中山大学数据科学与计算机学院

计算机科学与技术专业-人工智能

本科生实验报告

（2018-2019 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级	计科 2 班	专业（方向）	计算机科学与技术
学号	16337341	姓名	朱志儒

实验题目

贝叶斯网络

实验内容

· 算法原理

1) 概率

概率是对随机事件发生可能性的度量, 不像频率概率范畴所描述的对某个随机事件发生可能性的一个定值，而指的是一个变量。

2) 先验概率

先验概率是指根据以往的经验和分析得到的概率，在贝叶斯统计推断中，不确定数量的先验概率分布就是在考虑其他因素之前对该数量的置信程度的概率分布。

考虑 B 发生的情况下 A 发生的这一事件，事件 B 发生之前，对事件 A 的发生有一个基本的概率判断，称其为 A 的先验概率 P(A)。

3) 后验概率

后验概率是指一个随机变量基于实验或调查后得到的概率分布, 可通过贝叶斯公式用先验概率和似然函数计算得到。

考虑 B 发生的情况下 A 发生的这一事件, 事件 B 发生之前, 对事件 A 有先验概率判断, 事件 B 发生后，对事件 A 的发生概率将重新评估，称其为 A 的后验概率 P(A|B)。

4) 全概率公式

假设事件集合$\{B_n: n = 1, 2, 3, \dots\}$满足:

$\mathrm{B_i} \cap B_j = \emptyset, i \neq j$且$i, j = 1,2, \dots$且 $\mathrm{P}(B_i) > 0, P(B_j) > 0$;

$\mathrm{B_1} \cup B_2 \cup \dots = \Omega$;

则称事件集合$\{B_n: n = 1, 2, 3, \dots\}$是样本空间Ω的一个划分。

设$\{B_n: n = 1, 2, 3, \dots\}$是样本空间Ω的一个划分，A 为任一事件，则

$$\mathrm{P}(A) = \sum_{i=1}^{\infty} P(B_i)P(A|B_i)$$

上式即为全概率公式。

5) 贝叶斯公式

设试验 E 的样本空间为 S，A 为 E 的事件，$\mathrm{B_1}, B_2, \dots, B_n$为 S 的一个划分，且$\mathrm{P(A)} > 0, P(B_i) > 0 (i = 1,2, \dots, n)$，则

$$\mathrm{P}(B_i|A) = \frac{P(A|B_i)P(B_i)}{\sum_{j=1}^{n} P(A|B_j)P(B_j)}$$

上式即为贝叶斯公式。

6) 链式法则

$$\mathrm{P(X_1}, X_2, \dots, X_n) = P(X_n|X_1, X_2, \dots, X_{n-1})P(X_{n-1}|X_1, X_2, \dots, X_{n-2}) \cdots P(X_2|X_1)P(X_1)$$

7) 条件独立

若 A 和 B 在给定事件 C 的情况下条件独立，意思就是说，当事件 C 发生时，事件 A 是否发生和事件 B 是否发生对于条件概率分布而言是独立的，即当知道 C 发生时，知道 A 是否发生而无助于知道 B 是否发生，同样，知道 B 是否发生而无助于 A 是否发生。

用公式表示：$\mathrm{P(A,B|C)} = P(A|C)P(B|C)$或$\mathrm{P(A|B,C)} = P(A|C)$

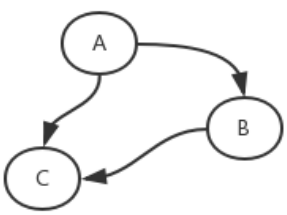
8) 贝叶斯网络

贝叶斯网络是一种模拟推理过程中因果关系的不确定性处理模型，它由网络拓扑图和条件概率表组成。

网络拓扑图结构是一个有向无环图，图中的节点表示随机变量，箭头表示节点之间的因果关系，连接两个节点的箭头表示这两个随机变量具有因果关系而不是条件独立的，其中一个节点表示“因”，另一个节点表示“果”。 而两个节点间若没有箭头相互连接一起时就称其随机变量彼此间为条件独立。将研究的问题中涉及的随机变量根据是否条件独立绘制在一个有向图中，形成贝叶斯网络，它主要用来描述随机变量之间的条件依赖，节点表示随机变量，箭头表示条件依赖。

条件概率表是由节点的条件概率$\mathrm{P}(\mathrm{X_i}|P_i)$得到的表，表中的每一行列出所有可能发生的$\mathrm{P_i}$，每一列列出所有可能发生的$\mathrm{X_i}$，任一行的概率和总为 1。得到条件概率表后就很容易地得知贝叶斯网络结构图中各节点间之因果关系。但条件概率表也存在一个很大的缺点：如果节点$\mathrm{X_i}$是有很多“因”所造成的“果”，那么计算其条件概率表将变得十分复杂。

一个简单的贝叶斯网络图：



联合概率分布：$\mathrm{P(A,B,C)} = P(A)P(B|A)P(C|A,B)$

条件概率表：

P(A):

A	0	1
Prob.	0.4	0.6

P(B|A):

A	B	
	0	1
0	0.32	0.68
1	0.54	0.46

P(C|A,B):

B	A	C	
		0	1
0	0	1.0	0.0
0	1	0.2	0.8
1	0	0.91	0.09
1	1	0.02	0.98

9) D-分离

D-分离方法可以快速判断两个节点之间是否条件独立，它分为三种情况：

a) 情况 1：

若 Z 是已知的，那么 X 与 Y 被阻断，即 X 与 Y 条件独立。

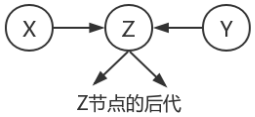
若 Z 是未知的，那么 X 与 Y 未被阻断，即 X 与 Y 不是条件独立。

b) 情况 2：

若 Z 是已知的，那么 X 与 Y 被阻断，即 X 与 Y 条件独立。

若 Z 是未知的，那么 X 与 Y 未被阻断，即 X 与 Y 不是条件独立。

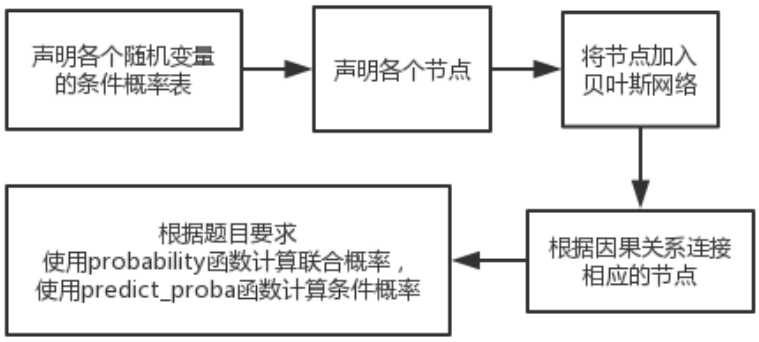
c) 情况 3：



若 Z 和 Z 的后代都是未知的，那么 X 与 Y 被阻断，即 X 与 Y 条件独立。

若 Z 已知或 Z 的某个后代是已知的，那么 X 与 Y 未被阻断，即 X 与 Y 不是条件独立。

· 流程图



· 关键代码

1) Task1：三门问题

```
class Task1:
    def __init__(self):
        # 声明各个节点的概率分布
        guest = DiscreteDistribution({'A': 1. / 3, 'B': 1. / 3, 'C': 1./3})
        prize = DiscreteDistribution({'A': 1. / 3, 'B': 1. / 3, 'C': 1./3})
        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])

        # 声明各个节点
        s1 = Node(guest, name="guest")
        s2 = Node(prize, name="prize")
        s3 = Node(monty, name="monty")

        # 构建贝叶斯网络
        self.model = BayesianNetwork("Monty Hall Problem")
        self.model.add_states(s1, s2, s3)
        self.model.add_edge(s1, s3)
        self.model.add_edge(s2, s3)
        self.model.bake()

    def get_A_C_B(self):
        # 计算 P(['A', 'C', 'B'])
        return self.model.probability(['A', 'C', 'B'])

    def get_A_C_A(self):
        # 计算 P(['A', 'C', 'A'])
        return self.model.probability(['A', 'C', 'A'])
```

2) Task2：Burglary

构建贝叶斯网络的步骤与 Task1 相同，就不再重复贴代码。

```
def get_J_M(self):
    # 计算 P(JohnCalls, MaryCalls)
    tmp = 0
    probability = 0.0
    while tmp != 8:
        select = []
        if tmp & 1:
            select.append('tb')
        else:
            select.append('fb')
        if tmp >> 1 & 1:
```

```
            select.append('te')
        else:
            select.append('fe')
        if tmp >> 2 & 1:
            select.append('ta')
        else:
            select.append('fa')
        tmp += 1
        select += ['tj', 'tm']
        # 计算联合概率的和以消去其他随机变量
        probability += self.model.probability(select)
    return probability

def get_B_E_A_J_M(self):
    # 计算 P(Burglary, Earthquake, Alarm, JohnCalls, MaryCalls)
    return self.model.probability(['tb', 'te', 'ta', 'tj', 'tm'])

def get_A_when_J_M(self):
    # 计算 P(Alarm | JohnCalls, MaryCalls)
    return self.model.predict_proba({'JohnCalls': 'tj', 'MaryCalls': 'tm'})[
2]

def get_J_NM_when_NB(self):
    # 计算 P(JohnCalls, ~MaryCalls | ~Burglary)
    tmp = 0
    probability = 0.0
    while tmp != 4:
        select = ['fb']
        if tmp & 1:
            select.append('te')
        else:
            select.append('fe')
        if tmp >> 1 & 1:
            select.append('ta')
        else:
            select.append('fa')
        tmp += 1
        select += ['tj', 'fm']
        # 计算联合概率的和以消去其他随机变量
        probability += self.model.probability(select)
    return probability / 0.999
```

3) Task3：Diagnosing

构建贝叶斯网络的步骤与 Task1 相同，就不再重复贴代码。

```
def get_p1(self):
    # 计算 P(Mortality='True' | PatientAge='0-30',
CTScanResult='Ischemic Stroke')
    return self.model.predict_proba({'PatientAge': '0', 'CTScanResult': 'CIS
'})[5]

def get_p2(self):
    # 计算 P(Disability='Severe' | PatientAge='65+',
MRIScanResult='Ischemic Stroke')
    return self.model.predict_proba({'PatientAge': '65', 'MRIScanResult': 'M
IS'})[6]

def get_p3(self):
    # 计算 P(StrokeType='Stroke Mimic' | PatientAge='65+',
CTScanResult='Hemmorraghic Stroke', MRIScanResult='Ischemic Stroke')
    return self.model.predict_proba({'PatientAge': '65', 'CTScanResult': 'CH
S', 'MRIScanResult': 'MIS'})[3]

def get_p4(self):
    # 计算 P(Mortality='False' | PatientAge='0-30',
Anticoagulants='Used', StrokeType='Stroke Mimic')
    return self.model.predict_proba({'PatientAge': '0', 'Anticoagulants': 'U
', 'StrokeType': 'SSM'})[5]

def get_p5(self):
    # 计算 P(PatientAge='0-30', CTScanResult='Ischemic Stroke',
MRIScanResult='Hemmorraghic Stroke', Anticoagulants='Used',
StrokeType='Stroke Mimic', Disability='Severe', Mortality ='False')
    return self.model.probability(['0', 'CIS', 'MHS', 'SSM', 'U', 'F', 'S'])
```

实验结果及分析

· 实验结果展示

a) Task1：三门问题

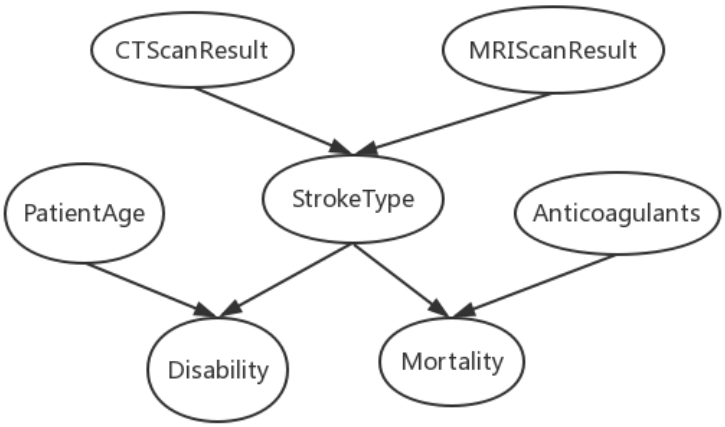
```
P(['A', 'C', 'B']): 0.11111111111111109
P(['A', 'C', 'A']): 0.0
```

b) Task2：Burglary

```
P(JohnCalls, MaryCalls): 0.0020841002390000014
P(Burglary, Earthquake, Alarm, JohnCalls, MaryCalls): 1.1969999999999995e-06
P(Alarm | JohnCalls, MaryCalls): 0.7606920388631567
P(JohnCalls, ~MaryCalls | ~Burglary): 0.049847948999999996
```

c) Task3：Diagnosing

贝叶斯网络结构图：



计算的概率如下：

```
P(Mortality='True' | PatientAge='0-30', CTScanResult='Ischemic Stroke'): 0.5948499999999999
P(Disability='Severe' | PatientAge='65+', MRIScanResult='Ischemic Stroke'): 0.421
P(StrokeType='Stroke Mimic' | PatientAge='65+', CTScanResult='Hemmorraghic Stroke', MRIScanResult='Ischemic Stroke'): 0.10000000000000045
P(Mortality='False' | PatientAge='0-30', Anticoagulants='Used', StrokeType='Stroke Mimic'): 0.1000000000000002
P(PatientAge='0-30', CTScanResult='Ischemic Stroke', MRIScanResult='Hemmorraghic Stroke',
  Anticoagulants='Used', StrokeType='Stroke Mimic', Disability='Severe', Mortality ='False'): 5.250000000000009e-06
```

思考题

对于刚刚讲到的 K2 算法，有什么可以改进的思路？

改进思路：

1) K2 算法的评分函数可修改为计算$\log g(i,\pi_i)$，而不是直接计算$g(i,\pi_i)$，以简化运算；

2) K2 算法对随机变量的序列的顺序十分敏感，对于不同的随机变量的序列，该算法将得到不同的最优结构，所以可以使用几组不同的随机变量序列作为输入，得到相应的几组不同的最优结构，比较这些最优结构，选择效果最好的结构作为输出。