The plugin otagrum: learning nonparametric Copula Bayesian Networks

Marvin LASSERRE

supervised by Pierre-Henri WUILLEMIN (LIP6) and Régis LEBRUN (Airbus CRT)

June 10, 2022

Context: learning high dimensional non-parametric continuous distributions

- **Why distributions ?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),

Context: learning high dimensional non-parametric continuous distributions

- **Why distributions ?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),

- **Why continuous ?** Because in applications such as physics, engineering or finance variables are often continuous,

Context: learning high dimensional non-parametric continuous distributions

- **Why distributions ?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),
- **Why continuous ?** Because in applications such as physics, engineering or finance variables are often continuous,
- **Why non-parametric ?** Because parametric models such as Gaussian are too restrictive for certain applications such as anomaly detection, risk analysis or reliability analysis.

Context: learning high dimensional non-parametric continuous distributions

- **Why distributions ?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),
- **Why continuous ?** Because in applications such as physics, engineering or finance variables are often continuous,
- **Why non-parametric ?** Because parametric models such as Gaussian are too restrictive for certain applications such as anomaly detection, risk analysis or reliability analysis.
- **Why high-dimensional ?** Because complex systems involve a large number of variables,

Context: learning high dimensional non-parametric continuous distributions

- **Why distributions ?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),
- **Why continuous ?** Because in applications such as physics, engineering or finance variables are often continuous,
- **Why non-parametric ?** Because parametric models such as Gaussian are too restrictive for certain applications such as anomaly detection, risk analysis or reliability analysis.
- **Why high-dimensional ?** Because complex systems involve a large number of variables,
- **Challenge:** Various non-parametric models exists to estimate a density but they are limited to a **few dimensions** (~ 5 variables),

Context: learning high dimensional non-parametric continuous distributions

- **Why distributions ?** Because we are faced with uncertainties (lack of information, inherently uncertain problems),
- **Why continuous ?** Because in applications such as physics, engineering or finance variables are often continuous,
- **Why non-parametric ?** Because parametric models such as Gaussian are too restrictive for certain applications such as anomaly detection, risk analysis or reliability analysis.
- **Why high-dimensional ?** Because complex systems involve a large number of variables,
- **Challenge:** Various non-parametric models exists to estimate a density but they are limited to a **few dimensions** (~ 5 variables),
- **Solution**: Use of Probabilistic Graphical Models (PGM) to break the joint distribution into a product of conditional distributions of **lesser dimensions**.

Modeling with copulas

Definition

Definition

- $\boldsymbol{U} = (U_1, \cdots, U_n)$, **continuous** random variable over $[0, 1]^n$,

Definition

- $\boldsymbol{U} = (U_1, \cdots, U_n)$, **continuous** random variable over $[0,1]^n$,

Definition (Copula Nelsen 2007)

A copula function is a cumulative distribution function on $[0,1]^n$:

$$C(u_1, \ldots, u_n) = \mathbb{P}(U_1 \leq u_1, \ldots, U_n \leq u_n)$$

with **uniform** one-dimensional marginals :

$$C(1, \ldots, u_i, \ldots, 1) = u_i.$$

Definition

- $\boldsymbol{U} = (U_1, \cdots, U_n)$, **continuous** random variable over $[0,1]^n$,

Definition (Copula Nelsen 2007)

A copula function is a cumulative distribution function on $[0,1]^n$:

$$C(u_1, \ldots, u_n) = \mathbb{P}(U_1 \leq u_1, \ldots, U_n \leq u_n)$$

with **uniform** one-dimensional marginals :

$$C(1, \ldots, u_i, \ldots, 1) = u_i.$$

- If C is **absolutely continuous**, a copula density function c exists :

$$c(\mathrm{x}) = \frac{\partial^n C}{\partial x_1 \cdots \partial x_n}(x_1, \cdots, x_n)$$

Sklar's theorem

Theorem (Sklar, 1959)

Sklar's theorem

Theorem (Sklar, 1959)

For any ***continuous*** *distribution* F *over* $X_1, \cdots, X_n$*, there exists a* ***unique*** *copula function* C*, such that:*

Sklar's theorem

Theorem (Sklar, 1959)

*For any **continuous** distribution F over $X_1, \cdots, X_n$, there exists a **unique** copula function C, such that:*

$$F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))$$

Sklar's theorem

Theorem (Sklar, 1959)

For any ***continuous*** *distribution F over $X_1, \cdots, X_n$, there exists a* ***unique*** *copula function C, such that:*

$$F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))$$

Moreover, if F is ***absolutely*** *continuous,*

$$f(x_1, \cdots, x_n) = c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)$$

Sklar's theorem

Theorem (Sklar, 1959)

For any ***continuous*** *distribution F over $X_1, \cdots, X_n$, there exists a* ***unique*** *copula function C, such that:*

$$F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))$$

Moreover, if F is ***absolutely*** *continuous,*

$$f(x_1, \cdots, x_n) = c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)$$

- Decomposition of the joint distribution into a copula function and a set of marginals : **more freedom for modeling**.

Sklar's theorem

Theorem (Sklar, 1959)

*For any **continuous** distribution F over $X_1, \cdots, X_n$, there exists a **unique** copula function C, such that:*

$$F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))$$

*Moreover, if F is **absolutely** continuous,*

$$f(x_1, \cdots, x_n) = c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)$$

- Decomposition of the joint distribution into a copula function and a set of marginals : **more freedom for modeling**.
- C encodes all the information about the dependencies between the variables: **interesting for independence tests**.

Sklar's theorem

Theorem (Sklar, 1959)

*For any **continuous** distribution F over $X_1, \cdots, X_n$, there exists a **unique** copula function C, such that:*

$$F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))$$

*Moreover, if F is **absolutely** continuous,*

$$f(x_1, \cdots, x_n) = c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)$$

- Decomposition of the joint distribution into a copula function and a set of marginals : **more freedom for modeling**.
- C encodes all the information about the dependencies between the variables: **interesting for independence tests**.
- Non-parametric estimation of the copula with the Empirical Bernstein Copula (EBC),

Sklar's theorem

Theorem (Sklar, 1959)

*For any **continuous** distribution F over $X_1, \cdots, X_n$, there exists a **unique** copula function C, such that:*

$$F(x_1, \cdots, x_n) = C(F_1(x_1), \cdots, F_n(x_n))$$

*Moreover, if F is **absolutely** continuous,*

$$f(x_1, \cdots, x_n) = c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i)$$

- Decomposition of the joint distribution into a copula function and a set of marginals : **more freedom for modeling**.
- C encodes all the information about the dependencies between the variables: **interesting for independence tests**.
- Non-parametric estimation of the copula with the Empirical Bernstein Copula (EBC),

⚠ C becomes hard to model for high dimensions !

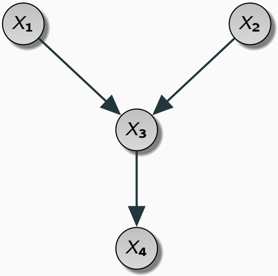
Modeling with Bayesian Networks

Bayesian Networks

- **Compact** representation of a joint probability distribution over a set of variables $\boldsymbol{X}$ using :

Bayesian Networks

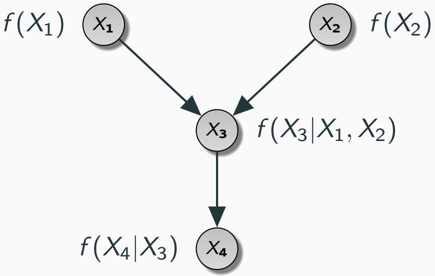
- **Compact** representation of a joint probability distribution over a set of variables $\boldsymbol{X}$ using :
  - A Directed Acyclic Graph (*DAG*),



$$\mathcal{I}_l(\mathcal{G}) = \{(X_i \perp \mathrm{ND}_i | \mathrm{Pa}_i)\}.$$

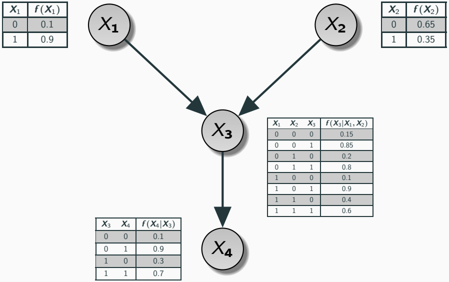
Bayesian Networks

- **Compact** representation of a joint probability distribution over a set of variables $\boldsymbol{X}$ using :
  - A Directed Acyclic Graph (*DAG*),
  - A set of Conditional Probability Distributions (*CPD*).



$$f(\boldsymbol{x}) = \prod_{i=1}^{n} f(x_i|\mathbf{pa}_i)$$

- **Compact** representation of a joint probability distribution over a set of variables $\boldsymbol{X}$ using :
  - A Directed Acyclic Graph (*DAG*),
  - A set of Conditional Probability Distributions (*CPD*).



X_1	$f(X_1)$
0	0.1
1	0.9

X_2	$f(X_2)$
0	0.65
1	0.35

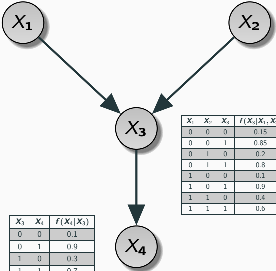
X_1	X_2	X_3	$f(X_3 \mid X_1, X_2)$
0	0	0	0.15
0	0	1	0.85
0	1	0	0.2
0	1	1	0.8
1	0	0	0.1
1	0	1	0.9
1	1	0	0.4
1	1	1	0.6

X_3	X_4	$f(X_4 \mid X_3)$
0	0	0.1
0	1	0.9
1	0	0.3
1	1	0.7

Discrete case : Conditional Probability Tables.

Bayesian Networks

- **Compact** representation of a joint probability distribution over a set of variables $\boldsymbol{X}$ using :
  - A Directed Acyclic Graph (*DAG*),
  - A set of Conditional Probability Distributions (*CPD*).



X_1	$f(X_1)$
0	0.1
1	0.9

X_2	$f(X_2)$
0	0.65
1	0.35

X_1	X_2	X_3	$f(X_3 \mid X_1, X_2)$
0	0	0	0.15
0	0	1	0.85
0	1	0	0.2
0	1	1	0.8
1	0	0	0.1
1	0	1	0.9
1	1	0	0.4
1	1	1	0.6

X_3	X_4	$f(X_4 \mid X_3)$
0	0	0.1
0	1	0.9
1	0	0.3
1	1	0.7

Discrete case : Conditional Probability Tables.

Continuous case : ???

Bayesian Networks and continuous data

- **Discretization** :
  1. Limited to only a few bins for fast inference and learning algorithms.
  2. Which one do we chose to minimize the loss of information ?
  3. How to a continuous model from there ?

Bayesian Networks and continuous data

- **Discretization** :
  1. Limited to only a few bins for fast inference and learning algorithms.
  2. Which one do we chose to minimize the loss of information ?
  3. How to a continuous model from there ?

- **Linear Gaussian Bayesian Networks (LGBN)** Lauritzen et al. 1989: $f(y|\boldsymbol{x}) = \mathcal{N}(y; \beta_0 + \sum_{i=1}^{k} \beta_i x_i, \sigma_y^2)$
  1. Good: Fast inference and learning algorithms,
  2. Bad: Strong model assumptions (Gaussian),

Bayesian Networks and continuous data

- **Discretization** :
  1. Limited to only a few bins for fast inference and learning algorithms.
  2. Which one do we chose to minimize the loss of information ?
  3. How to a continuous model from there ?

- **Linear Gaussian Bayesian Networks (LGBN)** Lauritzen et al. 1989: $f(y|\boldsymbol{x}) = \mathcal{N}(y; \beta_0 + \sum_{i=1}^{k} \beta_i x_i, \sigma_y^2)$
  1. Good: Fast inference and learning algorithms,
  2. Bad: Strong model assumptions (Gaussian),

- **Mixture models:** Langseth et al. 2012; Cortijo et al. 2016
  1. Good: Expressive models,
  2. Bad: Hard to learn

Copula Bayesian Networks (CBNs)

Copula Bayesian Networks : definition

Definition (Copula Bayesian Network, Elidan 2010)

Copula Bayesian Networks : definition

Definition (Copula Bayesian Network, Elidan 2010)

- $\mathcal{G}$: DAG over $\boldsymbol{X}$,

Copula Bayesian Networks : definition

Definition (Copula Bayesian Network, Elidan 2010)

- $\mathcal{G}$: DAG over $\boldsymbol{X}$,
- Θ_C : set of (local) copula densities c_i,

Copula Bayesian Networks : definition

Definition (Copula Bayesian Network, Elidan 2010)

- $\mathcal{G}$: DAG over $\boldsymbol{X}$,
- Θ_C : set of (local) copula densities c_i,
- Θ_f set of marginal densities f_i

Copula Bayesian Networks : definition

Definition (Copula Bayesian Network, Elidan 2010)

- $\mathcal{G}$: DAG over $\boldsymbol{X}$,
- Θ_C : set of (local) copula densities c_i,
- Θ_f set of marginal densities f_i

A Copula Bayesian Network (CBN) is a triplet $(\mathcal{G}, \Theta_C, \Theta_f)$

Copula Bayesian Networks : definition

Definition (Copula Bayesian Network, Elidan 2010)

- $\mathcal{G}$: DAG over $\boldsymbol{X}$,
- Θ_C : set of (local) copula densities c_i,
- Θ_f set of marginal densities f_i

A Copula Bayesian Network (CBN) is a triplet $(\mathcal{G}, \Theta_C, \Theta_f)$ which encodes a joint density $f(\boldsymbol{X})$ that factorizes over $\mathcal{G}$:

$$
\begin{aligned}
f(x_1, \cdots, x_n) &= c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i) \ \text{(Sklar)} \\
&= \prod_{i=1}^{n} R_i(F_i(x_i) | \boldsymbol{F}(\mathrm{pa}_{X_i})) \cdot f_i(x_i)
\end{aligned}
$$

where $R_i(u_i|\pi_i) = \frac{c_i(u_i, \boldsymbol{\pi}_i)}{c_i(\boldsymbol{\pi}_i)}$.

Copula Bayesian Networks : definition

Definition (Copula Bayesian Network, Elidan 2010)

- $\mathcal{G}$: DAG over $\boldsymbol{X}$,
- Θ_C : set of (local) copula densities c_i,
- Θ_f set of marginal densities f_i

A Copula Bayesian Network (CBN) is a triplet $(\mathcal{G}, \Theta_C, \Theta_f)$ which encodes a joint density $f(\boldsymbol{X})$ that factorizes over $\mathcal{G}$:

$$\begin{aligned} f(x_1, \cdots, x_n) &= c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i) \quad \text{(Sklar)} \\ &= \prod_{i=1}^{n} R_i(F_i(x_i) | \boldsymbol{F}(\mathrm{pa}_{X_i})) \cdot f_i(x_i) \end{aligned}$$

where $R_i(u_i|\pi_i) = \frac{c_i(u_i, \boldsymbol{\pi}_i)}{c_i(\boldsymbol{\pi}_i)}$.

- Same graphical language than classical BNs (same independences)

Copula Bayesian Networks : definition

Definition (Copula Bayesian Network, Elidan 2010)

- $\mathcal{G}$: DAG over $\boldsymbol{X}$,
- Θ_C : set of (local) copula densities c_i,
- Θ_f set of marginal densities f_i

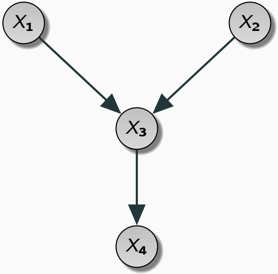
A Copula Bayesian Network (CBN) is a triplet $(\mathcal{G}, \Theta_C, \Theta_f)$ which encodes a joint density $f(\boldsymbol{X})$ that factorizes over $\mathcal{G}$:

$$
\begin{aligned}
f(x_1, \cdots, x_n) &= c(F_1(x_1), \cdots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i) \quad \text{(Sklar)} \\
&= \prod_{i=1}^{n} R_i(F_i(x_i) | \boldsymbol{F}(\mathrm{pa}_{X_i})) \cdot f_i(x_i)
\end{aligned}
$$

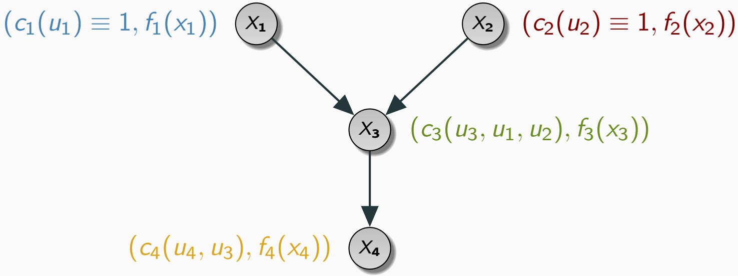
where $R_i(u_i|\pi_i) = \frac{c_i(u_i, \boldsymbol{\pi}_i)}{c_i(\boldsymbol{\pi}_i)}$.

- Same graphical language than classical BNs (same independences)
- Classic algorithms can be adapted for structural learning.

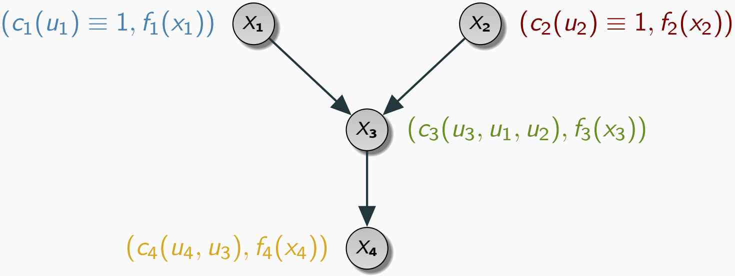
Copula Bayesian Networks : example



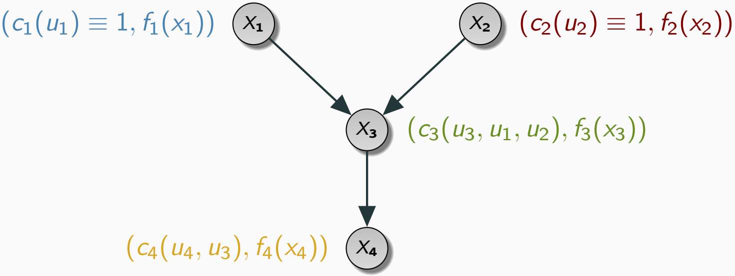
Copula Bayesian Networks : example



Copula Bayesian Networks : example

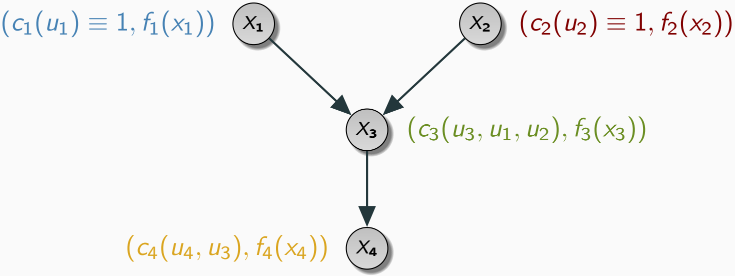


- $\Theta_C = \{c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3)\}$



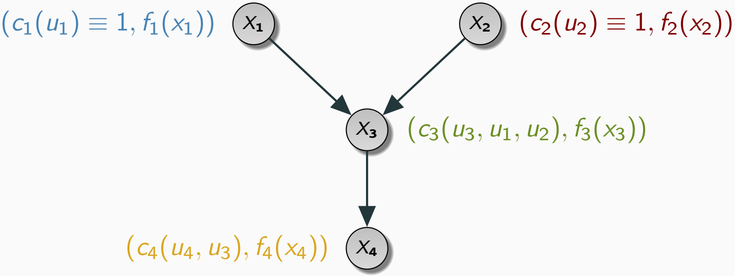
- $\Theta_C = \{c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3)\}$
- $\Theta_f = \{f_1(x_1), f_2(x_2), f_3(x_3), f_4(x_4)\}$

Copula Bayesian Networks : example



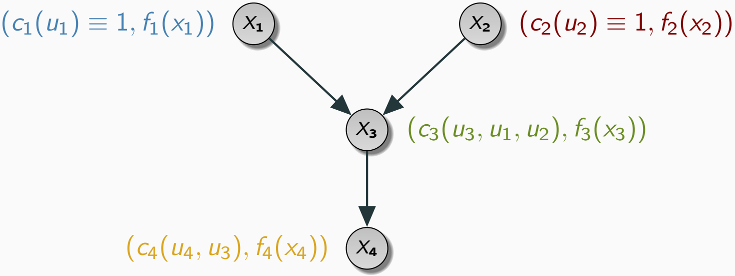
- $\Theta_C = \{c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3)\}$
- $\Theta_f = \{f_1(x_1), f_2(x_2), f_3(x_3), f_4(x_4)\}$
- $$\begin{aligned} f(x_1, x_2, x_3, x_4) = &[R_1(F_1(x_1))f_1(x_1)][R_2(F_2(x_2))f_2(x_2)] \\ &\times [R_3(F_3(x_3)|F_1(x_1), F_2(x_2))f_3(x_3)] \\ &\times [R_4(F_4(x_4)|F_3(x_3))f_4(x_4)] \end{aligned}$$

Copula Bayesian Networks : example



- $\Theta_C = \{c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3)\}$
- $\Theta_f = \{f_1(x_1), f_2(x_2), f_3(x_3), f_4(x_4)\}$
- $$\begin{aligned} f(x_1, x_2, x_3, x_4) = & [R_1(F_1(x_1))f_1(x_1)][R_2(F_2(x_2))f_2(x_2)] \\ & \times [R_3(F_3(x_3)|F_1(x_1), F_2(x_2))f_3(x_3)] \\ & \times [R_4(F_4(x_4)|F_3(x_3))f_4(x_4)] \end{aligned}$$
- Parametric copulas: Gaussian, Student, Dirichlet, ...

Copula Bayesian Networks : example



- $\Theta_C = \{c_1(u_1) \equiv 1, c_2(u_2) \equiv 1, c_3(u_3, u_1, u_2), c_4(u_4, u_3)\}$
- $\Theta_f = \{f_1(x_1), f_2(x_2), f_3(x_3), f_4(x_4)\}$
- $$\begin{aligned} f(x_1, x_2, x_3, x_4) = {} & [R_1(F_1(x_1))f_1(x_1)][R_2(F_2(x_2))f_2(x_2)] \\ & \times [R_3(F_3(x_3)|F_1(x_1), F_2(x_2))f_3(x_3)] \\ & \times [R_4(F_4(x_4)|F_3(x_3))f_4(x_4)] \end{aligned}$$
- Parametric copulas: Gaussian, Student, Dirichlet, ...
- Non-parametric copulas: Empirical Bernstein Copula (EBC)

Structure learning for CBNs

Learning algorithms

- CPC a continuous PC algorithm based on an independence test using Hellinger distance:
  - M. Lasserre et al. (May 2020). "Constraint-Based Learning for Non-Parametric Continuous Bayesian Networks". In: *FLAIRS 33 - 33rd Florida Artificial Intelligence Research Society Conference*. Miami, United States: AAAI, pp. 581–586
  - M. Lasserre et al. (2021a). "Constraint-based learning for non-parametric continuous bayesian networks". In: *Annals of Mathematics and Artificial Intelligence*, pp. 1–18

Learning algorithms

- CPC a continuous PC algorithm based on an independence test using Hellinger distance:
  - M. Lasserre et al. (May 2020). "Constraint-Based Learning for Non-Parametric Continuous Bayesian Networks". In: *FLAIRS 33 - 33rd Florida Artificial Intelligence Research Society Conference*. Miami, United States: AAAI, pp. 581–586
  - M. Lasserre et al. (2021a). "Constraint-based learning for non-parametric continuous bayesian networks". In: *Annals of Mathematics and Artificial Intelligence*, pp. 1–18
- CMIIC, an algorithm based on information theory:
  - M. Lasserre et al. (2021b). "Learning Continuous High-Dimensional Models using Mutual Information and Copula Bayesian Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 13, pp. 12139–12146

Learning algorithms

- CPC a continuous PC algorithm based on an independence test using Hellinger distance:
  - M. Lasserre et al. (May 2020). "Constraint-Based Learning for Non-Parametric Continuous Bayesian Networks". In: *FLAIRS 33 - 33rd Florida Artificial Intelligence Research Society Conference*. Miami, United States: AAAI, pp. 581–586
  - M. Lasserre et al. (2021a). "Constraint-based learning for non-parametric continuous bayesian networks". In: *Annals of Mathematics and Artificial Intelligence*, pp. 1–18
- CMIIC, an algorithm based on information theory:
  - M. Lasserre et al. (2021b). "Learning Continuous High-Dimensional Models using Mutual Information and Copula Bayesian Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 13, pp. 12139–12146
- Improvement of the state of the art algorithm (CBIC) by using mutual information to speed up the calculations.

Comparison method

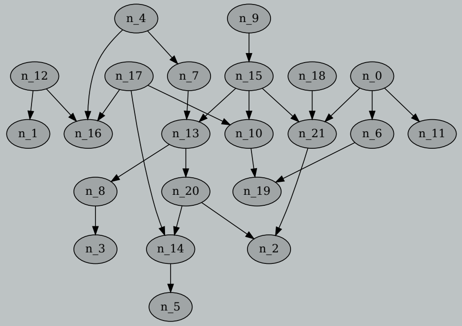
1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed : F-score et SHD.

Comparison method

→ 1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed : F-score et SHD.

Comparison method

→ 1. We generate **random** reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed : F-score et SHD.



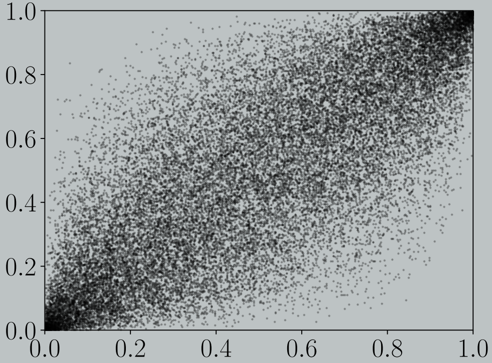
Number of nodes : n $\Longrightarrow$ Number of arcs : $\lfloor 1.2 \times n \rfloor$

Comparison method

1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed : F-score et SHD.

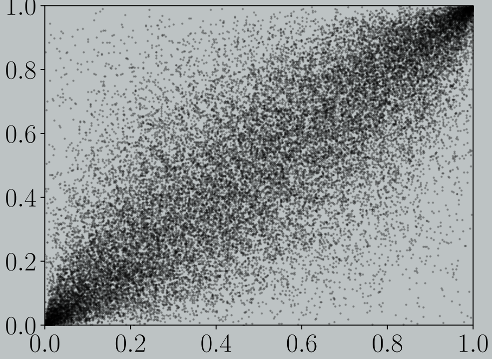
Comparison method

1. We generate random reference structures,
2. Copulas are parametrized : **Gaussian**, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed : F-score et SHD.



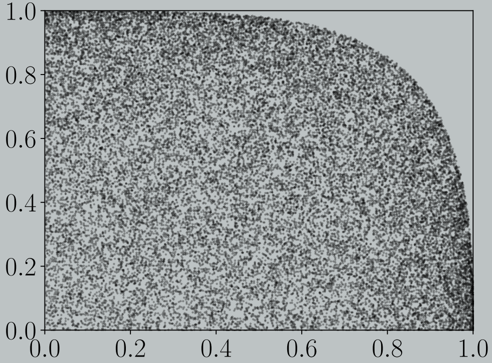
Comparison method

1. We generate random reference structures,
2. → Copulas are parametrized : Gaussian, **Student** or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed : F-score et SHD.



Comparison method

1. We generate random reference structures,
2. $\rightarrow$ Copulas are parametrized : Gaussian, Student or **Dirichlet**,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,
5. Structural scores are computed : F-score et SHD.



Comparison method

1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,

→ 3. Samples are generated from the CBN : forward-sampling,

4. A structure is learned from the generated data,
5. Structural scores are computed : F-score et SHD.

Comparison method

1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,

→ 4. A structure is learned from the generated data,

5. Structural scores are computed : F-score et SHD.

Comparison method

1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,

$\rightarrow$ 5. Structural scores are computed : F-score et SHD.

Comparison method

1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,

$\rightarrow$ 5. Structural scores are computed : **F-score** et SHD.

Comparison method

1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,

$\rightarrow$ 5. Structural scores are computed : **F-score** et SHD.

- **F-score** : skeleton (undirected structure)

Comparison method

1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,

$\rightarrow$ 5. Structural scores are computed : F-score et **SHD**.

- **F-score** : skeleton (undirected structure)
  - Skeleton perfectly retrieved : **F-score = 1**

Comparison method

1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,

→ 5. Structural scores are computed : F-score et **SHD**.

- **F-score** : skeleton (undirected structure)
  - Skeleton perfectly retrieved : **F-score = 1**
- **Structural Hamming Distance** (SHD) : CPDAG (skeleton + v-structures)

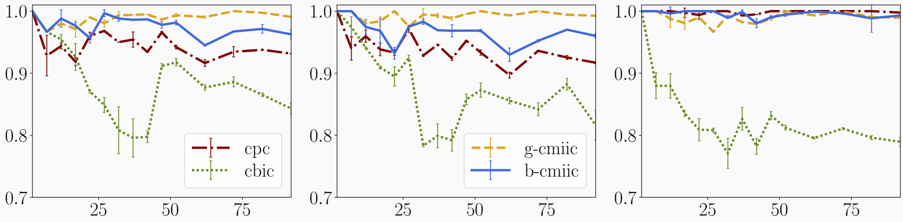
Comparison method

1. We generate random reference structures,
2. Copulas are parametrized : Gaussian, Student or Dirichlet,
3. Samples are generated from the CBN : forward-sampling,
4. A structure is learned from the generated data,

→ 5. Structural scores are computed : F-score et SHD.

- **F-score** : skeleton (undirected structure)
  - Skeleton perfectly retrieved : **F-score = 1**
- **Structural Hamming Distance** (SHD) : CPDAG (skeleton + v-structures)
  - CPDAG perfectly retrieved : **SHD = 0**

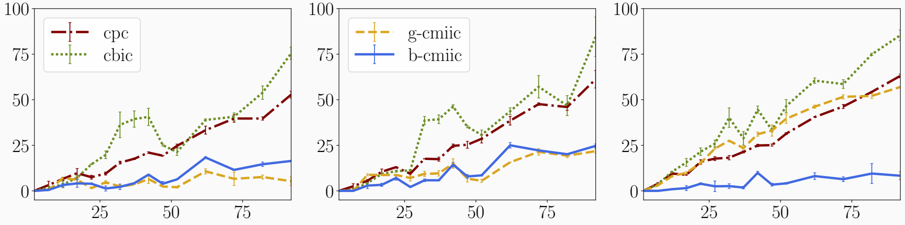
F-score evolution : random structures



(a) Gaussian case **(b)** Student case **(c)** Dirichlet case

F-score evolution for **CBIC**, **CPC**, **G-CMIIC** and **B-CMIIC** methods with respect to the **dimension** of the random structures. The results are averaged over 2 random structures of same dimension and over 5 different samples of size $m = 10^4$.

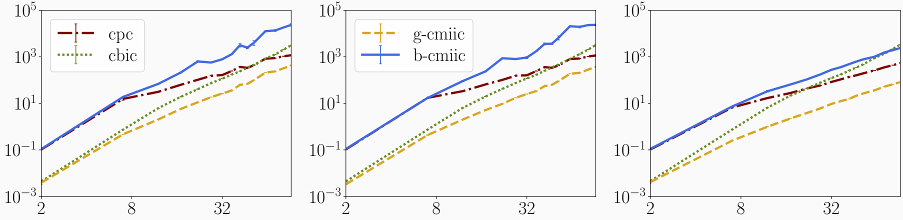
SHD evolution : random structures



(a) Gaussian case **(b)** Student case **(c)** Dirichlet case

SHD evolution for CBIC, CPC, G-CMIIC and B-CMIIC methods with respect to the **dimension** of the random structure. The results are averaged over 2 different structures of same dimension and over 5 different samples of size $m = 10^4$.

Temporal complexity



(a) Gaussian case **(b)** Student case **(c)** Dirichlet case

Learning time in seconds for CBIC, CPC, G-CMIIC et B-CMIIC with respect to the **dimension** of the random structures. The results are averaged over 2 different **random structures** of same dimension and over 5 different samples of size $m = 10^4$.

The otagrum module

otagrum: an open source library to learn CBNs

Two similar libraries (`C++`, python wrappers, open source):

otagrum: an open source library to learn CBNs

Two similar libraries (`C++`, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).

otagrum: an open source library to learn CBNs

Two similar libraries (C++, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

otagrum: an open source library to learn CBNs

Two similar libraries (`C++`, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

otagrum: an open source library to learn CBNs

Two similar libraries (C++, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

What does it contain ?

otagrum: an open source library to learn CBNs

Two similar libraries (`C++`, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

What does it contain ?

- A CBN class,

otagrum: an open source library to learn CBNs

Two similar libraries (`C++`, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

What does it contain ?

- A CBN class,
- Several learning algorithms,

otagrum: an open source library to learn CBNs

Two similar libraries (`C++`, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

What does it contain ?

- A CBN class,
- Several learning algorithms,
- A detailed documentation.

otagrum: an open source library to learn CBNs

Two similar libraries (`C++`, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

What does it contain ?

- A CBN class,
- Several learning algorithms,
- A detailed documentation.

Where to find it ?

otagrum: an open source library to learn CBNs

Two similar libraries (`C++`, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

What does it contain ?

- A CBN class,
- Several learning algorithms,
- A detailed documentation.

Where to find it ?

- Module : openturns/otagrum (GitHub)

otagrum: an open source library to learn CBNs

Two similar libraries (`C++`, python wrappers, open source):

- **OpenTURNS** deals with copulas and continuous distributions (available on GitHub, pip and conda).
- **aGrUM** deals with (discrete) graphical models (available on GitLab, pip and conda).

A module to rule them all: **otagrum**.

What does it contain ?

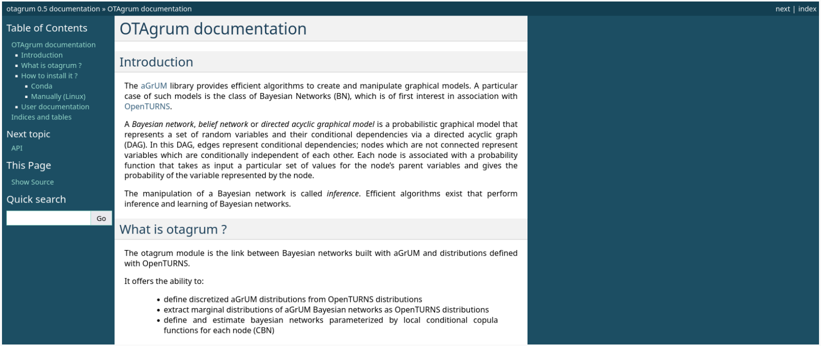
- A CBN class,
- Several learning algorithms,
- A detailed documentation.

Where to find it ?

- Module : openturns/otagrum (GitHub)
- Experiments : MLasserre/otagrum-experiments (GitHub)

otagrum: installation

- Online website : `https://openturns.github.io/otagrum/master/index.html`



otagrum 0.5 documentation » OTAgrum documentation　　　　next | index

Table of Contents

- OTAgrum documentation
  - Introduction
  - What is otagrum ?
  - How to install it ?
    - Conda
    - Manually (Linux)
  - User documentation
- Indices and tables

Next topic

API

This Page

Show Source

Quick search

Go

OTAgrum documentation

Introduction

The aGrUM library provides efficient algorithms to create and manipulate graphical models. A particular case of such models is the class of Bayesian Networks (BN), which is of first interest in association with OpenTURNS.

A *Bayesian network, belief network* or *directed acyclic graphical model* is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). In this DAG, edges represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other. Each node is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability of the variable represented by the node.

The manipulation of a Bayesian network is called *inference*. Efficient algorithms exist that perform inference and learning of Bayesian networks.

What is otagrum ?

The otagrum module is the link between Bayesian networks built with aGrUM and distributions defined with OpenTURNS.

It offers the ability to:

- define discretized aGrUM distributions from OpenTURNS distributions
- extract marginal distributions of aGrUM Bayesian networks as OpenTURNS distributions
- define and estimate bayesian networks parameterized by local conditional copula functions for each node (CBN)

- Can be easily installed using conda:

```
$ conda install -c conda-forge otagrum
```

- Or manually to have the development version.

otagrum: an example of use

Using OTaGrUM: The wine data set

Importing modules

Entrée [1]:

```
import openturns as ot
import openturns.viewer as otv

import pyAgrum as gum
import pyAgrum.lib.notebook as gnb

import otagrum as otagr
```

Loading data

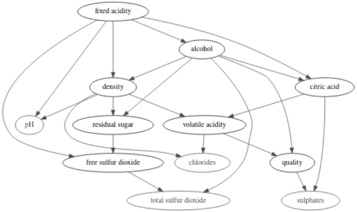
Entrée [2]:

```
data_ref = ot.Sample.ImportFromTextFile('winequality-red.csv', ";")
```

otagrum: an example of use

Structure learning with CBIC algorithm

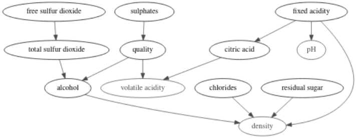
Entrée [3]:
```
learner = otagr.TabuList(data_ref, 2, 10, 2) # Creating a TabuList learner
cbic_dag = learner.learnDAG()                 # Learning DAG
gnb.showDot(cbic_dag.toDot())
```



otagrum: an example of use

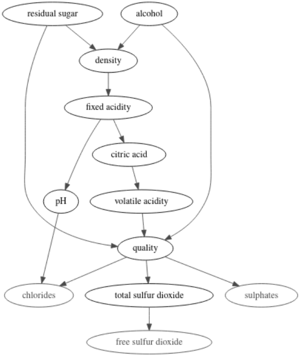
Structure learning with CPC algorithm

Entrée [4]:
```
learner = otagr.ContinuousPC(data_ref, 4, 0.05) # Using a CPC learner
cpc_dag = learner.learnDAG()                    # Learning DAG
gnb.showDot(cpc_dag.toDot())
```



Structure learning with CMIIC algorithm

Entrée [5]:
```
learner = otagr.ContinuousMIIC(data_ref) # Using a CMIIC learner
learner.setAlpha(0.04)                   # Setting the value of alpha
cmiic_dag = learner.learnDAG()           # Learning DAG
gnb.showDot(cmiic_dag.toDot())
```



otagrum: an example of use

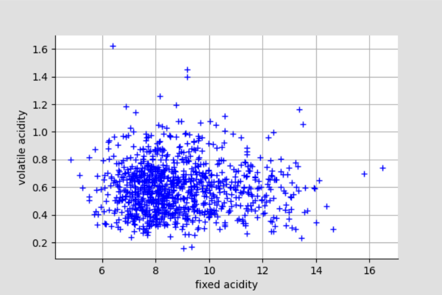
Parameter learning

Entrée [7]:
```
cpc_cbn = otagr.ContinuousBayesianNetworkFactory(ot.KernelSmoothing(ot.Histogram()),
                                                  ot.BernsteinCopulaFactory(),
                                                  cpc_dag,
                                                  0.05,
                                                  4,
                                                  False).build(data_ref)
```

Sampling the CBN

Entrée [9]:
```
sample = cpc_cbn.getSample(1000)
ot.VisualTest.DrawPairs(sample.getMarginal([0,1]))
```

Out[9]:



Conclusion & Future Works

Conclusion

Summary:

- CBNs allow to take advantage of conditional independences to reduce the global complexity,
- Using the Empirical Bernstein Copula we obtained non-parametric independence tests and non-parametric CBNs,
- We implemented learning algorithms for CBNs in the plugin otagrum,

Future works : Inference in CBNs

- It consists in finding:
$$f(\boldsymbol{T}|\boldsymbol{E}=\boldsymbol{e})$$
where $\boldsymbol{T}, \boldsymbol{E} \subset \boldsymbol{X}$ such that $\boldsymbol{T} \cap \boldsymbol{E}$.
- Use of sampling to make approximate inferences,
- Use of junction trees to make numerical integrations.

Thank you for your attention !

Bibliography

Cortijo, S. and C. Gonzales (2016). "Bayesian networks with conditional truncated densities". In: *The Twenty-Ninth International Flairs Conference* (cit. on pp. 27–29).

Elidan, G. (2010). "Copula bayesian networks". In: *Advances in neural information processing systems*, pp. 559–567 (cit. on pp. 31–38).

Langseth, H., T. D. Nielsen, R. Rumı, and A. Salmerón (2012). "Mixtures of truncated basis functions". In: *International Journal of Approximate Reasoning* 53.2, pp. 212–227 (cit. on pp. 27–29).

Lasserre, M., R. Lebrun, and P.-H. Wuillemin (May 2020). "Constraint-Based Learning for Non-Parametric Continuous Bayesian Networks". In: *FLAIRS 33 - 33rd Florida Artificial Intelligence Research Society Conference*. Miami, United States: AAAI, pp. 581–586 (cit. on pp. 47–49).

Lasserre, M., R. Lebrun, and P.-H. Wuillemin (2021a). "Constraint-based learning for non-parametric continuous bayesian networks". In: *Annals of Mathematics and Artificial Intelligence*, pp. 1–18 (cit. on pp. 47–49).

Lasserre, M., R. Lebrun, and P.-H. Wuillemin (2021b). "Learning Continuous High-Dimensional Models using Mutual Information and Copula Bayesian Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 13, pp. 12139–12146 (cit. on pp. 47–49).

Lauritzen, S. L. and N. Wermuth (1989). "Graphical models for associations between variables, some of which are qualitative and some quantitative". In: *The annals of Statistics*, pp. 31–57 (cit. on pp. 27–29).

Nelsen, R. B. (2007). *An introduction to copulas*. Springer Science & Business Media (cit. on pp. 9–12).