 state1=[0,0,1,1,1]
2 state2=[0,0,0,1,1]
3 print ("P(Alarm | JohnCalls, MaryCalls)=")
4 print prob(*state1)/prob(*state2)
5 print "\n"

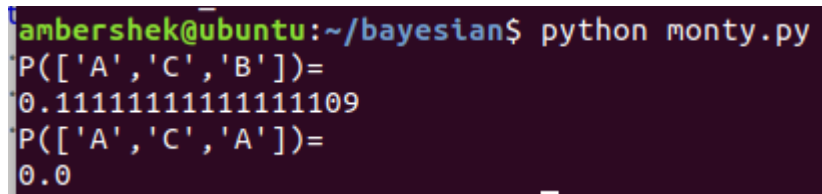
```

*Code 8 Call prob to calculate numerator and denominator and then divide to get conditional probability*

## 2 Results and Analysis

### 2.1 Result

#### 2.1.1 Task 1 – Monty Hall Problem



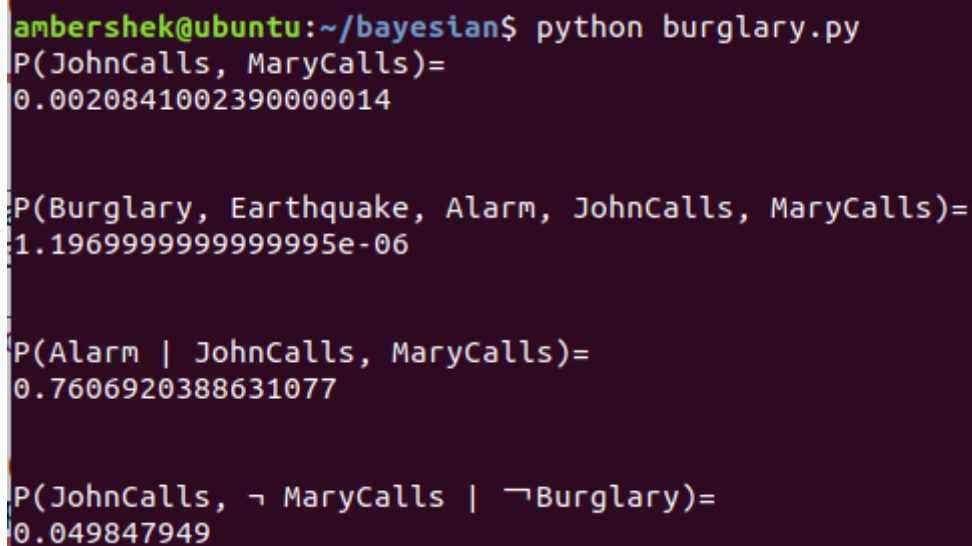
```

ambershek@ubuntu:~/bayesian$ python monty.py
P(['A', 'C', 'B'])=
0.1111111111111109
P(['A', 'C', 'A'])=
0.0

```

*Figure 2 Result conditional probabilities got to Monty Hall Problem*

#### 2.1.2 Task 2 – Burglary



```

ambershek@ubuntu:~/bayesian$ python burglary.py
P(JohnCalls, MaryCalls)=
0.0020841002390000014

P(Burglary, Earthquake, Alarm, JohnCalls, MaryCalls)=
1.1969999999999995e-06

P(Alarm | JohnCalls, MaryCalls)=
0.7606920388631077

P(JohnCalls, ¬ MaryCalls | ¬ Burglary)=
0.049847949

```

*Figure 3 Results of Burglary Problem*

#### 2.1.3 Task 3 – Diagnosing

The Bayesian network of diagnosing problem and result probabilities are shown below.

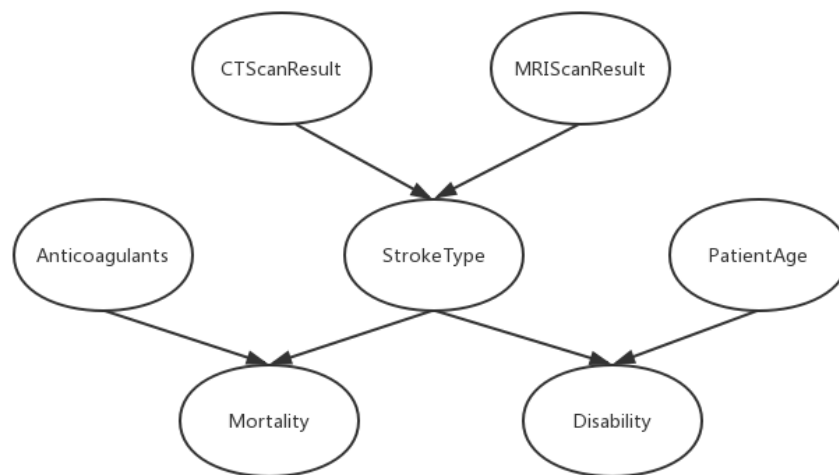


Figure 4 The Bayesian network of diagnosing problem

```

ambershek@ubuntu:~/bayesian$ python diagnosing.py
P(Mortality='True' | PatientAge='0-30' , CTScanResult='Ischemic Stroke')=
0.5948500000000003

P(Disability=' Severe ' | PatientAge='65+' , MRIScanResult=' Ischemic Stroke ')=
0.42100000000000003

P(StrokeType='Stroke Mimic' | PatientAge='65+' , CTScanResult='Hemorrhagic Stroke' , MRIScan
Result='Ischemic Stroke')=
0.10000000000000005

P(Mortality='False' | PatientAge='0-30' , Anticoagulants='Used' , StrokeType='Stroke Mimic')=
0.09999999999999992

P( PatientAge='0-30' , CTScanResult='Ischemic Stroke' , MRIScanResult=' Hemorrhagic Stroke' ,
Anticoagulants='Used' , StrokeType='Stroke Mimic' , Disability=' Severe' , Mortality='False'
)=
5.250000000000009e-06
  
```

Figure 5 Results of Diagnosing Problem

### 3 思考题

K2 has high computational cost and produces a significant number of extra arcs in the learned network.

The high computational cost is lead to by calculating  $g(i, \pi_i)$  (in line 7 in TA's slides) which requires many computational resources especially for nodes characterized by a great number of parents.

The extra arc problem arises especially when the network is characterized by a lot of root nodes (nodes without parents). During network learning, the algorithm tries to add parents to each of these nodes until it maximizes  $g(i, \pi_i)$ . The algorithm will add at least one arc to a root node because the value of the heuristic for this new structure is always better than the value of the previous structure.

The new proposed approach presented in Lamma's paper considers all the association rules containing a single item in the body and a single item in the head. In order to obtain the leverage of these rules [1]. The improved K2 considers all the possible two items rules and for each we compute the leverage. It first computes, for each stochastic variable  $V_i$ , the maximum and the minimum of the leverage of the association rules that have an item that refers to  $V_i$ . Let  $MaxLev(V_i)$  and  $MinLev(V_i)$  be these figures.

## Work Cited

- [1] Lamma, Evelina, Fabrizio Riguzzi, and Sergio Storari. "Improving the K2 Algorithm Using Association Rule Parameters."