simulate_categorical_network

June 12, 2025

1 Simulate Categorical Network

[2]:
```
# env: smile

import pandas as pd
import pysmile
import pysmile_license
import numpy as np
import os
```

[3]:
```
!pip show pysmile
```

```
Name: pysmile
Version: 2.4.0
Summary: BayesFusion's SMILE wrapper
Home-page: https://bayesfusion.com/
Author: BayesFusion LLC
Author-email: support@bayesfusion.com
License: PySMILE ACADEMIC is the Python wrapper for SMILE
SMILE: Structural Modeling, Inference and Learning Engine

This software can be used only with a valid license, obtained from BayesFusion,
LLC.
The following document contains full terms and conditions of SMILE Academic
license:
https://download.bayesfusion.com/license_academic.txt

SMILE Academic can be used without cost for academic teaching and research use.
We would like to stress that "academic teaching and research use" means using
the
software (1) for the purpose of academic teaching or research as part of an
academic program or an academic research project, and (2) by a user who is
at the time of use affiliated with an academic institution. In other words,
affiliation with an academic institution alone, research conducted at government
or industrial research centers, research conducted by members of university
faculty in consulting projects, or use in a commercial educational institution
DO NOT qualify as academic teaching and research use.
```

```
The documentation is available at:
http://support.bayesfusion.com/

Location: /home/jacob/anaconda3/envs/smile/lib/python3.11/site-packages
Requires:
Required-by:
```

1.1 Create simulated data from a network with discrete variables

To simulate categorical variables A, B, C, D with the following Bayesian network dependencies:

- $C \to A$
- $A, B \to D$

We'll:

1. Sample C independently.
2. Sample A conditioned on C.
3. Sample B independently.
4. Sample D conditioned on A and B.

[4]:
```
output_folder = "./simulated_data"
os.makedirs(output_folder, exist_ok=True)

simulated_dataset_path = "./simulated_data/simulated_cate_dataset.csv"
```

[5]:
```
import numpy as np
import pandas as pd

# Define categories
categories_C = ['c1', 'c2', 'c3']
categories_A = ['a1', 'a2', 'a3']
categories_B = ['b1', 'b2']
categories_D = ['d1', 'd2']

n_samples = 50000

# 1. Sample C independently
C = np.random.choice(categories_C, size=n_samples, p=[0.3, 0.4, 0.3])

# 2. Sample A conditioned on C
# Conditional probability table: P(A | C)
P_A_given_C = {
    'c1': [0.7, 0.2, 0.1],
    'c2': [0.2, 0.5, 0.3],
    'c3': [0.1, 0.3, 0.6],
}
A = np.array([
    np.random.choice(categories_A, p=P_A_given_C[c_val])
```

```
    for c_val in C
])

# 3. Sample B independently
B = np.random.choice(categories_B, size=n_samples, p=[0.6, 0.4])

# 4. Sample D conditioned on A and B
# Conditional probability table: P(D | A, B)
P_D_given_AB = {
    ('a1', 'b1'): [0.9, 0.1],
    ('a1', 'b2'): [0.6, 0.4],
    ('a2', 'b1'): [0.5, 0.5],
    ('a2', 'b2'): [0.4, 0.6],
    ('a3', 'b1'): [0.3, 0.7],
    ('a3', 'b2'): [0.2, 0.8],
}
D = np.array([
    np.random.choice(categories_D, p=P_D_given_AB[(a_val, b_val)])
    for a_val, b_val in zip(A, B)
])

# Create DataFrame
df = pd.DataFrame({'C': C, 'A': A, 'B': B, 'D': D})
print(df.head())

# Save to CSV
df.to_csv(simulated_dataset_path, index=False)
```

```
    C   A   B   D
0  c2  a2  b2  d2
1  c2  a3  b1  d1
2  c2  a2  b2  d2
3  c2  a1  b1  d1
4  c1  a2  b2  d2
```

[6]:
```
cate_ds = pysmile.learning.DataSet()

try:
    cate_ds.read_file(simulated_dataset_path)
except pysmile.SMILEException:
    print("Dataset load failed")
#endtry

print(f"Dataset has {cate_ds.get_variable_count()} variables (columns) "
    + f"and {cate_ds.get_record_count()} records (rows)")
```

```
Dataset has 4 variables (columns) and 50000 records (rows)
```

1.2 Structure learning from simulated dataset

[7]:
```
bayes_search = pysmile.learning.BayesianSearch()
bayes_search.set_iteration_count(50)
bayes_search.set_rand_seed(9876543)

## (1)
try:
    net1 = bayes_search.learn(cate_ds)
    print(f"1st Bayesian Search finished, structure score: {bayes_search.
 ↪get_last_score()}")
    net1.write_file("./simulated_data/learned_cate_net_bayes_1.xdsl")
except pysmile.SMILEException:
    print("Bayesian Search failed")
#endtry

## (2)
bayes_search = pysmile.learning.BayesianSearch()
bayes_search.set_iteration_count(50)
bayes_search.set_rand_seed(3456789)
try:
    net2 = bayes_search.learn(cate_ds)
    print(f"2nd Bayesian Search finished, structure score: {bayes_search.
 ↪get_last_score()}")
    net2.write_file("./simulated_data/learned_cate_net_bayes_2.xdsl")
except pysmile.SMILEException:
    print("Bayesian Search failed")
#endtry

## (3)
pc = pysmile.learning.PC()
try:
    pattern = pc.learn(cate_ds)
    net5 = pattern.make_network(cate_ds)
    print("PC finished, proceeding to parameter learning")
    net5.write_file("./simulated_data/learned_cate_net_pc.xdsl")
except pysmile.SMILEException:
    print("PC failed")
#endtry
```

```
1st Bayesian Search finished, structure score: -161415.45206879414
2nd Bayesian Search finished, structure score: -161321.79663454118
PC finished, proceeding to parameter learning
2nd Bayesian Search finished, structure score: -161321.79663454118
PC finished, proceeding to parameter learning
```

1.3 Use predefined network and simulated data for parameter learning

[8]:
```
def create_cpt_node(net, id, name, outcomes, x_pos, y_pos):

    handle = net.add_node(pysmile.NodeType.CPT, id)

    net.set_node_name(handle, name)
    net.set_node_position(handle, x_pos, y_pos, 85, 55)

    initial_outcome_count = net.get_outcome_count(handle)

    for i in range(0, initial_outcome_count):
        net.set_outcome_id(handle, i, outcomes[i])

    for i in range(initial_outcome_count, len(outcomes)):
        net.add_outcome(handle, outcomes[i])

    return handle
```

To simulate categorical variables A, B, C, D with the following Bayesian network dependencies:

- $C \rightarrow A$
- $A, B \rightarrow D$

[9]:
```
# Define categories
# categories_C = ['c1', 'c2', 'c3']
# categories_A = ['a1', 'a2', 'a3']
# categories_B = ['b1', 'b2']
# categories_D = ['d1', 'd2']

cate_net = pysmile.Network()

A = create_cpt_node(cate_net, "A", "A", ['a1', 'a2', 'a3'], 10, 20)
B = create_cpt_node(cate_net, "B", "B", ['b1', 'b2'], 10, 30)
C = create_cpt_node(cate_net, "C", "C", ['c1', 'c2', 'c3'], 10, 40)
D = create_cpt_node(cate_net, "D", "D", ['d1', 'd2'], 10, 50)

cate_net.add_arc(C, A)
cate_net.add_arc(A, D)
cate_net.add_arc(B, D)

cate_net.write_file("./simulated_data/predefined_cate_net.xdsl")
```

[10]:
```
em = pysmile.learning.EM()

try:
    matching = cate_ds.match_network(cate_net)
except pysmile.SMILEException:
```

```
    print("Can't automatically match network with dataset")
#endtry

em.set_uniformize_parameters(False)
em.set_randomize_parameters(False)
em.set_eq_sample_size(0)

try:
    em.learn(cate_ds, cate_net, matching)
except pysmile.SMILEException:
    print("EM failed")
#endtry

print("EM finished")
cate_net.write_file("./simulated_data/simulated_data_em_cate.xdsl")
print("Complete.")
```

```
EM finished
Complete.
```