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

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 \( \mathcal{G} = (\mathcal{V}, \mathcal{A}) \) where \( \mathcal{V} \) 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 \( \theta = \{ P(X_i|\mathbf{X}_{Pa(i)}) \} \) defines a CPD for each node of the graph, where \( P_{Pa(i)} \) is the set of parents of \( X_i \) in the graph \( \mathcal{G} \). This allows to represent the joint probability distribution \( P(\mathbf{x}) \) as:

\[
P(\mathbf{x}) = \prod_{i=1}^{n} P(X_i|\mathbf{X}_{Pa(i)}).
\]

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 common 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 \( \mathcal{G} \) 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 \( \theta \). PyBNesian also implements conditional Bayesian networks. Note that these networks can be defined using different types of CPDs to support different types of data.
of data. The utility of conditional Bayesian networks is to represent a conditional distribution $P(x|y)$ instead of a joint distribution $P(x)$. 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 $X^{(t)}$, where the random variable $X$ 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, $P(x^{(1:k)})$, and a transition Bayesian network that represents a distribution $P(x^{(k+1:T)}|x^{(1:k-1)})$. This allows to factorize the probability distribution of $P(x^{(1:T)})$ as:

$$P(x^{(1:T)}) = P(x^{(1:k)}) \prod_{t=k+1}^{T} P(x^{(t)}|x^{(t-k-1)})$$

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 appropriate 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 non-parametric 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.


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 probabilistic graphical models, that includes Bayesian networks. However, it only implements discrete Bayesian networks. It is mainly implemented in Cython, to improve performance.

- **LGPy** [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-
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. 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

![High-level organization of the library modules. An arrow s → t indicates that module t uses module s to implement its functionality.](image-url)
are usually not the same. For example, the library \texttt{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 \texttt{pgmpy} and \texttt{pomegranate} that are dedicated to probabilistic graphical models, which include models other than Bayesian networks, such as Markov networks, junction trees, etc. \texttt{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 \texttt{PyBNesian} and \texttt{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 \texttt{PyBNesian} and \texttt{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 \texttt{bnlearn}'s\cite{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: $\chi^2$ test (for discrete data) and partial linear correlation test \cite{6,7} (for continuous data). The execution times for both libraries are shown in Fig. 2. These results show that \texttt{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 ($\texttt{PyBNesian}$ was approximately 11.35 times faster). In contrast, the smallest relative difference was for the HC with BIC on continuous data ($\texttt{PyBNesian}$ was approximately 1.78 times faster).

6. Illustrative examples

An illustrative example is shown in the manual: \url{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: \url{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

\textbf{David Atienza}: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing – original draft. \textbf{Concha Bielza}: Project administration, Supervision, Resources, Writing – review & editing. \textbf{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.

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

A.2. Current code version

See Table 4.

Table 4
Code metadata.

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Software metadata description</th>
<th>Please fill in this column</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Current code version</td>
<td>0.4.2</td>
</tr>
<tr>
<td>C2</td>
<td>Permanent link to code/ repository of this code version</td>
<td><a href="https://github.com/davenza/PyBNesian/tree/v0.4.2">https://github.com/davenza/PyBNesian/tree/v0.4.2</a></td>
</tr>
<tr>
<td>C3</td>
<td>Legal Code License</td>
<td>MIT License</td>
</tr>
<tr>
<td>C4</td>
<td>Code versioning system used</td>
<td>git</td>
</tr>
<tr>
<td>C5</td>
<td>Software code languages, tools, and services used</td>
<td>Python 3.6–3.9, OpenCL 1.2+</td>
</tr>
<tr>
<td>C6</td>
<td>Compilation requirements, operating environments &amp; dependencies</td>
<td>Compatible Python, Clang or GCC with C++17 support, OpenCL 1.2+, numPy, pybind11, Apache PyArrow</td>
</tr>
<tr>
<td>C8</td>
<td>Support email for questions</td>
<td><a href="mailto:datienza@ifi.upm.es">datienza@ifi.upm.es</a></td>
</tr>
</tbody>
</table>

References


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.