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

变量消元

（1）原理

变量消除原理：

- 变量消除法通过利用联合概率分布的分解来降低推理的复杂度。

给定一个联合概率分布:$P(A,B,C,D)=P(A)P(B|A)P(C|B)P(D|C)$

此时需要求$P(B)$，由概率定义可得到以下公式

$$P(B)=\sum_A\sum_C\sum_D P(A)P(B|A)P(C|B)P(D|C)$$

考虑到A,C,D之间的相关关系，将不相关的概率求和分解，得到以下公式：

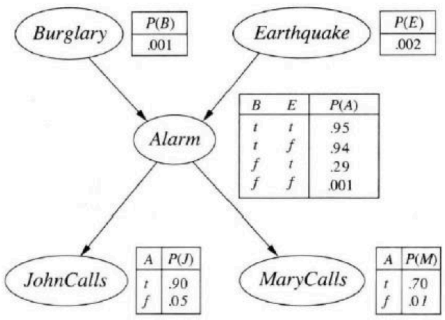
$$\begin{aligned}P(B)&=\sum_A P(A)P(B|A)\sum_C P(C|B)\sum_D P(D|C)\\&=\sum_A P(A)P(B|A)\sum_C P(C|B)\end{aligned}$$

通过变量消除，计算复杂度大大降低，提升推断效率。

（2）TASK关键代码（python带注释）

Task Burglary

- 网络图：



补全函数：

- **inference**：作出精确推断
  - 先对evidence进行赋值(调用restrict函数)，替换成新的factor
  - 对需要消除的变量依次操作
    - 对于含有要消除的变量(这里设为X)的factor，依次相乘(调用multiply函数)，并替换为新的factor
    - 将新的factor进行sum_out操作，即把含有X的概率对应相加，消除X。

```
def inference(factor_list, query_variables,
                  ordered_list_of_hidden_variables, evidence_list):
        #对证据进行限制
        for ev in evidence_list:
            # Your code here
            tmp_list = factor_list.copy()
            for i in range(len(factor_list)):
                if ev in factor_list[i].var_list:
                    tmp_list.append(factor_list[i].restrict(ev,
evidence_list[ev]))
                    tmp_list.remove(factor_list[i])
            factor_list = tmp_list
        #变量消除
        for var in ordered_list_of_hidden_variables:
            # Your code here
            first = False
            tmp_list = factor_list.copy()
            for i in range(len(factor_list)):
                if var in factor_list[i].var_list:
                    tmp_list.remove(factor_list[i])
                    if not first:
                        new_factor = factor_list[i]
                        first = True
                    else: new_factor = new_factor.multiply(factor_list[i])
            tmp_list.append(new_factor.sum_out(var))
            factor_list = tmp_list

        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()
```

- **same：判断两个factor的概率对应位置是否符合乘法的规则**

```
def same(self, f1, f2, list1, list2):
        for i, j in zip(list1, list2):
            if f1[i] != f2[j]:
                return False
        return True
```

- **multiply：两个factor相乘**
  - 先求得两个factor含有的相同变量的列表
  - 对于列表中的每个变量，在两个factor中找到概率表，对于每一项概率判断是否对应，进行相乘，生成新的概率。

$$f_3(A,B,C) = f_1(A,B) * f_2(B,C)$$

其中 $f_3(a,b,c) = f_1(a,b) * f_2(b,c)$，对应相乘。

```
def multiply(self, factor):
        '''function that multiplies with another factor'''
        # Your code here
        common = [i for i in self.var_list if i in factor.var_list]
        if not len(common): return self

        index1, index2 = [self.var_list.index(i) for i in common],
[factor.var_list.index(i) for i in common]
        new_list = self.var_list + [i for i in factor.var_list if i not in
self.var_list]
        new_cpt = {}
        for i in self.cpt:
            for j in factor.cpt:
                if self.same(i, j, index1, index2):
                    jStr = ''
                    for s in range(len(j)):
                        if s not in index2: jStr += j[s]
                    new_cpt[i + jStr] = self.cpt[i] * factor.cpt[j]
        new_node = Node('f' + str(new_list), new_list)
        new_node.set_cpt(new_cpt)
        return new_node
```

- **sum_out：对factor中某一个变量进行求和，从而消去它。**

- 假设需要消除的变量为A，对$f(A,B)$:

$$f^*(B)=\sum_A f(a,B)$$

```
def sum_out(self, variable):
        '''function that sums out a variable given a factor'''
        # Your code here
        new_var_list = [j for j in self.var_list if j != variable]
        var_index = self.var_list.index(variable)
        new_cpt = {}
        for i in self.cpt:
            new_cpt[i[:var_index] + i[var_index+1:]] =
new_cpt.get(i[:var_index] + i[var_index+1:],0) + self.cpt[i]
        new_node = Node('f' + str(new_var_list), new_var_list)
        new_node.set_cpt(new_cpt)
        return new_node
```

- **restrict：对变量进行赋值**
  - 假设需要消除的变量为A=a，对$f(A,B)$:

$$f^*(B)=f(a,B)$$

```
def restrict(self, variable, value):
        '''function that restricts a variable to some value
        in a given factor'''
        # Your code here
        new_var_list = [j for j in self.var_list if j != variable]
        var_index = self.var_list.index(variable)
        new_cpt = {}
        for i in self.cpt:
            if i[var_index] == chr(value + ord('0')):
                new_cpt[i[:var_index] + i[var_index+1:]] = self.cpt[i]
        new_node = Node('f' + str(new_var_list), new_var_list)
        new_node.set_cpt(new_cpt.copy())
        return new_node
```

（3）结果展示

- 输出：

```
print("P(A) **********")
VariableElimination.inference([B, E, A, J, M], ['A'], ['B', 'E', 'J', 'M'],
{})

print("P(B | J, ~M) **********")
VariableElimination.inference([B, E, A, J, M], ['B'], ['E', 'A'], {'J':1,
'M':0})
```

- 结果：

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

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