School of Data and Computer Science, Sun Yat-sen University

E09 Variable Elimination

18340149 孙新梦

2020 年 11 月 11 日

目录

1 VE

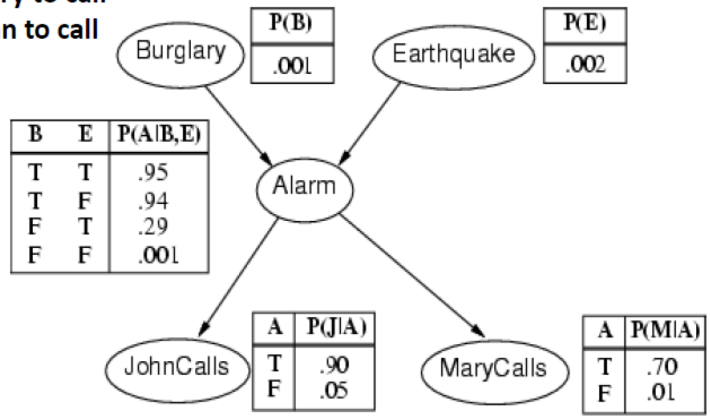
The burglary example is described as following:

- **A burglary can set the alarm off**
- **An earthquake can set the alarm off**
- **The alarm can cause Mary to call**
- **The alarm can cause John to call**

Note that these tables only provide the probability that Xi is true.

(E.g., Pr(A is true|B,E))

The probability that Xi is false is 1- these values



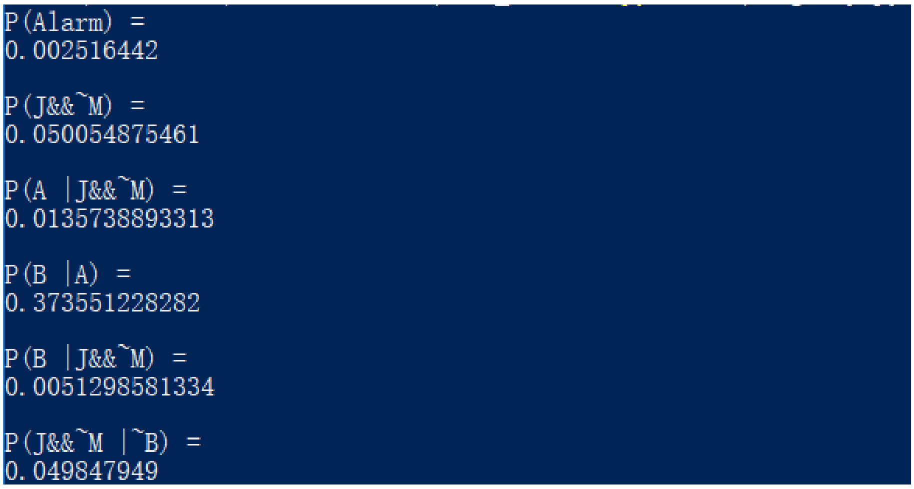
P(B)
.001

P(E)
.002

B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01



```
P(Alarm) =
0.002516442

P(J&&~M) =
0.050054875461

P(A |J&&~M) =
0.0135738893313

P(B |A) =
0.373551228282

P(B |J&&~M) =
0.0051298581334

P(J&&~M |~B) =
0.049847949
```

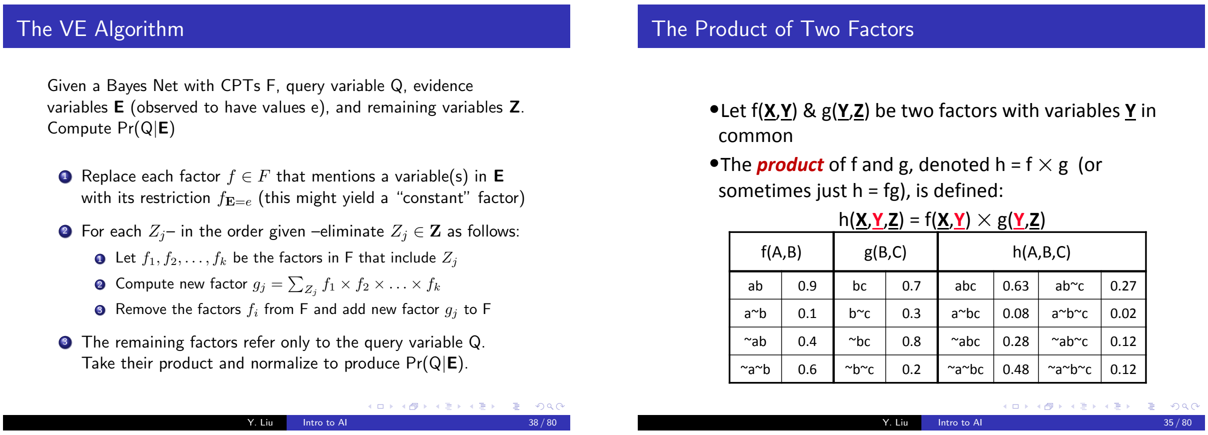
Here is a VE template for you to solve the burglary example:

```
class VariableElimination:
    @staticmethod
    def inference(factorList, queryVariables,
    orderedListOfHiddenVariables, evidenceList):
        for ev in evidenceList:
            #Your code here
        for var in orderedListOfHiddenVariables:
            #Your code here
        print "RESULT:"
        res = factorList[0]
        for factor in factorList[1:]:
            res = res.multiply(factor)
        total = sum(res.cpt.values())
        res.cpt = {k: v/total for k, v in res.cpt.items()}
        res.printInf()
    @staticmethod
    def printFactors(factorList):
        for factor in factorList:
            factor.printInf()
class Util:
    @staticmethod
    def to_binary(num, len):
        return format(num, '0' + str(len) + 'b')
class Node:
    def __init__(self, name, var_list):
        self.name = name
        self.varList = var_list
        self.cpt = {}
    def setCpt(self, cpt):
        self.cpt = cpt
    def printInf(self):
        print "Name = " + self.name
        print " vars " + str(self.varList)
        for key in self.cpt:
            print "   key: " + key + " val : " + str(self.cpt[key])
        print ""
    def multiply(self, factor):
        """function that multiplies with another factor"""
        #Your code here
        new_node = Node("f" + str(newList), newList)
        new_node.setCpt(new_cpt)
        return new_node
    def sumout(self, variable):
        """function that sums out a variable given a factor"""
        #Your code here
        new_node = Node("f" + str(new_var_list), new_var_list)
        new_node.setCpt(new_cpt)
        return new_node
    def restrict(self, variable, value):
        """function that restricts a variable to some value
        in a given factor"""
        #Your code here
        new_node = Node("f" + str(new_var_list), new_var_list)
        new_node.setCpt(new_cpt)
        return new_node
# create nodes for Bayes Net
B = Node("B", ["B"])
E = Node("E", ["E"])
A = Node("A", ["A", "B","E"])
J = Node("J", ["J", "A"])
M = Node("M", ["M", "A"])

# Generate cpt for each node
B.setCpt({'0': 0.999, '1': 0.001})
E.setCpt({'0': 0.998, '1': 0.002})
A.setCpt({'111': 0.95, '011': 0.05, '110':0.94,'010':0.06,
'101':0.29,'001':0.71,'100':0.001,'000':0.999})
J.setCpt({'11': 0.9, '01': 0.1, '10': 0.05, '00': 0.95})
M.setCpt({'11': 0.7, '01': 0.3, '10': 0.01, '00': 0.99})

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})
```

2 Task

- You should implement 4 functions: `inference`, `multiply`, `sumout` and `restrict`. You can turn to Figure 1 and Figure 2 for help.
- Please hand in a file named E09_YourNumber.pdf, and send it to ai_2020@foxmail.com



The VE Algorithm

Given a Bayes Net with CPTs F, query variable Q, evidence variables **E** (observed to have values e), and remaining variables **Z**. Compute Pr(Q|**E**)

1. Replace each factor $f \in F$ that mentions a variable(s) in **E** with its restriction $f_{\mathbf{E}=e}$ (this might yield a "constant" factor)
2. For each Z_j– in the order given –eliminate $Z_j \in \mathbf{Z}$ as follows:
   1. Let $f_1, f_2, \ldots, f_k$ be the factors in F that include Z_j
   2. Compute new factor $g_j = \sum_{Z_j} f_1 \times f_2 \times \ldots \times f_k$
   3. Remove the factors f_i from F and add new factor g_j to F
3. The remaining factors refer only to the query variable Q. Take their product and normalize to produce Pr(Q|**E**).

Y. Liu Intro to AI 38 / 80

The Product of Two Factors

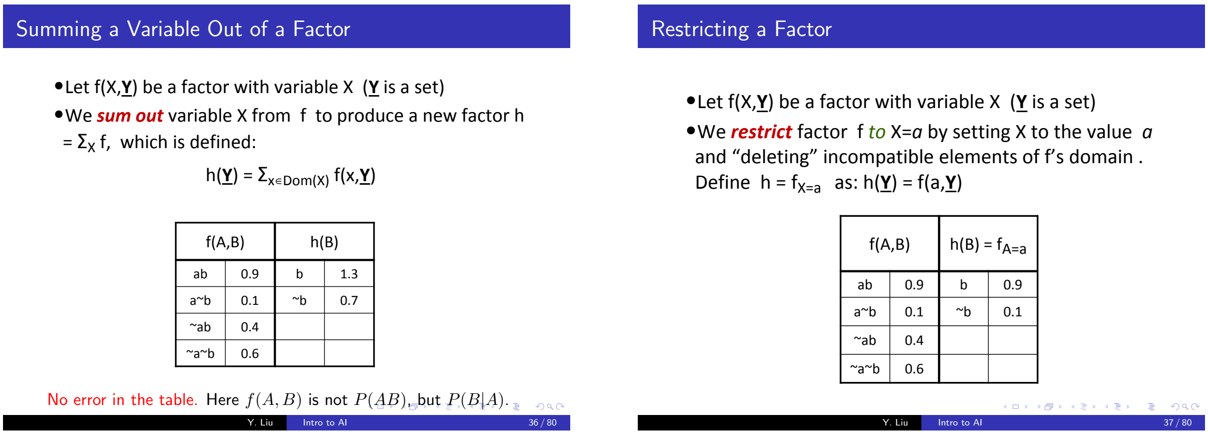
- Let f(**X**,**Y**) & g(**Y**,**Z**) be two factors with variables **Y** in common
- The ***product*** of f and g, denoted h = f × g (or sometimes just h = fg), is defined:

h(**X**,**Y**,**Z**) = f(**X**,**Y**) × g(**Y**,**Z**)

f(A,B)		g(B,C)		h(A,B,C)			
ab	0.9	bc	0.7	abc	0.63	ab~c	0.27
a~b	0.1	b~c	0.3	a~bc	0.08	a~b~c	0.02
~ab	0.4	~bc	0.8	~abc	0.28	~ab~c	0.12
~a~b	0.6	~b~c	0.2	~a~bc	0.48	~a~b~c	0.12

Y. Liu Intro to AI 35 / 80

图 1: VE and Product



Summing a Variable Out of a Factor

- Let f(X,**Y**) be a factor with variable X (**Y** is a set)
- We ***sum out*** variable X from f to produce a new factor h = Σ$_X$ f, which is defined:

h(**Y**) = Σ$_{x \in \text{Dom}(X)}$ f(x,**Y**)

f(A,B)		h(B)	
ab	0.9	b	1.3
a~b	0.1	~b	0.7
~ab	0.4		
~a~b	0.6		

No error in the table. Here $f(A, B)$ is not $P(AB)$, but $P(B|A)$.

Y. Liu Intro to AI 36 / 80

Restricting a Factor

- Let f(X,**Y**) be a factor with variable X (**Y** is a set)
- We ***restrict*** factor f *to* X=*a* by setting X to the value *a* and "deleting" incompatible elements of f's domain . Define h = f$_{X=a}$ as: h(**Y**) = f(a,**Y**)

f(A,B)		h(B) = f$_{A=a}$	
ab	0.9	b	0.9
a~b	0.1	~b	0.1
~ab	0.4		
~a~b	0.6		

Y. Liu Intro to AI 37 / 80

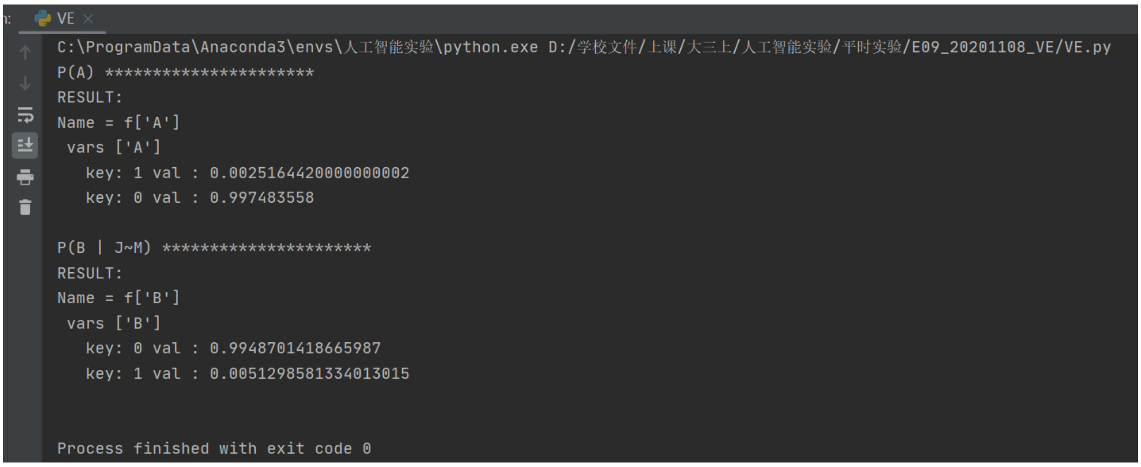
图 2: Sumout and Restrict

3 Codes and Results

```python
"""孙新梦


    18340149
    AI E09
"""


class VariableElimination:
    @staticmethod
    def inference(factorList, queryVariables,
                  orderedListOfHiddenVariables, evidenceList):

        # step1.restrict. 把每个因子里面含有证据的替换取值，创建新的表
        for ev in evidenceList:  # 遍历证据
            for factor in factorList:  # 遍历所有因子找出含有证据的相关因子
                if ev in factor.varList:
                    # 使用函数替换变量的值并创建新的restrictcpt
                    factorList.append(factor.restrict(ev, evidenceList[ev]))
                    # 删除原先的factor
                    factorList.remove(factor)

        # step2.elimation按照给定的顺序进行变量消除算法.
        for var in orderedListOfHiddenVariables:  # 消除顺序遍历变量
            # 含有目标变量的要加入elimationList
            elimationList = list(filter(lambda factor: var in factor.varList, factorList))

            new_var = elimationList[0]
            for eli in elimationList:
                for factor in factorList:
                    if factor.name == eli.name:
                        factorList.remove(factor)
                # 第一个之后乘起来
                if eli != elimationList[0]:
                    new_var = new_var.multiply(eli)
            new_var = new_var.sumout(var)   #加和
            factorList.append(new_var)

        #计算结果
        print("RESULT:")
        res = factorList[0]
        for factor in factorList[1:]:
            res = res.multiply(factor)
        total = sum(res.cpt.values())
        res.cpt = {k: v / total for k, v in res.cpt.items()}
        res.printInf()

    @staticmethod
    def printFactors(factorList):
        for factor in factorList:
            factor.printInf()


class Util:
    @staticmethod
    def to_binary(num, len):
        return format(num, '0' + str(len) + 'b')


class Node:
    def __init__(self, name, var_list):
        self.name = name
        self.varList = var_list
        self.cpt = {}

    def setCpt(self, cpt):
        self.cpt = cpt

    def printInf(self):
        print("Name␣=␣" + self.name)
        print("␣vars␣" + str(self.varList))
        for key in self.cpt:
            print("␣␣␣key:␣" + key + "␣val␣:␣" + str(self.cpt[key]))
        print("")

    def multiply(self, factor):
        """function that multiplies with another factor"""
        # Your code here
        # 两个节点的表相乘
        newlist = [var for var in self.varList]
        new_cpt = {}

        # 计算公共变量的索引
        index1 = []
        index2 = []
        for var1 in self.varList:
            for var2 in factor.varList:
                if var1 == var2:
                    index1.append(self.varList.index(var1))
                    index2.append(factor.varList.index(var2))
                else:
                    newlist.append(var2)

        for k1, v1 in self.cpt.items():
            for k2, v2 in factor.cpt.items():
                flag = True  # 表示两个项能否做乘积，要求之前记录的公共部分相同都为之类00
                for i in range(len(index1)):
                    if k1[index1[i]] != k2[index2[i]]:
                        flag = False
                        break
                    if flag:
                        new_key = k1  # 以为蓝本，添加除了相同的项之外的符号k1k2
                        for i in range(len(k2)):
                            if i in index2:
                                continue
                            new_key += k2[i]

                    new_cpt[new_key] = v1 * v2

        theList = []
        for letter in newlist:
            theList.append(letter)
        #print(theList)

        #new_node = Node(”f” + str(newList), newList)  # 为两个节点的合并去重newlistlist
        new_node = Node(”f”+str(theList), theList)
        new_node.setCpt(new_cpt)  # 新的为两个产生的cptnewcpt

        return new_node

    def sumout(self, variable):
        ”””function that sums out a variable given a factor”””
        # Your code here
        new_var_List = [var for var in self.varList]
        new_var_List.remove(variable)
        new_cpt = {}

        # index需要被加和的遍历的下标:
        index = self.varList.index(variable)

        # 遍历字典的键值对，把目标变量相同的加在一起cpt
        for k, v in self.cpt.items():
            # 没有记录过这一项新增上来，
            if k[:index] + k[index + 1:] not in new_cpt.keys():
                new_cpt[k[:index] + k[index + 1:]] = v
            else:  # 有了加和在原来基础上
                new_cpt[k[:index] + k[index + 1:]] += v

        theList = []
        for letter in new_var_List:
            theList.append(letter)
        #print(theList)

        new_node = Node(”f” + str(theList), theList)
        new_node.setCpt(new_cpt)
        return new_node

    def restrict(self, variable, value):
        ”””function that restricts a variable to some value
        in a given factor”””
        # Your code here
        new_var_List = [var for var in self.varList]
        new_var_List.remove(variable)
        new_cpt = {}

        # index需要被限制的变量下标:
        index = self.varList.index(variable)

        # 把和相同的的值存放进新字典valuevariable
        for k, v in self.cpt.items():
            if k[index] == str(value):
                new_cpt[k[:index] + k[index + 1:]] = v

        theList = []
        for letter in new_var_List:
            theList.append(letter)
        #print(theList)

        new_node = Node("f" + str(theList), theList)
        new_node.setCpt(new_cpt)
        return new_node


# create nodes for Bayes Net
B = Node("B", ["B"])
E = Node("E", ["E"])
A = Node("A", ["A", "B", "E"])
J = Node("J", ["J", "A"])
M = Node("M", ["M", "A"])

# Generate cpt for each node
B.setCpt({'0': 0.999, '1': 0.001})
E.setCpt({'0': 0.998, '1': 0.002})
A.setCpt({'111': 0.95, '011': 0.05, '110': 0.94, '010': 0.06,
          '101': 0.29, '001': 0.71, '100': 0.001, '000': 0.999})
J.setCpt({'11': 0.9, '01': 0.1, '10': 0.05, '00': 0.95})
M.setCpt({'11': 0.7, '01': 0.3, '10': 0.01, '00': 0.99})

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})
```

结果截图



```
C:\ProgramData\Anaconda3\envs\人工智能实验\python.exe D:/学校文件/上课/大三上/人工智能实验/平时实验/E09_20201108_VE/VE.py
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


Process finished with exit code 0
```

4　体会感想

本次实验实现VE算法的过程，是对理论课的内容以及上次的理论作业一个更深刻的体会。VE算法在计算概率的时候非常常用，算法思想是首先把证据的值替换进去，之后按照给出的顺序逐步做乘积加和的消除变量，得到新的factor替换使用的factor，之后剩下的变量归一化就是所求概率。