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

E10 Variable Elimination

17341137 Zhenpeng Song

November 15, 2019

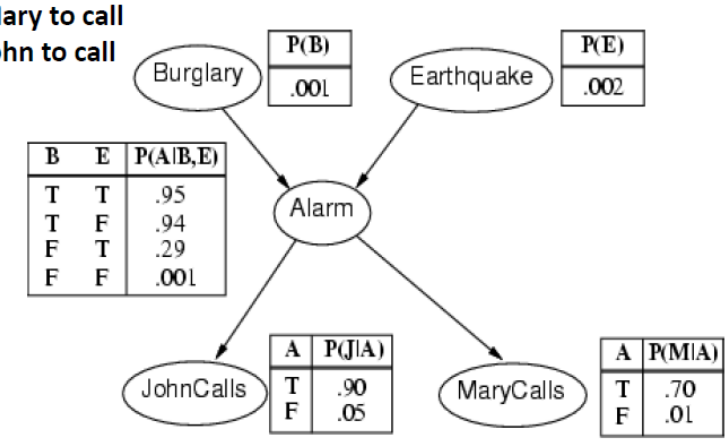
Contents

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
```

```
9           print "RESULT:"
10          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_201901@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

Figure 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 $P(B|A)$.

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

Figure 2: Sumout and Restrict

3 Codes and Results

3.1 Codes

3.1.1 VE.py

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

        # Restriction
        for evidence in evidenceList:
            for factor in factorList:
                if evidence in factor.varList:
                    factorList.append(factor.restrict(
                        evidence, evidenceList[evidence]))
                    factorList.remove(factor)

        # Elimination
        for variable in orderdListOfHiddenVariables:
            # Those factors, whose variable list
            # contains target variable should be
            # added into elimination list.
            eliminationList = list(
                filter(lambda factor: variable in factor.varList,
                       factorList))

            new_var = eliminationList[0]
            for e in eliminationList:
                for i in factorList:
                    if i.name == e.name:
                        factorList.remove(i)
                if not e == eliminationList[0]:
```

```
29                      new_var = new_var.multiply(e)
30
31              new_var = new_var.sumout(variable)
32              factorList.append(new_var)
33
34          # Calculate the Result
35          print("RESULT:")
36          res = factorList[0]
37          for factor in factorList[1:]:
38              res = res.multiply(factor)
39
40          total = sum(res.cpt.values())
41          res.cpt = {k: v/total for k, v in res.cpt.items()}
42          res.printInf()
43
44      @staticmethod
45      def printFactors(factorList):
46          for factor in factorList:
47              factor.printInf()
48
49
50  class Util:
51      @staticmethod
52      def to_binary(num, len):
53          return format(num, '0' + str(len) + 'b')
54
55
56  class Node:
57      def __init__(self, name, var_list):
58          self.name = name
59          self.varList = var_list
60          self.cpt = {}
61
```

```
62      def setCpt(self, cpt):
63          self.cpt = cpt
64
65      def printInf(self):
66          print("Name = " + self.name)
67          print(" vars " + str(self.varList))
68          for key in self.cpt:
69              print("   key: " + key + " val : " + str(self.cpt[key]))
70          print()
71
72      def multiply(self, factor):
73          # Function to multiplies with another factor.
74          newList = [var for var in self.varList]
75          new_cpt = {}
76
77          # To store the same variables of two factors
78          idx1 = []
79          idx2 = []
80          for var1 in self.varList:
81              for var2 in factor.varList:
82                  if var1 == var2:
83                      idx1.append(self.varList.index(var1))
84                      idx2.append(factor.varList.index(var2))
85                  else:
86                      newList.append(var2)
87
88          # multiplying
89          for k1, v1 in self.cpt.items():
90              for k2, v2 in factor.cpt.items():
91                  # flag to determine which two cpts
92                  # should be multiplied together
93                  flag = True
94                  for i in range(len(idx1)):
```

```
95                      # Check value of each variable
96                      if k1[idx1[i]] != k2[idx2[i]]:
97                          flag = False
98                          break
99                  if flag:
100                     # new key in new cpt is
101                     # built sequentially
102                     new_key = k1
103                     for i in range(len(k2)):
104                         if i in idx2: continue
105                         new_key += k2[i]
106                     new_cpt[new_key] = v1 * v2
107
108         new_node = Node("f" + str(newList), newList)
109         new_node.setCpt(new_cpt)
110         return new_node
111
112     def sumout(self, variable):
113         # Fuction to sum out a variable given a factor.
114         new_var_list = [var for var in self.varList]
115         new_var_list.remove(variable)
116         new_cpt = {}
117
118         # To store the index of the variable to sum out
119         idx = self.varList.index(variable)
120
121         # For each value of the target variable,
122         # sum it up to build a new cpt dict.
123         for k, v in self.cpt.items():
124             if k[:idx] + k[idx+1:] not in new_cpt.keys():
125                 new_cpt[k[:idx] + k[idx+1:]] = v
126             else: new_cpt[k[:idx] + k[idx+1:]] += v
127
```

```
128         new_node = Node("f" + str(new_var_list), new_var_list)
129         new_node.setCpt(new_cpt)
130         return new_node
131 
132     def restrict(self, variable, value):
133         # Function to restrict a variable
134         # in a given factor to some value.
135         new_var_list = [i for i in self.varList]
136         new_var_list.remove(variable)
137         new_cpt = {}
138 
139         # To store the index of the variable to restrict
140         idx = self.varList.index(variable)
141 
142         # Only choose the same value as the parameter
143         # to build the new cpt
144         for k, v in self.cpt.items():
145             if k[idx] == str(value):
146                 new_cpt[k[:idx] + k[idx+1:]] = v
147 
148         new_node = Node("f" + str(new_var_list), new_var_list)
149         new_node.setCpt(new_cpt)
150         return new_node
```

3.1.2 Sol_10.py

```
from VE import *
# Nodes for Bayes Net
# B      J
#  \.   /'
#    A
#  /'   \.
# E      M
B = Node("B", ["B"])
E = Node("E", ["E"])
A = Node("A", ["A", "B", "E"])
J = Node("J", ["J", "A"])
M = Node("M", ["M", "A"])

# Cpts 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})

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

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

3.2 Results

```
λ python Sol_10.py
P(A) **********
RESULT:
Name = f['A']
 vars ['A']
   key: 1 val : 0.0025164420000000002
   key: 0 val : 0.997483558

P(J && ~M) **********
RESULT:
Name = f['J', 'A', 'A']
 vars ['J', 'A', 'A']
   key: 1 val : 0.05064931327459069
   key: 0 val : 0.9493506867254093

P(A | J && ~M) **********
RESULT:
Name = f['A']
 vars ['A']
   key: 1 val : 0.013573889331307633
   key: 0 val : 0.9864261106686925
```

Figure 3: res1

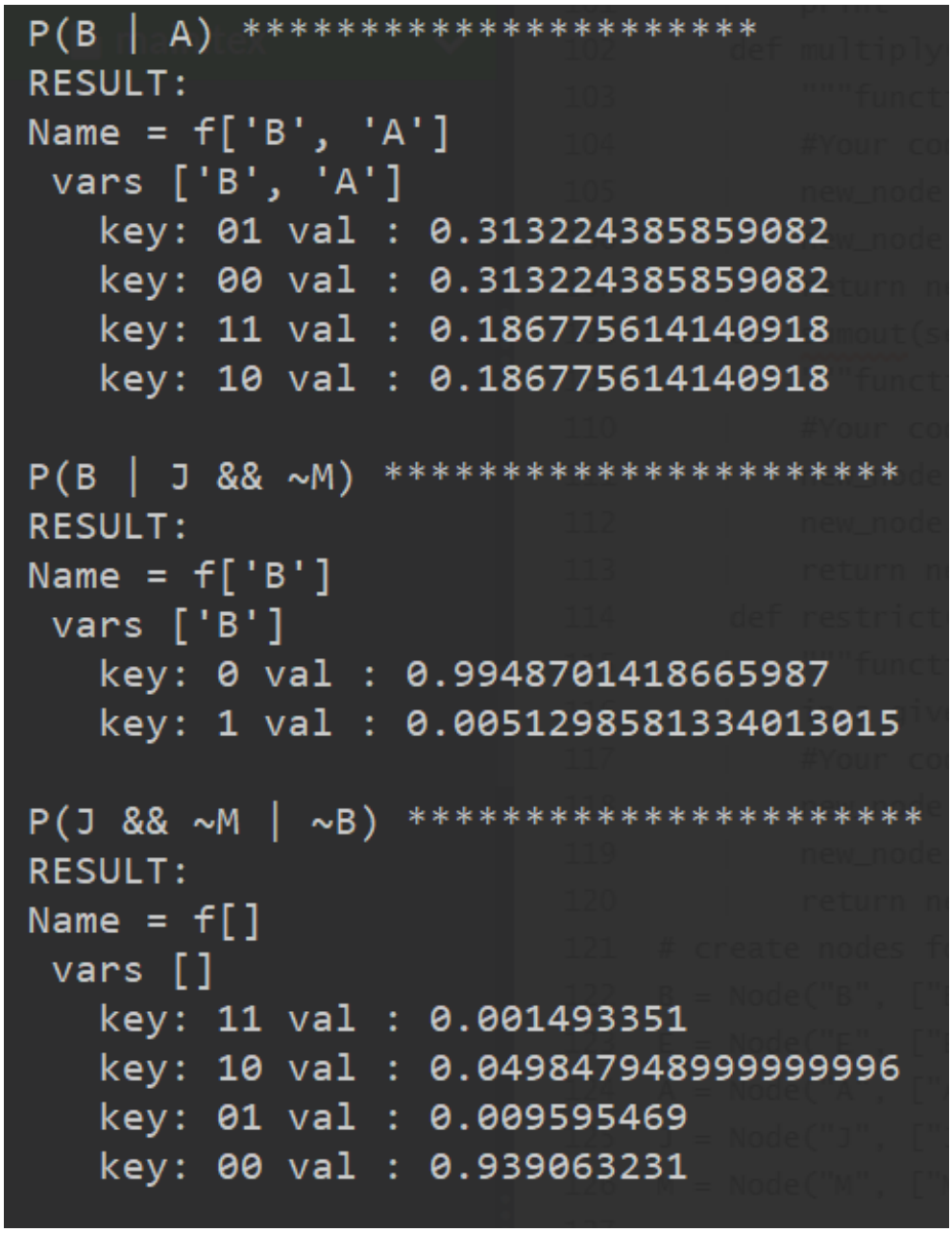


Figure 4: res2