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

E10 Variable Elimination

17341111 Xuehai Liu

November 17, 2019

Contents

1 VE

The burglary example is described as following:

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

Figure 1: VE and Product

Figure 2: Sumout and Restrict

3 Codes and Results

Codes:

```
class VariableElimination:
    @staticmethod
    def inference(factorList, queryVariables,
    orderedListOfHiddenVariables, evidenceList):
        for key, value in evidenceList.items():
            new_factor_list = []
            for factor in factorList:
                if(key in factor.varList):
                    newnode = factor.restrict(key,value)
                    new_factor_list.append(newnode)
                else:
                    new_factor_list.append(factor)

            factorList = new_factor_list


        for var in orderedListOfHiddenVariables:
            #Your code here
            mulNode = None
            new_factor_list =[]

            for factor in factorList:
                if(var in factor.varList):
                    new_factor_list.append(factor)
            res = new_factor_list[0]
            factorList.remove(res)
            for factor in new_factor_list[1:]:
                res = res.multiply(factor)
                factorList.remove(factor)
            res = res.sumout(var)
```

```
            factorList.append(res)
            #print ("RESULT:")
            #for factor in factorList:
            #    factor.printInf()

        #print("result")
        #for factor in factorList:
        #    factor.printInf()
        new_factor_list = []
        for factor in factorList:
            if factor.varList != []:
                new_factor_list.append(factor)
        factorList = new_factor_list

        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_cpt = {}
        newList = []
        samenode = ""
        index1 = 0
        index2 = 0
        for i,var in enumerate(self.varList):
            if var not in factor.varList:
                newList.append(var)
            else:
                index1 = i
                newList.append(var)
        for i,var in enumerate(factor.varList):
            if var not in newList:
                newList.append(var)
            else:
                index2 = i

        for string, prob1 in self.cpt.items():
```

```
            for string2, prob2 in factor.cpt.items():
                newstring = ''
                if (string2[index2] == string[index1]):
                    for i in string:
                        newstring += i
                    for i,ch in enumerate(string2):
                        if(i != index2 ):
                            newstring += ch
                    if (newstring not in new_cpt.keys()):
                        new_cpt[newstring] = prob1 * prob2
        #new_cpt =
        tempstr = ""
        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_var_list = []
        indexnum = 0
        for i,var in enumerate(self.varList):
            if(var != variable):
                new_var_list.append(var)
            else:
                indexnum = i
        new_cpt = {}
        strlen = len(self.varList)
        strings = ""

        #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})
        for string in self.cpt.keys():
```

```
            tempstr = list (string)
            tempstr[indexnum] = ""
            newstr = ""
            zero = ""
            one = ""
            for i,a in enumerate(tempstr):
                if(i==indexnum):
                    zero += "0"
                    one += "1"
                else:
                    newstr += a
                    zero += a
                    one += a
            if(newstr not in new_cpt.keys()):
                new_cpt[newstr] = self.cpt[zero]+self.cpt[one]


        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_var_list =[]
        indexnum = 0
        for i,var in enumerate(self.varList):
            if(var != variable):
                new_var_list.append(var)
            else:
                indexnum = i


        new_cpt ={}
```

```
        for string in self.cpt.keys():
            news = ""
            flag = 0
            for i,ch in enumerate(string):
                if(i==indexnum):
                    if(ch == str(value)):
                        flag = 1
                else:
                    news += ch
            if(news not in new_cpt.keys() and flag):
                new_cpt[news] = self.cpt[string]

        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"])

#B.multiply(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})
```

Results:

```
P(A) **********
Name = f['A']
 vars ['A']
   key: 1 val : 0.0025164420000000002
   key: 0 val : 0.997483558

P(J~M) **********
Name = f['J', 'M']
 vars ['J', 'M']
   key: 11 val : 0.002084100239
   key: 10 val : 0.050054875461
   key: 01 val : 0.009652244741
   key: 00 val : 0.938208779559

P(A | J~M) **********
Name = f['A']
 vars ['A']
   key: 1 val : 0.013573889331307631
   key: 0 val : 0.9864261106686925
```

Figure 3: Q1 - Q3

```
P(B | A) **********
Name = f['B']
 vars ['B']
   key: 0 val : 0.626448771718164
   key: 1 val : 0.373551228281836

P(B | J~M) **********
Name = f['B']
 vars ['B']
   key: 0 val : 0.9948701418665987
   key: 1 val : 0.0051298581334013015

P(J~M | ~B) **********
Name = f['J', 'M']
 vars ['J', 'M']
   key: 11 val : 0.0014933510000000002
   key: 10 val : 0.049847949
   key: 01 val : 0.009595469
   key: 00 val : 0.939063231
```

Figure 4: Q4 - Q6

Figure 5: Answer