tsBNgen

Release 1.0.0

Manie Tadayon

Sep 08, 2020

CONTENTS:

CHAPTER ONE

DESCRIPTION

tsBNgen is a Python library to generate time series data based on an arbitrary dynamic Bayesian network. The intention behind writing tsBNgen is to let researchers geenrate time series data according to arbitrary model they want.

tsBNgen is released under the MIT license.

Following are the features and capabilities of this software:

- It handles discrete nodes, continous nodes and hybrid (Mixture of discrete and continuous) network.
- It uses multinomila distribution for the discrete nodes and Gaussian distribution for the continuous nodes.
- It handles arbitrary Bayesian network structure.
- It supports arbitrary loopback values.
- The code can be modified easily to handle arbitrary static and temporal structures.

CHAPTER

TWO

INSTRUCTION

Either clone this repository https://github.com/manitadayon/tsBNgen or install the package using

```
pip install tsBNgen.
```

Then import necessary libraries using the following commands:

```
from tsBNgen import *
from tsBNgen.tsBNgen import *
```

There are in general two functions you should be running if you want to generate data:

- BN_data_gen(): Use this function under the following conditions: custom_time variable is not specified and the value of the loopback for all the variables is at most 1.

Note: condition 1 describes the classical dynamic Bayesian network in which some nodes at time t-1 are connected to themselves at time t.

- BN_sample_gen_loopback():
  1. custom_time is not specified and you want the loopback value for some nodes to be at most 2.
  2. custom_time is specified and it is at least equal to the maximum loopback value of the loopbacks2.

Following are the explanation of the some of the variables and parameters in tsBNgen:

- **T** : Length of each time series.
- **N** : Number of samples.
- **N_level** : list. Number of possible levels for discrete nodes.
- **Mat** : data-frame. Adjacency matrix for each time point.
- **Node_Type** :list. Type of each variable in Bayesian Network.
- **CPD** : dict. Conditonal Probability Distribution for initial time point.
- **Parent** : dict. Identifying parent of each node in Bayesian network at initial time.
- **CPD2** : dict. Conditonal Probability Distribution.
- **Parent2** : dict. Identifying parent of each node in Bayesian network.
- **loopbacks** : dict. Describing the temporal interconnection between nodes.
- **CPD3** : dict. Conditonal Probability Distribution. Use this entry when BN_sample_gen_loopback() is called.
- **Parent3** : dict. Identifying parent of each node in Bayesian network. Use this entry whenBN_sample_gen_loopback() is called.

- **loopback2** : dict. Describing the temporal interconnection between nodes. Use this entry whenBN_sample_gen_loopback() is called.

Example 1

```
from tsBNgen import *
from tsBNgen.tsBNgen import *
T=20
N=2000
N_level=[2,4]
Mat=pd.DataFrame(np.array(([0,1,1],[0,0,1],[0,0,0])))
Node_Type=['D','D','C']
CPD={'0':[0.6,0.4],'01':[[0.5,0.3,0.15,0.05],[0.1,0.15,0.3,0.45]],'012':{'mu0':10,
↪'sigma0':2,'mu1':30,'sigma1':5,
    'mu2':50,'sigma2':5,'mu3':70,'sigma3':5,'mu4':15,'sigma4':5,'mu5':50,'sigma5':5,
↪'mu6':70,'sigma6':5,'mu7':90,'sigma7':3
}}
Parent={'0':[],'1':[0],'2':[0,1]}

CPD2={'00':[[0.7,0.3],[0.2,0.8]],'011':[[0.7,0.2,0.1,0],[0.6,0.3,0.05,0.05],[0.35,0.5,
↪0.15,0],
[0.2,0.3,0.4,0.1],[0.3,0.3,0.2,0.2],[0.1,0.2,0.3,0.4],[0.05,0.15,0.3,0.5],[0,0.05,0.
↪25,0.7]],'012':{'mu0':10,'sigma0':2,'mu1':30,'sigma1':5,
    'mu2':50,'sigma2':5,'mu3':70,'sigma3':5,'mu4':15,'sigma4':5,'mu5':50,'sigma5':5,
↪'mu6':70,'sigma6':5,'mu7':90,'sigma7':3
}}

Parent2={'0':[0],'1':[0,1],'2':[0,1]}
loopbacks={'00':[1],'11':[1]}
Parent2={'0':[0],'1':[0,1],'2':[0,1]}
Time_series1=tsBNgen(T,N,N_level,Mat,Node_Type,CPD,Parent,CPD2,Parent2,loopbacks)
Time_series1.BN_data_gen()
```

Example 2

```
from tsBNgen import *
from tsBNgen.tsBNgen import *
T=10
N=1000
N_level=[2,4]
Mat=pd.DataFrame(np.array(([0,1,0],[0,0,1],[0,0,0])))
Node_Type=['D','D','C']
CPD={'0':[0.5,0.5],'01':[[0.6,0.3,0.05,0.05],[0.1,0.2,0.3,0.4]],'12':{'mu0':10,'sigma0
↪':5,'mu1':30,'sigma1':5,
    'mu2':60,'sigma2':5,'mu3':80,'sigma3':5}}
Parent={'0':[],'1':[0],'2':[1]}

CPD2={'00':[[0.7,0.3],[0.3,0.7]],'0011':[[0.7,0.2,0.1,0],[0.5,0.4,0.1,0],[0.45,0.45,0.
↪1,0],
[0.3,0.4,0.2,0.1],[0.4,0.4,0.1,0.1],[0.2,0.3,0.3,0.2],[0.2,0.3,0.3,0.2],[0.1,0.2,0.3,
↪0.4],[0.3,0.4,0.2,0.1],[0.2,0.2,0.4,0.2],
 [0.2,0.1,0.4,0.3],[0.05,0.15,0.3,0.5],[0.1,0.3,0.3,0.3],[0,0.1,0.3,0.6],[0,0.1,0.2,0.
↪7],[0,0,0.3,0.7]],'112':{'mu0':10,'sigma0':2,'mu1':30,'sigma1':2,
    'mu2':50,'sigma2':2,'mu3':60,'sigma3':5,'mu4':20,'sigma4':2,'mu5':25,'sigma5':5,
↪'mu6':50,'sigma6':5,'mu7':60,'sigma7':5,
   'mu8':40,'sigma8':5,'mu9':50,'sigma9':5,'mu10':70,'sigma10':5,'mu11':85,'sigma11
↪':2,'mu12':60,'sigma12':5,
```

(continues on next page)

(continued from previous page)

```
    'mu13':60,'sigma13':5,'mu14':80,'sigma14':3,'mu15':90,'sigma15':3}}

Parent2={'0':[0],'1':[0,0,1],'2':[1,1]}
loopbacks={'00':[1], '01':[1],'11':[1],'12':[1]}

Time_series2=tsBNgen(T,N,N_level,Mat,Node_Type,CPD,Parent,CPD2,Parent2,loopbacks)
Time_series2.BN_data_gen()
```

CHAPTER

THREE

TSBNGEN

3.1 tsBNgen package

3.1.1 Submodules

3.1.2 tsBNgen.tsBNgen module

class tsBNgen.tsBNgen.**tsBNgen**(*T*, *N*, *N_level*, *Mat*, *Node_Type*, *CPD*, *Parent*, *CPD2*, *Parent2*, *loopbacks*, *CPD3=None*, *Parent3=None*, *loopbacks2=None*, *custom_time=0*)

Bases: `object`

BFS(*Row*)

Perform Breadth-first search for the given node(row).

Parameters **Row** (*int*) – Corresponds to the row (node) in adjacency matrix.

Returns The node and all its children.

Return type list

BN_data_gen()

It uses Initial_sample for initial time(t=0) and BN_sample for time point t=1 up to time t=T (length of time series)

Parameters **None** –

Returns None

Return type None

Raises **Exception** – Parent of a discrete node cannot be continuous

Notes

Use this function under the following conditions: custom_time variable is not specified and the value of the loopback for all the variables is at most 1

BN_sample()

Generate samples for all the nodes after the initial time.

Parameters **None** –

Returns

Return type None

Notes

Use this function to generate samples if the loopback values are at most one. Loopback=1 means that a node at time t is connected to the node at t-1.

Raises **`Exception`** – Parent of a discrete node cannot be continuous

`BN_sample_gen_loopback``()`

Generate time series data for all the nodes for all time. See Notes for the more information.

Parameters **`None`** –

Returns

Return type None

Raises **`Exception`** – Parent of a discrete node cannot be continuous

Notes

This is more general form of BN_data_gen that supports only two different BN structures or loopback value of maximum one for all the nodes.

`BN_sample_loopback``()`

Generate samples for all the nodes given CPD3 and Parent3 are used.

Parameters **`None`** –

Returns

Return type None

Raises **`Exception`** – Parent of a discrete node cannot be continuous

Notes

Use this function when you want to incorporate three BNs.

`Child``(`*Row*`)`

Finds the children of the node specified by the row of the adjacency matrix.

Parameters **`Row`** (*`int`*) – The row in the adjacency matrix, corresponding to the same node in a Bayesian network.

Returns All the children of the given node.

Return type list

`DAG_ordering``()`

Find the topological ordering of the graph

Parameters **`None`** –

Returns

Return type None

`Gaussian_select``(`*index1*, *index2*, *ii=0*`)`

`Initial_sample``()`

Generate samples for all the nodes at initial time (t=0)

Parameters **`None`** –

Returns

Return type None

Raises **`Exception`** – Parent of a discrete node cannot be continuous

`Level_multiplied`()

`Multinomial_Select`(*index1*, *index2*, *ii=0*)

Generate sample according to the Multinomial distribution.

Parameters

- **`index1`** (*`string`*) – key values of dictionary in CPD/CPD2/CPD3
- **`index2`** (*`int`*) – Determine which CPD entry to select.
- **`ii`** (*`int`*) – The node to generate the sample for. It defaults to 0.

Returns The new generated sample.

Return type int

`Role_Assignment`()

Identify the root node.

Parameters **`None`** –

Returns

Return type None

`Roots_length`()

Identifying the number of root nodes.

Parameters **`None`** –

Returns Number of root nodes.

Return type int

`Valid_BN`(*parent*)

Verify whether the parent-child relationships between nodes are valid.

Parameters **`parent`** (*`dict`*) – dictionary where the keys are the nodes and the values are the list of parents.

Returns

Return type None

Raises **`Exception`** – "Parent of a discrete node cannot be continuous"

`continous_cpd`()

`static int_to_str`(*List*)

Concatenates list elements to a string.

Parameters **`List`** (*`list`*) –

Returns concatenated list elements as a string.

Return type string

`parents_len`(*Node*)

`static zero_loc`(*List*)

Find the index of zero values in a list.

Parameters `List` (*list*) –

Returns indices of zero values in a list.

Return type list

3.1.3 Module contents

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX