```python
import os
from typing import *
```

```python
os.getcwd()
# Setting the baseline:
os.chdir('/development/projects/statisticallyfit/github/learningmathstat/PythonNeuralNetNLP'

curPath: str = os.getcwd() + "/src/CausalNexStudy/"

dataPath: str = curPath + "data/student/"


print("curPath = ", curPath, "\n")
print("dataPath = ", dataPath, "\n")
```

```
curPath =  /development/projects/statisticallyfit/github/learningmathstat/PythonNeuralNetNLP

dataPath =  /development/projects/statisticallyfit/github/learningmathstat/PythonNeuralNetNL
```

```python
import sys
# Making files in utils folder visible here: to import my local print functions for nn.Modu
sys.path.append(os.getcwd() + "/src/utils/")
# For being able to import files within CausalNex folder
sys.path.append(curPath)

sys.path
```

```
['/development/projects/statisticallyfit/github/learningmathstat/PythonNeuralNetNLP/src/Caus
 '/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python37.zip',
 '/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7',
 '/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/lib-dynload',
 '',
 '/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages',
 '/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/IPython
 '/home/statisticallyfit/.ipython',
 '/development/projects/statisticallyfit/github/learningmathstat/PythonNeuralNetNLP/src/util
 '/development/projects/statisticallyfit/github/learningmathstat/PythonNeuralNetNLP/src/Caus
```

1/ Structure Learning

Structure from Domain Knowledge

We can manually define a structure model by specifying the relationships between different features. First we must create an empty structure model.

```python
from causalnex.structure import StructureModel

structureModel: StructureModel = StructureModel()
structureModel
```

```
<causalnex.structure.structuremodel.StructureModel at 0x7f6d14067fd0>
```

Next we can specify the relationships between features. Let us assume that experts tell us the following causal relationships are known (where G1 is grade in semester 1):

- `health` $\longrightarrow$ `absences`
- `health` $\longrightarrow$ `G1`

```python
structureModel.add_edges_from([
    ('health', 'absences'),
    ('health', 'G1')
])
```

Visualizing the Structure

```python
structureModel.edges
```

```
OutEdgeView([('health', 'absences'), ('health', 'G1')])
```

```python
structureModel.nodes
```

```
NodeView(('health', 'absences', 'G1'))
```

```python
from IPython.display import Image
from causalnex.plots import plot_structure, NODE_STYLE, EDGE_STYLE

viz = plot_structure(
    structureModel,
    graph_attributes={"scale": "0.5"},
    all_node_attributes=NODE_STYLE.WEAK,
    all_edge_attributes=EDGE_STYLE.WEAK)
filename_first = curPath + "structure_model_first.png"

viz.draw(filename_first)
Image(filename_first)
```

```
/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/pygraphvi
Warning: node 'absences', graph '%3' size too small for label
Warning: node 'G1', graph '%3' size too small for label

  warnings.warn(b"".join(errors).decode(self.encoding), RuntimeWarning)
```

Figure 1: png

Learning the Structure

Can use CausalNex to learn structure model from data, when number of variables grows or domain knowledge does not exist. (Algorithm used is the NOTEARS algorithm). * NOTE: not always necessary to train / test split because structure learning should be a joint effort between machine learning and domain experts.

First must pre-process the data so the NOTEARS algorithm can be used.

Preparing the Data for Structure Learning

```
import pandas as pd
from pandas.core.frame import DataFrame
```

```python
fileName: str = dataPath + 'student-por.csv'
data: DataFrame = pd.read_csv(fileName, delimiter = ';')

data.head(10)
```

school

sex

age

address

famsize

Pstatus

Medu

Fedu

Mjob

Fjob

...

famrel

freetime

goout

Dalc

Walc

health

absences

G1

G2

G3

0

GP

F

18

U

GT3

A

4

4

at_home

teacher

...

4

3

4

1

1

3

4

0

11

11

1

GP

F

17

U

GT3

T

1

1

at_home

other

...

5

3

3

1

1

3

2

9

11

11

2

GP

F

15

U

LE3

T

1

1

at_home

other

...

4

3

2

2

3

3

6

12

13

12

3

GP

F

15

U

GT3

T

4

2

health

services

. . .

3

2

2

1

1

5

0

14

14

14

4

GP

F

16

U

GT3

T

3

3

other

other

. . .
4
3
2
1
2
5
0
11
13
13
5
GP
M
16
U
LE3
T
4
3
services
other
. . .
5
4
2
1
2
5
6
12

12

13

6

GP

M

16

U

LE3

T

2

2

other

other

...

4

4

4

1

1

3

0

13

12

13

7

GP

F

17

U

GT3

A

4

4

other

teacher

...

4

1

4

1

1

1

2

10

13

13

8

GP

M

15

U

LE3

A

3

2

services

other

...

4

2

2

1

1

1

0

15

16

17

9

GP

M

15

U

GT3

T

3

4

other

other

...

5

5

1

1

1

5

0

12

12

13

10 rows × 33 columns

Can see the features are numeric and non-numeric. Can drop sensitive features like gender that we do not want to include in our model.

```
iDropCol: List[int] = ['school','sex','age','Mjob', 'Fjob','reason','guardian']

data = data.drop(columns = iDropCol)
data.head(5)
```

address

famsize

Pstatus

Medu

Fedu

traveltime

studytime

failures

schoolsup

famsup

...

famrel

freetime

goout

Dalc

Walc

health

absences

G1

G2

G3

0

U

GT3

A

4

4

2

2

0

yes

no

. . .

4

3

4

1

1

3

4

0

11

11

1

U

GT3

T

1

1

1

2

0

no

yes

. . .

5

3

3

1

1

3

2

9

11

11

2

U

LE3

T

1

1

1

2

0

yes

no

...

4

3

2

2

3

3

6

12

13

12

3

U

GT3

T

4

2

1

3

0

no

yes

...

3

2

2

1

1

5

0

14

14

14

4

U

GT3

T

3

3

1

2

0

no

yes

…

4

3

2

1

2

5

0

11

13

13

5 rows × 26 columns

Next we want tomake our data numeric since this is what the NOTEARS algorithm expects. We can do this by label-encoding the non-numeric variables (to make them also numeric, like the current numeric variables).

```
import numpy as np


structData: DataFrame = data.copy()

# This operation below excludes all column variables that are number variables (so keeping o
structData.select_dtypes(exclude=[np.number]).head(5)
```

address

famsize

Pstatus

schoolsup

famsup

paid

activities

nursery

higher

internet

romantic

0

U

GT3

A

yes

no

no

no

yes

yes

no

no

1

U

GT3

T

no

yes

no

no

no

yes

yes

no

2

U

LE3

T

yes

no

no

no

yes

yes

yes

no

3

U

GT3

T

no

yes

no

yes

yes

yes

yes

yes

4

U

GT3

T

no

yes

no

no

yes

yes

no

no

```
# Getting the names of the categorical variables (columns)
structData.select_dtypes(exclude=[np.number]).columns
```

```
Index(['address', 'famsize', 'Pstatus', 'schoolsup', 'famsup', 'paid',
       'activities', 'nursery', 'higher', 'internet', 'romantic'],
      dtype='object')
```

```
namesOfCategoricalVars: List[str] = list(structData.select_dtypes(exclude=[np.number]).colu
namesOfCategoricalVars
```

```
['address',
 'famsize',
 'Pstatus',
 'schoolsup',
 'famsup',
 'paid',
 'activities',
 'nursery',
 'higher',
 'internet',
 'romantic']
```

```
from sklearn.preprocessing import LabelEncoder

labelEncoder: LabelEncoder = LabelEncoder()

# NOTE: structData keeps also the numeric columns, doesn't exclude them! just updates the n
for varName in namesOfCategoricalVars:
    structData[varName] = labelEncoder.fit_transform(y = structData[varName])
```

```
structData.head(5)
```

address

famsize

Pstatus

Medu

Fedu

traveltime

studytime

failures

schoolsup

famsup

…

famrel

freetime

goout

Dalc

Walc

health

absences

G1

G2

G3

0

1

0

0

4

4

2

2

0

1

0

. . .

4

3

4

1

1

3

4

0

11

11

1

1

0

1

1

1

1

2

0

0

1

...

5

3

3

1

1

3

2

9

11

11

2

1

1

1

1

1

1

2

0

1

0

...

4

3

2

2

3

3

6

12

13

12

3

1

0

1

4

2

1

3

0

0

1

...

3

2

2

1

1

5

0

14

14

14

4

1

0

1

3

3

1

2

0

0

1

...

4

3

2

1

2

5

0

11

13

13

5 rows × 26 columns

```
# Going to compare the converted numeric values to their previous categorical values:
namesOfCategoricalVars
```

```
['address',
 'famsize',
 'Pstatus',
 'schoolsup',
 'famsup',
 'paid',
 'activities',
 'nursery',
 'higher',
 'internet',
 'romantic']
```

```python
categData: DataFrame = data.select_dtypes(exclude=[np.number])
```

```python
# The different values of Address variable (R and U)
np.unique(categData['address'])
```

```
array(['R', 'U'], dtype=object)
```

```python
np.unique(categData['famsize'])
```

```
array(['GT3', 'LE3'], dtype=object)
```

```python
np.unique(categData['Pstatus'])
```

```
array(['A', 'T'], dtype=object)
```

```python
np.unique(categData['schoolsup'])
```

```
array(['no', 'yes'], dtype=object)
```

```python
np.unique(categData['famsup'])
```

```
array(['no', 'yes'], dtype=object)
```

```python
np.unique(categData['paid'])
```

```
array(['no', 'yes'], dtype=object)
```

```python
np.unique(categData['activities'])
```

```
array(['no', 'yes'], dtype=object)
```

```
np.unique(categData['nursery'])
```

```
array(['no', 'yes'], dtype=object)
```

```
np.unique(categData['higher'])
```

```
array(['no', 'yes'], dtype=object)
```

```
np.unique(categData['internet'])
```

```
array(['no', 'yes'], dtype=object)
```

```
np.unique(categData['romantic'])
```

```
array(['no', 'yes'], dtype=object)
```

```
# A numeric column:
np.unique(data['Medu'])
```

```
array([0, 1, 2, 3, 4])
```

```
# All the values we convert in structData are binary, so testing how a non-binary one gets
testMultivals: List[str] = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

assert list(labelEncoder.fit_transform(y = testMultivals)) == [0, 1, 2, 3, 4, 5, 6, 7]
```

Now apply the NOTEARS algo to learn the structure:

```
#from src.utils.Clock import *

def clock(startTime, endTime):
    elapsedTime = endTime - startTime
    elapsedMins = int(elapsedTime / 60)
    elapsedSecs = int(elapsedTime - (elapsedMins * 60))
    return elapsedMins, elapsedSecs

from causalnex.structure.notears import from_pandas
import time

startTime: float = time.time()

structureModelLearned = from_pandas(X = structData)

print(f"Time taken = {clock(startTime = startTime, endTime = time.time())}")
```

```
Time taken = (6, 1)
```

```python
# Now visualize it:
viz = plot_structure(
    structureModelLearned,
    graph_attributes={"scale": "0.5"},
    all_node_attributes=NODE_STYLE.WEAK,
    all_edge_attributes=EDGE_STYLE.WEAK)
filename_learned = curPath + "structure_model_learnedStructure.png"

viz.draw(filename_learned)
Image(filename_learned)
```

```
/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/pygraphvi
Warning: node 'famsize', graph '%3' size too small for label
Warning: node 'Pstatus', graph '%3' size too small for label
Warning: node 'Medu', graph '%3' size too small for label
Warning: node 'Fedu', graph '%3' size too small for label
Warning: node 'traveltime', graph '%3' size too small for label
Warning: node 'studytime', graph '%3' size too small for label
Warning: node 'failures', graph '%3' size too small for label
Warning: node 'schoolsup', graph '%3' size too small for label
Warning: node 'famsup', graph '%3' size too small for label
Warning: node 'paid', graph '%3' size too small for label
Warning: node 'activities', graph '%3' size too small for label
Warning: node 'nursery', graph '%3' size too small for label
Warning: node 'higher', graph '%3' size too small for label
Warning: node 'internet', graph '%3' size too small for label
Warning: node 'romantic', graph '%3' size too small for label
Warning: node 'famrel', graph '%3' size too small for label
Warning: node 'freetime', graph '%3' size too small for label
Warning: node 'goout', graph '%3' size too small for label
Warning: node 'Dalc', graph '%3' size too small for label
Warning: node 'Walc', graph '%3' size too small for label
Warning: node 'health', graph '%3' size too small for label
Warning: node 'absences', graph '%3' size too small for label
Warning: node 'G1', graph '%3' size too small for label
Warning: node 'G2', graph '%3' size too small for label
Warning: node 'G3', graph '%3' size too small for label

  warnings.warn(b"".join(errors).decode(self.encoding), RuntimeWarning)
```

Can apply thresholding here to prune the algorithm's resulting fully connected

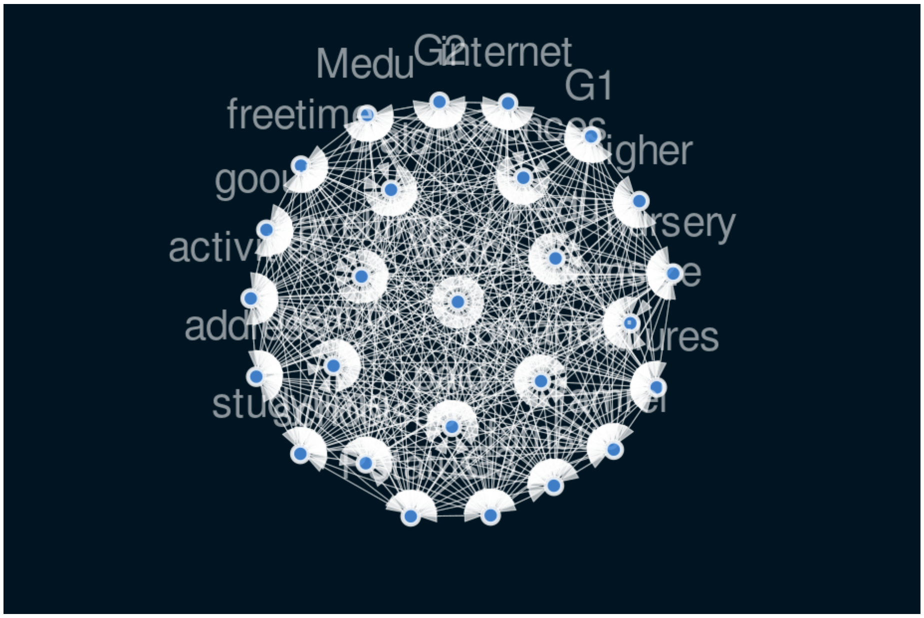

Figure 2: png

graph. Thresholding can be applied either by specifying the value for the parameter `w_threshold` in `from_pandas` or we can remove the edges by calling the structure model function `remove_edges_below_threshold`.

```
structureModelPruned = structureModelLearned.copy()
structureModelPruned.remove_edges_below_threshold(threshold = 0.8)

# Now visualize it:
viz = plot_structure(
    structureModelPruned,
    graph_attributes={"scale": "0.5"},
    all_node_attributes=NODE_STYLE.WEAK,
    all_edge_attributes=EDGE_STYLE.WEAK)
filename_pruned = curPath + "structure_model_pruned.png"
viz.draw(filename_pruned)
Image(filename_pruned)
```

```
/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/pygraphvi
Warning: node 'absences', graph '%3' size too small for label
Warning: node 'G1', graph '%3' size too small for label
Warning: node 'famsize', graph '%3' size too small for label
Warning: node 'Pstatus', graph '%3' size too small for label
```

```
Warning: node 'famrel', graph '%3' size too small for label
Warning: node 'Medu', graph '%3' size too small for label
Warning: node 'Fedu', graph '%3' size too small for label
Warning: node 'traveltime', graph '%3' size too small for label
Warning: node 'studytime', graph '%3' size too small for label
Warning: node 'failures', graph '%3' size too small for label
Warning: node 'schoolsup', graph '%3' size too small for label
Warning: node 'famsup', graph '%3' size too small for label
Warning: node 'paid', graph '%3' size too small for label
Warning: node 'activities', graph '%3' size too small for label
Warning: node 'nursery', graph '%3' size too small for label
Warning: node 'higher', graph '%3' size too small for label
Warning: node 'internet', graph '%3' size too small for label
Warning: node 'romantic', graph '%3' size too small for label
Warning: node 'freetime', graph '%3' size too small for label
Warning: node 'goout', graph '%3' size too small for label
Warning: node 'Dalc', graph '%3' size too small for label
Warning: node 'Walc', graph '%3' size too small for label
Warning: node 'health', graph '%3' size too small for label
Warning: node 'G2', graph '%3' size too small for label
Warning: node 'G3', graph '%3' size too small for label

  warnings.warn(b"".join(errors).decode(self.encoding), RuntimeWarning)
/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/pygraphvi
Warning: node 'absences', graph '%3' size too small for label
Warning: node 'G1', graph '%3' size too small for label
Warning: node 'G2', graph '%3' size too small for label
Warning: node 'G3', graph '%3' size too small for label
Warning: node 'famsize', graph '%3' size too small for label
Warning: node 'Pstatus', graph '%3' size too small for label
Warning: node 'famrel', graph '%3' size too small for label
Warning: node 'Medu', graph '%3' size too small for label
Warning: node 'Fedu', graph '%3' size too small for label
Warning: node 'traveltime', graph '%3' size too small for label
Warning: node 'studytime', graph '%3' size too small for label
Warning: node 'failures', graph '%3' size too small for label
Warning: node 'schoolsup', graph '%3' size too small for label
Warning: node 'famsup', graph '%3' size too small for label
Warning: node 'paid', graph '%3' size too small for label
Warning: node 'activities', graph '%3' size too small for label
Warning: node 'nursery', graph '%3' size too small for label
Warning: node 'higher', graph '%3' size too small for label
Warning: node 'internet', graph '%3' size too small for label
Warning: node 'romantic', graph '%3' size too small for label
Warning: node 'freetime', graph '%3' size too small for label
Warning: node 'goout', graph '%3' size too small for label
```

```
Warning: node 'Dalc', graph '%3' size too small for label
Warning: node 'Walc', graph '%3' size too small for label
Warning: node 'health', graph '%3' size too small for label

  warnings.warn(b"".join(errors).decode(self.encoding), RuntimeWarning)
```

Figure 3: png

Comparing the freshly learned model with the pruned model:

```
structureModelLearned.adj
```

```
AdjacencyView({'address': {'famsize': {'origin': 'learned', 'weight': 0.07172400411745194},
```

```
structureModelPruned.degree
```

```
DiDegreeView({'address': 2, 'famsize': 0, 'Pstatus': 3, 'Medu': 1, 'Fedu': 0, 'traveltime':
```

```
structureModelLearned.edges
```

```
OutEdgeView([('address', 'famsize'), ('address', 'Pstatus'), ('address', 'Medu'), ('address'
```

```
NodeView(('address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'traveltime', 'studytime', 'failu
```

```python
assert structureModelLearned.node == structureModelLearned.nodes
```

```python
structureModelLearned.nodes
```

```
NodeView(('address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'traveltime', 'studytime', 'failu
```

```python
assert structureModelPruned.node == structureModelPruned.nodes
```

```python
structureModelPruned.nodes
```

```
NodeView(('address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'traveltime', 'studytime', 'failu
```

```python
structureModelLearned.out_degree
```

```
OutDegreeView({'address': 25, 'famsize': 25, 'Pstatus': 25, 'Medu': 25, 'Fedu': 25, 'travelt
```

```python
structureModelPruned.out_degree
```

```
OutDegreeView({'address': 2, 'famsize': 0, 'Pstatus': 3, 'Medu': 0, 'Fedu': 0, 'traveltime':
```

```python
structureModelLearned.out_edges
```

```
OutEdgeView([('address', 'famsize'), ('address', 'Pstatus'), ('address', 'Medu'), ('address'
```

AdjacencyView({'address': {'famsize': {'origin': 'learned', 'weight': 0.07172400411745194},

```
 'famsup',
 'paid',
 'activities',
 'nursery',
 'higher',
 'internet',
 'romantic',
 'famrel',
 'freetime',
 'goout',
 'Dalc',
 'Walc',
 'health',
 'absences',
 'G1',
 'G2',
 'G3']
```

```
list(structureModelPruned.neighbors(n = 'address'))
```

```
['absences', 'G1']
```

```
# TODO: what does negative weight mean?
# TODO: why are weights not probabilities?
list(structureModelLearned.adjacency())[:2]
```

```
[('address',
  {'famsize': {'origin': 'learned', 'weight': 0.07172400411745194},
   'Pstatus': {'origin': 'learned', 'weight': 0.027500652131841753},
   'Medu': {'origin': 'learned', 'weight': 0.4329609981782503},
   'Fedu': {'origin': 'learned', 'weight': 0.10940724573937048},
   'traveltime': {'origin': 'learned', 'weight': -0.3080468648891065},
   'studytime': {'origin': 'learned', 'weight': 0.22858517407180592},
   'failures': {'origin': 'learned', 'weight': 0.06633709792506814},
   'schoolsup': {'origin': 'learned', 'weight': 2.265558640319601e-06},
   'famsup': {'origin': 'learned', 'weight': 4.164128335492464e-06},
   'paid': {'origin': 'learned', 'weight': 2.6188325902813357e-06},
   'activities': {'origin': 'learned', 'weight': 8.921883360997223e-06},
   'nursery': {'origin': 'learned', 'weight': 1.0431757754516237e-06},
   'higher': {'origin': 'learned', 'weight': 0.2175470691398659},
   'internet': {'origin': 'learned', 'weight': 4.631899217412905e-07},
   'romantic': {'origin': 'learned', 'weight': 2.1163994047249527e-05},
   'famrel': {'origin': 'learned', 'weight': 0.2713375883408355},
   'freetime': {'origin': 'learned', 'weight': 0.11768720419459214},
   'goout': {'origin': 'learned', 'weight': 0.16392393831724242},
```

```
   'Dalc': {'origin': 'learned', 'weight': 0.11663243893798651},
   'Walc': {'origin': 'learned', 'weight': 0.16559963300289912},
   'health': {'origin': 'learned', 'weight': 0.20294893185551394},
   'absences': {'origin': 'learned', 'weight': 1.0400949529066366},
   'G1': {'origin': 'learned', 'weight': 1.006295091882122},
   'G2': {'origin': 'learned', 'weight': 0.15007496882413057},
   'G3': {'origin': 'learned', 'weight': 0.223096391377955}}),
 ('famsize',
  {'address': {'origin': 'learned', 'weight': 2.57364988344861e-06},
   'Pstatus': {'origin': 'learned', 'weight': -5.39386360384519e-07},
   'Medu': {'origin': 'learned', 'weight': -0.0016220902698672792},
   'Fedu': {'origin': 'learned', 'weight': -0.024651044459558742},
   'traveltime': {'origin': 'learned', 'weight': 0.25181986913147913},
   'studytime': {'origin': 'learned', 'weight': 0.07404468489673609},
   'failures': {'origin': 'learned', 'weight': -0.00011631802985936184},
   'schoolsup': {'origin': 'learned', 'weight': 7.582265421368856e-07},
   'famsup': {'origin': 'learned', 'weight': 8.083571741711851e-06},
   'paid': {'origin': 'learned', 'weight': 5.982031984826393e-07},
   'activities': {'origin': 'learned', 'weight': 1.1369901568939202e-05},
   'nursery': {'origin': 'learned', 'weight': 1.3604190036451818e-06},
   'higher': {'origin': 'learned', 'weight': 3.4544721166046257e-07},
   'internet': {'origin': 'learned', 'weight': 1.985563914894138e-06},
   'romantic': {'origin': 'learned', 'weight': 2.9757663553056567e-05},
   'famrel': {'origin': 'learned', 'weight': 0.23128615865426996},
   'freetime': {'origin': 'learned', 'weight': 0.023554521782170514},
   'goout': {'origin': 'learned', 'weight': -0.089444259197238},
   'Dalc': {'origin': 'learned', 'weight': 0.272822548840043},
   'Walc': {'origin': 'learned', 'weight': 0.21200668687560334},
   'health': {'origin': 'learned', 'weight': 0.07702410821801904},
   'absences': {'origin': 'learned', 'weight': -0.1488343695903593},
   'G1': {'origin': 'learned', 'weight': 0.5361350969644317},
   'G2': {'origin': 'learned', 'weight': 0.032840481295506055},
   'G3': {'origin': 'learned', 'weight': 0.03510912683115285}})]
```

```python
# TODO: what does negative weight mean?
# TODO: why are weights not probabilities?
list(structureModelPruned.adjacency())
```

```
[('address',
  {'absences': {'origin': 'learned', 'weight': 1.0400949529066366},
   'G1': {'origin': 'learned', 'weight': 1.006295091882122}}),
 ('famsize', {}),
 ('Pstatus',
  {'famrel': {'origin': 'learned', 'weight': 0.8402877660070628},
   'absences': {'origin': 'learned', 'weight': -1.0538754156321408},
```

```
  'G1': {'origin': 'learned', 'weight': 1.261362346111696}}),
 ('Medu', {}),
 ('Fedu', {}),
 ('traveltime', {}),
 ('studytime', {'G1': {'origin': 'learned', 'weight': 0.8636139137063454}}),
 ('failures',
  {'absences': {'origin': 'learned', 'weight': 0.9395791571697139}}),
 ('schoolsup', {'G1': {'origin': 'learned', 'weight': -0.8015184747758134}}),
 ('famsup', {}),
 ('paid', {'absences': {'origin': 'learned', 'weight': -1.0534625350951718}}),
 ('activities', {}),
 ('nursery', {}),
 ('higher',
  {'Medu': {'origin': 'learned', 'weight': 0.9842407795725915},
   'G1': {'origin': 'learned', 'weight': 2.6906165356962597}}),
 ('internet',
  {'absences': {'origin': 'learned', 'weight': 0.8369080746968736}}),
 ('romantic', {}),
 ('famrel', {}),
 ('freetime', {}),
 ('goout', {}),
 ('Dalc', {'Walc': {'origin': 'learned', 'weight': 0.8623769618608512}}),
 ('Walc', {}),
 ('health', {}),
 ('absences', {}),
 ('G1', {'G2': {'origin': 'learned', 'weight': 0.8893123602483163}}),
 ('G2', {'G3': {'origin': 'learned', 'weight': 0.884705682463779}}),
 ('G3', {})]
```

```
structureModelLearned.get_edge_data(u = 'address', v = 'G1') # something!
```

```
{'origin': 'learned', 'weight': 1.006295091882122}
```

```
structureModelPruned.get_edge_data(u = 'address', v = 'G1') # something!
```

```
{'origin': 'learned', 'weight': 1.006295091882122}
```

```
structureModelLearned.get_edge_data(u = 'Feduromantic', v = 'absences') # nothing!
```

```
structureModelPruned.get_edge_data(u = 'Feduromantic', v = 'absences') # nothing!
```

```
list(structureModelLearned.get_target_subgraph(node = 'absences').adjacency())[:2]
```

```
[('address',
  {'famsize': {'origin': 'learned', 'weight': 0.07172400411745194},
   'Pstatus': {'origin': 'learned', 'weight': 0.027500652131841753},
   'Medu': {'origin': 'learned', 'weight': 0.4329609981782503},
   'Fedu': {'origin': 'learned', 'weight': 0.10940724573937048},
   'traveltime': {'origin': 'learned', 'weight': -0.3080468648891065},
   'studytime': {'origin': 'learned', 'weight': 0.22858517407180592},
   'failures': {'origin': 'learned', 'weight': 0.06633709792506814},
   'schoolsup': {'origin': 'learned', 'weight': 2.265558640319601e-06},
   'famsup': {'origin': 'learned', 'weight': 4.164128335492464e-06},
   'paid': {'origin': 'learned', 'weight': 2.6188325902813357e-06},
   'activities': {'origin': 'learned', 'weight': 8.921883360997223e-06},
   'nursery': {'origin': 'learned', 'weight': 1.0431757754516237e-06},
   'higher': {'origin': 'learned', 'weight': 0.2175470691398659},
   'internet': {'origin': 'learned', 'weight': 4.631899217412905e-07},
   'romantic': {'origin': 'learned', 'weight': 2.1163994047249527e-05},
   'famrel': {'origin': 'learned', 'weight': 0.2713375883408355},
   'freetime': {'origin': 'learned', 'weight': 0.11768720419459214},
   'goout': {'origin': 'learned', 'weight': 0.16392393831724242},
   'Dalc': {'origin': 'learned', 'weight': 0.11663243893798651},
   'Walc': {'origin': 'learned', 'weight': 0.16559963300289912},
   'health': {'origin': 'learned', 'weight': 0.20294893185551394},
   'absences': {'origin': 'learned', 'weight': 1.0400949529066366},
   'G1': {'origin': 'learned', 'weight': 1.006295091882122},
   'G2': {'origin': 'learned', 'weight': 0.15007496882413057},
   'G3': {'origin': 'learned', 'weight': 0.223096391377955}}),
 ('famsize',
  {'address': {'origin': 'learned', 'weight': 2.57364988344861e-06},
   'Pstatus': {'origin': 'learned', 'weight': -5.39386360384519e-07},
   'Medu': {'origin': 'learned', 'weight': -0.0016220902698672792},
   'Fedu': {'origin': 'learned', 'weight': -0.024651044459558742},
   'traveltime': {'origin': 'learned', 'weight': 0.25181986913147913},
   'studytime': {'origin': 'learned', 'weight': 0.07404468489673609},
   'failures': {'origin': 'learned', 'weight': -0.00011631802985936184},
   'schoolsup': {'origin': 'learned', 'weight': 7.582265421368856e-07},
   'famsup': {'origin': 'learned', 'weight': 8.083571741711851e-06},
   'paid': {'origin': 'learned', 'weight': 5.982031984826393e-07},
   'activities': {'origin': 'learned', 'weight': 1.1369901568939202e-05},
   'nursery': {'origin': 'learned', 'weight': 1.3604190036451818e-06},
   'higher': {'origin': 'learned', 'weight': 3.4544721166046257e-07},
   'internet': {'origin': 'learned', 'weight': 1.985563914894138e-06},
   'romantic': {'origin': 'learned', 'weight': 2.9757663553056567e-05},
   'famrel': {'origin': 'learned', 'weight': 0.23128615865426996},
   'freetime': {'origin': 'learned', 'weight': 0.023554521782170514},
   'goout': {'origin': 'learned', 'weight': -0.089444259197238},
   'Dalc': {'origin': 'learned', 'weight': 0.272822548840043},
```

```
  'Walc': {'origin': 'learned', 'weight': 0.21200668687560334},
  'health': {'origin': 'learned', 'weight': 0.07702410821801904},
  'absences': {'origin': 'learned', 'weight': -0.1488343695903593},
  'G1': {'origin': 'learned', 'weight': 0.5361350969644317},
  'G2': {'origin': 'learned', 'weight': 0.032840481295506055},
  'G3': {'origin': 'learned', 'weight': 0.03510912683115285}})]
```

```
list(structureModelPruned.get_target_subgraph(node = 'absences').adjacency())
```

```
[('address',
  {'absences': {'origin': 'learned', 'weight': 1.0400949529066366},
   'G1': {'origin': 'learned', 'weight': 1.006295091882122}}),
 ('Pstatus',
  {'famrel': {'origin': 'learned', 'weight': 0.8402877660070628},
   'absences': {'origin': 'learned', 'weight': -1.0538754156321408},
   'G1': {'origin': 'learned', 'weight': 1.261362346111696}}),
 ('Medu', {}),
 ('studytime', {'G1': {'origin': 'learned', 'weight': 0.8636139137063454}}),
 ('failures',
  {'absences': {'origin': 'learned', 'weight': 0.9395791571697139}}),
 ('schoolsup', {'G1': {'origin': 'learned', 'weight': -0.8015184747758134}}),
 ('paid', {'absences': {'origin': 'learned', 'weight': -1.0534625350951718}}),
 ('higher',
  {'Medu': {'origin': 'learned', 'weight': 0.9842407795725915},
   'G1': {'origin': 'learned', 'weight': 2.6906165356962597}}),
 ('internet',
  {'absences': {'origin': 'learned', 'weight': 0.8369080746968736}}),
 ('famrel', {}),
 ('absences', {}),
 ('G1', {'G2': {'origin': 'learned', 'weight': 0.8893123602483163}}),
 ('G2', {'G3': {'origin': 'learned', 'weight': 0.884705682463779}}),
 ('G3', {})]
```

In the above structure some relations appear intuitively correct: * `Pstatus` affects `famrel` - if parents live apart, the quality of family relationship may be poor as a result * `internet` affects `absences` - the presence of internet at home may cause stduents to skip class. * `studytime` affects `G1` - longer studytime should have a positive effect on a student's grade in semester 1 (`G1`).

However there are some relations that are certainly incorrect: * `higher` affects `Medu` (Mother's education) - this relationship does not make sense as students who want to pursue higher education does not affect mother's education. It could be the OTHER WAY AROUND.

To avoid these erroneous relationships we can re-run the structure learning with some added constraints. Using the method `from_pandas` from `causalnex.structure.notears` to set the argument `tabu_edges`, with the edge (from –> to) which we do not want to include in the graph.

```
# Reruns the analysis from the structure data, just not including this edge.
# NOT modifying the previous `structureModel`.
structureModel: StructureModel = from_pandas(structData, tabu_edges=[("higher", "Medu")], w_
```

Now the `higher --> Medu` relationship is **no longer** in the graph.

```
# Now visualize it:
viz = plot_structure(
    structureModel,
    graph_attributes={"scale": "0.5"},
    all_node_attributes=NODE_STYLE.WEAK,
    all_edge_attributes=EDGE_STYLE.WEAK)
filename_noHigherMedu = curPath + "structure_model_learnedStructure_noHigherMedu.png"
viz.draw(filename_noHigherMedu)
Image(filename_noHigherMedu)
```

Modifying the Structure (after structure learning)

To correct erroneous relationships, we can incorporate domain knowledge into the model after structure learning. We can modify the structure model through adding and deleting the edges. For example we can add and remove edges with the function `add_edge(u_of_edges, v_of_edges)` that adds a causal relationship from `u` to `v`, where * `u_of_edge` = causal node * `v_of_edge` = effect node

and if the relation doesn't exist it will be created.

```
# NOTE the learning of the graph is different each time so these assertions may not be true
assert not structureModel.has_edge(u = 'higher', v = 'Medu')

# Adding causal relationship from health to paid (used to failures -> G1 ??)
structModeTestEdges = structureModel.copy()

# No edge, showing creation effect
assert not structModeTestEdges.has_edge(u ='health', v ='paid')
structModeTestEdges.add_edge(u_of_edge ='health', v_of_edge ='paid')
assert structModeTestEdges.has_edge(u ='health', v ='paid')
assert {'origin': 'unknown'} == structModeTestEdges.get_edge_data(u ='health', v ='paid')
```

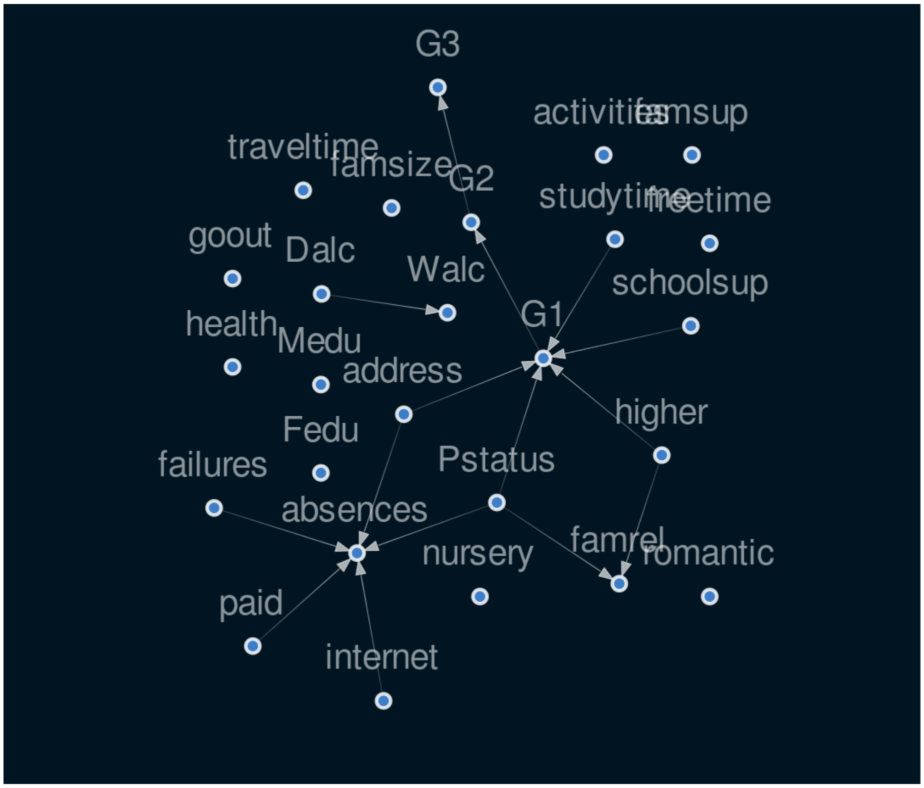
G3
activities
famsup
traveltime
famsize
G2
studytime
freetime
goout
Dalc
Walc
schoolsup
health
Medu
G1
address
higher
Fedu
failures
Pstatus
absences
nursery
famrel
romantic
paid
internet

Figure 4: png

```python
# Has edge, showing replacement effect
assert structModeTestEdges.has_edge(u ='higher', v ='G1')
prevEdge = structModeTestEdges.get_edge_data(u ='higher', v ='G1')
prevEdge
```

```
{'origin': 'learned', 'weight': 2.7243556829495947}
```

```python
structModeTestEdges.add_edge(u_of_edge ='higher', v_of_edge ='G1')
assert structModeTestEdges.has_edge(u ='higher', v ='G1')
curEdge = structModeTestEdges.get_edge_data(u ='higher', v ='G1')
curEdge
assert prevEdge == curEdge
```

```python
# Has edge, showing removal effect
assert structModeTestEdges.has_edge(u ='higher', v ='famrel')
structModeTestEdges.get_edge_data(u ='higher', v ='famrel')
```

```
{'origin': 'learned', 'weight': 0.8896329694730597}
```

```python
structModeTestEdges.remove_edge(u ='higher', v ='famrel')
assert not structModeTestEdges.has_edge(u ='higher', v ='famrel')
```

Can now visualize the updated structure:

```python
viz = plot_structure(
    structModeTestEdges,
    graph_attributes={"scale": "0.5"},
    all_node_attributes=NODE_STYLE.WEAK,
    all_edge_attributes=EDGE_STYLE.WEAK)
filename_testEdges = curPath + "structureModel_testedges.png"
viz.draw(filename_testEdges)
Image(filename_testEdges)
```

```python
# Previous one:
Image(curPath + "structure_model_learnedStructure_noHigherMedu.png")
```

```python
# Just doing same operations on the current graph, after tutorial:
structureModel.add_edge(u_of_edge = 'failures', v_of_edge = 'G1')
# structureModel.remove_edge(u = 'Pstatus', v = 'G1')
# structureModel.remove_edge(u = 'address', v='G1')

viz = plot_structure(
```

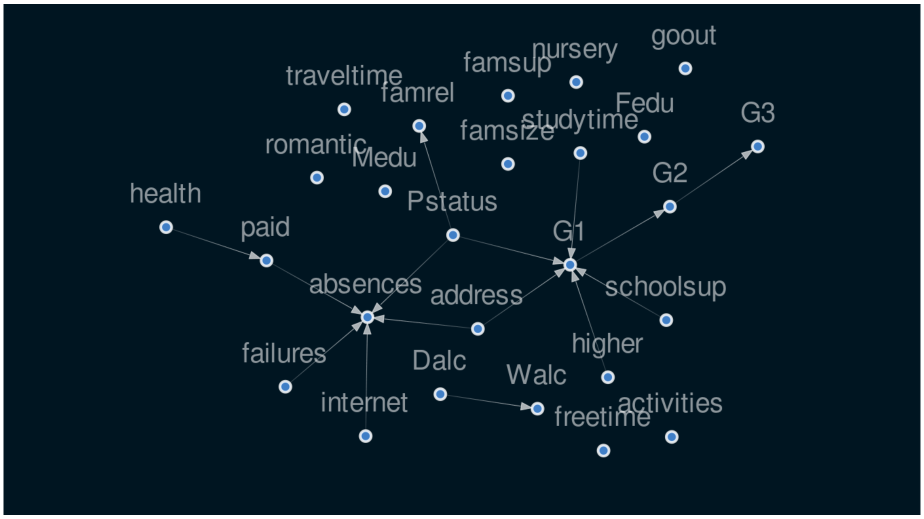

Figure 5: png

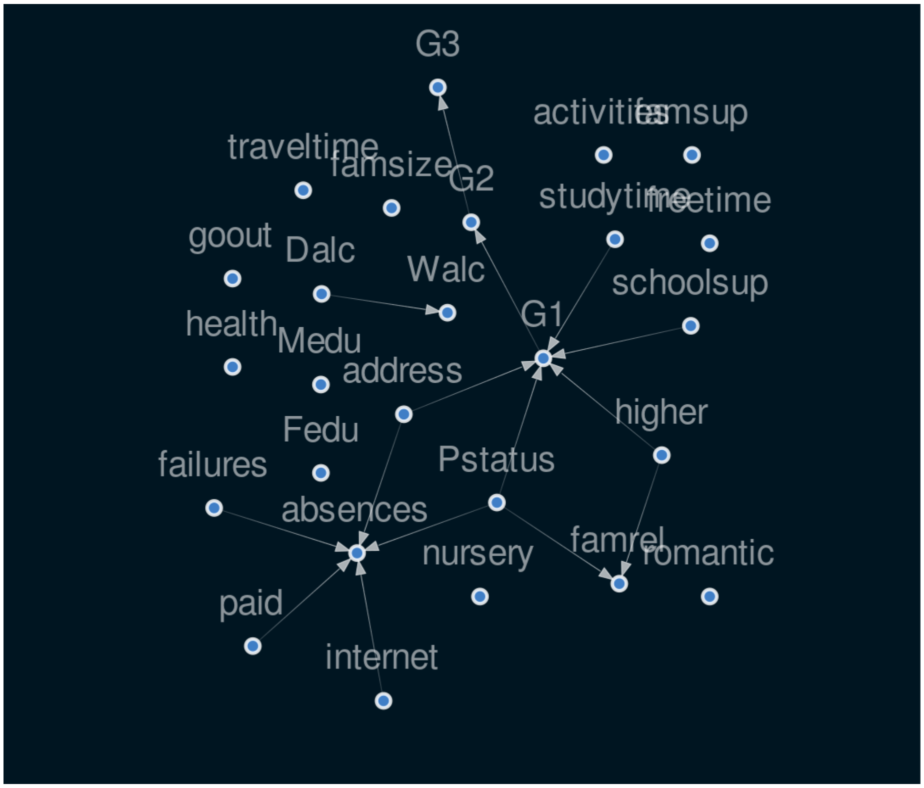

Figure 6: png

```
    structureModel,
    graph_attributes={"scale": "0.5"},
    all_node_attributes=NODE_STYLE.WEAK,
    all_edge_attributes=EDGE_STYLE.WEAK)
filename_updateEdge = curPath + "structureModel_updated.png"
viz.draw(filename_updateEdge)
Image(filename_updateEdge)
```

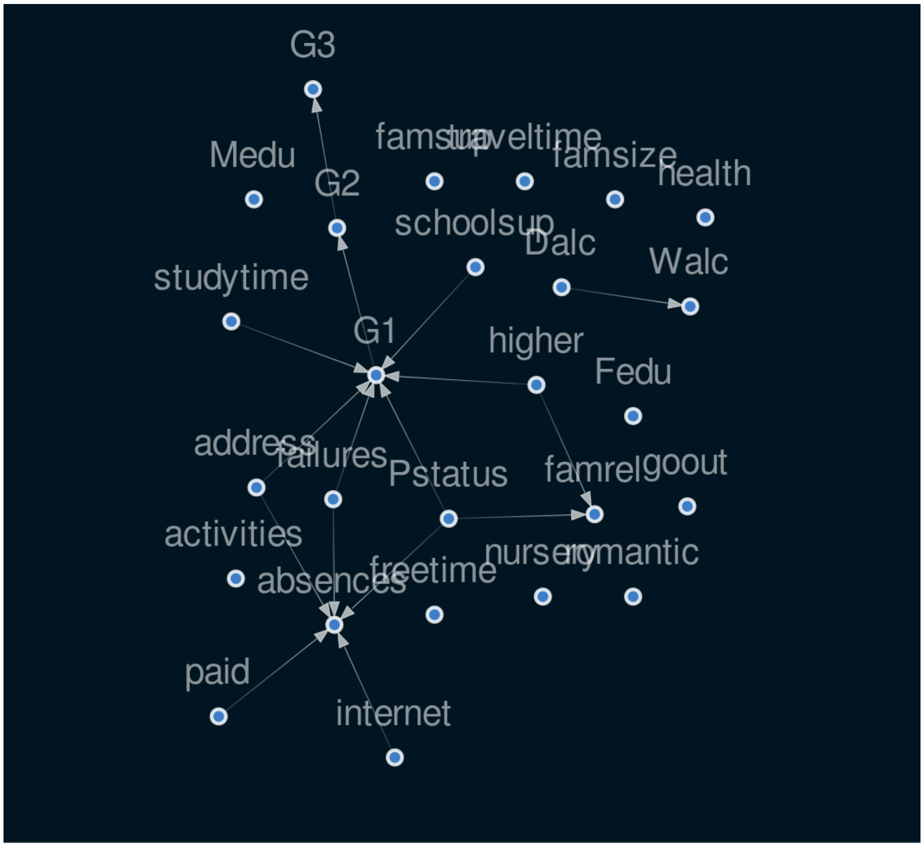

Figure 7: png

Can see there are two separate subgraphs in the above plot: `Dalc -> Walc` and the other big subgraph. We can retrieve the largest subgraph easily by calling `get_largest_subgraph()`:

```
newStructModel: StructureModel = structureModel.get_largest_subgraph()

# Now visualize:
viz = plot_structure(
```

```
    newStructModel,
    graph_attributes={"scale": "0.5"},
    all_node_attributes=NODE_STYLE.WEAK,
    all_edge_attributes=EDGE_STYLE.WEAK)
filename_finalStruct = curPath + "finalStruct.png"
viz.draw(filename_finalStruct)
Image(filename_finalStruct)
```

```
/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/pygraphvi
Warning: node 'absences', graph '%3' size too small for label
Warning: node 'G1', graph '%3' size too small for label
Warning: node 'Pstatus', graph '%3' size too small for label
Warning: node 'famrel', graph '%3' size too small for label
Warning: node 'studytime', graph '%3' size too small for label
Warning: node 'failures', graph '%3' size too small for label
Warning: node 'schoolsup', graph '%3' size too small for label
Warning: node 'paid', graph '%3' size too small for label
Warning: node 'higher', graph '%3' size too small for label
Warning: node 'internet', graph '%3' size too small for label
Warning: node 'G2', graph '%3' size too small for label
Warning: node 'G3', graph '%3' size too small for label

  warnings.warn(b"".join(errors).decode(self.encoding), RuntimeWarning)
/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/pygraphvi
Warning: node 'absences', graph '%3' size too small for label
Warning: node 'G1', graph '%3' size too small for label
Warning: node 'G2', graph '%3' size too small for label
Warning: node 'G3', graph '%3' size too small for label
Warning: node 'Pstatus', graph '%3' size too small for label
Warning: node 'famrel', graph '%3' size too small for label
Warning: node 'studytime', graph '%3' size too small for label
Warning: node 'failures', graph '%3' size too small for label
Warning: node 'schoolsup', graph '%3' size too small for label
Warning: node 'paid', graph '%3' size too small for label
Warning: node 'higher', graph '%3' size too small for label
Warning: node 'internet', graph '%3' size too small for label

  warnings.warn(b"".join(errors).decode(self.encoding), RuntimeWarning)
```

```
# Showing that within the same subgraph, we can query by two different nodes and get the sam
assert newStructModel.get_target_subgraph(node = 'G1').adj == newStructModel.get_target_subg

# NOTE key way how to find all unique subgraphs: going by nodes, for each node, if the curr
```

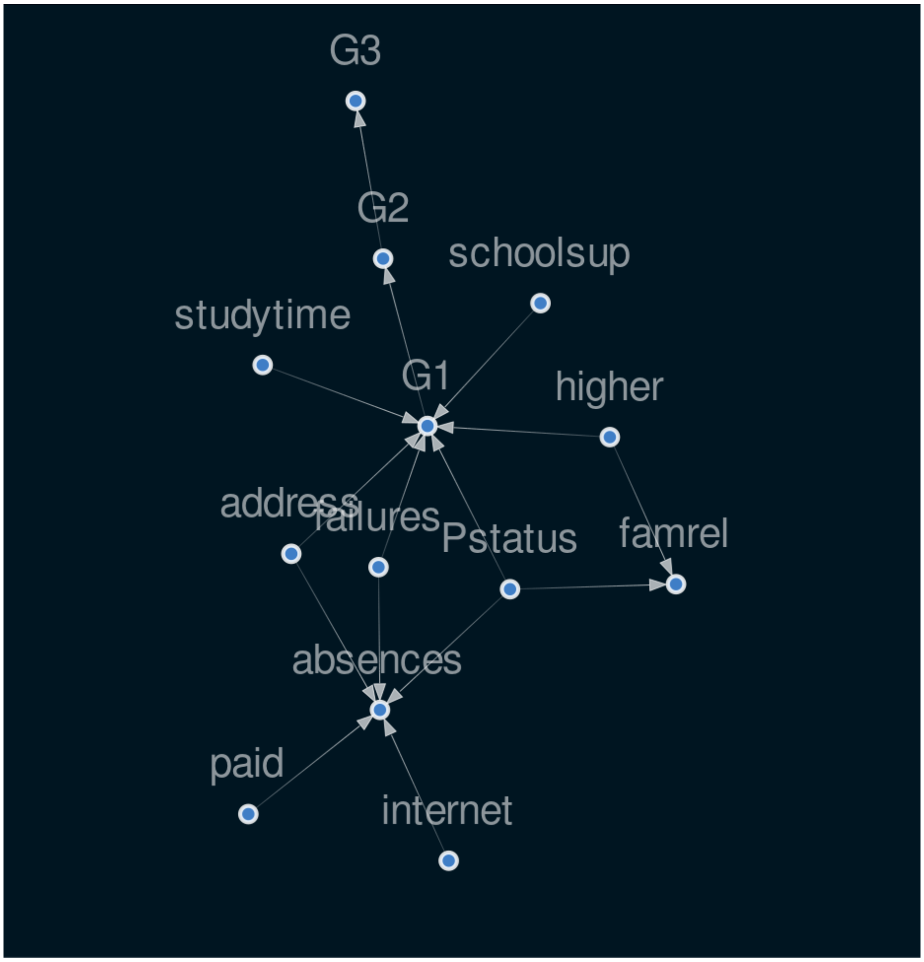
G3
G2
schoolsup
studytime
G1
higher
address
failures
Pstatus
famrel
absences
paid
internet

Figure 8: png

After deciding on how the final structure model should look, we can instantiate a BayesianNetwork:

```
from causalnex.network import BayesianNetwork

bayesNet: BayesianNetwork = BayesianNetwork(structure = newStructModel)
bayesNet.cpds
```

```
{}
```

```
bayesNet.edges
#bayesNet.node_states
```

```
[('address', 'absences'),
 ('address', 'G1'),
 ('G1', 'G2'),
 ('Pstatus', 'famrel'),
 ('Pstatus', 'absences'),
 ('Pstatus', 'G1'),
 ('studytime', 'G1'),
 ('failures', 'absences'),
 ('failures', 'G1'),
 ('schoolsup', 'G1'),
 ('paid', 'absences'),
 ('higher', 'famrel'),
 ('higher', 'G1'),
 ('internet', 'absences'),
 ('G2', 'G3')]
```

```
assert set(bayesNet.nodes) == set(list(iter(newStructModel.node)))
bayesNet.nodes
```

```
['address',
 'absences',
 'G1',
 'Pstatus',
 'famrel',
 'studytime',
 'failures',
 'schoolsup',
 'paid',
 'higher',
 'internet',
 'G2',
 'G3']
```

Can now learn the conditional probability distribution of different features in this `BayesianNetwork`

2/ Fitting the Conditional Distribution of the Bayesian Network

Preparing the Discretised Data

Any continuous features should be discretised prior to fitting the Bayesian Network, since CausalNex networks support only discrete distributions.

Should make numerical features categorical by discretisation then give the buckets meaningful labels. ## 1. Reducing Cardinality of Categorical Features To reduce cardinality of categorical features (reduce number of values they take on), can define a map `{oldValue: newValue}` and use this to update the feature we will discretise. Example: for the `studytime` feature, if the studytime is more than 2 then categorize it as `long-studytime` and the rest of the values are binned under `short_studytime`.

```
discrData: DataFrame = data.copy()

# Getting unique values per variable
dataVals = {var: data[var].unique() for var in data.columns}
dataVals
```

```
{'address': array(['U', 'R'], dtype=object),
 'famsize': array(['GT3', 'LE3'], dtype=object),
 'Pstatus': array(['A', 'T'], dtype=object),
 'Medu': array([4, 1, 3, 2, 0]),
 'Fedu': array([4, 1, 2, 3, 0]),
 'traveltime': array([2, 1, 3, 4]),
 'studytime': array([2, 3, 1, 4]),
 'failures': array([0, 3, 1, 2]),
 'schoolsup': array(['yes', 'no'], dtype=object),
 'famsup': array(['no', 'yes'], dtype=object),
 'paid': array(['no', 'yes'], dtype=object),
 'activities': array(['no', 'yes'], dtype=object),
 'nursery': array(['yes', 'no'], dtype=object),
 'higher': array(['yes', 'no'], dtype=object),
 'internet': array(['no', 'yes'], dtype=object),
 'romantic': array(['no', 'yes'], dtype=object),
 'famrel': array([4, 5, 3, 1, 2]),
 'freetime': array([3, 2, 4, 1, 5]),
```

```
 'goout': array([4, 3, 2, 1, 5]),
 'Dalc': array([1, 2, 5, 3, 4]),
 'Walc': array([1, 3, 2, 4, 5]),
 'health': array([3, 5, 1, 2, 4]),
 'absences': array([ 4,  2,  6,  0, 10,  8, 16, 14,  1, 12, 24, 22, 32, 30, 21, 15,  9,
        18, 26,  7, 11,  5, 13,  3]),
 'G1': array([ 0,  9, 12, 14, 11, 13, 10, 15, 17,  8, 16, 18,  7,  6,  5,  4, 19]),
 'G2': array([11, 13, 14, 12, 16, 17,  8, 10, 15,  9,  7,  6, 18, 19,  0,  5]),
 'G3': array([11, 12, 14, 13, 17, 15,  7, 10, 16,  9,  8, 18,  6,  0,  1,  5, 19])}
```

```
failuresMap = {v: 'no_failure' if v == [0] else 'yes_failure'
               for v in dataVals['failures']} # 0, 1, 2, 3 (number of failures)
failuresMap
```

```
{0: 'no_failure', 3: 'yes_failure', 1: 'yes_failure', 2: 'yes_failure'}
```

```
studytimeMap = {v: 'short_studytime' if v in [1,2] else 'long_studytime'
                for v in dataVals['studytime']}
studytimeMap
```

```
{2: 'short_studytime',
 3: 'long_studytime',
 1: 'short_studytime',
 4: 'long_studytime'}
```

Once we have defined the maps `{oldValue:  newValue}` we can update each feature, applying the map transformation. The `map` function applies the given dictionary as a rule to the called dictionary.

```
discrData['failures'] = discrData['failures'].map(failuresMap)
discrData['failures']
```

```
0       no_failure
1       no_failure
2       no_failure
3       no_failure
4       no_failure
          ...
644    yes_failure
645     no_failure
646     no_failure
647     no_failure
648     no_failure
Name: failures, Length: 649, dtype: object
```

```
discrData['studytime'] = discrData['studytime'].map(studytimeMap)
discrData['studytime']
```

```
0      short_studytime
1      short_studytime
2      short_studytime
3       long_studytime
4      short_studytime
            ...
644     long_studytime
645    short_studytime
646    short_studytime
647    short_studytime
648    short_studytime
Name: studytime, Length: 649, dtype: object
```

3. Discretising Numeric Features

To make numeric features categorical, they must first by discretised. The `causalnex.discretiser.Discretiser` helper class supports several discretisation methods. Here, the `fixed` method will be applied, providing static values that define the bucket boundaries. For instance, `absences` will be discretised into buckets `< 1`, `1 to 9`, and `>= 10`. Each bucket will be labelled as an integer, starting from zero.

```
from causalnex.discretiser import Discretiser

# Many values in absences, G1, G2, G3
dataVals
```

```
{'address': array(['U', 'R'], dtype=object),
 'famsize': array(['GT3', 'LE3'], dtype=object),
 'Pstatus': array(['A', 'T'], dtype=object),
 'Medu': array([4, 1, 3, 2, 0]),
 'Fedu': array([4, 1, 2, 3, 0]),
 'traveltime': array([2, 1, 3, 4]),
 'studytime': array([2, 3, 1, 4]),
 'failures': array([0, 3, 1, 2]),
 'schoolsup': array(['yes', 'no'], dtype=object),
 'famsup': array(['no', 'yes'], dtype=object),
 'paid': array(['no', 'yes'], dtype=object),
 'activities': array(['no', 'yes'], dtype=object),
 'nursery': array(['yes', 'no'], dtype=object),
 'higher': array(['yes', 'no'], dtype=object),
```

```
 'internet': array(['no', 'yes'], dtype=object),
 'romantic': array(['no', 'yes'], dtype=object),
 'famrel': array([4, 5, 3, 1, 2]),
 'freetime': array([3, 2, 4, 1, 5]),
 'goout': array([4, 3, 2, 1, 5]),
 'Dalc': array([1, 2, 5, 3, 4]),
 'Walc': array([1, 3, 2, 4, 5]),
 'health': array([3, 5, 1, 2, 4]),
 'absences': array([ 4,  2,  6,  0, 10,  8, 16, 14,  1, 12, 24, 22, 32, 30, 21, 15,  9,
        18, 26,  7, 11,  5, 13,  3]),
 'G1': array([ 0,  9, 12, 14, 11, 13, 10, 15, 17,  8, 16, 18,  7,  6,  5,  4, 19]),
 'G2': array([11, 13, 14, 12, 16, 17,  8, 10, 15,  9,  7,  6, 18, 19,  0,  5]),
 'G3': array([11, 12, 14, 13, 17, 15,  7, 10, 16,  9,  8, 18,  6,  0,  1,  5, 19])}
```

```
discrData['absences'] = Discretiser(method = 'fixed', numeric_split_points = [1,10]).transfo

assert (np.unique(discrData['absences']) == np.array([0,1,2])).all()


discrData['G1'] = Discretiser(method = 'fixed', numeric_split_points = [10]).transform(data
assert (np.unique(discrData['G1']) == np.array([0,1])).all()


discrData['G2'] = Discretiser(method = 'fixed', numeric_split_points = [10]).transform(data
assert (np.unique(discrData['G2']) == np.array([0,1])).all()

discrData['G3'] = Discretiser(method = 'fixed', numeric_split_points = [10]).transform(data
assert (np.unique(discrData['G3']) == np.array([0,1])).all()
```

4. Create Labels for Numeric Features

To make the discretised categories more readable, we can map the category labels onto something more meaningful in the same way we mapped category feature values.

```
absencesMap = {0: "No-absence", 1:"Low-absence", 2:"High-absence"}

G1Map = {0: "Fail", 1: "Pass"}
G2Map = {0: "Fail", 1: "Pass"}
G3Map = {0: "Fail", 1: "Pass"}

discrData['absences'] = discrData['absences'].map(absencesMap)
discrData['absences']
```

```
0      Low-absence
1      Low-absence
2      Low-absence
3       No-absence
4       No-absence
          ...
644    Low-absence
645    Low-absence
646    Low-absence
647    Low-absence
648    Low-absence
Name: absences, Length: 649, dtype: object
```

```
discrData['G1'] = discrData['G1'].map(G1Map)
discrData['G1']
```

```
0      Fail
1      Fail
2      Pass
3      Pass
4      Pass
       ...
644    Pass
645    Pass
646    Pass
647    Pass
648    Pass
Name: G1, Length: 649, dtype: object
```

```
discrData['G2'] = discrData['G2'].map(G2Map)
discrData['G2']
```

```
0      Pass
1      Pass
2      Pass
3      Pass
4      Pass
       ...
644    Pass
645    Pass
646    Pass
647    Pass
648    Pass
Name: G2, Length: 649, dtype: object
```

```python
discrData['G3'] = discrData['G3'].map(G3Map)
discrData['G3']
```

```
0      Pass
1      Pass
2      Pass
3      Pass
4      Pass
       ...
644    Pass
645    Pass
646    Fail
647    Pass
648    Pass
Name: G3, Length: 649, dtype: object
```

```python
# Now for reference later get the discrete data values also:
discrDataVals = {var: discrData[var].unique() for var in discrData.columns}
discrDataVals
```

```
{'address': array(['U', 'R'], dtype=object),
 'famsize': array(['GT3', 'LE3'], dtype=object),
 'Pstatus': array(['A', 'T'], dtype=object),
 'Medu': array([4, 1, 3, 2, 0]),
 'Fedu': array([4, 1, 2, 3, 0]),
 'traveltime': array([2, 1, 3, 4]),
 'studytime': array(['short_studytime', 'long_studytime'], dtype=object),
 'failures': array(['no_failure', 'yes_failure'], dtype=object),
 'schoolsup': array(['yes', 'no'], dtype=object),
 'famsup': array(['no', 'yes'], dtype=object),
 'paid': array(['no', 'yes'], dtype=object),
 'activities': array(['no', 'yes'], dtype=object),
 'nursery': array(['yes', 'no'], dtype=object),
 'higher': array(['yes', 'no'], dtype=object),
 'internet': array(['no', 'yes'], dtype=object),
 'romantic': array(['no', 'yes'], dtype=object),
 'famrel': array([4, 5, 3, 1, 2]),
 'freetime': array([3, 2, 4, 1, 5]),
 'goout': array([4, 3, 2, 1, 5]),
 'Dalc': array([1, 2, 5, 3, 4]),
 'Walc': array([1, 3, 2, 4, 5]),
 'health': array([3, 5, 1, 2, 4]),
 'absences': array(['Low-absence', 'No-absence', 'High-absence'], dtype=object),
```

```
 'G1': array(['Fail', 'Pass'], dtype=object),
 'G2': array(['Pass', 'Fail'], dtype=object),
 'G3': array(['Pass', 'Fail'], dtype=object)}
```

5. Train / Test Split

Must train and test split data to help validate findings. Split 90% train and 10% test.

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(discrData,
                               train_size = 0.9, test_size = 0.10,
                               random_state = 7)
```

3/ Model Probability

With the learnt structure model and discretised data, we can now fit the probability distribution of the Bayesian Network.

First Step: The first step is to specify all the states that each node can take. Can be done from data or can provide dictionary of node values. Here, we use the full dataset to avoid cases where states in our test set do not exist in the training set. In the real world, those states would need to be provided using the dictionary method.

```
import copy


# First 'copying' the object so previous state is preserved:
# SOURCE: https://www.geeksforgeeks.org/copy-python-deep-copy-shallow-copy/
bayesNetNodeStates = copy.deepcopy(bayesNet)
assert not bayesNetNodeStates == bayesNet, "Deepcopy bayesnet object must work"
# bayesNetNodeStates = BayesianNetwork(bayesNet.structure)

bayesNetNodeStates: BayesianNetwork = bayesNetNodeStates.fit_node_states(df = discrData)
bayesNetNodeStates.node_states
```

```
{'address': {'R', 'U'},
 'famsize': {'GT3', 'LE3'},
 'Pstatus': {'A', 'T'},
 'Medu': {0, 1, 2, 3, 4},
 'Fedu': {0, 1, 2, 3, 4},
```

```
 'traveltime': {1, 2, 3, 4},
 'studytime': {'long_studytime', 'short_studytime'},
 'failures': {'no_failure', 'yes_failure'},
 'schoolsup': {'no', 'yes'},
 'famsup': {'no', 'yes'},
 'paid': {'no', 'yes'},
 'activities': {'no', 'yes'},
 'nursery': {'no', 'yes'},
 'higher': {'no', 'yes'},
 'internet': {'no', 'yes'},
 'romantic': {'no', 'yes'},
 'famrel': {1, 2, 3, 4, 5},
 'freetime': {1, 2, 3, 4, 5},
 'goout': {1, 2, 3, 4, 5},
 'Dalc': {1, 2, 3, 4, 5},
 'Walc': {1, 2, 3, 4, 5},
 'health': {1, 2, 3, 4, 5},
 'absences': {'High-absence', 'Low-absence', 'No-absence'},
 'G1': {'Fail', 'Pass'},
 'G2': {'Fail', 'Pass'},
 'G3': {'Fail', 'Pass'}}
```

Fit Conditional Probability Distributions

The `fit_cpds` method of `BayesianNetwork` accepts a dataset to learn the conditional probability distributions (CPDs) of **each node** along with a method of how to do this fit.

```
# Copying the object information
bayesNetCPD: BayesianNetwork = copy.deepcopy(bayesNetNodeStates)

# Fitting the CPDs
bayesNetCPD: BayesianNetwork = bayesNetCPD.fit_cpds(data = train,
                                                    method = "BayesianEstimator",
                                                    bayes_prior = "K2")
```

```
/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/pandas/co
  object.__getattribute__(self, name)
/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/pandas/co
  return object.__setattr__(self, name, value)
/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/pgmpy/est
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing
```

```
See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ix-indexer-is-deprecate
  states = sorted(list(self.data.ix[:, variable].dropna().unique()))
/development/bin/python/conda3_ana/envs/pybayesian_env/lib/python3.7/site-packages/pgmpy/est
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ix-indexer-is-deprecate
  state_count_data = data.ix[:, variable].value_counts()
```

```
bayesNetCPD.cpds
```

```
{'address':
 address
 R        0.302048
 U        0.697952,
 'absences': Pstatus              A                                              \
 address             R
 failures     no_failure                    yes_failure
 internet             no        yes                 no              yes
 paid                 no   yes   no       yes          no       yes        no
 absences
 High-absence        0.2  0.25  0.2  0.333333         0.2  0.333333  0.333333
 Low-absence         0.4  0.50  0.4  0.333333         0.4  0.333333  0.333333
 No-absence          0.4  0.25  0.4  0.333333         0.4  0.333333  0.333333

 Pstatus                                     ...          T               \
 address                       U             ...          R               U
 failures             no_failure             ... yes_failure     no_failure
 internet                     no             ...         yes             no
 paid              yes          no       yes ...          no  yes        no
 absences                                    ...
 High-absence  0.333333   0.200000  0.333333 ...    0.148148  0.2   0.061224
 Low-absence   0.333333   0.666667  0.333333 ...    0.518519  0.6   0.612245
 No-absence    0.333333   0.133333  0.333333 ...    0.333333  0.2   0.326531

 Pstatus
 address
 failures                           yes_failure
 internet               yes                  no             yes
 paid          yes        no       yes          no   yes        no       yes
 absences
```

```
High-absence  0.25  0.109312  0.071429    0.142857  0.25  0.323529  0.222222
Low-absence   0.25  0.473684  0.714286    0.428571  0.25  0.470588  0.555556
No-absence    0.50  0.417004  0.214286    0.428571  0.50  0.205882  0.222222

[3 rows x 32 columns],
'G1': Pstatus                A                                                \
address                R
failures      no_failure
higher                no
schoolsup             no                             yes
studytime long_studytime short_studytime long_studytime short_studytime
G1
Fail            0.666667        0.333333            0.5             0.5
Pass            0.333333        0.666667            0.5             0.5

Pstatus                                                             \
address
failures
higher               yes
schoolsup             no                             yes
studytime long_studytime short_studytime long_studytime short_studytime
G1
Fail            0.333333        0.222222            0.5             0.5
Pass            0.666667        0.777778            0.5             0.5

Pstatus                                   ...             T                  \
address                                   ...             U
failures     yes_failure                  ...    no_failure
higher                no                  ...           yes
schoolsup             no                  ...           yes
studytime long_studytime short_studytime  ... long_studytime short_studytime
G1                                        ...
Fail            0.666667        0.666667  ...       0.222222        0.285714
Pass            0.333333        0.333333  ...       0.777778        0.714286

Pstatus                                                             \
address
failures     yes_failure
higher                no
schoolsup             no                             yes
studytime long_studytime short_studytime long_studytime short_studytime
G1
Fail            0.666667        0.789474            0.5        0.666667
Pass            0.333333        0.210526            0.5        0.333333

Pstatus
```

```
address
failures
higher              yes
schoolsup            no                            yes
studytime long_studytime short_studytime long_studytime short_studytime
G1
Fail           0.571429        0.652174            0.5        0.666667
Pass           0.428571        0.347826            0.5        0.333333

[2 rows x 64 columns],
'Pstatus':
Pstatus
A        0.119454
T        0.880546,
'famrel': Pstatus         A                   T
higher         no       yes        no       yes
famrel
1        0.142857  0.061538  0.064516  0.023758
2        0.142857  0.092308  0.048387  0.045356
3        0.285714  0.092308  0.161290  0.159827
4        0.357143  0.461538  0.419355  0.503240
5        0.071429  0.292308  0.306452  0.267819,
'studytime':
studytime
long_studytime   0.204778
short_studytime  0.795222,
'failures':
failures
no_failure   0.837884
yes_failure  0.162116,
'schoolsup':
schoolsup
no         0.887372
yes        0.112628,
'paid':
paid
no    0.938567
yes   0.061433,
'higher':
higher
no      0.114334
yes     0.885666,
'internet':
internet
no        0.230375
yes       0.769625,
```

```
'G2': G1   Fail Pass
G2
Fail  0.5  0.5
Pass  0.5  0.5,
'G3': G2   Fail Pass
G3
Fail  0.5  0.5
Pass  0.5  0.5}
```

```
# The size of the tables depends on how many connections a node has
Image(filename_finalStruct)
```

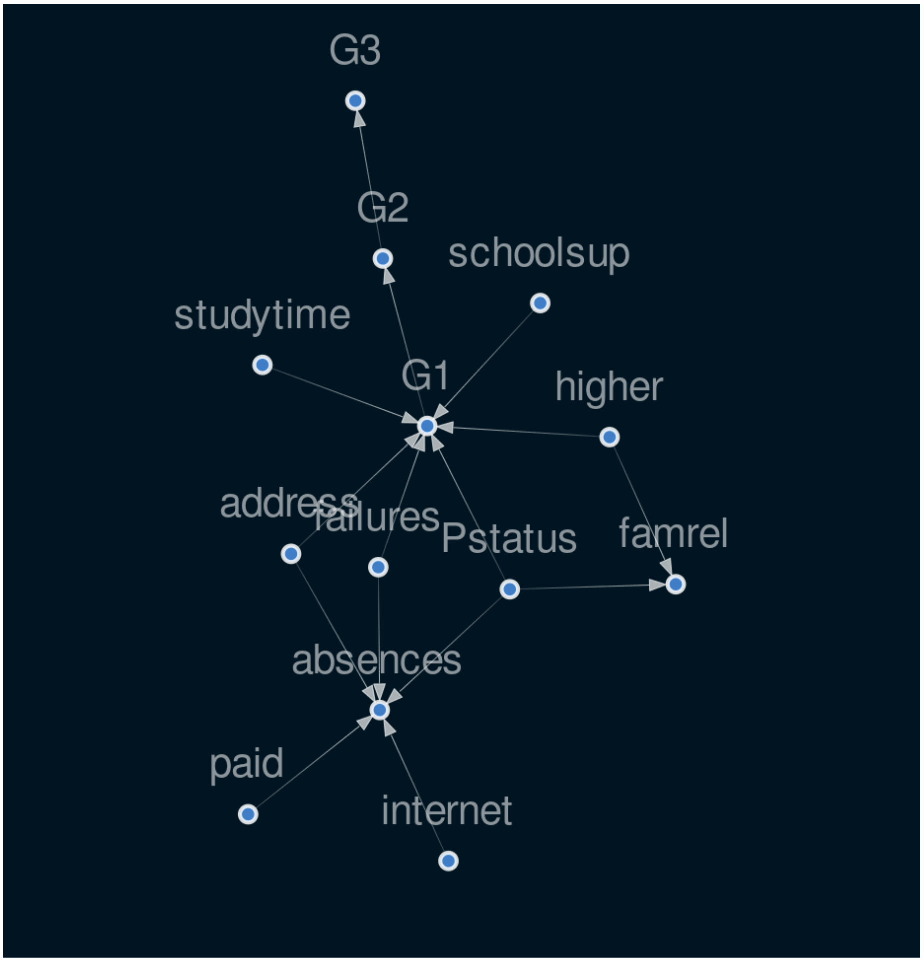

Figure 9: png

```python
# G1 has many connections so its table holds all the combinations of conditional probabilit
bayesNetCPD.cpds['G1']
```

Pstatus

A

…

T

address

R

…

U

failures

no_failure

yes_failure

…

no_failure

yes_failure

higher

no

yes

no

…

yes

no

yes

schoolsup

no

yes

no

yes

no

…

yes

no

yes

no

yes

studytime

long_studytime

short_studytime

long_studytime

short_studytime

long_studytime

short_studytime

long_studytime

short_studytime

long_studytime

short_studytime

. . .

long_studytime

short_studytime

long_studytime

short_studytime

long_studytime

short_studytime

long_studytime

short_studytime

long_studytime

short_studytime

G1

Fail

0.666667

0.333333

0.5

0.5

0.333333

0.222222

0.5

0.5

0.666667

0.666667

...

0.222222

0.285714

0.666667

0.789474

0.5

0.666667

0.571429

0.652174

0.5

0.666667

Pass

0.333333

0.666667

0.5

0.5

0.666667

0.777778

0.5

0.5

0.333333

0.333333

...

0.777778

0.714286

0.333333

0.210526

0.5

0.333333

0.428571

0.347826

0.5

0.333333

2 rows × 64 columns

```
bayesNetCPD.cpds['absences']
```

Pstatus

A

...

T

address

R

U

...

R

U

failures

no_failure

yes_failure

no_failure

...

yes_failure

no_failure

yes_failure

internet

no

yes

no

yes

no

...

yes

no

yes

no

yes

paid

no

yes

no

yes

no

yes

no

yes

no

yes

...

no

yes

no

yes

no

yes

no

yes

no

yes

absences

High-absence

0.2

0.25

0.2

0.333333

0.2

0.333333

0.333333

0.333333

0.200000

0.333333

. . .

0.148148

0.2

0.061224

0.25

0.109312

0.071429

0.142857

0.25

0.323529

0.222222

Low-absence

0.4

0.50

0.4

0.333333

0.4

0.333333

0.333333

0.333333

0.666667

0.333333

...

0.518519

0.6

0.612245

0.25

0.473684

0.714286

0.428571

0.25

0.470588

0.555556

No-absence

0.4

0.25

0.4

0.333333

0.4

0.333333

0.333333

0.333333

0.133333

0.333333

...

0.333333

0.2

0.326531

0.50

0.417004

0.214286

0.428571

0.50

0.205882

0.222222

3 rows × 32 columns

```
# Studytime variable is a singular ndoe so its table is small, no conditional probabilities
bayesNetCPD.cpds['studytime']
```

studytime

long_studytime

0.204778

short_studytime

0.795222

```
# Pstatus has only outgoing nodes, no incoming nodes so has no conditional probabilities.
bayesNetCPD.cpds['Pstatus']
```

Pstatus

A

0.119454

T

0.880546

```
# Famrel has two incoming nodes (PStatus and higher) so models their conditional probabilit
bayesNetCPD.cpds['famrel']
```

Pstatus

A

T

higher

no

yes

no

yes

famrel

1

0.142857

0.061538

0.064516

0.023758

2

0.142857

0.092308

0.048387

0.045356

3

0.285714

0.092308

0.161290

0.159827

4

0.357143

0.461538

0.419355

0.503240

5

0.071429

0.292308

0.306452

0.267819

```
bayesNetCPD.cpds['G2']
```

G1

Fail

Pass

G2

Fail

0.5

0.5

Pass

0.5

0.5

```
bayesNetCPD.cpds['G3']
```

G2

Fail

Pass

G3

Fail

0.5

0.5

Pass

0.5

0.5

The CPD dictionaries are multiindexed so the `loc` functino can be a useful way to interact with them:

```
# TODO: https://hyp.is/_95epIOuEeq_HdeYjzCPXQ/causalnex.readthedocs.io/en/latest/03_tutoria
discrData.loc[1:5,['address', 'G1', 'paid', 'higher']]
```

address

G1

paid

higher

1

U

Fail

no

yes

2

U

Pass

no

yes

3

U

Pass

no

yes

4

U

Pass

no

yes

5

U

Pass

no

yes

Predict the State given the Input Data

The `predict` method of `BayesianNetwork` allos us to make predictions based on the data using the learnt network. For example we want to predict if a student passes of failes the exam based on the input data. Consider an incoming student data like this:

```
# Row number 18
discrData.loc[18, discrData.columns != 'G1']
```

```
address                    U
famsize                  GT3
Pstatus                    T
Medu                       3
Fedu                       2
traveltime                 1
studytime     short_studytime
failures         yes_failure
schoolsup                 no
famsup                   yes
paid                     yes
activities               yes
nursery                  yes
higher                   yes
internet                 yes
romantic                  no
famrel                     5
freetime                   5
goout                      5
Dalc                       2
Walc                       4
health                     5
absences         Low-absence
G2                      Fail
G3                      Fail
Name: 18, dtype: object
```

Based on this data, want to predict if this particular student (in row 18) will succeed on their exam. Intuitively expect this student not to succeed because they spend shorter amount of study time and have failed in the past.

There are two kinds of prediction methods: * `predict_probability(data, node)`: Predict the **probability of each possible state of a node**, based on some input data. * `predict(data, node)`: Predict the **state of a node** based on some input data, using the Bayesian Network.

```
predictionProbs = bayesNetCPD.predict_probability(data = discrData, node = 'G1')
predictionProbs
```

G1_Pass

G1_Fail

0

0.777778

0.222222

1

0.882051

0.117949

2

0.714286

0.285714

3

0.968254

0.031746

4

0.882051

0.117949

...

...

...

644

0.600000

0.400000

645

0.882051

0.117949

646

0.882051

0.117949

647

0.882051

0.117949

648

0.750000

0.250000

649 rows × 2 columns

```
# Student 18 passes with probability 0.358, and fails with prob 0.64
predictionProbs.loc[18, :]
```

```
G1_Pass    0.347826
G1_Fail    0.652174
Name: 18, dtype: float64
```

```
# This function does predictions for ALL observations (all students)
predictions = bayesNetCPD.predict(data = discrData, node = 'G1')
predictions
```

G1_prediction

0

Pass

1

Pass

2

Pass

3

Pass

4

Pass

…

…

644

Pass

645

Pass

646

Pass

647

Pass

648

Pass

649 rows × 1 columns

```
predictions.loc[18, :]
```

```
G1_prediction    Fail
Name: 18, dtype: object
```

Compare this prediction to the ground truth:

```
print(f"Student 18 is predicted to {predictions.loc[18, 'G1_prediction']}")
print(f"Ground truth for student 18 is {discrData.loc[18, 'G1']}")
```

```
Student 18 is predicted to Fail
Ground truth for student 18 is Fail
```

4/ Model Quality

To evaluate the quality of the model that has been learned, CausalNex supports two main approaches: Classification Report and Reciever Operating Characteristics (ROC) / Area Under the ROC Curve (AUC). ## Measure 1: Classification Report To obtain a classification report using a BN, we need to provide a test set and the node we are trying to classify. The classification report predicts the target node for all rows (observations) in the test set and evaluate how well those predictions are made, via the model.

```
from causalnex.evaluation import classification_report
```

```
classification_report(bn = bayesNetCPD, data = test, node = 'G1')
```

precision

recall

f1-score

support

G1_Fail

0.777778

0.583333

0.666667

12

G1_Pass

0.910714

0.962264

0.935780

53

micro avg

0.892308

0.892308

0.892308

65

macro avg

0.844246

0.772799

0.801223

65

weighted avg

0.886172

0.892308

0.886097

65

Interpret Results of classification report: this report shows that the model can classify reasonably well whether a student passs the exam. For predictions where the student fails, the precision is adequate but recall is bad. This implies that we can rely on predictions for `G1_Fail` but we are likely to miss some of the predictions we should have made. Perhaps these missing predictions are a result of something missing in our structure * ALERT - explore graph structure when the recall is bad

ROC / AUC

The ROC and AUC can be obtained with `roc_auc` method within CausalNex metrics module. ROC curve is computed by micro-averaging predictions made across all states (classes) of the target node.

```
from causalnex.evaluation import roc_auc

roc, auc = roc_auc(bn = bayesNetCPD, data = test, node = 'G1')

print(f"ROC = \n{roc}\n")
print(f"AUC = {auc}")
```

```
ROC =
[(0.0, 0.0), (0.0, 0.1076923076923077), (0.0, 0.16923076923076924), (0.046153846153846156, (

AUC = 0.9123076923076924
```

High value of AUC gives confidence in model performance

5/ Querying Marginals

After iterating over our model structure, CPDs, and validating our model quality, we can **query our model under different observations** to gain insights.

Baseline Marginals

To query the model for baseline marginals that reflect the population as a whole, a `query` method can be used.

First: update the model using the complete dataset since the one we currently have is built only from training data.

```
# Copy object:
bayesNetFull = copy.deepcopy(bayesNetCPD)

# Fitting CPDs with full data
bayesNetFull: BayesianNetwork = bayesNetFull.fit_cpds(data = discrData,
                                                      method = "BayesianEstimator",
                                                      bayes_prior = "K2")
```

```
WARNING:root:Replacing existing CPD for address
WARNING:root:Replacing existing CPD for absences
WARNING:root:Replacing existing CPD for G1
WARNING:root:Replacing existing CPD for Pstatus
WARNING:root:Replacing existing CPD for famrel
WARNING:root:Replacing existing CPD for studytime
WARNING:root:Replacing existing CPD for failures
WARNING:root:Replacing existing CPD for schoolsup
WARNING:root:Replacing existing CPD for paid
WARNING:root:Replacing existing CPD for higher
WARNING:root:Replacing existing CPD for internet
WARNING:root:Replacing existing CPD for G2
WARNING:root:Replacing existing CPD for G3
```

Get warnings, showing we are replacing the previously existing CPDs

Second: For inference, must create a new `InferenceEngine` from our `BayesianNetwork`, which lets us query the model. The query method will compute the marginal likelihood of all states for all nodes. Query lets us get the marginal distributions, marginalizing to get rid of the conditioning variable(s) for each node variable.

```
from causalnex.inference import InferenceEngine


eng = InferenceEngine(bn = bayesNetFull)
eng
```

```
<causalnex.inference.inference.InferenceEngine at 0x7f6cca0b69d0>
```

Query the baseline marginal distributions, which means querying marginals **as learned from data**:

```
marginalDistLearned: Dict[str, Dict[str, float]] = eng.query()
marginalDistLearned
```

```
{'address': {'R': 0.3041474654377881, 'U': 0.6958525345622117},
 'absences': {'High-absence': 0.1278149471852898,
  'Low-absence': 0.5034849294152204,
  'No-absence': 0.36870012339948993},
 'G1': {'Fail': 0.2614871976647877, 'Pass': 0.7385128023352121},
 'Pstatus': {'A': 0.12442396313364057, 'T': 0.8755760368663592},
 'famrel': {1: 0.03724247501855778,
  2: 0.04846203869543736,
  3: 0.15602529390568748,
```

```
  4: 0.4814761637760789,
  5: 0.2767940286042384},
 'studytime': {'long_studytime': 0.20430107526881724,
  'short_studytime': 0.7956989247311828},
 'failures': {'no_failure': 0.8448540706605223,
  'yes_failure': 0.1551459293394777},
 'schoolsup': {'no': 0.8940092165898619, 'yes': 0.10599078341013828},
 'paid': {'no': 0.9385560675883257, 'yes': 0.06144393241167435},
 'higher': {'no': 0.10752688172043012, 'yes': 0.8924731182795699},
 'internet': {'no': 0.2334869431643625, 'yes': 0.7665130568356374},
 'G2': {'Fail': 0.4999999999999999, 'Pass': 0.4999999999999999},
 'G3': {'Fail': 0.4999999999999999, 'Pass': 0.4999999999999999}}
```

```
marginalDistLearned['address']
```

```
{'R': 0.3041474654377881, 'U': 0.6958525345622117}
```

```
marginalDistLearned['G1']
```

```
{'Fail': 0.2614871976647877, 'Pass': 0.7385128023352121}
```

Output tells us that `P(G1=Fail) ~ 0.25` and `P(G1 = Pass) ~ 0.75`. As a quick sanity check can compute what proportion of our data are `Fail` and `Pass`, should give nearly the same result:

```
import numpy as np

labels, counts = np.unique(discrData['G1'], return_counts = True)

print(list(zip(labels, counts)))
print('\nProportion failures = {}'.format(counts[0] / sum(counts)))
print('\nProportion passes = {}'.format(counts[1] / sum(counts)))
```

```
[('Fail', 157), ('Pass', 492)]

Proportion failures = 0.24191063174114022

Proportion passes = 0.7580893682588598
```

Marginals After Observations

Can query the marginal likelihood of states in our network, **given observations**.

TODO is this using the Bayesian update rule?

These observations can be made anywhere in the network and their impact will be propagated through to the node of interest.

```
# Reminding of the data types for each variable:
discrDataVals
```

```
{'address': array(['U', 'R'], dtype=object),
 'famsize': array(['GT3', 'LE3'], dtype=object),
 'Pstatus': array(['A', 'T'], dtype=object),
 'Medu': array([4, 1, 3, 2, 0]),
 'Fedu': array([4, 1, 2, 3, 0]),
 'traveltime': array([2, 1, 3, 4]),
 'studytime': array(['short_studytime', 'long_studytime'], dtype=object),
 'failures': array(['no_failure', 'yes_failure'], dtype=object),
 'schoolsup': array(['yes', 'no'], dtype=object),
 'famsup': array(['no', 'yes'], dtype=object),
 'paid': array(['no', 'yes'], dtype=object),
 'activities': array(['no', 'yes'], dtype=object),
 'nursery': array(['yes', 'no'], dtype=object),
 'higher': array(['yes', 'no'], dtype=object),
 'internet': array(['no', 'yes'], dtype=object),
 'romantic': array(['no', 'yes'], dtype=object),
 'famrel': array([4, 5, 3, 1, 2]),
 'freetime': array([3, 2, 4, 1, 5]),
 'goout': array([4, 3, 2, 1, 5]),
 'Dalc': array([1, 2, 5, 3, 4]),
 'Walc': array([1, 3, 2, 4, 5]),
 'health': array([3, 5, 1, 2, 4]),
 'absences': array(['Low-absence', 'No-absence', 'High-absence'], dtype=object),
 'G1': array(['Fail', 'Pass'], dtype=object),
 'G2': array(['Pass', 'Fail'], dtype=object),
 'G3': array(['Pass', 'Fail'], dtype=object)}
```

```
# Reminder of nodes you CAN query (for instance putting 'health' in the dictionary argument
bayesNetFull.nodes
```

```
['address',
 'absences',
 'G1',
 'Pstatus',
```

```
 'famrel',
 'studytime',
 'failures',
 'schoolsup',
 'paid',
 'higher',
 'internet',
 'G2',
 'G3']
```

```
marginalDistObs_biasPass: Dict[str, Dict[str, float]] = eng.query({'studytime': 'long_studyt

# Seeing if biasing in favor of failing will influence the observed marginals:
marginalDistObs_biasFail: Dict[str, Dict[str, float]] = eng.query({'studytime': 'short_study

# Higher probability of passing when have the above observations, since they are another se
marginalDistLearned['G1']
```

```
{'Fail': 0.2614871976647877, 'Pass': 0.7385128023352121}
```

```
marginalDistObs_biasPass['G1']
```

```
{'Fail': 0.07373430443712227, 'Pass': 0.9262656955628777}
```

```
marginalDistObs_biasFail['G1']
```

```
{'Fail': 0.7243863093775379, 'Pass': 0.27561369062246216}
```

```
marginalDistLearned['G2']
```

```
{'Fail': 0.4999999999999999, 'Pass': 0.4999999999999999}
```

```
# G2 and G3 nodes don't show bias probability because they are not many conditionals on the
marginalDistObs_biasPass['G2']
```

```
{'Fail': 0.5, 'Pass': 0.5}
```

```
marginalDistObs_biasFail['G2']
```

```
{'Fail': 0.5, 'Pass': 0.5}
```

```
marginalDistLearned['G3']
```

```
{'Fail': 0.4999999999999999, 'Pass': 0.4999999999999999}
```

```
marginalDistObs_biasPass['G3']
```

```
{'Fail': 0.5, 'Pass': 0.5}
```

```
marginalDistObs_biasFail['G3']
```

```
{'Fail': 0.5, 'Pass': 0.5}
```

Looking at difference in likelihood of `G1` based on just `studytime`. See that students who study longer are more likely to pass on their exam:

```
marginalDist_short = eng.query({'studytime':'short_studytime'})
marginalDist_long = eng.query({'studytime': 'long_studytime'})

print('Marginal G1 | Short Studytime', marginalDist_short['G1'])
print('Marginal G1 | Long Studytime', marginalDist_long['G1'])
```

```
Marginal G1 | Short Studytime {'Fail': 0.2817997392562336, 'Pass': 0.7182002607437664}
Marginal G1 | Long Studytime {'Fail': 0.18237519357178764, 'Pass': 0.8176248064282124}
```

Interventions with Do Calculus

Do-Calculus, allows us to specify interventions.

Updating a Node Distribution

Can apply an intervention to any node in our data, updating its distribution using a `do` operator, which means asking our mdoel "what if" something were different.

For example, can ask what would happen if 100% of students wanted to go on to do higher education.

```
print("'higher' marginal distribution before DO: ", eng.query()['higher'])

# Make the intervention on the network
eng.do_intervention(node = 'higher', state = {'yes': 1.0, 'no': 0.0}) # all students yes

print("'higher' marginal distribution after DO: ", eng.query()['higher'])
```

```
'higher' marginal distribution before DO:  {'no': 0.10752688172043012, 'yes': 0.89247311827
'higher' marginal distribution after DO:  {'no': 0.0, 'yes': 1.0000000000000002}
```

Resetting a Node Distribution

We can reset any interventions that we make using `reset_intervention` method and providing the node we want to reset:

```
eng.reset_do('higher')
```

```
eng.query()['higher'] # same as before
```

```
{'no': 0.10752688172043012, 'yes': 0.8924731182795699}
```

Effect of DO on Marginals

We can use `query` to find the effect that an intervention has on our marginal likelihoods of OTHER variables, not just on the INTERVENED variable.

Example 1: change 'higher' and check grade 'G1' (how the likelihood of achieving a pass changes if 100% of students wanted to do higher education)

Answer: if 100% of students wanted to do higher education (as opposed to 90% in our data population) , then we estimate the pass rate would increase from 74.7% to 79.3%.

```
print('marginal G1', eng.query()['G1'])

eng.do_intervention(node = 'higher', state = {'yes':1.0, 'no': 0.0})
print('updated marginal G1', eng.query()['G1'])
```

```
marginal G1 {'Fail': 0.2614871976647877, 'Pass': 0.7385128023352121}
updated marginal G1 {'Fail': 0.22096538189680157, 'Pass': 0.7790346181031987}
```

```
# This is how we know it is 90% of the population that does higher education:
eng.reset_do('higher')
```

```
eng.query()['higher']
```

```
{'no': 0.10752688172043012, 'yes': 0.8924731182795699}
```

```
# OR:
labels, counts = np.unique(discrData['higher'], return_counts = True)
counts / sum(counts)
```

```
array([0.10631741, 0.89368259])
```

Example 2: change 'higher' and check grade 'G1' (how the likelihood of achieving a pass changes if 80% of students wanted to do higher education)

```
eng.reset_do('higher')

print('marginal G1', eng.query()['G1'])

eng.do_intervention(node = 'higher', state = {'yes':0.8, 'no': 0.2})
print('updated marginal G1', eng.query()['G1']) # fail is actually higher!!!!
```

```
marginal G1 {'Fail': 0.2614871976647877, 'Pass': 0.7385128023352121}
updated marginal G1 {'Fail': 0.2963359592252558, 'Pass': 0.7036640407747445}
```