中山大学数据科学与计算机学院

人工智能本科生实验报告

（2018-2019）学年秋季学期

教学班级	16级计科二班	专业（方向）	计算机科学与技术
学号	16337334	姓名	周启恒

贝叶斯网络

（1）原理

定义

- 贝叶斯网络是一种概率图模型，由有向无环图表示。
- 互相连接两个节点，代表此两个随机变量是具有因果关系或是非条件独立的。而两个节点间不相互连接的情况就称其随机变量彼此间为条件独立。
- 利用变量间的独立性和条件独立关系，把复杂的联合概率分布分解成一系列相对简单的模块，减少为定义完全联合概率分布所指定的概率数目，从而可以降低知识获取和概率推理的复杂度的一种模型。

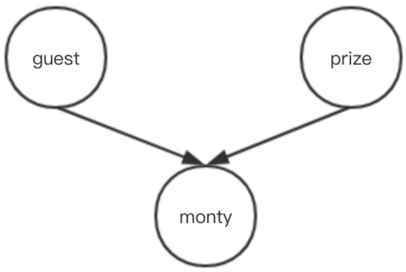
贝叶斯网络学习的步骤

- 变量定义：对问题进行建模，找出所需的变量
- 结构学习：确定各变量之间的连接关系（因果关系）
- 参数学习：发现变量之间相互关联的量化关系

（2）TASK关键代码（python带注释）

Task1 三门问题

- 网络图，Guest为选手选择的门，Prize为奖品所在的门，Monty为主持人打开的门。

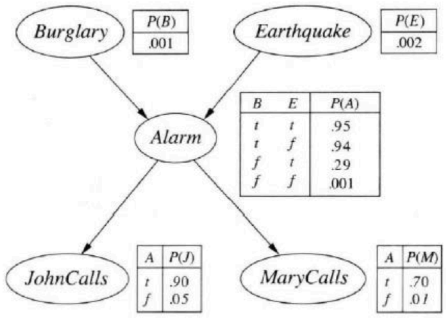



```python
import pomegranate as pg

guestDistribution = pg.DiscreteDistribution( {'A':1./3, 'B':1./3, 'C':1./3} )
prizeDistribution = pg.DiscreteDistribution( {'A':1./3, 'B':1./3, 'C':1./3} )
#生成条件概率表
table = []
for prize in ['A', 'B', 'C']:
    for guest in ['A', 'B', 'C']:
        for monty in ['A', 'B', 'C']:
            tmpPro = 0.0 if(monty == guest or monty == prize) else 0.5 if(guest == prize) else 1.0
            table.append([prize, guest, monty, tmpPro])
montyCondition = pg.ConditionalProbabilityTable(table, [guestDistribution, prizeDistribution])
#初始化三个节点
s1 = pg.Node(guestDistribution, name='guest')
s2 = pg.Node(prizeDistribution, name='prize')
s3 = pg.Node(montyCondition, name='monty')
#构建贝叶斯网络
model = pg.BayesianNetwork('Monty Hall Problem')
#增加三个节点
model.add_nodes(s1, s2, s3)
#增加边
model.add_edge(s1, s3)
model.add_edge(s2, s3)
model.bake()
#调用函数得到概率
print(model.probability(['A','C','B']))
print(model.probability(['A','C','A']))
```

Task2 Burglary

- 网络图:




```
#TASK2
Burglary = pg.DiscreteDistribution({'B':0.001, '~B':0.999})
Earthquake = pg.DiscreteDistribution({'E':0.002, '~E':0.998})
Alarm = pg.ConditionalProbabilityTable([
    ['B', 'E', 'A', 0.95],
    ['B', 'E', '~A', 0.05],

    ['B', '~E', 'A', 0.94],
    ['B', '~E', '~A', 0.06],

    ['~B', 'E', 'A', 0.29],
    ['~B', 'E', '~A', 0.71],

    ['~B', '~E', 'A', 0.001],
    ['~B', '~E', '~A', 0.999]], [Burglary, Earthquake]
)

John = pg.ConditionalProbabilityTable([
    ['A', 'J', 0.90],
    ['A', '~J', 0.10],
    ['~A', 'J', 0.05],
    ['~A', '~J', 0.95]], [Alarm]
)

Mary = pg.ConditionalProbabilityTable([
    ['A', 'M', 0.70],
    ['A', '~M', 0.30],
    ['~A', 'M', 0.01],
    ['~A', '~M', 0.99]], [Alarm]
)

task2_model = pg.BayesianNetwork('task2')
s_b = pg.Node(Burglary, name='burglary')
s_e = pg.Node(Earthquake, name='earthquake')
s_a = pg.Node(Alarm, name='alarm')
s_j = pg.Node(John, name='john')
```

```
s_m = pg.Node(Mary, name='mary')

task2_model.add_nodes(s_b, s_e, s_a, s_j, s_m)
task2_model.add_edge(s_b, s_a)
task2_model.add_edge(s_e, s_a)
task2_model.add_edge(s_a, s_j)
task2_model.add_edge(s_a, s_m)
task2_model.bake()
```

- **求联合概率**

递归计算联合概率，将未赋值的变量的所有情况遍历。

$$P(A,B) = \sum_C \sum_D \sum_E P(A,B,c,d,e)$$

```
'''
eventList:存储没有被赋值的变量
toDo:需要计算的概率
domain_list:各个变量的取值
'''
def getProbability(model, eventList, index, toDo, domain_list):
    if not len(eventList): return 0
    if(index == len(eventList)):
        return model.probability(toDo)
    result = 0
    for i in domain_list[eventList[index]]:
        toDo[eventList[index]] = i
        result += getProbability(model, eventList, index+1, toDo,
domain_list)
    return result
```

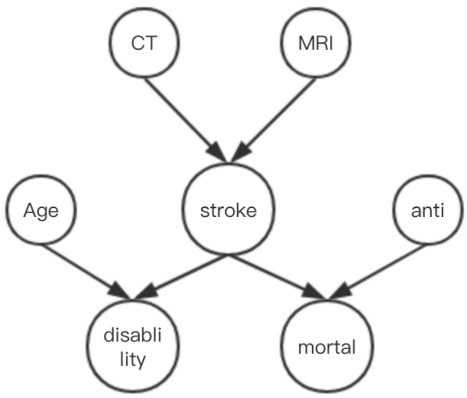
```
def getEventList(toDo):
    eventList = []
    for i in range(len(toDo)):
        if toDo[i] == '':
            eventList.append(i)
    return eventList

toDo = ['', '', '', 'J', 'M']
eventList = getEventList(toDo)
#P(JohnCalls, MaryCalls)
result1 = getProbability(task2_model, eventList, 0, toDo, domain_list)
#P(Burglary, Earthquake, Alarm, JohnCalls, MaryCalls)
result2 = task2_model.probability(['B', 'E', 'A', 'J', 'M'])
#P(Alarm | JohnCalls, MaryCalls)
toDo = ['', '', 'A', 'J', 'M']
eventList = getEventList(toDo)
```

```
result3 = getProbability(task2_model, eventList3, 0, toDo, domain_list) /
result1
#P(JohnCalls, ¬ MaryCalls | ¬Burglary)
toDo4 = ['~B', '', '', 'J', '~M']
eventList4 = getEventList(toDo4)
result4 = getProbability(task2_model, eventList4, 0, toDo4, domain_list) /
0.999
```

TASK3

- 贝叶斯网络图



- 按照网络结构编写代码构建贝叶斯网络（代码略）
- 各变量取值范围

```
p_domain = ['0-30', '31-65', '65+']
c_domain = ['IS', 'HS']
m_domain = ['IS', 'HS']
s_domain = ['IS', 'HS', 'SM']
a_domain = ['U', 'N']
mo_domain = ['T', 'F']
d_domain = ['Neg', 'Mod', 'Sev']

domain_list = [p_domain, c_domain, m_domain, s_domain, a_domain, mo_domain,
d_domain]
```

- 计算概率

```
#p1 = P(Mortality='True' | PatientAge='0-30' , CTScanResult='Ischemic
Stroke')
toDo = ['0-30', 'IS', '', '', '', 'T', '']
eventList = getEventList(toDo)
result1 = getProbability(task3_model, eventList, 0, toDo, domain_list)
toDo = ['0-30', 'IS', '', '', '', '', '']
eventList = getEventList(toDo)
result1 /= getProbability(task3_model, eventList, 0, toDo, domain_list)
```

```python
#p2 = P(Disability=' Severe ' | PatientAge='65+' , MRIScanResult=' Ischemic Stroke ')
toDo = ['65+', '', 'IS', '', '', '', 'Sev']
eventList = getEventList(toDo)
result2 = getProbability(task3_model, eventList, 0, toDo, domain_list)
toDo = ['65+', '', 'IS', '', '', '', '']
eventList = getEventList(toDo)
result2 /= getProbability(task3_model, eventList, 0, toDo, domain_list)
```

```python
#p3 = P(StrokeType='Stroke Mimic' | PatientAge='65+' , CTScanResult='Hemmorraghic Stroke' , MRIScanResult='Ischemic Stroke')
toDo = ['65+', 'HS', 'IS', 'SM', '', '', '']
eventList = getEventList(toDo)
result3 = getProbability(task3_model, eventList, 0, toDo, domain_list)
toDo = ['65+', 'HS', 'IS', '', '', '', '']
eventList = getEventList(toDo)
result3 /= getProbability(task3_model, eventList, 0, toDo, domain_list)
```

```python
#p4 = P(Mortality='False' | PatientAge='0-30', Anticoagulants=’Used’, StrokeType='Stroke Mimic')
toDo = ['0-30', '', '', 'SM', 'U', 'F', '']
eventList = getEventList(toDo)
result4 = getProbability(task3_model, eventList, 0, toDo, domain_list)
toDo = ['0-30', '', '', 'SM', 'U', '', '']
eventList = getEventList(toDo)
result4 /= getProbability(task3_model, eventList, 0, toDo, domain_list)
```

```python
#p5 = P(PatientAge='0-30', CTScanResult='Ischemic Stroke', MRIScanResult=' 'Hemmorraghic Stroke' , Anticoagulants=’Used’, StrokeType='Stroke Mimic' , Disability=' Severe' , Mortality ='False' )
toDo = ['0-30', '', '', 'SM', 'U', 'F', '']
result5 = task3_model.probability(['0-30','IS','HS','SM','U','F','Sev'])
```

思考题

K2算法的优化思路

- 对于某个节点，寻找其父节点列表。可利用一些规则提前判断两节点是否相关，在寻找父节点的过程中可减少一些搜索步骤。