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

E10 Variable Elimination

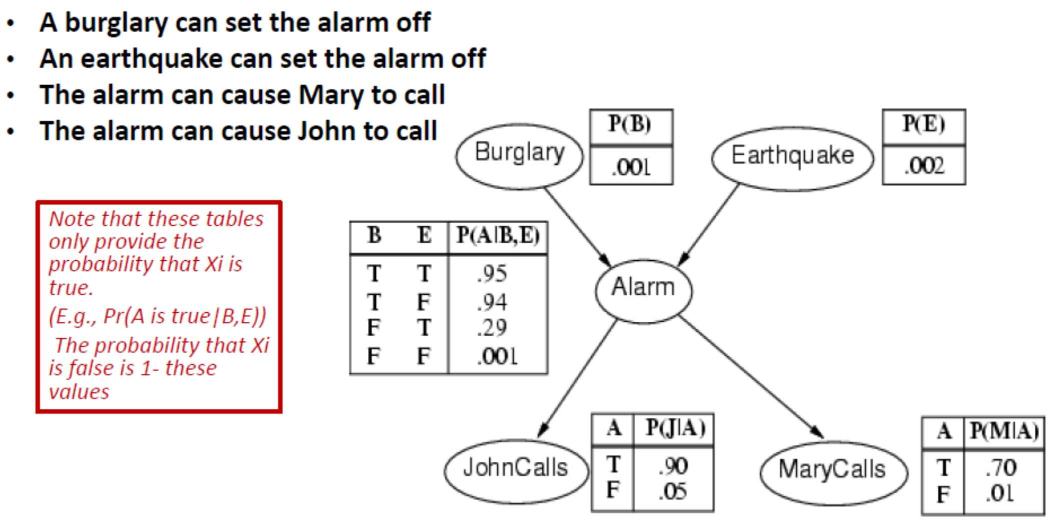
17341203 Yixin Zhang

November 16, 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

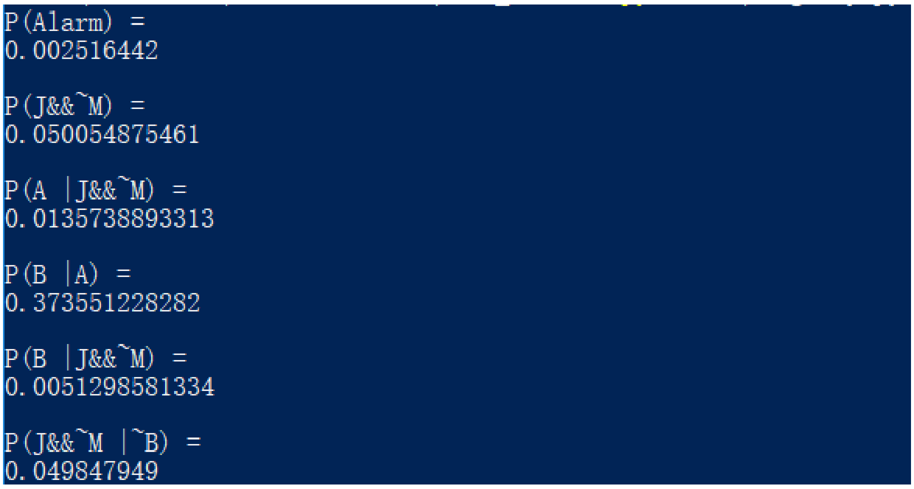
P(B)
.001

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



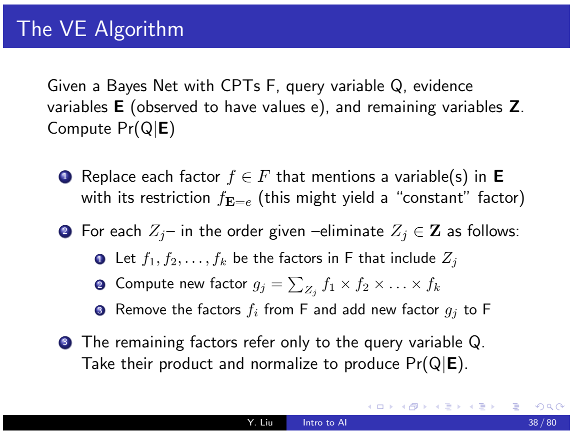
```
P(Alarm) =
0.002516442

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_201901@foxmail.com

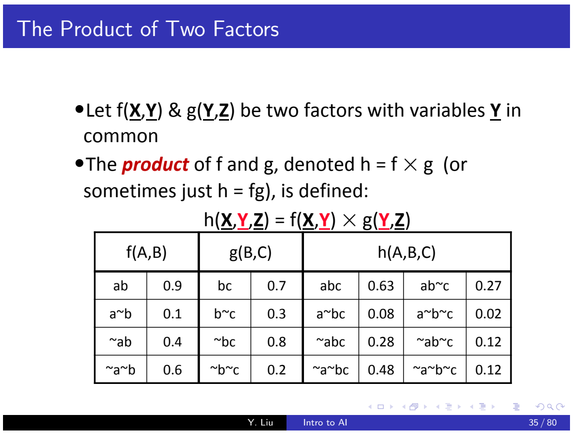


The VE Algorithm

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

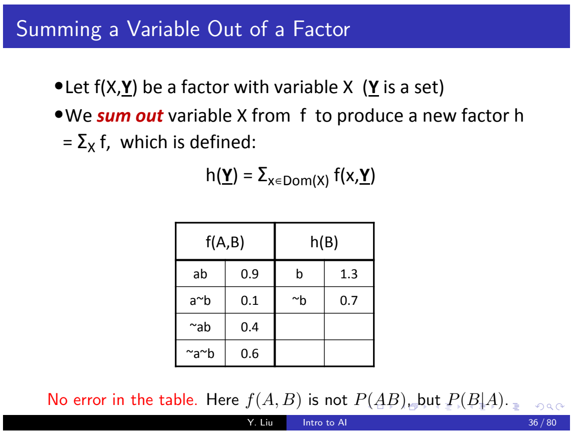
- Let f(**X**,**Y**) & g(**Y**,**Z**) be two factors with variables **Y** in common
- The ***product*** of f and g, denoted h = f × g (or sometimes just h = fg), is defined:

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

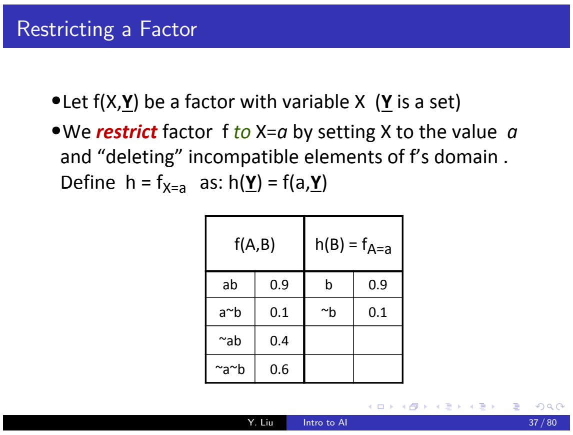
- Let f(X,**Y**) be a factor with variable X (**Y** is a set)
- We ***sum out*** variable X from f to produce a new factor h = Σ_X f, which is defined:

h(**Y**) = $\Sigma_{x \in \mathrm{Dom}(X)}$ 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 Overview

The most difficult part in the code is the implementation of 'multiply' method. There, the behaviour of 'multiply' is quite similar to that of natural join in relational databases.

Moreover, there are some special cases that must be taken care of in 'multiply'. For example, when two factors which are to multiply have no common variables, or one of the factors' variable list is empty, 'multiply' method must work correctly.

3.2 Code

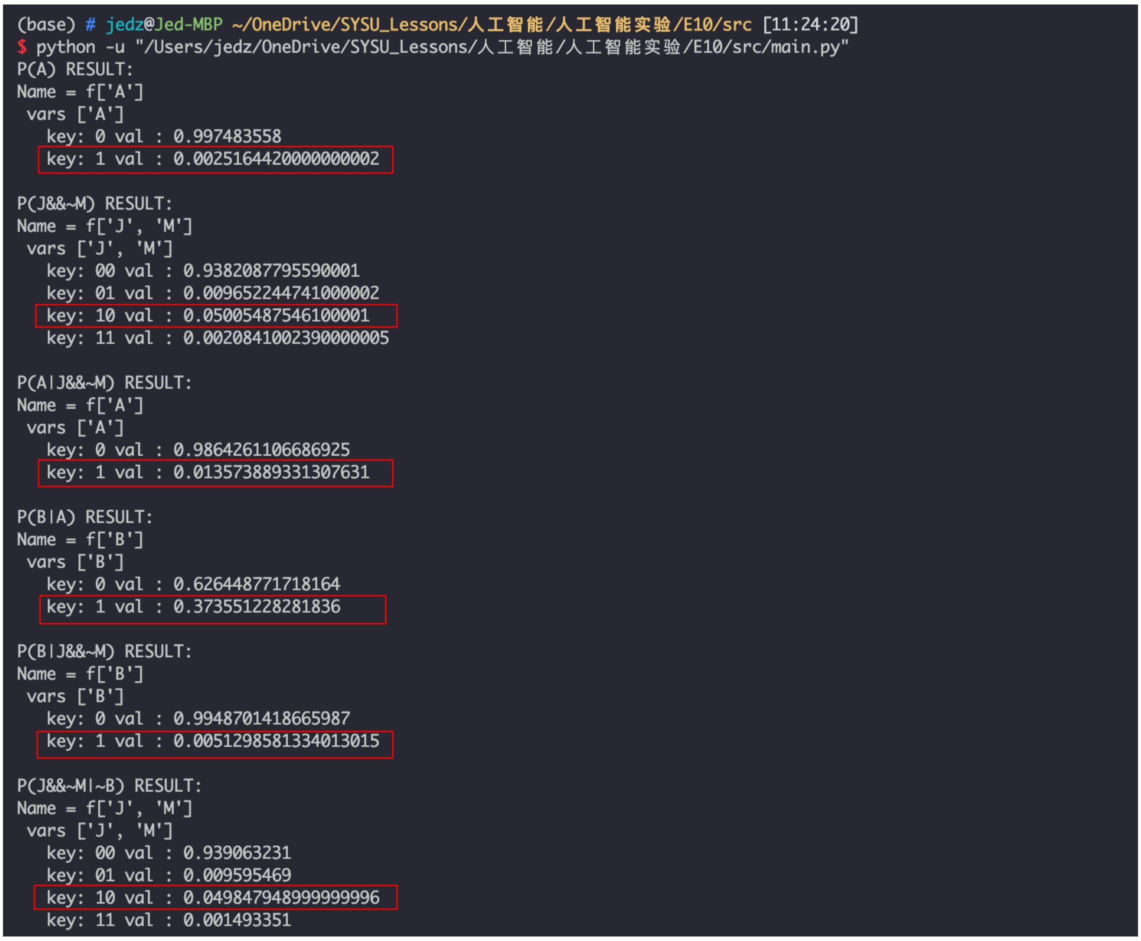
```
# -*- coding: utf-8 -*-
# @Author: Jed Zhang
# @Date: 2019-11-16 10:49:40
```

```python

class VariableElimination:
    @staticmethod
    def inference(factorList, queryVariables, orderedListOfHiddenVariables, evidenceList):
        # Step 1: 用证据取代factor中相关变量的值
        for ev in evidenceList:
            for i, factor in enumerate(factorList):
                if ev in factor.varList: # 因子中有变量在证据中
                    factorList[i] = factor.restrict(ev, evidenceList[ev])

        # Step 2: 按顺序依次消除变量
        for var in orderedListOfHiddenVariables: # var就是课件里的Zj
            corresponding_factors = [factor for factor in factorList if var in factor.varList]
            if corresponding_factors:
                new_factor = corresponding_factors[0]
                factorList.remove(new_factor)
                for factor in corresponding_factors[1:]: # 从第二个开始累乘
                    new_factor = new_factor.multiply(factor)
                    factorList.remove(factor)
                new_factor = new_factor.sumout(var) # 对变量求和从而消除该变量
            factorList.append(new_factor)

        # Step 3: 归一化并显示结果
        print("RESULT:")
        res = factorList[0]
        for factor in factorList[1:]:
            res = res.multiply(factor)
        total = sum(res.cpt.values()) # 归一化分母
        res.cpt = {k: v/total for k, v in res.cpt.items()}
        res.printInf()

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

```

```python

# 并没有用到Util类
# 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 = {} # 由setCpt函数输入

    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"""
        # 使用了类似关系型数据库中“自然连接”的操作
        var_intersection = sorted(list(set(self.varList) & set(factor.varList))) # 两个factor的变量交集
        new_var_list = self.varList + [var for var in factor.varList if var not in var_intersection]
        tup_index1 = [self.varList.index(x) for x in var_intersection]
        tup_index2 = [factor.varList.index(x) for x in var_intersection]

        merge_tup = list(zip(tup_index1, tup_index2))
        new_cpt = {}
        for key1 in self.cpt:
          for key2 in factor.cpt:
               flag = True
               for m in merge_tup:
                   if key1[m[0]] != key2[m[1]]: # 不符合自然连接条件，跳过当前key对
                      flag = False
                      break
               if flag:
                   # key1+temp组成新的key
                   temp = key2
                   for m in merge_tup:
                      temp = list(key2)
                      temp[m[1]] = 'x' # 用x标记该字符将要删除
                      temp = ''.join(temp).replace('x', '')
                   new_cpt[key1+temp] = self.cpt[key1] * factor.cpt[key2]

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

    def sumout(self, variable):
        """function that sums out a variable given a factor"""
        pos = self.varList.index(variable) # 要求和的变量的序号
        new_var_list = self.varList[:pos] + self.varList[pos+1:]

        new_cpt_keyset = sorted(list(set([k[:pos]+k[pos+1:] for k in self.cpt.keys()]))) # 新变量列表的组合构成的集合
        new_cpt = {}
        for new_key in new_cpt_keyset:
          new_value = 0
          for value in ['0', '1']: # 本例中变量只有两种取值
               new_value += self.cpt[new_key[:pos] + value + new_key[pos:]] # 在原来的CPT中进行累加
          new_cpt[new_key] = new_value

        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"""
        pos = self.varList.index(variable) # 要限制的变量的序号
        new_var_list = self.varList[:pos] + self.varList[pos+1:]

        new_cpt_keyset = sorted(list(set([k[:pos]+k[pos+1:] for k in self.cpt.keys()]))) # 新变量列表的组合构成的集合
        new_cpt = {}
        for new_key in new_cpt_keyset:
            new_cpt[new_key] = self.cpt[new_key[:pos] + value + new_key[pos:]]

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


if __name__ == '__main__':
    # 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})

    # 注意：下面evidenceList中将变量的取值统一成字符串的'1'的'0'，而不是数字。
    print("P(A)", end=' ')
    VariableElimination.inference([B,E,A,J,M], ['A'], ['B', 'E', 'J','M'], {})

```

```python
    print("P(J&&~M)", end=' ')
    VariableElimination.inference([B,E,A,J,M], ['J', '~M'], ['B', 'E', 'A'], {})

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

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

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

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

3.3 Results



```
(base) # jedz@Jed-MBP ~/OneDrive/SYSU_Lessons/人工智能/人工智能实验/E10/src [11:24:20]
$ python -u "/Users/jedz/OneDrive/SYSU_Lessons/人工智能/人工智能实验/E10/src/main.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['J', 'M']
 vars ['J', 'M']
   key: 00 val : 0.9382087795590001
   key: 01 val : 0.009652244741000002
   key: 10 val : 0.050054875461000001
   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 : 0.005129858133401301 5

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
```