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

**变量消元**

## 实验内容

* **算法原理**

1. **因子：**

因子是随机变量的一个函数，它将随机变量映射到一个条件概率表。例如，f(A,B)可以表示P(A|B)或P(B|A)，取决于具体的贝叶斯网络。

1. **相乘操作：**

因子f(A, B)和因子g(B, C)具有相同的变量B，这两个因子相乘得到新的因子h(A, B, C)，即。

1. **求和操作：**

对于因子f(A, B)，对随机变量A求和即可从f(A, B)得到h(B)。由于f(A, B)是条件概率表而不是联合概率表，所以求和后h(B)可能存在大于1的概率，归一化后即可得到正确的概率。

1. **限制操作：**

对于因子f(A, B)，限制f(A, B)中的A = a，即在f(A, B)的条件概率表中选择A = a的项而忽视其他的项，从而得到新的因子h(B)。

1. **变量消元算法：**

根据给定的证据列表，限制所有已知的随机变量。

根据给定的消元顺序，对每个需要消元的变量Zj执行如下操作：

1. 找到包含变量Zj的所有因子；
2. 先将找到的所有因子相乘，再对变量Zj求和得到新的因子gj；
3. 从因子列表中删除之前找到的所有因子，再将新的因子gj加入因子列表。

将其他随机变量消去后，只剩下包含查询变量的因子，将这些因子相乘再进行归一化就可以得到目标条件概率。

* **伪代码**

输入：N—贝叶斯网络；E—证据变量；e—证据变量的取值；

Q—查询变量；p—消元顺序，包含所有不在中的变量。

输出：P(Q | E = e)。

1. U←N中所有概率分布的集合；
2. 在U的因子中，将证据变量E设置为其观测值e；
3. ：
4. 设Z为p中排在最前面的变量，将Z从p中删去；
5. 从U中删去所有涉及Z的因子，设这些因子是；
6. ；
7. ；
8. 将h加入U
9. end while；
10. 将U中所有因子相乘，得到一个Q的因子h(Q)；
11. ；

* **关键代码**

变量消元算法：

1. **def** inference(factor\_list, query\_variables, ordered\_list\_of\_hidden\_variables, evidence\_list):
2. # 根据给定的evidence\_list限制所有已知的随机变量
3. **for** evkey, value **in** evidence\_list.items():
4. **for** i **in** range(len(factor\_list)):
5. **if** evkey **in** factor\_list[i].var\_list:
6. factor\_list[i] = factor\_list[i].restrict(evkey, str(value))
7. # 根据给定的消元顺序逐个消元
8. **for** var **in** ordered\_list\_of\_hidden\_variables:
9. currnet\_list = []
10. # 找到包含所要消除的变量的所有因子
11. **for** i **in** range(len(factor\_list)):
12. **if** var **in** factor\_list[i].var\_list:
13. currnet\_list.append(factor\_list[i])
14. # 将找到的所有因子从因子列表factor\_list中删除
15. **for** factor **in** currnet\_list:
16. factor\_list.remove(factor)
17. # 将所有找到的因子相乘再对于所要消除的变量求和得到新的因子
18. new\_factor = currnet\_list.pop()
19. **while** len(currnet\_list) != 0:
20. new\_factor = new\_factor.multiply(currnet\_list.pop())
21. # 将新得到的因子加入因子列表factor\_list
22. factor\_list.append(new\_factor.sum\_out(var))
23. **print**("RESULT: ")
24. res = factor\_list[0]
25. **for** factor **in** factor\_list[1:]:
26. res = res.multiply(factor)
27. total = sum(res.cpt.values())
28. res.cpt = {k: v / total **for** k, v **in** res.cpt.items()}
29. res.print\_inf()

限制操作：

1. **def** restrict(self, variable, value):
2. '''function that restricts a variable to some value in a given factor'''
3. # 找到变量的下标
4. index = self.var\_list.index(variable)
5. new\_var\_list = deepcopy(self.var\_list)
6. # 删除因子中已限制的变量
7. new\_var\_list.remove(variable)
8. new\_cpt = {}
9. **for** key, prob **in** self.cpt.items():
10. **if** key[index] == value:
11. # 新的条件概率表中只保留限制变量的值为value的概率
12. kkey = key[:index] + key[index + 1:]
13. new\_cpt[kkey] = prob
14. new\_node = Node('f' + str(new\_var\_list), new\_var\_list)
15. new\_node.set\_cpt(new\_cpt)
16. **return** new\_node

求和操作：

1. **def** sum\_out(self, variable):
2. '''function that sums out a variable given a factor'''
3. # 找到变量variable的下标
4. index = self.var\_list.index(variable)
5. new\_var\_list = deepcopy(self.var\_list)
6. # 删除因子中变量variable
7. new\_var\_list.remove(variable)
8. new\_cpt = {}
9. # 对变量variable求和
10. **for** key1, prob1 **in** self.cpt.items():
11. **for** key2, prob2 **in** self.cpt.items():
12. **if** key1[:index] == key2[:index] **and** key1[index + 1:] == key2[index + 1:] **and** key1[index] != key2[index]:
13. # 只对相互匹配的求和
14. new\_key = key1[:index] + key1[index + 1:]
15. **if** new\_key **not** **in** new\_cpt.keys():
16. # 对条件概率表遍历两遍导致每个结果会出现两次，只需保存一次即可
17. new\_cpt[new\_key] = prob1 + prob2
18. new\_node = Node('f' + str(new\_var\_list), new\_var\_list)
19. new\_node.set\_cpt(new\_cpt)
20. **return** new\_node

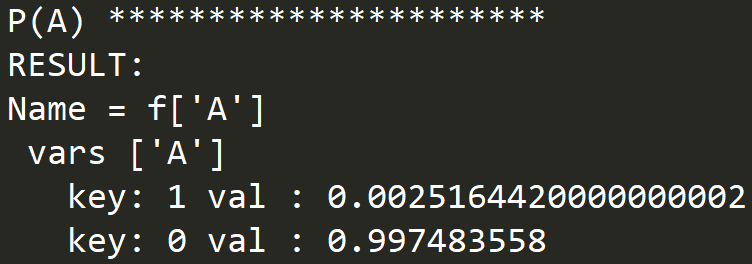
相乘操作：

1. **def** multiply(self, factor):
2. '''function that multiplies with another factor'''
3. index1 = 0
4. index2 = 0
5. new\_list = []
6. new\_cpt = {}
7. # 找到两个因子中相同的变量
8. **for** var **in** self.var\_list:
9. **if** var **in** factor.var\_list:
10. # 得到这个变量在两个因子中下标
11. index1 = self.var\_list.index(var)
12. index2 = factor.var\_list.index(var)
13. new\_list = self.var\_list + factor.var\_list[:index2] + factor.var\_list[index2 + 1:]
14. **break**
15. **for** key1, prob1 **in** self.cpt.items():
16. **for** key2, prob2 **in** factor.cpt.items():
17. **if** key1[index1] == key2[index2]:
18. # 遍历两个条件概率表，找到相匹配的key
19. new\_key = key1 + key2[:index2] + key2[index2 + 1:]
20. # 两个概率相乘以组成新的条件概率表
21. new\_cpt[new\_key] = prob1 \* prob2
22. new\_node = Node('f' + str(new\_list), new\_list)
23. new\_node.set\_cpt(new\_cpt)
24. **return** new\_node

## 实验结果及分析

* **实验结果展示**

计算P(A)：



计算P(B|J, ~M)：

