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。

2) 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h(A, B, C) = $f(A, B) \times g(B, C)$。

3) 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。

4) 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)。

5) 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 Z_j 执行如下操作：

a) 找到包含变量 Z_j 的所有因子$\{f_1, f_2, \dots, f_n\}$；

b) 先将找到的所有因子相乘，再对变量 Z_j 求和得到新的因子 g_j；

c) 从因子列表中删除之前找到的所有因子，再将新的因子 g_j 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Q ∪ E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. while $(p \neq \emptyset)$：

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$\{f_1, f_2, \dots, f_k\}$；

6. $g \leftarrow \prod_{i=1}^{k} f_i$；

7. $h \leftarrow \sum_Z g$；

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. return $\frac{h(Q)}{\sum_Q h(Q)}$；

· 关键代码

变量消元算法：

```
def inference(factor_list, query_variables, ordered_list_of_hidden_variables, evidence_list):
    # 根据给定的 evidence_list 限制所有已知的随机变量
    for evkey, value in evidence_list.items():
        for i in range(len(factor_list)):
            if evkey in factor_list[i].var_list:
                factor_list[i] = factor_list[i].restrict(evkey, str(value))
    # 根据给定的消元顺序逐个消元
    for var in ordered_list_of_hidden_variables:
        currnet_list = []
        # 找到包含所要消除的变量的所有因子
        for i in range(len(factor_list)):
            if var in factor_list[i].var_list:
                currnet_list.append(factor_list[i])
        # 将找到的所有因子从因子列表 factor_list 中删除
        for factor in currnet_list:
            factor_list.remove(factor)
        # 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
        new_factor = currnet_list.pop()
        while len(currnet_list) != 0:
            new_factor = new_factor.multiply(currnet_list.pop())
        # 将新得到的因子加入因子列表 factor_list
        factor_list.append(new_factor.sum_out(var))
```

```
    print("RESULT: ")
    res = factor_list[0]
    for factor in factor_list[1:]:
        res = res.multiply(factor)
    total = sum(res.cpt.values())
    res.cpt = {k: v / total for k, v in res.cpt.items()}
    res.print_inf()
```

限制操作：

```
def restrict(self, variable, value):
    '''function that restricts a variable to some value in a given factor'''
    # 找到变量的下标
    index = self.var_list.index(variable)
    new_var_list = deepcopy(self.var_list)
    # 删除因子中已限制的变量
    new_var_list.remove(variable)
    new_cpt = {}
    for key, prob in self.cpt.items():
        if key[index] == value:
            # 新的条件概率表中只保留限制变量的值为 value 的概率
            kkey = key[:index] + key[index + 1:]
            new_cpt[kkey] = prob
    new_node = Node('f' + str(new_var_list), new_var_list)
    new_node.set_cpt(new_cpt)
    return new_node
```

求和操作：

```
def sum_out(self, variable):
    '''function that sums out a variable given a factor'''
    # 找到变量 variable 的下标
    index = self.var_list.index(variable)
    new_var_list = deepcopy(self.var_list)
    # 删除因子中变量 variable
    new_var_list.remove(variable)
    new_cpt = {}
    # 对变量 variable 求和
    for key1, prob1 in self.cpt.items():
        for key2, prob2 in self.cpt.items():
            if key1[:index] == key2[:index] and key1[index + 1:] == key2[ind
ex + 1:] and key1[index] != key2[index]:
                # 只对相互匹配的求和
```

```
                    new_key = key1[:index] + key1[index + 1:]
                    if new_key not in new_cpt.keys():
                        # 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
                        new_cpt[new_key] = prob1 + prob2
        new_node = Node('f' + str(new_var_list), new_var_list)
        new_node.set_cpt(new_cpt)
        return new_node
```

相乘操作：

```
def multiply(self, factor):
    '''function that multiplies with another factor'''
    index1 = 0
    index2 = 0
    new_list = []
    new_cpt = {}
    # 找到两个因子中相同的变量
    for var in self.var_list:
        if var in factor.var_list:
            # 得到这个变量在两个因子中下标
            index1 = self.var_list.index(var)
            index2 = factor.var_list.index(var)
            new_list = self.var_list + factor.var_list[:index2] + factor.var
_list[index2 + 1:]
            break
    for key1, prob1 in self.cpt.items():
        for key2, prob2 in factor.cpt.items():
            if key1[index1] == key2[index2]:
                # 遍历两个条件概率表，找到相匹配的 key
                new_key = key1 + key2[:index2] + key2[index2 + 1:]
                # 两个概率相乘以组成新的条件概率表
                new_cpt[new_key] = prob1 * prob2
    new_node = Node('f' + str(new_list), new_list)
    new_node.set_cpt(new_cpt)
    return new_node
```

实验结果及分析

· 实验结果展示

计算 P(A)：

```
P(A) ********************
RESULT:
Name = f['A']
 vars ['A']
   key: 1 val : 0.0025164420000000002
   key: 0 val : 0.997483558
```

计算 P(B|J, ~M)：

```
P(B | J, ~M) **********************
RESULT:
Name = f['B']
 vars ['B']
   key: 0 val : 0.9948701418665987
   key: 1 val : 0.0051298581334013015
```