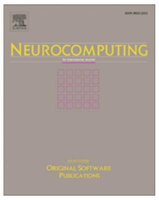
Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



Original software publication

PyBNesian: An extensible python package for Bayesian networks

David Atienza [*], Concha Bielza, Pedro Larrañaga

Universidad Politécnica de Madrid, Departamento de Inteligencia Artificial, 28660 Boadilla del Monte, Spain

ARTICLE INFO

Article history:
Received 10 March 2022
Revised 20 May 2022
Accepted 27 June 2022
Available online 08 July 2022
Communicated by Zidong Wang

Keywords:
Bayesian networks
Kernel density estimation
Dynamic models
Conditional independence

ABSTRACT

Bayesian networks are probabilistic graphical models that are commonly used to represent the uncertainty in data. The PyBNesian package provides an implementation for many different types of Bayesian network models and some variants, such as conditional Bayesian networks and dynamic Bayesian networks. In addition, the package can be easily extended with new components that can interoperate with those already implemented. Furthermore, the package also implements other related models such as kernel density estimation using OpenCL 1.2+ to enable GPU acceleration. PyBNesian is totally free and open-source under the MIT license.

1. Introduction

Bayesian networks [19,13,15] are probabilistic graphical models that can represent multivariate probability distributions by factorizing them using local conditional probability distributions (CPDs). This factorization takes advantage of the conditional independence between the random variables. Bayesian networks are especially useful to visually represent high-dimensional probability distributions with a small number of parameters. Bayesian networks can be created from human experts or learned from data.

In this paper, we present a novel software package to use Bayesian networks. The PyBNesian package provides an implementation of Bayesian networks that is easy to use, while also achieving competitive performance. In addition, the package is designed to be fully extensible, so new components can easily be developed which extend the features currently supported by the package. These extensions can also interoperate with the components already implemented in the package. This reduces the amount of code and time needed to create new Bayesian networks developments.

2. Background

Formally, a Bayesian network is a tuple $\mathscr{B} = (\mathscr{G}, \boldsymbol{\theta})$ where $\mathscr{G} = (V, A)$ is a directed acyclic graph (DAG) with a set of nodes $V = \{1, \ldots, n\}$ and a set of arcs $A \subseteq V \times V$. A Bayesian network represents the probability distribution, $P(\mathbf{x})$, of a multivariate random variable $\mathbf{X} = (X_1, \ldots, X_n)$. The set $\boldsymbol{\theta} = \{P(x_i|\mathbf{x}_{Pa_{\mathscr{G}}(i)})\}$ defines a CPD for each node of the graph, where $Pa_{\mathscr{G}}(i)$ is the set of parents of X_i in the graph $\mathscr{G}$. This allows to represent the joint probability distribution $P(\mathbf{x})$ as:

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(x_i|\mathbf{x}_{Pa_{\mathscr{G}}(i)}). \tag{1}$$

Many different types of Bayesian networks have been proposed in the literature, that can support different types of data: discrete, continuous and hybrid data. This is possible using appropriate types of CPDs. The most commmon types of CPDs are conditional probability tables for discrete data, and linear Gaussian CPDs for continuous data. This gives rise to the class of discrete Bayesian networks and Gaussian Bayesian networks [24,8], respectively.

There exists other variants of the Bayesian network model, that are usually needed to solve different types of problems. The conditional Bayesian networks [13] are designed to represent a conditional probability distribution of the form $P(\mathbf{x}|\mathbf{y})$, where the variables $\mathbf{Y}$ (which are always observed in data) are denoted interface variables. The conditional Bayesian networks differ from the normal Bayesian networks because the nodes of the graph $\mathscr{G}$ are split into two different groups $V = V_N \cup V_I$, where V_N (normal nodes) and V_I (interface nodes) are associated to the variables $\mathbf{X}$ and $\mathbf{Y}$, respectively. Then, an interface node is not allowed to have any parents or an associated CPD in $\boldsymbol{\theta}$. PyBNesian also implements conditional Bayesian networks. Note that these networks can be defined using different types of CPDs to support different types

* Corresponding author.
E-mail addresses: datienza@fi.upm.es (D. Atienza), mcbielza@fi.upm.es (C. Bielza), pedro.larranaga@fi.upm.es (P. Larrañaga).

https://doi.org/10.1016/j.neucom.2022.06.112
0925-2312/© 2022 The Author(s). Published by Elsevier B.V.

of data. The utility of conditional Bayesian networks is to represent a conditional distribution $P(\mathbf{x}|\mathbf{y})$ instead of a joint distribution $P(\mathbf{x})$. To the best of our knowledge, no other open-source implementation of conditional Bayesian networks exists.

Another interesting variant is the dynamic Bayesian network model [16,26]. This is especially suited to model temporal/sequential data. The sequential data is denoted as $\mathbf{X}^{(1:T)}$, where the random variable $\mathbf{X}$ is observed from time slice 1 to time slice T. The dynamic Bayesian networks usually make the k-Markovian assumption, where the probability distribution of a sequence slice depends at most on the previous k slices. A dynamic Bayesian network comprises an initial Bayesian network that represents the probability distribution of the first slices k of the sequence, $P(\mathbf{x}^{(1:k)})$, and a transition Bayesian network that represents a distribution $P(\mathbf{x}^{(t)}|\mathbf{x}^{(t-k:t-1)})$. This allows to factorize the probability distribution of $P(\mathbf{x}^{(1:T)})$ as:

$$P\left(\mathbf{x}^{(1:T)}\right) = P\left(\mathbf{x}^{(1:k)}\right)\prod_{t=k+1}^{T} P\left(\mathbf{x}^{(t)}|\mathbf{x}^{(t-k:t-1)}\right). \tag{2}$$

`PyBNesian` implements dynamic Bayesian networks using conditional Bayesian networks as transition networks.

3. Software framework

`PyBNesian` is organized in different modules:

- **Bayesian networks module:** this module implements many different Bayesian network types: discrete Bayesian networks, Gaussian Bayesian networks, conditional linear Gaussian Bayesian networks, kernel density estimation Bayesian networks [10] and semiparametric Bayesian networks [3]. Each Bayesian network type defines different CPDs and appropiate arc restrictions. Also, recall that each Bayesian network type can be used with the different variants of Bayesian networks described in Section 2.
- **Graph module:** this module implements different types of graphs: undirected graphs, directed graphs, partially directed acyclic graphs and DAGs. The implementation of these graphs is tailored to facilitate the development of probabilistic graphical models, and Bayesian networks in particular. In addition, it implements a set of conditional graphs, which provide support for interface and normal nodes and are used by conditional Bayesian networks.
- **Learning module:** this module implements methods for learning Bayesian networks from data. It includes parameter and structure learning. The parameter learning is performed using maximum likelihood estimation. The structure learning can be performed using greedy hill-climbing, PC stable [5], MMPC [28], MMHC [29] and dynamic MMHC [27] (for dynamic Bayesian networks). The behavior of these algorithms can be customized using different learning operators, learning score functions and conditional independence tests.
- **Kernel density estimation module:** this module implements kernel density estimation [22,30,4] and conditional kernel density estimation using Gaussian kernels. In addition, it implements three different bandwidth selection techniques: normal reference rule, Scott's rule and UCV criterion [22,30,4]. This module is implemented using OpenCL 1.2+ whenever possible, so GPUs can accelerate the computation, which is a common trend in other machine learning packages in the last few years, e.g., Tensorflow [1] and PyTorch [18].

Fig. 1 illustrates the high-level organization of these modules. The Bayesian networks module needs the functionality of the graph module to implement Bayesian networks. In addition, the kernel density estimation module is used to implement some nonparametric CPDs and nonparametric/semiparametric Bayesian networks. Finally, the learning module uses the implementation of the Bayesian networks module to incrementally build the learned Bayesian network from data.

In addition, `PyBNesian` allows the creation of extensions that can easily interoperate with the already implemented components in the package. Thus, new components can be created for the following components:

- CPDs.
- Bayesian network types.
- Conditional independence tests.
- Learning score functions.
- Learning operators.
- Bandwidth selection techniques.

A guide on creating extensions for `PyBNesian` is available at https://pybnesian.readthedocs.io/en/latest/extending.html.

`PyBNesian` is implemented almost entirely in C++ for the best possible performance. The package `pybind11` [11] is used to enable fast interoperability between Python and C++. In addition, data transfers between C++ and Python are performed using `Apache Arrow`, which almost completely eliminates the overhead of data copy operations.

4. Related work

There exist other open-source implementations of Bayesian networks. In this section we review some of the most common ones.

- `bnlearn` [23] is probably the most mature project in the Bayesian network community, and implements a comprehensive list of techniques for learning Bayesian networks. It implements most of the Bayesian networks in the state-of-the-art such as discrete Bayesian networks, Gaussian Bayesian networks and conditional linear Gaussian Bayesian networks [14]. Also, it implements many different score functions and conditional independence tests used for learning Bayesian networks models from data. This includes some interesting features, such as support for ordinal data. The `bnlearn` package is available for the R language, and it is partially developed in R and C, to improve its performance. However, it may be complex to add extensions to the package.
- `pcalg` [12,9] is an R package that mainly implements constraint-based learning algorithms and causal inference. It is implemented in R with some parts in C++ to improve its performance.
- `pgmpy` [2] is a Python package of probabilistic graphical models. It includes Bayesian networks, but with full support only for discrete Bayesian networks.
- `pomegranate` [21] is a Python package of probabilitic graphical models, that includes Bayesian networks. However, it only implements discrete Bayesian networks. It is mainly implemented in Cython, to improve performance.
- `LGNpy` [17] is a Python package that implements Gaussian Bayesian networks.

Table 1 compares the functionalities implemented in these libraries. We consider a library to be extensible if it is possible for user-defined code to interoperate with the library algorithms without having to change the library code. Also, we compared the functionalities implemented in three different topics: repre-

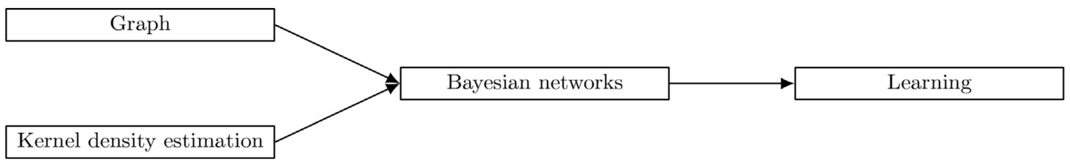


Fig. 1. High-level organization of the library modules. An arrow $s \rightarrow t$ indicates that module t uses module s to implement its functionality.

Table 1
Summary of the functionalities implemented by each Bayesian network (BN) library.

Feature	PyBNesian	bnlearn	pcalg	pgmpy	pomegranate	LGNpy
Extensible design	✓	X	✓[a]	✓	X	X
			Representation support			
Discrete BNs	✓	✓	X	✓	✓	X
Continuous BNs						
Gaussian BNs	✓	✓	X	X	X	✓
Kernel density estimation BNs	✓	X	X	X	X	X
Semiparametric BNs	✓	X	X	X	X	X
Hybrid BNs						
Conditional linear Gaussian BNs	✓	✓	X	X	X	X
Hybrid semiparametric BNs	✓	X	X	X	X	X
Conditional BNs	✓	X	X	X	X	X
Dynamic BNs	✓	X	X	✓	X	X
			Learning support			
Parameter learning						
Maximum likelihood estimate	✓	✓	X	✓	✓	✓
Bayesian estimate	X	✓	X	✓	X	X
Structure learning algorithms						
Greedy hill-climbing	✓	✓	X	✓	✓[b]	X
PC	✓	✓	✓	✓	X	X
Max–min parent children	✓	✓	X	✓	X	X
Max–min hill-climbing	✓	✓	X	✓	X	X
Learning score fucntions						
K2	X	✓	X	✓	X	X
Bayesian Dirichlet equivalent	✓	✓	X	✓	X	X
Bayesian Information Criterion	✓	✓	X	✓	✓	X
Bayesian Gaussian equivalent	✓	✓	X	X	X	X
Predictive log-likelihood	✓	✓	X	X	X	X
Conditional independence tests						
χ^2	✓	✓	X	✓	X	X
Mutual information	✓[c]	✓	X	X	X	X
Partial linear correlation	✓	✓	✓	✓	X	X
			Inference support			
Exact inference	X	X	X	✓	X	✓
Approximate inference	X	✓	X	✓	✓	X

[a] Custom conditional independence tests can be provided to the PC algorithm.
[b] A greedy algorithm is implemented, although it is not a standard greedy hill-climbing.
[c] Two different implementations: a mutual information test which assumes Gaussianity or a nonparametric one that estimates the mutual information with a nearest-neighbors procedure (CMIknn) [20].

Table 2
Properties of the Bayesian networks used for comparison.

Bayesian network	Nodes	Arcs
DIABETES	413	602
ARTH150	107	150

sentation, learning and inference. Representation support is the ability to access and modify the components of different types of Bayesian networks, which are always composed of a DAG and a set of CPDs, as we detailed in Section 2. Note that pcalg does not support the representation of any type of Bayesian network, because it does not implement the representation of CPDs. Learning support is the ability to learn the parameters and the structure of the Bayesian network from data. In addition, some structure learning algorithms make use of score functions and conditional independence tests. In this comparison, we included the most common algorithms, scores and conditional independence tests. In this topic, the most developed library is bnlearn, which implements a large set of learning algorithms, scores and conditional independence tests. However, PyBNesian also includes other learning algorithms (such as DMMHC), learning scores (such as a holdout/cross-validated likelihood) and conditional independence tests (such as RCoT [25]) which are not present in the table. Finally, inference support allows to perform queries, which can be performed using exact or approximate algorithms. Note that PyBNesian does not yet support inference, but it can sample new data from a Bayesian network, which is the basis of many approximate inference algorithms.

We would also like to point out that the comparison of functionalities has many subtleties because the focus of the libraries

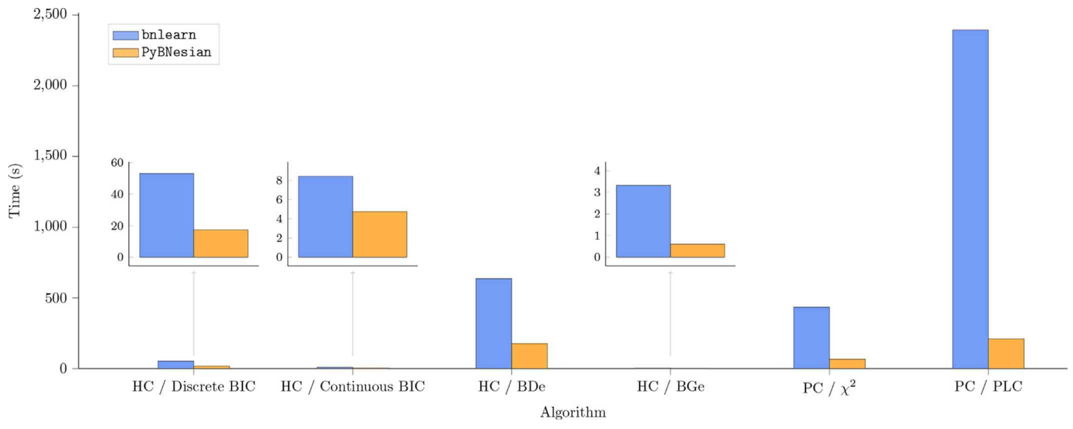


Fig. 2. Mean execution times of different structure learning algorithms for `bnlearn` (blue) and `PyBNesian` (orange). The comparison is performed on a Ubuntu 16.04 machine with 32 GB of RAM and a CPU Intel 6700 K (4 GHz).

are usually not the same. For example, the library `pcalg` is focused on constraint-based learning algorithms. Given this specific focus, it is reasonable that it lacks some other functionalities. On the other hand, there are libraries which a much broader focus, such as `pgmpy` and `pomegranate` that are dedicated to probabilistic graphical models, which include models other than Bayesian networks, such as Markov networks, junction trees, etc. `PyBNesian` in particular focuses on the implementation of all topics related to Bayesian networks, but not other similar models.

5. Performance analysis

In this section we compare the performance of `PyBNesian` and `bnlearn`, the most mature and efficient state-of-the-art implementation. The comparison is performed using the versions 0.4.2 and 4.7 for `PyBNesian` and `bnlearn`, respectively. To compare the performance of both libraries, a large discrete Bayesian network (DIABETES) and a large continuous Bayesian network (ARTH150) were selected from the `bnlearn`'s [23] Bayesian network repository. Table 2 describes the properties of both Bayesian networks.

We independently sampled 20 different datasets of 10,000 instances from each Bayesian network. Then, we measured the time required to learn a Bayesian network model from these datasets with two algorithms implemented in both libraries: greedy hill-climbing and PC. For this, we used the score functions implemented in both libraries: BIC (discrete and continuous), BDe and BGe. For the PC algorithm, we used the following independence tests that are implemented in both libraries: χ^2 test (for discrete data) and partial linear correlation test [6,7] (for continuous data). The execution times for both libraries are shown in Fig. 2. These results show that `PyBNesian` offers a competitive implementation of Bayesian networks compared with a state-of-the-art package. The largest relative difference between the two libraries occurred for the PC algorithm with the partial linear correlation test (`PyBNesian` was approximately 11.35 times faster). In contrast, the smallest relative difference was for the HC with BIC on continuous data (`PyBNesian` was approximately 1.78 times faster).

6. Illustrative examples

An illustrative example is shown in the manual: https://pybnesian.readthedocs.io/en/latest/pybnesian.html. The organization of the package and the description of all the functionalities can be found in the API reference: https://pybnesian.readthedocs.io/en/latest/api.html.

7. Conclusions

In this paper, we presented a novel Python package for Bayesian networks. The package is easy to use and allows creating extensions that can easily interoperate with the already implemented components. Therefore, we hope this package can speed up the development of research on Bayesian network models and its application.

CRediT authorship contribution statement

David Atienza: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing – original draft. **Concha Bielza:** Project administration, Supervision, Resources, Writing – review & editing. **Pedro Larrañaga:** Project administration, Supervision, Resources, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially supported by the Ministry of Education, Culture and Sport through the grant FPU16/00921, by the Spanish Ministry of Science and Innovation through the PID2019-109247 GB-I00 project, and by the BBVA Foundation (2019 Call) through the "Score-based nonstationary temporal Bayesian networks. Applications in climate and neuroscience" (BAYES-CLIMA-NEURO) project.

Appendix A. Required metadata

A.1. Current executable software version

See Table 3.

Table 3
Software metadata.

Nr.	Software metadata description	Please fill in this column
S1	Current software version	0.4.2
S2	Permanent link to executables of this version	https://github.com/davenza/PyBNesian/releases/tag/v0.4.2
S3	Legal Software License	MIT License
S4	Computing platform/ Operating System	Linux, OS X, Windows
S5	Installation requirements & dependencies	Python 3.6–3.9, OpenCL 1.2+, numpy, pybind11, Apache PyArrow
S6	Link to user manual	https://pybnesian.readthedocs.io/en/latest/
S7	Support email for questions	datienza@fi.upm.es

A.2. Current code version

See Table 4.

Table 4
Code metadata.

Nr.	Software metadata description	Please fill in this column
C1	Current code version	0.4.2
C2	Permanent link to code/ repository used of this code version	https://github.com/davenza/PyBNesian/tree/v0.4.2
C3	Legal Code License	MIT License
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Python 3.6–3.9, OpenCL 1.2+
C6	Compilation requirements, operating environments & dependencies	Compatible Python, Clang or GCC with C++17 support, OpenCL 1.2+, numpy, pybind11, Apache PyArrow
C7	Link to developer documentation/manual	https://pybnesian.readthedocs.io/en/latest/
C8	Support email for questions	datienza@fi.upm.es

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, Software available from tensorflow.org, 2015.

[2] A. Ankan, A. Panda, pgmpy: Probabilistic graphical models using Python, in: Proceedings of the 14th Python in Science Conference, 2015, pp. 6–11.

[3] D. Atienza, C. Bielza, P. Larrañaga, Semiparametric Bayesian networks, Inf. Sci. 584 (2022) 564–582.

[4] J.E. Chacón, T. Duong, Multivariate Kernel Smoothing and its Applications, Chapman and Hall/CRC, 2018.

[5] D. Colombo, M.H. Maathuis, Order-independent constraint-based causal structure learning, J. Mach. Learn. Res. 15 (2014) 3921–3962.

[6] R.A. Fisher, Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population, Biometrika 10 (1915) 507–521.

[7] R.A. Fisher, On the probable error of a coefficient of correlation deduced from a small sample, Metron, 1921, pp. 3–32.

[8] D. Geiger, D. Heckerman, Learning Gaussian networks, in: Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence, 1994, pp. 235–243.

[9] A. Hauser, P. Bühlmann, Characterization and greedy learning of interventional Markov equivalence classes of directed acyclic graphs, J. Mach. Learn. Res. 13 (2012) 2409–2464.

[10] R. Hofmann, V. Tresp, Discovering structure in continuous variables using Bayesian networks, in: Proceedings of Advances in Neural Information Processing Systems 8, 1995, pp. 500–506.

[11] W. Jakob, J. Rhinelander, D. Moldovan, pybind11 – seamless operability between C++11 and Python, 2017. https://github.com/pybind/pybind11.

[12] M. Kalisch, M. Mächler, D. Colombo, M.H. Maathuis, P. Bühlmann, Causal inference using graphical models with the R package pcalg, J. Stat. Softw. 47 (2012) 1–26.

[13] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, The MIT Press, 2009.

[14] S.L. Lauritzen, N. Wermuth, Graphical models for associations between variables, some of which are qualitative and some quantitative, Ann. Stat. 17 (1989) 31–57.

[15] M. Maathuis, M. Drton, S. Lauritzen, M. Wainwright, Handbook of Graphical Models, first ed., CRC Press, 2018.

[16] K.P. Murphy, Dynamic Bayesian Networks: Representation, Inference and Learning (Ph.D. thesis), University of California, Berkeley, 2002.

[17] P. Ostwal, 2020. Lgnpy: v1.0.0.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, 2019, pp. 8024–8035.

[19] J. Pearl, Probabilistic Reasoning in Intelligent Systems, Morgan Kaufmann Publishers, 1988.

[20] J. Runge, Conditional independence testing based on a nearest-neighbor estimator of conditional mutual information, in: Proceedings of the 21st International Conference on Artificial Intelligence and Statistics, vol. 84, 2018, pp. 938–947.

[21] J. Schreiber, Pomegranate: Fast and flexible probabilistic modeling in Python, J. Mach. Learn. Res. 18 (2018) 1–6.

[22] D.W. Scott, Multivariate Density Estimation: Theory, Practice, and Visualization, second ed., Wiley, 2015.

[23] M. Scutari, Learning Bayesian networks with the bnlearn R package, J. Stat. Softw. 35 (2010) 1–22.

[24] R.D. Shachter, C.R. Kenley, Gaussian influence diagrams, Manage. Sci. 35 (1989) 527–550.

[25] E.V. Strobl, K. Zhang, S. Visweswaran, Approximate kernel-based conditional independence tests for fast non-parametric causal discovery, J. Causal Inference 7 (2019) 1–24.

[26] G. Trabelsi, New Structure Learning Algorithms and Evaluation Methods for Large Dynamic Bayesian Networks (Ph.D. thesis), Université de Nantes, 2013.

[27] G. Trabelsi, P. Leray, M. Ben Ayed, A.M. Alimi, Dynamic MMHC: A local search algorithm for dynamic Bayesian network structure learning, in: Advances in Intelligent Data Analysis XII, 2013, pp. 392–403.

[28] I. Tsamardinos, C.F. Aliferis, A. Statnikov, Time and sample efficient discovery of Markov blankets and direct causal relations, in: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, pp. 673–678.

[29] I. Tsamardinos, L.E. Brown, C.F. Aliferis, The max-min hill-climbing Bayesian network structure learning algorithm, Mach. Learn. 65 (2006) 31–78.

[30] M.P. Wand, M.C. Jones, Kernel Smoothing, Chapman and Hall/CRC, 1994.

David Atienza received his BS in Computer Science from Universidad de Burgos, Spain, in 2014, his MS in Artificial Intelligence from Universidad Politécnica de Madrid, Spain, in 2016, and his PhD degree in Artificial Intelligence from Universidad Politécnica de Madrid, Spain, in 2021. His research interests include Bayesian networks, nonparametric models, probabilistic graphical models, density estimation, anomaly detection and real applications to the Industry 4.0 paradigm.

Concha Bielza received her M.S. degree in Mathematics from Universidad Complutense de Madrid, in 1989 and her Ph.D. degree in Computer Science from Universidad Politécnica de Madrid, in 1996. She is currently a Full Professor of Statistics and Operations Research with the Universidad Politécnica de Madrid. Her research interests are primarily in the areas of probabilistic graphical models, decision analysis, classification models, and real applications. She has published more than 150 papers in impact factor journals and has supervised 20 PhD theses. She was awarded the 2014 UPM Research Prize and the 2020 machine learning award of Amity University (India).

Pedro Larrañaga is Full Professor in Computer Science and Artificial Intelligence at the Universidad Politécnica de Madrid. He received his MSc degree in Mathematics from the University of Valladolid and his PhD degree in Computer Science from the University of the Basque Country. He has published more than 200 papers in impact factor journals and has supervised 33 PhD theses. He is fellow of the European Association for Artificial Intelligence and of the Academia Europea. He was awarded the 2013 Spanish National Prize in Computer Science and the prize of the Spanish Association for Artificial Intelligence in 2018.