

# E10 Variable Elimination

---

17341015 Hongzheng Chen

November 14, 2019

## Contents

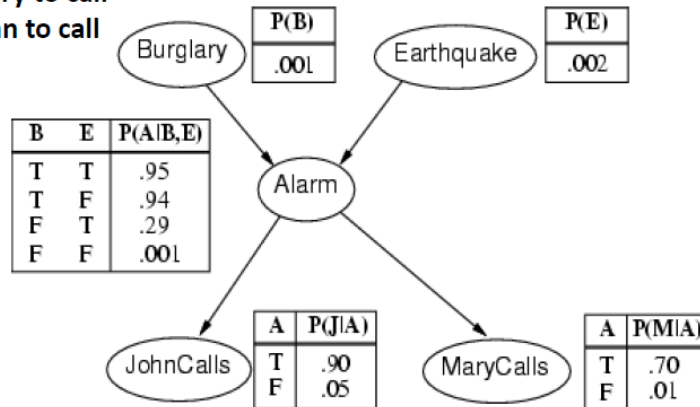
<b>1</b>	<b>VE</b>	<b>2</b>
<b>2</b>	<b>Task</b>	<b>4</b>
<b>3</b>	<b>Codes and Results</b>	<b>4</b>

# 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  $X_i$  is true.  
(E.g.,  $Pr(A \text{ is true} | B, E)$ )  
The probability that  $X_i$  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]
        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 `E10_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  $\Pr(Q|E)$

- Replace each factor  $f \in F$  that mentions a variable(s) in  $E$  with its restriction  $f_{E=e}$  (this might yield a "constant" factor)
- For each  $Z_j$  in the order given –eliminate  $Z_j \in Z$  as follows:
  - Let  $f_1, f_2, \dots, f_k$  be the factors in  $F$  that include  $Z_j$
  - Compute new factor  $g_j = \sum_{Z_j} f_1 \times f_2 \times \dots \times f_k$
  - Remove the factors  $f_i$  from  $F$  and add new factor  $g_j$  to  $F$
- 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  $f(\underline{X}, \underline{Y})$  &  $g(\underline{Y}, \underline{Z})$  be two factors with variables  $\underline{Y}$  in common
- The **product** of  $f$  and  $g$ , denoted  $h = f \times g$  (or sometimes just  $h = fg$ ), is defined:

$$h(\underline{X}, \underline{Y}, \underline{Z}) = f(\underline{X}, \underline{Y}) \times g(\underline{Y}, \underline{Z})$$

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

Figure 1: VE and Product

### Summing a Variable Out of a Factor

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

$$h(\underline{Y}) = \sum_{\underline{X} \in \text{Dom}(\underline{X})} f(\underline{X}, \underline{Y})$$

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  $P(A, B)$ , but  $P(B|A)$ .

### Restricting a Factor

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

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		

Figure 2: Sumout and Restrict

## 3 Codes and Results

Codes are listed below. Notice **Python 3** is used!

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

```

for ev in evidenceList:
    for i,node in enumerate(factorList):
        if ev in node.varList:
            factorList[i] = node.restrict(ev,evidenceList[ev])
for var in orderedListOfHiddenVariables:
    # for node in factorList:
    #     print(node.name,end=" ")
    # print()
    newFactorList = []
    for node in factorList:
        if var in node.varList:
            newFactorList.append(node)
    res = newFactorList[0]
    factorList.remove(res)
    for factor in newFactorList[1:]:
        res = res.multiply(factor)
        factorList.remove(factor)
    res = res.sumout(var)
    factorList.append(res)
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()

def get_new_cpt_var(num):
    if num == 0: # be careful!
        return [""]
    cpt_var = []
    format_spec = "{0:0" + str(num) + "b}"
    for i in range(2**num):
        cpt_var.append(format_spec.format(i))
    return cpt_var

class Node:
    def __init__(self, name, var_list):
        self.name = name
        # the first var is itself, others are dependency
        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"""
    var_list_1 = self.varList.copy()
    var_list_2 = factor.varList.copy()
    new_var_list = list(set(var_list_1 + var_list_2)) # take a union
    new_cpt = {}
    cpt_var = get_new_cpt_var(len(new_var_list))
    for var in cpt_var:
        var_dict = {}
        for i,v in enumerate(new_var_list):
            var_dict[v] = var[i]
        item = ""
        for var1 in self.varList:
            item += var_dict[var1]
        f1 = self.cpt[item]
        item = ""
        for var2 in factor.varList:
            item += var_dict[var2]
        f2 = factor.cpt[item]
        new_cpt[var] = f1 * f2
    new_node = Node("f" + str(new_var_list), new_var_list)
    new_node.setCpt(new_cpt)
    # print("{} multiply {} -> {}".format(self.name, factor.name, new_node.name))
    return new_node

def sumout(self, variable):
    """function that sums out a variable given a factor"""
    index = self.varList.index(variable)
    new_var_list = self.varList.copy()
    new_var_list.remove(variable)
    cpt_var = get_new_cpt_var(len(new_var_list))
    new_cpt = {}
    for var in cpt_var:
        sumup = 0
        for curr in ["0", "1"]:
            origin_var = var[:index] + curr + var[index:]
            sumup += self.cpt[origin_var]
        new_cpt[var] = sumup
    new_node = Node("f" + str(new_var_list), new_var_list)
    new_node.setCpt(new_cpt)
    # print("{} sumout {} -> {}".format(self.name, variable, new_node.name))
    return new_node

def restrict(self, variable, value):
    """function that restricts a variable to some value
    in a given factor"""
    index = self.varList.index(variable)
    new_var_list = self.varList.copy()
    new_var_list.remove(variable)
    cpt_var = get_new_cpt_var(len(new_var_list))
    new_cpt = {}
    for var in cpt_var:
        origin_var = var[:index] + str(value) + var[index:]
        new_cpt[var] = self.cpt[origin_var]

```

```

        new_node = Node("f" + str(new_var_list), new_var_list)
        new_node.setCpt(new_cpt)
        # print("{} restricts {} to {} -> {}".format(self.name,variable,value,new_node.name
        ↪ ))
        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(J ~M) *****")
VariableElimination.inference([B,E,A,J,M], ['J','M'], ['B','E','A'], {})

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

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

```

```
chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/ArtificialIntelligence/E10_VE
chhzh123@DESKTOP-PV2UBJL:/mnt/d/Assignments/ArtificialIntelligence/E10_VE$ python VE.py
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 : 0.013573889331307631

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['M', 'J']
vars ['M', 'J']
key: 00 val : 0.939063231
key: 01 val : 0.049847948999999996
key: 10 val : 0.009595469
key: 11 val : 0.001493351

chhzh123@DESKTOP-PV2UBJL:/mnt/d/Assignments/ArtificialIntelligence/E10_VE$
```