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

E09 Variable Elimination

18340215 张天祎

2020 年 11 月 26 日

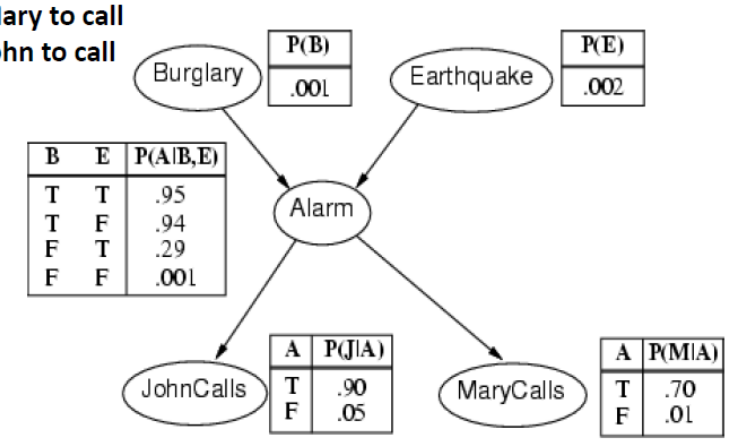
目录

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(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]
```

```
11          for factor in factorList[1:]:
12              res = res.multiply(factor)
13          total = sum(res.cpt.values())
14          res.cpt = {k: v/total for k, v in res.cpt.items()}
15          res.printInf()
16      @staticmethod
17      def printFactors(factorList):
18          for factor in factorList:
19              factor.printInf()
20  class Util:
21      @staticmethod
22      def to_binary(num, len):
23          return format(num, '0' + str(len) + 'b')
24  class Node:
25      def __init__(self, name, var_list):
26          self.name = name
27          self.varList = var_list
28          self.cpt = {}
29      def setCpt(self, cpt):
30          self.cpt = cpt
31      def printInf(self):
32          print "Name = " + self.name
33          print " vars " + str(self.varList)
34          for key in self.cpt:
35              print "    key: " + key + " val : " + str(self.cpt[key])
36          print ""
37      def multiply(self, factor):
38          """function that multiplies with another factor"""
39          #Your code here
40          new_node = Node("f" + str(newList), newList)
41          new_node.setCpt(new_cpt)
42          return new_node
43      def sumout(self, variable):
44          """function that sums out a variable given a factor"""
45          #Your code here
46          new_node = Node("f" + str(new_var_list), new_var_list)
47          new_node.setCpt(new_cpt)
```

```
48          return new_node
49      def restrict(self, variable, value):
50          """function that restricts a variable to some value
51          in a given factor"""
52          #Your code here
53          new_node = Node("f" + str(new_var_list), new_var_list)
54          new_node.setCpt(new_cpt)
55          return new_node
56  # create nodes for Bayes Net
57  B = Node("B", ["B"])
58  E = Node("E", ["E"])
59  A = Node("A", ["A", "B","E"])
60  J = Node("J", ["J", "A"])
61  M = Node("M", ["M", "A"])
62
63  # Generate cpt for each node
64  B.setCpt({'0': 0.999, '1': 0.001})
65  E.setCpt({'0': 0.998, '1': 0.002})
66  A.setCpt({'111': 0.95, '011': 0.05, '110':0.94,'010':0.06,
67  '101':0.29,'001':0.71,'100':0.001,'000':0.999})
68  J.setCpt({'11': 0.9, '01': 0.1, '10': 0.05, '00': 0.95})
69  M.setCpt({'11': 0.7, '01': 0.3, '10': 0.01, '00': 0.99})
70
71  print "P(A) **********"
72  VariableElimination.inference([B,E,A,J,M], ['A'], ['B', 'E', 'J','M'], {})
73
74  print "P(B | J~M) **********"
75  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**).

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

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**) = $\Sigma_{x \in Dom(X)}$ 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 $P(B|A)$.

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		

图 2: Sumout and Restrict

3 Codes and Results

```
1  import copy
2  class VariableElimination:
3      def printFactors(factorList):
4          for factor in factorList:
5              factor.printInf()
6
7      def inference(factorList, queryVariables,
8      orderedListOfHiddenVariables, evidenceList):
9          for ev in evidenceList:
10             #Your code here
11             for pos, factor in enumerate(factorList):
12                 factorList[pos] = factor.restrict(ev,
13                 # 注意这里需要转成str
```

```
14                  str(evidenceList[ev]))
15          print()
16          for var in orderedListOfHiddenVariables:
17              #Your code here
18              temList= []
19              for pos, factor in enumerate(factorList):
20                  if var in factor.varList:
21                      temList.append(factor)
22              for factor in temList:
23                  factorList.remove(factor)
24              curr = temList[0]
25              for i in temList[1:]:
26                  curr = curr.multiply(i)
27              curr = curr.sumout(var)
28              factorList.append(curr)
29          print("RESULT:")
30          res = factorList[0]
31          for factor in factorList[1:]:
32              res = res.multiply(factor)
33          total = sum(res.cpt.values())
34          res.cpt = {k: v/total for k, v in res.cpt.items()}
35          res.printInf()
36
37  class Util:
38      def to_binary(num, len):
39          return "{0:0>{1:}b}".format(num,len)
40      def lfind(l, val):
41          for pos, i in enumerate(l):
42              if i==val:
43                  return pos
44          return -1
45
46
47  class Node:
48      def __init__(self, name, var_list):
49          self.name = name
50          self.varList = var_list
```

```
51          self.cpt = {}
52      def setCpt(self, cpt):
53          self.cpt = cpt
54      def printInf(self):
55          print("Name = " + self.name)
56          print("  vars " + str(self.varList))
57          for key in self.cpt:
58              print("    key: " + str(key) + " val : " + str(self.cpt[key]))
59          print()
60      def multiply(self, factor):
61          """function that multiplies with another factor"""
62          #Your code here
63          newList = list(set(self.varList+factor.varList))
64          new_cpt = {}
65          length = len(newList)
66          len1 = len(self.varList)
67          len2 = len(factor.varList)
68          map1 = {}
69          map2 = {}
70          for pos, i in enumerate(self.varList):
71              map1[pos] = Util.lfind(newList, i)
72          for pos, i in enumerate(factor.varList):
73              map2[pos] = Util.lfind(newList, i)
74          for i in range(pow(2, length)):
75              curr = Util.to_binary(i, length)
76              str1 = ''
77              str2 = ''
78              for i in range(len1):
79                  str1 += curr[map1[i]]
80              for i in range(len2):
81                  str2 += curr[map2[i]]
82              new_cpt[curr] = self.cpt[str1]*factor.cpt[str2]
83          new_node = Node("f" + str(newList), newList)
84          new_node.setCpt(new_cpt)
85          return new_node
86      def sumout(self, variable):
87          """function that sums out a variable given a factor"""
```

```
88          #Your code here
89          node0 = self.restrict(variable,'0')
90          node1 = self.restrict(variable,'1')
91          new_var_list = copy.deepcopy(node0.varList)
92          new_cpt = copy.deepcopy(node0.cpt)
93          for key,p in new_cpt.items():
94              new_cpt[key] += node1.cpt[key]
95
96          new_node = Node("f" + str(new_var_list), new_var_list)
97          new_node.setCpt(new_cpt)
98          return new_node
99      def restrict(self, variable, value):
100         """function that restricts a variable to some value
101         in a given factor"""
102         #Your code here
103         if variable not in self.varList:
104             return self
105         new_var_list = copy.deepcopy(self.varList) # 这里必须要用深复制
106         new_var_list.remove(variable)
107         length = len(new_var_list)
108         new_cpt = {}
109         pos = Util.lfind(self.varList, variable)
110         for i in range(pow(2, length)):
111             new_cpt[Util.to_binary(i, length)] = 0
112         for key, p in self.cpt.items():
113             if key[pos] == value:
114                 key = key[0:pos]+key[pos+1:]
115                 new_cpt[key] = p
116         new_node = Node("f" + str(new_var_list), new_var_list)
117         new_node.setCpt(new_cpt)
118         return new_node
119
120 # create nodes for Bayes Net
121 B = Node("B", ["B"])
122 E = Node("E", ["E"])
123 A = Node("A", ["A", "B","E"])
124 J = Node("J", ["J", "A"])
```

```
125 M = Node("M", ["M", "A"])
126
127 # Generate cpt for each node
128 B.setCpt({'0': 0.999, '1': 0.001})
129 E.setCpt({'0': 0.998, '1': 0.002})
130 A.setCpt({'111': 0.95, '011': 0.05, '110':0.94,'010':0.06,
131 '101':0.29,'001':0.71,'100':0.001,'000':0.999})
132 J.setCpt({'11': 0.9, '01': 0.1, '10': 0.05, '00': 0.95})
133 M.setCpt({'11': 0.7, '01': 0.3, '10': 0.01, '00': 0.99})
134
135 print("P(A) **********")
136 VariableElimination.inference([B,E,A,J,M], ['A'], ['B','E','J','M'], {})
137
138 print("P(J~M) **********")
139 VariableElimination.inference([B,E,A,J,M], ['J','M'], ['B','E','A'], {})
140
141 print("P(A | J~M) **********")
142 VariableElimination.inference([B,E,A,J,M], ['A'], ['B','E'], {'J':1,'M':0})
143
144 print("P(B | A) **********")
145 VariableElimination.inference([B,E,A,J,M], ['B'], ['E','J','M'], {'A':1})
146
147 print("P(B | J~M) **********")
148 VariableElimination.inference([B,E,A,J,M], ['B'], ['E','A'], {'J':1,'M':0})
149
150 print("P(J~M | ~B) **********")
151 VariableElimination.inference([B,E,A,J,M], ['J','M'], ['E','A'], {'B':0})
```

结果如下：

```
P(A) ********************

RESULT:
Name = f['A']
 vars ['A']
   key: 0 val : 0.997483558
   key: 1 val : 0.0025164420000000002

P(J~M) **********************

RESULT:
Name = f['M', 'J']
 vars ['M', 'J']
   key: 00 val : 0.9382087795590001
   key: 01 val : 0.05005487546100001
   key: 10 val : 0.009652244741000002
   key: 11 val : 0.0020841002390000005

P(A | J~M) **********************

RESULT:
Name = f['A']
 vars ['A']
   key: 0 val : 0.9864261106686925
   key: 1 val : 0.013573889331307633
```

```
P(B | A) **********************

RESULT:
Name = f['B']
 vars ['B']
   key: 0 val : 0.626448771718164
   key: 1 val : 0.373551228281836

P(B | J~M) **********************

RESULT:
Name = f['B']
 vars ['B']
   key: 0 val : 0.9948701418665987
   key: 1 val : 0.0051298581334013015

P(J~M | ~B) **********************

RESULT:
Name = f['J', 'M']
 vars ['J', 'M']
   key: 00 val : 0.939063231
   key: 01 val : 0.009595469
   key: 10 val : 0.049847948999999996
   key: 11 val : 0.001493351
```

在各个结果表中寻找对应的概率，和问题原结果是吻合的。