BayesianNetwork.py

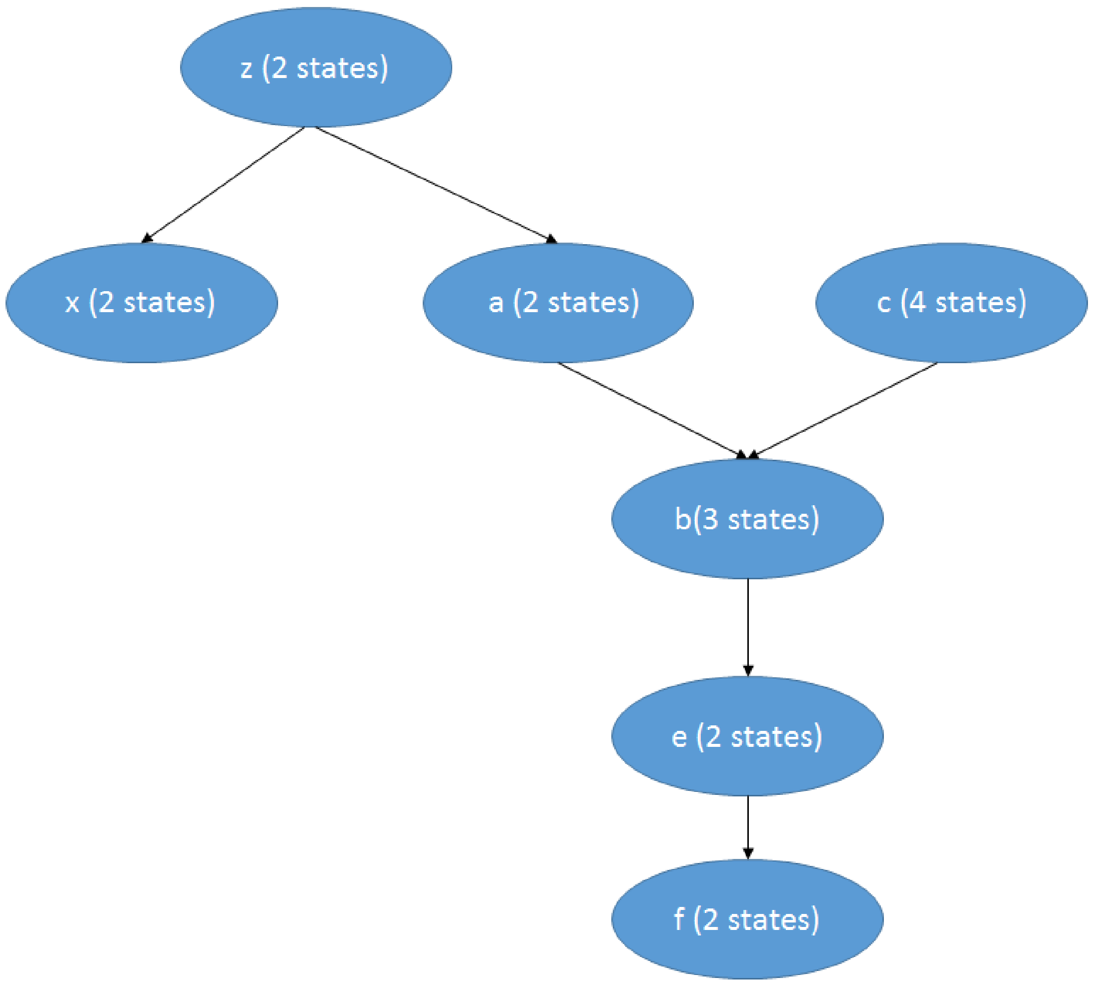
Authors: Swapnil Chhabra, Kyle Hundman
Date: May 6, 2014

Description:

BayesionNetwork.py is a module for implementing a simple Bayesian Network and doing inference on various nodes within the network based on user-provided evidence. Various functionalities are offered to the user to customize and alter their network (more detail provided below).

The module allows for building, editing and saving a Bayesian network of any complexity. Evidence can be set on multiple nodes (either child or parent) and inference can be performed on all nodes.

The following example is used throughout this documentation.



Marginal and Conditional Probabilities for the network shown above :

P(z):

z1	z2
0.45	0.55

P(x):

	z1	z2
x1	0.52	0.85
x2	0.48	0.15

P(a):

	z1	z2
a1	0.32	0.25
a2	0.68	0.75

P(c):

c1	c2	c3	c4
0.1	0.2	0.3	0.4

P(b):

	a1c1	a1c2	a1c3	a1c4	a2c1	a2c2	a2c3	a2c4
b1	0.2	0.1	0.01	0.2	0.33	0.3	0.2	0.9
b2	0.4	0.5	0.01	0.1	0.33	0.1	0.7	0.05
b3	0.4	0.4	0.98	0.7	0.34	0.6	0.1	0.05

P(e):

	b1	b2	b3
e1	0.10	0.30	0.60
e2	0.90	0.70	0.40

P(f):

	e1	e2
f1	0.20	0.55
f2	0.80	0.45

<u>Program Inputs and Outputs:</u>

<u>JSON Formatting:</u>

Two methods are available for building and initializing a network:

1. Loading one JSON file in specified formats
2. Calling functions (detailed below) which allow for entering information directly into the program

For the first method,one JSON file is needed:

- <u>Format</u>:
  [{"Name": "Node_Name","Parents":[List of Parent Nodes],"States":[State_Names],"cpt": [Marginal or Conditional Probabilities]}]

- <u>Example:</u>
  For parent Node "z":
  {"Name": "z","Parents":[],"States":["z1","z2"],"cpt": [0.45, 0.55]}

  For child Node "x":
  {"Name": "x","Parents":["z"],"States":["x1","x2"],"cpt": [[0.52, 0.48],[0.85,0.15]]}

- Above two examples (dictionaries) will be stored in a list, resulting in a list of dictionaries (each dictionary represents one node

Notes:

- **Conditional probabilities for child node must respect order that parent nodes were input.**
- **Parent nodes must be entered in the JSON file before their children**

  - <u>Example</u>: For the child node 'b' with parents 'a' and 'c'

{"States": ["b1", "b2", "b3"], "cpt": [[[0.2, 0.4, 0.4], [0.33, 0.33, 0.34]], [[0.1, 0.5, 0.4], [0.3, 0.1, 0.6]], [[0.01, 0.01, 0.98], [0.2, 0.7, 0.1]], [[0.2, 0.1, 0.7], [0.9, 0.05, 0.05]]], "Parents": ["c", "a"], "Name": "b"}

Functionalities:

Load in a network in the specified JSON format:

```
>>>load("filepath")
```

Create node by inputting node information directly into program*:

```
>>>createNode()
```

*All parents need to be created before their children

Delete a node from network (children must be deleted before their parent node is deleted):

```
>>>deleteNode(node)
```

Set evidence for various nodes:

```
>>> evidence = getUserInput()
```

Do inference on network using nodes and evidence (‘cpts’ will always be names ‘cpts’, evidence is label assigned to getUserInput() function:

```
>>> doInference(cpts + setEvidenceList(evidence))
```

Save node to JSON file and quit current program:

```
>>> save("filepath")
```

Test Case:

If the user would like to load pre-existing ‘nodes’ and ‘network’ JSON files, quickly do inference, and save a network, the following lines of code can be run:

```
>>>load("filepath")
>>>evidences = getUserInput()
>>>doInference(cpts + setEvidenceList(evidences))
>>>save("filepath")
```

Example:

We have the option of starting out with a partially built network and add/delete nodes to it or start out from scratch. In this example we start out with a partial network of 3 nodes (z,a,x) corresponding to the figure shown above:
z->a and z--x

The input json that contains this partial network will look like this:

[{"Name": "z","Parents":[],"States":["z1","z2"],"cpt": [0.45, 0.55]},{"Name": "x","Parents":["z"],"States":["x1","x2"],"cpt": [[0.52, 0.48],[0.85,0.15]]},
{"Name": "a","Parents":["z"],"States":["a1","a2"],"cpt":[[0.32, 0.68],[0.25,0.75]]}]

Note: To build the JSON file please enter information for the parent nodes prior to information for the child nodes.

We will now build this network to the complete one shown in the figure above. We will run an inference based on user provided evidence and then build an output json that captures nodes and probabilities for the full network.

Since we will be adding four additional nodes we will call the createNode function those many times.

Note: We must work our way from the top to the bottom. (user input shown in bold orange)

Python Code:

```
load("C:/Users/Swap Chhabra/workspace/Test1/input2.json") #loads in z,x,a
load("C:/Users/Swap Chhabra/workspace/Test1/input2.json") #loads in z,x,a
createNode() #create c [ 0.1,  0.2,  0.3,  0.4]
createNode() #create b [[[ 0.2 ,  0.4 ,  0.4 ],[ 0.33,  0.33,  0.34]],[[ 0.1 ,  0.5 ,  0.4 ],[ 0.3 ,  0.1 ,  0.6 ]],[[ 0.01,  0.01,  0.98],[ 0.2 ,  0.7 ,  0.1 ]],[[ 0.2 ,  0.1 ,  0.7 ],[ 0.9 ,  0.05,  0.05]]]
createNode() #create e [[.1, .9],[.3,.7],[.6,.4]]
createNode() #create f [[.2, .8],[.55,.45]]
deleteNode('b') #won't work because it is a parent
deleteNode('f') #will work because nothing is dependent on it
evidences = getUserInput()
doInference(cpts + setEvidenceList(evidences))
save("C:/Users/Swap Chhabra/workspace/Test1/output.json")
```

User interface:
Starting out with our partial network that we loaded using the input file:

```
> Please enter name of new node (parent nodes must be created before their children): c
>Please enter the number of parents for new node (parent nodes must be created before children): 0
>Please enter number of states for new node: 4
>Please enter name for state 1: c1
>Please enter name for state 2: c2
>Please enter name for state 3: c3
>Please enter name for state 4: c4
>Please enter marginal probability table for node c in the form of a list: [ 0.1,  0.2,  0.3,  0.4]
```

```
>Please enter name of new node (parent nodes must be created before their children): b
>Please enter the number of parents for new node (parent nodes must be created before
children): 2
>Please enter number of states for new node: 3
>Please enter name for state 1: b1
>Please enter name for state 2: b2
>Please enter name for state 3: b3
>Please enter name of parent 1 in order that parent was entered: a
>Please enter name of parent 2 in order that parent was entered: c
>Please enter conditional probability table with respect to the order that parents were entered:
 ['a', 'c'][[[ 0.2 ,  0.4 ,  0.4 ],[ 0.33,  0.33,  0.34]],[[ 0.1 ,  0.5 ,  0.4 ],[ 0.3 ,  0.1 ,  0.6 ]],[[ 0.01,
0.01,  0.98],[ 0.2 ,  0.7 ,  0.1 ]],[[ 0.2 ,  0.1 ,  0.7 ],[ 0.9 ,  0.05,  0.05]]]
>Please enter name of new node (parent nodes must be created before their children): e
>Please enter the number of parents for new node (parent nodes must be created before
children): 1
>Please enter number of states for new node: 2
>Please enter name for state 1: e1
>Please enter name for state 2: e2
>Please enter name of parent 1 in order that parent was entered: b
>Please enter conditional probability table with respect to the order that parents were entered:
 ['b'][[.1, .9],[.3,.7],[.6,.4]]
>Please enter name of new node (parent nodes must be created before their children): f
>Please enter the number of parents for new node (parent nodes must be created before
children): 1
>Please enter number of states for new node: 2
>Please enter name for state 1: f1
>Please enter name for state 2: f2
>Please enter name of parent 1 in order that parent was entered: e
>Please enter conditional probability table with respect to the order that parents were entered:
 ['e'][[.2, .8],[.55,.45]]
```

At this point we can modify the network by deleting existing nodes or proceed forward with inference. To delete nodes this program allows only deletion of nodes that don't have any dependencies

```
deleteNode('b') #won't work because it is a parent
```

We will get the following message:
"This node cannot be deleted because it has child nodes that are dependent on it.
Please delete child nodes of this node before deleting this node.

```
deleteNode('f') #will work because nothing is dependent on it
```

In this example we have deleted node 'f'

Now that we have built the network lets run an inference on it.

```
>How many nodes do you have evidence for? 3

>Enter name of node 1: x
>Enter total number of states for this node: 2
>Which of these 2 states for node x has evidence? Enter one number: 1

>Enter name of node 2: c
>Enter total number of states for this node: 4
>Which of these 4 states for node c has evidence? Enter one number: 3

>Enter name of node 3: e
>Enter total number of states for this node: 2
>Which of these 2 states for node e has evidence? Enter one number: 2
```

Output

The program outputs the resulting probabilities of the entire network.

```
a -> [ 0.17774657  0.82225343]
b -> [ 0.2123795   0.57052025  0.21710025]
c -> [ 0.  0.  1.  0.]
e -> [ 0.  1.]
x -> [ 1.  0.]
z -> [ 0.32608114  0.67391886]
```

Note that we deleted node ‘f’.
We also get the all the joint probabilities (not shown)

Note that we also create an output json which we may be reloaded to do the following:
1. modify the network i.e. add or delete nodes using the functions:
    - createNode() or
    - deleteNode(‘Node’) functions
2. run another inference using the functions:
    - evidences = getUserInput()
    - doInference(cpts + setEvidenceList(evidences))