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

E09 Variable Elimination

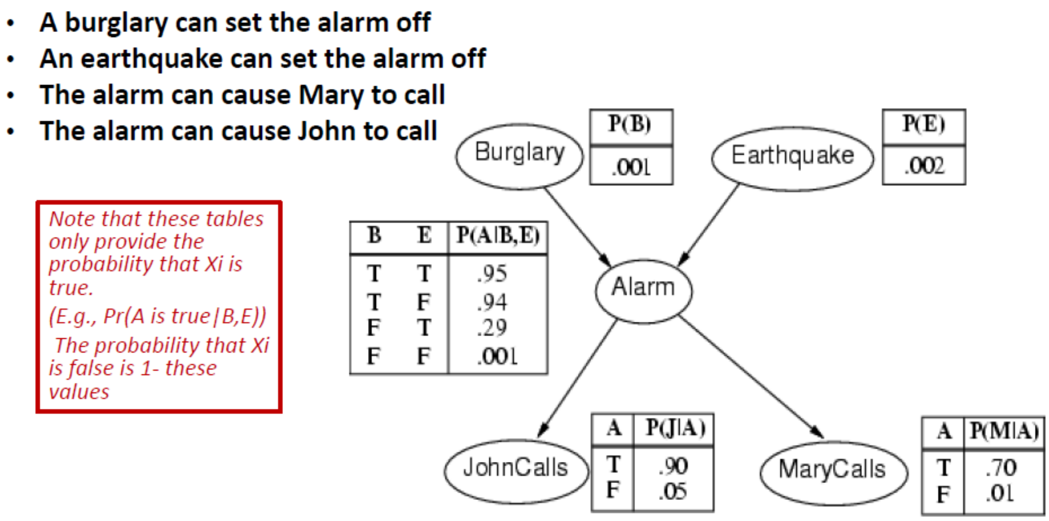
18340013 Conghao Chen

November 11, 2020

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

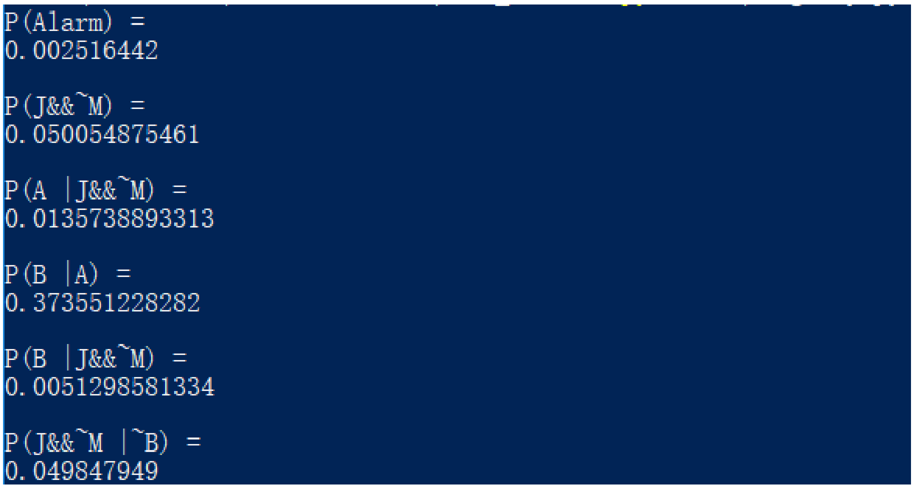
P(B)
.001

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
```

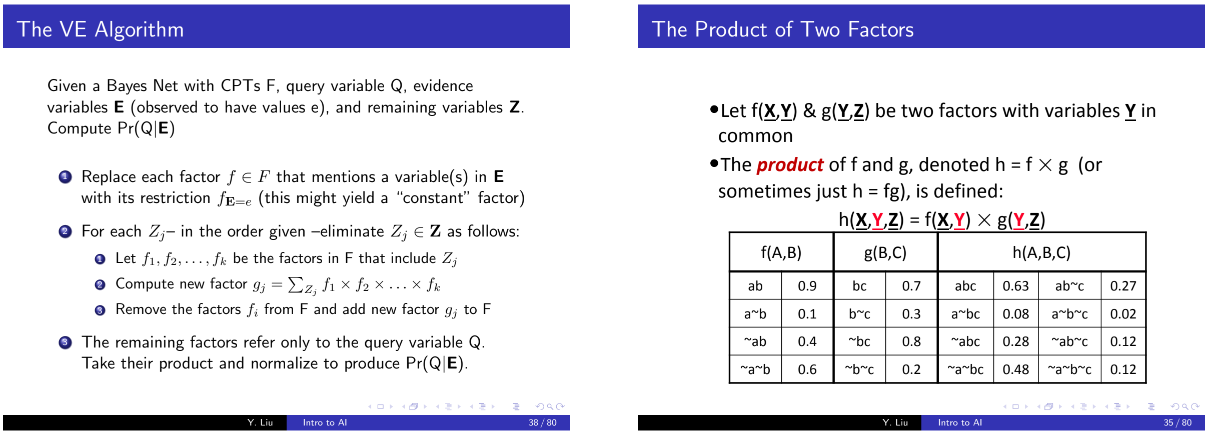
Here is a VE template for you to solve the burglary example:

```
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

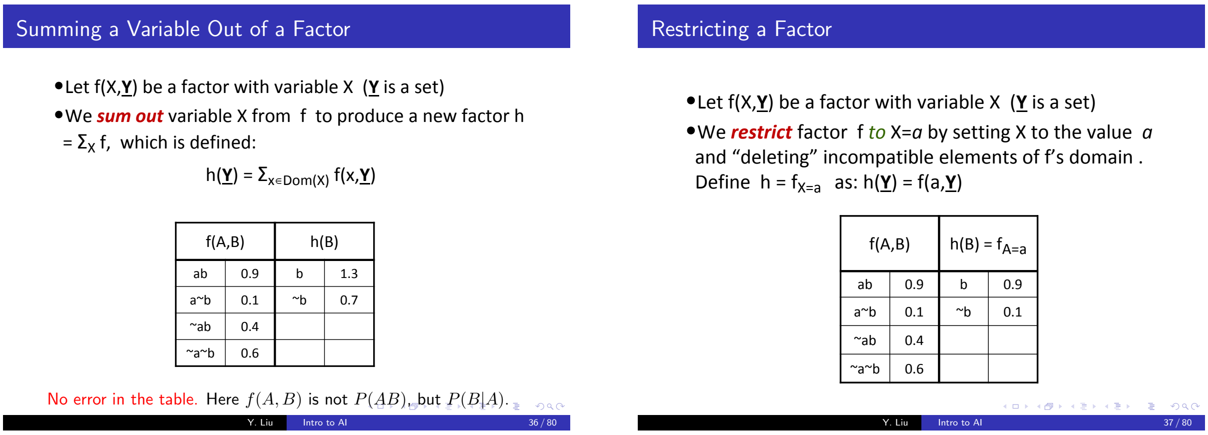
- Let f(**X**,**Y**) & g(**Y**,**Z**) be two factors with variables **Y** in common
- The ***product*** of f and g, denoted h = f × g (or sometimes just h = fg), is defined:

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

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

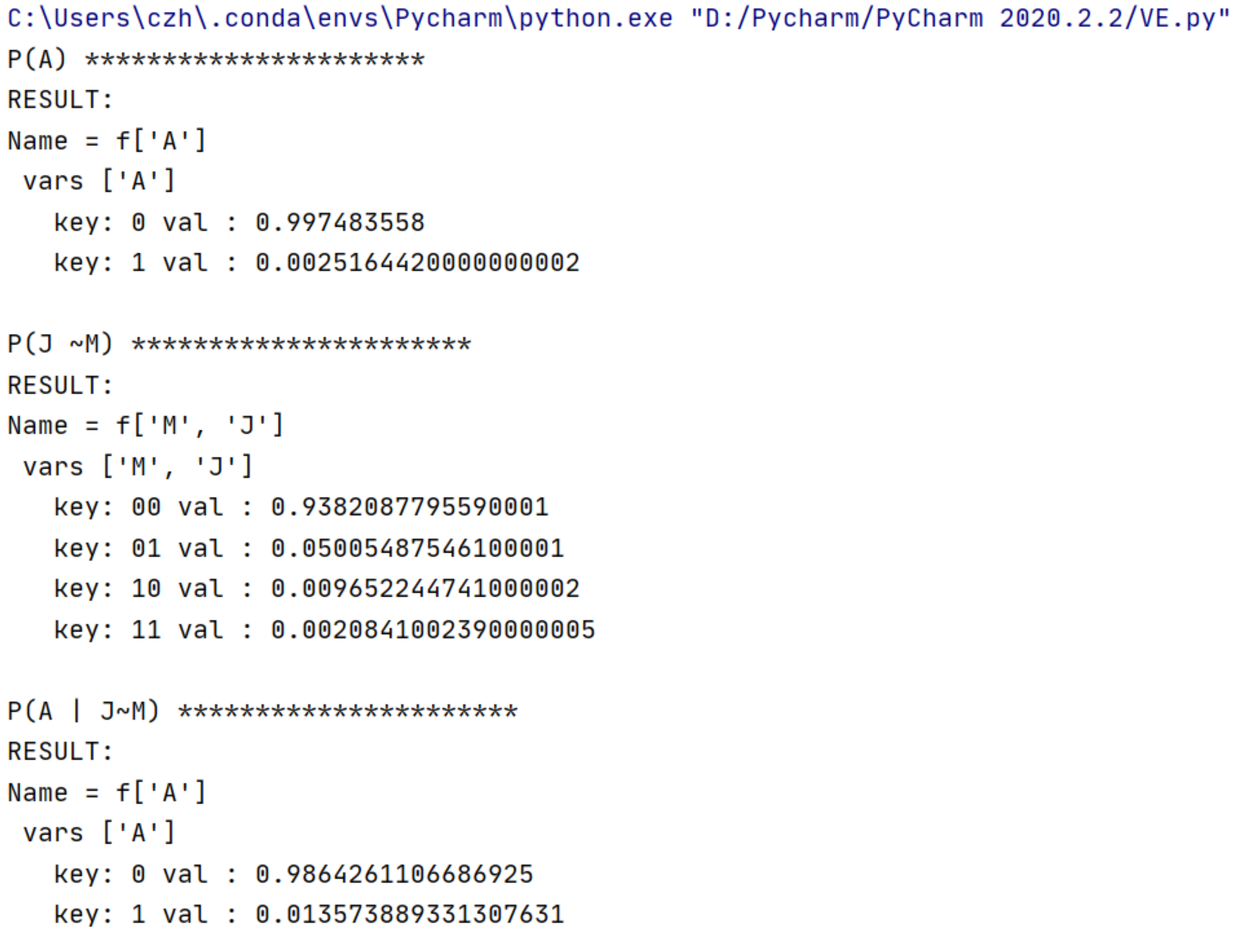
```python
class VariableElimination:
    @staticmethod
    def inference(factorList, queryVariables,
      orderedListOfHiddenVariables, evidenceList):
        for ev in evidenceList:
            for i,node in enumerate(factorList):
                if ev in node.varList:
                  factorList[i] =
                    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"""
            var1 = self.varList.copy()
            var2 = factor.varList.copy()
            var3 = list(set(var1 + var2))
            # take a union
            ncpt = {}
            var4 = get_new_cpt_var(len(var3))
            for var in var4:
                vardict = {}
                for i,v in enumerate(var3):
                    vardict[v] = var[i]
                item = ""
                for var1 in self.varList:
                    item += vardict[var1]
                f1 = self.cpt[item]
                item = ""
                for var2 in factor.varList:
                    item += vardict[var2]
                f2 = factor.cpt[item]
                ncpt[var] = f1 * f2
            nnode = Node("f" + str(var3), var3)
            nnode.setCpt(ncpt)
            # print("{} multiply {} ->
                {}".format(self.name,factor.name,nnode.name))
            return nnode

    def sumout(self, variable):
            """function that sums out a variable given a factor"""
            index = self.varList.index(variable)
            var5 = self.varList.copy()
            var5.remove(variable)
            var6 = get_new_cpt_var(len(var5))
            nncpt = {}
```

```
        for var in var6:
            sumup = 0
            for cur in ["0","1"]:
                var0 = var[:index] + cur + var[index:]
                sumup += self.cpt[var0]
            nncpt[var] = sumup
        nnnode = Node("f" + str(var5), var5)
        nnnode.setCpt(nncpt)
        # print("{} sumout {} ->
          {}".format(self.name,variable,nnnode.name))
        return nnnode

    def restrict(self, variable, value):
        """function that restricts a variable to some value
        in a given factor"""
        index = self.varList.index(variable)
        var7 = self.varList.copy()
        var7.remove(variable)
        var8 = get_new_cpt_var(len(var7))
        nnncpt = {}
        for var in var8:
            var00 = var[:index] + str(value) + var[index:]
            nnncpt[var] = self.cpt[var00]
        nnnnode = Node("f" + str(var7), var7)
        nnnnode.setCpt(nnncpt)
        # print("{} restricts {} to {} ->
          {}".format(self.name,variable,value,nnnnode.name))
        return nnnnode


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

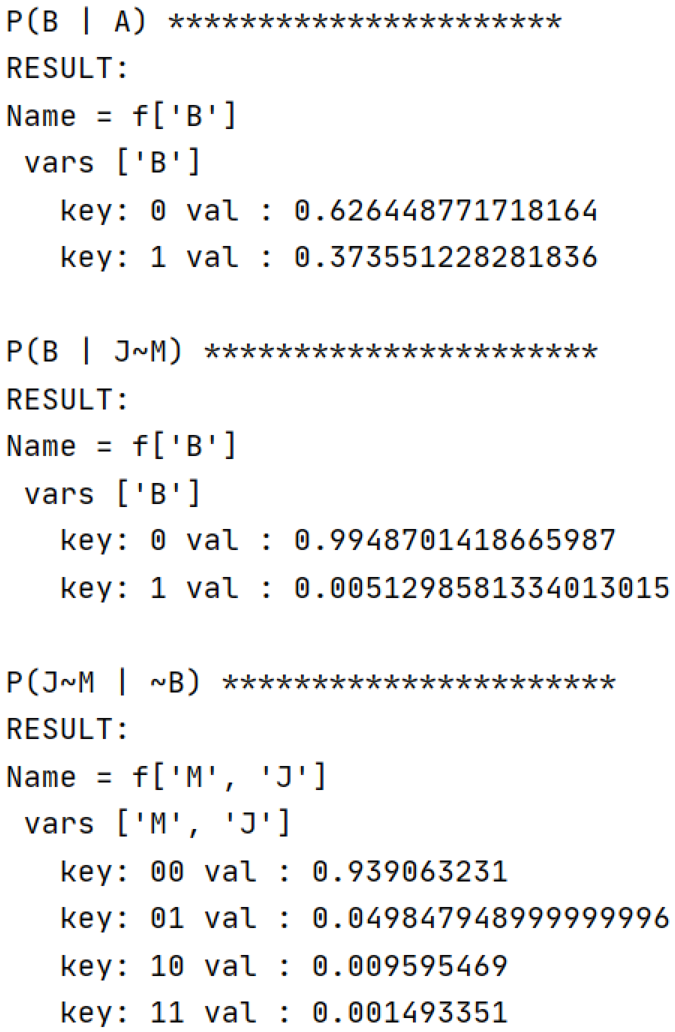
```
A = Node(”A”, [”A”, ”B”, ”E”])
J = Node(”J”, [”J”, ”A”])
M = Node(”M”, [”M”, ”A”])

# 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})
```

4 Results

文件里所给代码为 Python2 版本，我将其改为了 Python3 版本。（因为更习惯于用 Python3）



```
C:\Users\czh\.conda\envs\Pycharm\python.exe "D:/Pycharm/PyCharm 2020.2.2/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 : 0.050054875461000001
    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
```