Dashboard / My courses / COSC367-2020S2 / Weekly quizzes / 8. Belief networks and probabilistic inference

Started on	Thursday, 17 September 2020, 9:16 PM
State	Finished
Completed on	Monday, 28 September 2020, 5:06 PM
Time taken	10 days 18 hours
Marks	6.00/6.00
Grade	**100.00** out of 100.00

Information

Introduction

This quiz is on Bayesian (or belief) networks and probabilistic inference. The questions ask you to either write a program or express a problem in the form of a network. Please note that you are expected to be able to solve these problems manually (using only pen and paper) before you attempt to write a program for them.

Optional activity

The Belief Networks applet is an educational tool provided by AIspace. The tutorials are available here: http://www.aispace.org/bayes/help/tutorials.shtml. You can download the applet from the same place. The applet is useful to check your understanding of the topic. For instance load the network related to Example 8.15 in the textbook and answer the queries using pen and paper and then check your answers with the applet.

Information

Representation of belief networks in Python

A belief (or Bayesian) network is represented by a dictionary. The keys are the names of variables. The values are dictionaries themselves. The second level dictionaries have two keys: `'Parents'` whose value is a list of the names of the variables that are the parents of the current variable, and `'CPT'` whose value is a dictionary again. The keys of the third level dictionaries are tuples of Booleans which correspond to possible assignments of values to the parents of the current node (in the order they are listed) and the values are real numbers representing the probability of the current node being true given the specified assignment to the parents.

Notes

- Variable names are case sensitive.
- If a node does not have any parents, the value of `'Parents'` must be an empty list and the only key of the third level dictionary is the empty tuple.
- For simplicity, we assume that all the variables are Boolean.

Example

The following is the representation of the *alarm network* presented in the lecture notes.

```
network = {
    'Burglary': {
        'Parents': [],
        'CPT': {
            (): 0.001,
         }
    },

    'Earthquake': {
        'Parents': [],
        'CPT': {
            (): 0.002,
        }
    },

    'Alarm': {
        'Parents': ['Burglary','Earthquake'],
        'CPT': {
            (True,True): 0.95,
            (True,False): 0.94,
            (False,True): 0.29,
            (False,False): 0.001,
        }
    },

    'John': {
        'Parents': ['Alarm'],
        'CPT': {
            (True,): 0.9,
            (False,): 0.05,
        }
    },

    'Mary': {
        'Parents': ['Alarm'],
        'CPT': {
            (True,): 0.7,
            (False,): 0.01,
        }
    },
}
```

Question **1**

Correct

Mark 1.00 out of 1.00

Write a function `joint_prob(network, assignment)` that given a belief network and a complete assignment of all the variables in the network, returns the probability of the assignment. The data structure of the network is as described above. The assignment is a dictionary where keys are the variable names and the values are either `True` or `False`.

For example:

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td><pre>from student_answer import joint_prob

network = {
    'A': {
        'Parents': [],
        'CPT': {
            (): 0.2
            }},
}

p = joint_prob(network, {'A': True})
print("{:.5f}".format(p))</pre></td><td><pre>0.20000</pre></td></tr>
<tr><td><pre>from student_answer import joint_prob

network = {
    'A': {
        'Parents': [],
        'CPT': {
            (): 0.2
            }},
}

p = joint_prob(network, {'A': False})
print("{:.5f}".format(p))</pre></td><td><pre>0.80000</pre></td></tr>
<tr><td><pre>from student_answer import joint_prob

network = {
    'A': {
        'Parents': [],
        'CPT': {
            (): 0.1
            }},

    'B': {
        'Parents': ['A'],
        'CPT': {
            (True,): 0.8,
            (False,): 0.7,
            }},
    }

p = joint_prob(network, {'A': False, 'B':True})
print("{:.5f}".format(p))</pre></td><td><pre>0.63000</pre></td></tr>
<tr><td><pre>from student_answer import joint_prob

network = {
    'A': {
        'Parents': [],
        'CPT': {
            (): 0.1
            }},

    'B': {
        'Parents': ['A'],
        'CPT': {
            (True,): 0.8,
            (False,): 0.7,
            }},
    }

p = joint_prob(network, {'A': False, 'B':False})
print("{:.5f}".format(p))</pre></td><td><pre>0.27000</pre></td></tr>
</table>

Test	Result

`from student_answer import joint_prob`
 	
`network = {`
`    'Burglary': {`
`        'Parents': [],`
`        'CPT': {`
`            (): 0.001`
`            }},`
 	
`    'Earthquake': {`
`        'Parents': [],`
`        'CPT': {`
`            (): 0.002,`
`            }},`
`    'Alarm': {`
`        'Parents': ['Burglary','Earthquake'],`
`        'CPT': {`
`            (True,True): 0.95,`
`            (True,False): 0.94,`
`            (False,True): 0.29,`
`            (False,False): 0.001,`
`            }},`
 	
`    'John': {`
`        'Parents': ['Alarm'],`
`        'CPT': {`
`            (True,): 0.9,`
`            (False,): 0.05,`
`            }},`
 	
`    'Mary': {`
`        'Parents': ['Alarm'],`
`        'CPT': {`
`            (True,): 0.7,`
`            (False,): 0.01,`
`            }},`
`    }`
 	
`p = joint_prob(network, {'John': True, 'Mary': True,`
`                         'Alarm': True, 'Burglary': False,`
`                         'Earthquake': False})`
`print("{:.8f}".format(p))`
	0.00062811

Answer: (penalty regime: 0, 15, ... %)

```
def joint_prob(network, assignment):
    probability = 1
    for i, j in assignment.items():
        value = network[i]
        plist = []
        for x in value['Parents']:
            plist.append(assignment[x])
        plist = tuple(plist)
        if assignment[i] == True:
            probability = probability * value['CPT'][plist]
        if assignment[i] == False:
            probability = probability * (1- value['CPT'][plist])
    return probability

```

	Test	Expected	Got	

	Test	Expected	Got	

✔	`from student_answer import joint_prob`
	
	`network = {`
	`    'A': {`
	`        'Parents': [],`
	`        'CPT': {`
	`            (): 0.2`
	`            }},`
	`}`
	
	`p = joint_prob(network, {'A': True})`
	`print("{:.5f}".format(p))`
		0.20000	0.20000	✔

✔	`from student_answer import joint_prob`
	
	`network = {`
	`    'A': {`
	`        'Parents': [],`
	`        'CPT': {`
	`            (): 0.2`
	`            }},`
	`}`
	
	`p = joint_prob(network, {'A': False})`
	`print("{:.5f}".format(p))`
		0.80000	0.80000	✔

✔	`from student_answer import joint_prob`
	
	`network = {`
	`    'A': {`
	`        'Parents': [],`
	`        'CPT': {`
	`            (): 0.1`
	`            }},`
	
	`    'B': {`
	`        'Parents': ['A'],`
	`        'CPT': {`
	`            (True,): 0.8,`
	`            (False,): 0.7,`
	`            }},`
	`    }`
	
	`p = joint_prob(network, {'A': False, 'B':True})`
	`print("{:.5f}".format(p))`
		0.63000	0.63000	✔

✔	`from student_answer import joint_prob`
	
	`network = {`
	`    'A': {`
	`        'Parents': [],`
	`        'CPT': {`
	`            (): 0.1`
	`            }},`
	
	`    'B': {`
	`        'Parents': ['A'],`
	`        'CPT': {`
	`            (True,): 0.8,`
	`            (False,): 0.7,`
	`            }},`
	`    }`
	
	`p = joint_prob(network, {'A': False, 'B':False})`
	`print("{:.5f}".format(p))`
		0.27000	0.27000	✔

	Test	Expected	Got	

✔	<pre>from student_answer import joint_prob
	
	network = {
	    'Burglary': {
	        'Parents': [],
	        'CPT': {
	            (): 0.001
	            }},
	
	    'Earthquake': {
	        'Parents': [],
	        'CPT': {
	            (): 0.002,
	            }},
	    'Alarm': {
	        'Parents': ['Burglary','Earthquake'],
	        'CPT': {
	            (True,True): 0.95,
	            (True,False): 0.94,
	            (False,True): 0.29,
	            (False,False): 0.001,
	            }},
	
	    'John': {
	        'Parents': ['Alarm'],
	        'CPT': {
	            (True,): 0.9,
	            (False,): 0.05,
	            }},
	
	    'Mary': {
	        'Parents': ['Alarm'],
	        'CPT': {
	            (True,): 0.7,
	            (False,): 0.01,
	            }},
	    }
	
	p = joint_prob(network, {'John': True, 'Mary': True,
	                         'Alarm': True, 'Burglary': False,
	                         'Earthquake': False})
	print("{:.8f}".format(p))</pre>
		0.00062811	0.00062811	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Write a function `query(network, query_var, evidence)` that given a belief network, the name of a variable in the network, and some evidence, returns the posterior distribution of `query_var`. The parameter `network` is a belief network whose data structure was described earlier. The parameter `query_var` is the name of the variable we are interested in and is of type string. The parameter `evidence` is a dictionary whose elements are assignments to some variables in the network; the keys are the name of the variables and the values are Boolean.

The function must return a pair of real numbers where the first element is the probability of `query_var` being false given the evidence and the second element is the probability of `query_var` being true given the evidence.

Note: Please remember to include the joint probability function (from the previous question) and relevant import statements in your answer.

Hints

This is inference by enumeration. You need to use the joint probability function developed in the previous question. You have to perform the operation once for `query_var` being true and once for false. You have to sum over all possible values of "hidden" variables. The following gives you the set of hidden variables:

```
hidden_vars = network.keys() - evidence.keys() - {query_var}
```

All possible assignments to hidden variables can be obtained by:

```
for values in itertools.product((True, False), repeat=len(hidden_vars)):
    hidden_assignments = {var:val for var,val in zip(hidden_vars, values)}
```

Remarks

1. When the argument evidence is an empty dictionary we are (semantically) asking for the prior probability of query_var. The algorithm, however, remains the same.
2. This algorithm is very close to the mathematical definition of inference over the network and therefore it's easy to understand and implement. However, this is not an efficient algorithm (constant memory, O(n2^n) time). Using *factors* would be a much more efficient approach (see the textbook). Note that none of the test cases are very large, so for this question it doesn't make a difference what approach is taken.

For example:

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td><pre>from student_answer import query

network = {
    'A': {
        'Parents': [],
        'CPT': {
            (): 0.2
            }},
}

answer = query(network, 'A', {})
print("P(A=true) = {:.5f}".format(answer[True]))
print("P(A=false) = {:.5f}".format(answer[False]))</pre></td><td><pre>P(A=true) = 0.20000
P(A=false) = 0.80000</pre></td></tr>
<tr><td><pre>from student_answer import query

network = {
    'A': {
        'Parents': [],
        'CPT': {
            (): 0.1
            }},

    'B': {
        'Parents': ['A'],
        'CPT': {
            (True,): 0.8,
            (False,): 0.7,
            }},
    }

answer = query(network, 'B', {'A': False})
print("P(B=true|A=false) = {:.5f}".format(answer[True]))
print("P(B=false|A=false) = {:.5f}".format(answer[False]))</pre></td><td><pre>P(B=true|A=false) = 0.70000
P(B=false|A=false) = 0.30000</pre></td></tr>
</table>

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td><pre>from student_answer import query

network = {
    'A': {
        'Parents': [],
        'CPT': {
            (): 0.1
            }},

    'B': {
        'Parents': ['A'],
        'CPT': {
            (True,): 0.8,
            (False,): 0.7,
            }},
    }

answer = query(network, 'B', {})
print("P(B=true) = {:.5f}".format(answer[True]))
print("P(B=false) = {:.5f}".format(answer[False]))</pre></td><td><pre>P(B=true) = 0.71000
P(B=false) = 0.29000</pre></td></tr>
<tr><td><pre>from student_answer import query

network = {
    'Burglary': {
        'Parents': [],
        'CPT': {
            (): 0.001
            }},

    'Earthquake': {
        'Parents': [],
        'CPT': {
            (): 0.002,
            }},
    'Alarm': {
        'Parents': ['Burglary','Earthquake'],
        'CPT': {
            (True,True): 0.95,
            (True,False): 0.94,
            (False,True): 0.29,
            (False,False): 0.001,
            }},

    'John': {
        'Parents': ['Alarm'],
        'CPT': {
            (True,): 0.9,
            (False,): 0.05,
            }},

    'Mary': {
        'Parents': ['Alarm'],
        'CPT': {
            (True,): 0.7,
            (False,): 0.01,
            }},
    }

answer = query(network, 'Burglary', {'John': True, 'Mary': True})
print("Probability of a burglary when both\n"
      "John and Mary have called: {:.3f}".format(answer[True]))</pre></td><td><pre>Probability of a burglary
when both
John and Mary have called:
0.284</pre></td></tr>
</table>

Test

```
from student_answer import query

network = {
    'Burglary': {
        'Parents': [],
        'CPT': {
            (): 0.001
            }},

    'Earthquake': {
        'Parents': [],
        'CPT': {
            (): 0.002,
            }},
    'Alarm': {
        'Parents': ['Burglary','Earthquake'],
        'CPT': {
            (True,True): 0.95,
            (True,False): 0.94,
            (False,True): 0.29,
            (False,False): 0.001,
            }},

    'John': {
        'Parents': ['Alarm'],
        'CPT': {
            (True,): 0.9,
            (False,): 0.05,
            }},

    'Mary': {
        'Parents': ['Alarm'],
        'CPT': {
            (True,): 0.7,
            (False,): 0.01,
            }},
    }

answer = query(network, 'John', {'Mary': True})
print("Probability of John calling if\n"
      "Mary has called: {:.5f}".format(answer[True]))
```

Result

```
Probability of John calling
if
Mary has called: 0.17758
```

Answer: (penalty regime: 0, 15, ... %)

```
import itertools

def query(network, query_var, evidence):
    #what is evidence? hidden variable?

    probability_true = 0
    probability_false = 0
    hidden_vars = network.keys() - evidence.keys() - {query_var}
    for values in itertools.product((True, False), repeat=len(hidden_vars)):
        hidden_assignments = {var:val for var,val in zip(hidden_vars, values)}
        probability_true += joint_prob(network, dict({ query_var: True}, **hidden_assignments, **evic
        probability_false += joint_prob(network, dict({ query_var: False}, **hidden_assignments, **ev

    weighted_true = probability_true/(probability_false + probability_true)
    weighted_false = 1 -weighted_true

    formatted = { True : weighted_true, False : weighted_false}
    #print("Hello")
    #print(formatted)
    return formatted
    #print(probability_true)
    #print(probability_false)






def joint_prob(network, assignment):
    probability = 1
    #for every node in network
    for i, j in assignment.items():
        #get the info of each node
        value = network[i]
```

```
        plist = []
        #check the parents of each node to get conditional prob
        for x in value['Parents']:
            plist.append(assignment[x])
        #these are the required conditions for each node probability that i want to check
        plist = tuple(plist)
        #get the probability value if its true or false
        if assignment[i] == True:
            probability = probability * value['CPT'][plist]
        if assignment[i] == False:
            probability = probability * (1- value['CPT'][plist])
    return probability
```

	Test	Expected	Got	

✔	from student_answer import query
	
	network = {
	    'A': {
	        'Parents': [],
	        'CPT': {
	            (): 0.2
	            }},
	}
	
	answer = query(network, 'A', {})
	print("P(A=true) = {:.5f}".format(answer[True]))
	print("P(A=false) = {:.5f}".format(answer[False]))
		P(A=true) = 0.20000
		P(A=false) = 0.80000
			P(A=true) = 0.20000
			P(A=false) = 0.80000
				✔

✔	from student_answer import query
	
	network = {
	    'A': {
	        'Parents': [],
	        'CPT': {
	            (): 0.1
	            }},
	
	    'B': {
	        'Parents': ['A'],
	        'CPT': {
	            (True,): 0.8,
	            (False,): 0.7,
	            }},
	    }
	
	answer = query(network, 'B', {'A': False})
	print("P(B=true|A=false) = {:.5f}".format(answer[True]))
	print("P(B=false|A=false) = {:.5f}".format(answer[False]))
		P(B=true|A=false) = 0.70000
		P(B=false|A=false) = 0.30000
			P(B=true|A=false) = 0.70000
			P(B=false|A=false) = 0.30000
				✔

	Test	Expected	Got	

✔	`from student_answer import query`
	
	`network = {`
	`    'A': {`
	`        'Parents': [],`
	`        'CPT': {`
	`            (): 0.1`
	`            }},`
	
	`    'B': {`
	`        'Parents': ['A'],`
	`        'CPT': {`
	`            (True,): 0.8,`
	`            (False,): 0.7,`
	`            }},`
	`    }`
	
	`answer = query(network, 'B', {})`
	`print("P(B=true) = {:.5f}".format(answer[True]))`
	`print("P(B=false) = {:.5f}".format(answer[False]))`
		`P(B=true) = 0.71000`
		`P(B=false) = 0.29000`
			`P(B=true) = 0.71000`
			`P(B=false) = 0.29000`
				✔

✔	`from student_answer import query`
	
	`network = {`
	`    'Burglary': {`
	`        'Parents': [],`
	`        'CPT': {`
	`            (): 0.001`
	`            }},`
	
	`    'Earthquake': {`
	`        'Parents': [],`
	`        'CPT': {`
	`            (): 0.002,`
	`            }},`
	`    'Alarm': {`
	`        'Parents': ['Burglary','Earthquake'],`
	`        'CPT': {`
	`            (True,True): 0.95,`
	`            (True,False): 0.94,`
	`            (False,True): 0.29,`
	`            (False,False): 0.001,`
	`            }},`
	
	`    'John': {`
	`        'Parents': ['Alarm'],`
	`        'CPT': {`
	`            (True,): 0.9,`
	`            (False,): 0.05,`
	`            }},`
	
	`    'Mary': {`
	`        'Parents': ['Alarm'],`
	`        'CPT': {`
	`            (True,): 0.7,`
	`            (False,): 0.01,`
	`            }},`
	`    }`
	
	`answer = query(network, 'Burglary', {'John': True, 'Mary': True})`
	`print("Probability of a burglary when both\n"`
	`      "John and Mary have called: {:.3f}".format(answer[True]))`
		Probability of a burglary when both John and Mary have called: 0.284	Probability of a burglary when both John and Mary have called: 0.284	✔

	Test	Expected	Got	

✔	`from student_answer import query`
	
	`network = {`
	`    'Burglary': {`
	`        'Parents': [],`
	`        'CPT': {`
	`            (): 0.001`
	`            }},`
	
	`    'Earthquake': {`
	`        'Parents': [],`
	`        'CPT': {`
	`            (): 0.002,`
	`            }},`
	`    'Alarm': {`
	`        'Parents':`
	`['Burglary','Earthquake'],`
	`        'CPT': {`
	`            (True,True): 0.95,`
	`            (True,False): 0.94,`
	`            (False,True): 0.29,`
	`            (False,False): 0.001,`
	`            }},`
	
	`    'John': {`
	`        'Parents': ['Alarm'],`
	`        'CPT': {`
	`            (True,): 0.9,`
	`            (False,): 0.05,`
	`            }},`
	
	`    'Mary': {`
	`        'Parents': ['Alarm'],`
	`        'CPT': {`
	`            (True,): 0.7,`
	`            (False,): 0.01,`
	`            }},`
	`    }`
	
	`answer = query(network, 'John', {'Mary':`
	`True})`
	`print("Probability of John calling if\n"`
	`      "Mary has called:`
	`{:.5f}".format(answer[True]))`
		`Probability of John`
		`calling if`
		`Mary has called:`
		`0.17758`
			`Probability of John`
			`calling if`
			`Mary has called:`
			`0.17758`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

Consider a medical test for a certain disease that is very rare, striking only 1 in 100,000 people. Suppose the probability of testing positive if the person has the disease is 99%, as is the probability of testing negative when the person does not have the disease.

Express these facts in the form of a (causal) belief network. Use variable names `'Disease'` and `'Test'`. Assign the network to the variable `network`.

Important: Supply the `query` function and all the functions and modules it depends on (e.g. `joint_prob`) from the previous questions.

Comment: After solving the problem, you may find the value of P(having disease| positive test), which is essentially the *precision* of the test, counter-intuitive; one may expect this value to be much higher. Observe that the probability of returning positive regardless of the disease is about 1%, which is quite high compared to how rare the disease is. A good test for this rare disease must have a much higher *specificity*, which is the probability of returning negative when the person does not have the disease. You can explore this by changing the values in the CPTs (more specifically, making the value corresponding to `Disease` being False in the CPT of `Test` smaller).

For example:

Test	Result

from student_answer import query, network
 	
answer = query(network, 'Disease', {'Test': True})
print("The probability of having the disease\n"
      "if the test comes back positive: {:.8f}"
      .format(answer[True]))
	The probability of having the disease
	if the test comes back positive: 0.00098903

from student_answer import query, network
 	
answer = query(network, 'Disease', {'Test': False})
print("The probability of having the disease\n"
      "if the test comes back negative: {:.8f}"
      .format(answer[True]))
	The probability of having the disease
	if the test comes back negative: 0.00000010

Answer: (penalty regime: 0, 15, ... %)

```
import itertools








def query(network, query_var, evidence):
    #what is evidence? hidden variable?

    probability_true = 0
    probability_false = 0
    hidden_vars = network.keys() - evidence.keys() - {query_var}
    for values in itertools.product((True, False), repeat=len(hidden_vars)):
        hidden_assignments = {var:val for var,val in zip(hidden_vars, values)}
        probability_true += joint_prob(network, dict({ query_var: True}, **hidden_assignments, **evi
        probability_false += joint_prob(network, dict({ query_var: False}, **hidden_assignments, **e

    weighted_true = probability_true/(probability_false + probability_true)
    weighted_false = 1 -weighted_true

    formatted = { True : weighted_true, False : weighted_false}
    #print("Hello")
    #print(formatted)
    return formatted
    #print(probability_true)
    #print(probability_false)






def joint_prob(network, assignment):
    probability = 1
    #for every node in network
    for i, j in assignment.items():
        #get the info of each node
        value = network[i]
        plist = []
        #check the parents of each node to get conditional prob
        for x in value['Parents']:
            plist.append(assignment[x])
        #these are the required conditions for each node probability that i want to check
```

```
        plist = tuple(plist)
        #get the probability value if its true or false
        if assignment[i] == True:
            probability = probability * value['CPT'][plist]
        if assignment[i] == False:
            probability = probability * (1- value['CPT'][plist])
    return probability






network = {
    'Disease': {
        'Parents': [],
        'CPT': {
            (): 0.00001
            }},

    'Test': {
        'Parents': ['Disease'],
        'CPT': {
            (True,): 0.99,
            (False,): 0.01,
            }},
    }

```

	Test	Expected	Got	

✔	`from student_answer import query, network`
	
	`answer = query(network, 'Disease', {'Test': True})`
	`print("The probability of having the disease\n"`
	`      "if the test comes back positive: {:.8f}"`
	`      .format(answer[True]))`
		The probability of having the disease
		if the test comes back positive: 0.00098903
			The probability of having the disease
			if the test comes back positive: 0.00098903
				✔

✔	`from student_answer import query, network`
	
	`answer = query(network, 'Disease', {'Test': False})`
	`print("The probability of having the disease\n"`
	`      "if the test comes back negative: {:.8f}"`
	`      .format(answer[True]))`
		The probability of having the disease
		if the test comes back negative: 0.00000010
			The probability of having the disease
			if the test comes back negative: 0.00000010
				✔

✔	`from student_answer import query, network`
	
	`answer = query(network, 'Test', {})`
	`print("The probability of testing positive\n"`
	`      "is {:.8f}".format(answer[True]))`
		The probability of testing positive
		is 0.01000980
			The probability of testing positive
			is 0.01000980
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 1.00 out of 1.00

Consider two medical tests, A and B, for a virus. Test A is 95% effective at recognising the virus when the virus is present, but has a 10% false positive rate (indicating that the virus is present, when it is not). Test B is 90% effective at recognizing the virus, but has a 5% false positive rate. The two tests use independent methods of identifying the virus. The virus is carried by 1% of all people.

Express these facts in the form of a (causal) belief network. Use variable names `'A'`, `'B'`, and `'Virus'`. Assign the network to the variable `network`.

Important: Supply the `query` function and all the functions and modules it depends on (e.g. `joint_prob`) from the previous questions.

For example:

Test	Result

`from student_answer import query, network`
 	
`answer = query(network, 'Virus', {'A': True})`
`print("The probability of carrying the virus\n"`
`      "if test A is positive: {:.5f}"`
`      .format(answer[True]))`
	`The probability of carrying the virus`
	`if test A is positive: 0.08756`

`from student_answer import query, network`
 	
`answer = query(network, 'Virus', {'B': True})`
`print("The probability of carrying the virus\n"`
`      "if test B is positive: {:.5f}"`
`      .format(answer[True]))`
	`The probability of carrying the virus`
	`if test B is positive: 0.15385`

Answer: (penalty regime: 0, 15, ... %)

```
import itertools







def query(network, query_var, evidence):
    #what is evidence? hidden variable?

    probability_true = 0
    probability_false = 0
    hidden_vars = network.keys() - evidence.keys() - {query_var}
    for values in itertools.product((True, False), repeat=len(hidden_vars)):
        hidden_assignments = {var:val for var,val in zip(hidden_vars, values)}
        probability_true += joint_prob(network, dict({ query_var: True}, **hidden_assignments, **evid
        probability_false += joint_prob(network, dict({ query_var: False}, **hidden_assignments, **ev

    weighted_true = probability_true/(probability_false + probability_true)
    weighted_false = 1 -weighted_true

    formatted = { True : weighted_true, False : weighted_false}
    #print("Hello")
    #print(formatted)
    return formatted
    #print(probability_true)
    #print(probability_false)






def joint_prob(network, assignment):
    probability = 1
    #for every node in network
    for i, j in assignment.items():
        #get the info of each node
        value = network[i]
        plist = []
        #check the parents of each node to get conditional prob
        for x in value['Parents']:
            plist.append(assignment[x])
        #these are the required conditions for each node probability that i want to check
        plist = tuple(plist)
        #get the probability value if its true or false
        if assignment[i] == True:
            probability = probability * value['CPT'][plist]
        if assignment[i] == False:
            probability = probability * (1- value['CPT'][plist])
    return probability
```

```
    return probability




network = {
    'Virus': {
        'Parents': [],
        'CPT': {
            (): 0.01
            }},

    'A': {
        'Parents': ['Virus'],
        'CPT': {
            (True,): 0.95,
            (False,): 0.1,
            }},
    'B': {
        'Parents': ['Virus'],
        'CPT': {
            (True,): 0.90,
            (False,): 0.05,
            }}
    }
```

	Test	Expected	Got	

✔	`from student_answer import query, network`
	
	`answer = query(network, 'Virus', {'A': True})`
	`print("The probability of carrying the virus\n"`
	`      "if test A is positive: {:.5f}"`
	`      .format(answer[True]))`
		The probability of carrying the virus
		if test A is positive: 0.08756
			The probability of carrying the virus
			if test A is positive: 0.08756
				✔

✔	`from student_answer import query, network`
	
	`answer = query(network, 'Virus', {'B': True})`
	`print("The probability of carrying the virus\n"`
	`      "if test B is positive: {:.5f}"`
	`      .format(answer[True]))`
		The probability of carrying the virus
		if test B is positive: 0.15385
			The probability of carrying the virus
			if test B is positive: 0.15385
				✔

✔	`from student_answer import query, network`
	
	`answer = query(network, 'Virus', {'A':True, 'B': True})`
	`print("The probability of carrying the virus\n"`
	`      "if both test A and B are positive: {:.5f}"`
	`      .format(answer[True]))`
		The probability of carrying the virus
		if both test A and B are positive: 0.63333
			The probability of carrying the virus
			if both test A and B are positive: 0.63333
				✔

✔	`from student_answer import query, network`
	
	`answer = query(network, 'Virus', {'A':False, 'B': False})`
	`print("The probability of not carrying the virus\n"`
	`      "if both test A and B are negative: {:.5f}"`
	`      .format(answer[True]))`
		The probability of not carrying the virus
		if both test A and B are negative: 0.00006
			The probability of not carrying the virus
			if both test A and B are negative: 0.00006
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **5**

Correct

Mark 1.00 out of 1.00

Create a belief network with five random variables A, B, C, D, and E with the following properties:

- A and C are independent of any other variable (and each other).
- D and E depend on each other unless B is given.

For example:

Test	Result

`from student_answer import network`
 	
`print(sorted(network.keys()))`
	`['A', 'B', 'C', 'D', 'E']`

Answer: (penalty regime: 0, 15, ... %)

Reset answer

```



network = {
    'A': {
        'Parents': [],
        'CPT': {
            (): 0.2,
            }},
    'C': {
        'Parents': [],
        'CPT': {
            (): 0.1,
            }},
    'D': {
        'Parents': ['B'],
        'CPT': {
            (True,): 0.2,
            (False,): 0.3
            }},
    'E': {
        'Parents': ['B'],
        'CPT': {
            (True,): 0.1,
            (False,): 0.2
            }},
    'B': {
        'Parents': [],
        'CPT': {
            (): 0.2,
            }}
    }

```

	Test	Expected	Got	

✔	`from student_answer import network`
	
	`print(sorted(network.keys()))`
		`['A', 'B', 'C', 'D', 'E']`	`['A', 'B', 'C', 'D', 'E']`	✔

✔	`from student_answer import network`
	
	`print(network['E']['Parents'])`
		`['B']`	`['B']`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **6**

Correct

Mark 1.00 out of 1.00

Consider the belief network given in the answer box with three random variables A, B, and C. The topology of the network implies the following:

- A influences B
- A influences C
- B and C are conditionally independent given A

Without modifying the topology of the network, change the CPTs such that B and C become independent (unconditionally).

Notes

- You can achieve this by making B independent of A or by making C independent of A. While you can do this by simply removing one of the arcs (i.e. parents), here you are asked to do this without changing the topology/parents and by only changing the CPTs.
- When hand-designing belief networks, there is no point in changing CPTs in order to make two variables independent; instead you can (and should) modify the topology.
- When the topology of the network is hand-designed but the CPTs are obtained by looking at data (machine learning), then the values obtained for CPTs may effectively make two variables independent. For example in this network if A is a disease and B and C are some tests, when designing the topology, you may consider A as influencing both B and C but after you use data to obtain the values in CPTs, in turns out that B is independent of A (i.e. does not provide useful information).

For example:

Test	Result

`from student_answer import network`
 	
`print(sorted(network.keys()))`
	`['A', 'B', 'C']`

Answer: (penalty regime: 0, 15, ... %)

Reset answer

```
network = {
    #is there an algorithm or someting i should be following?
    'A': {
        'Parents': [],
        'CPT': {
            (): 0.1
            }},
    'B': {
        'Parents': ['A'],
        'CPT': {
            (False,): 0.2,
            (True,): 0.2
            }},

    'C': {
        'Parents': ['A'],
        'CPT': {
            (False,): 0.4,
            (True,): 0.4
            }},
}
```

	Test	Expected	Got	

✔	`from student_answer import network`
	
	`print(sorted(network.keys()))`
		`['A', 'B', 'C']`	`['A', 'B', 'C']`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

◄ 7. Local and global search

Jump to...

9. Machine learning with naive Bayes nets ►