Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Multivariate time-series modelling and forecasting with high-order dynamic Bayesian networks applied in industrial settings

PhD THESIS

Author

David Quesada López
MSc Artificial Intelligence

2023

Departamento de inteligencia artificial

Escuela Técnica Superior de Ingenieros Informáticos

Multivariate time-series modelling and forecasting with high-order dynamic Bayesian networks applied in industrial settings

Author

David Quesada López
MSc Artificial Intelligence

PhD supervisors

Concha Bielza
PhD Computer Science

Pedro Larrañaga
PhD Computer Science

2023

Thesis Committee

President:

Member:

Member:

Member:

Secretary:

Agradecimientos

Cuando miras los agradecimientos de las tesis doctorales, cada persona los escribe a su manera. En mi caso, me gustaría usar los agradecimientos de esta tesis para recopilar los nombres de las personas que me han acompañado hasta este punto. Quiero dejar todos estos nombres escritos junto al mío para la posteridad. Que quede constancia de que fueron, son y serán parte de mi vida.

En primer lugar a mis directores de tesis, Concha Bielza y Pedro Larrañaga, que me dieron la oportunidad de tomar este camino y sin la ayuda de los cuales este trabajo habría sido imposible. Dentro del ámbito de la enseñanza, me gustaría recordar también a los profesores del IES Vega del Jarama, en particular Orlando, Miguel Recio, Sonsoles y María Ángeles Polanco, y a los de la Universidad Complutense de Madrid, en particular a Rafael Caballero Roldán. No me olvido de cómo me marcaron.

A mi familia de Madrid: mi madre María Antonia López, mi padre Cristóbal Quesada, mi hermano Miguel Ángel y mi tita Ana María. Mi familia de Jaén: mi tita Paqui, mi tita Maria José, mi tito Juan y mis primos José, Juan, Javi e Ildefonso, junto con mis abuelos Juan, María, Ildefonso y Encarna. Por último mi familia del país donde sea que se hayan decidido ir ahora: mi hermana Ana Belén, su marido François, y mis sobrino y sobrinas Abel, Amaya y Alba. Os quiero de corazón.

A mis amigos, los de siempre: Diego, Pacheco, Óscar, Iñaki, Adrián, Romila, Carlos, Nacho, Iván y Alberto, y los del *TeamSpeak*: Rayco, Adri, Marco, Waka, Ale, Alexito y Javi. No me olvido tampoco de la muchacha Noelia ni del becario Denis, junto con Paun, Stas, Rafa, Ozek, Ana y Bog. No podría haber encontrado gente mejor de la que rodearme ni aposta. Os quiero a todos.

A mis compañeros del CIG, pilar fundamental para poder sobrellevar el camino del doctorado: Gabriel, Vicente, Carlos, Laura, Fer, Irene, David, Bojan, Alaia, Esteban, Mario, Carlos Li y Enrique. No me olvido tampoco de Santiago Fernández Prieto, Marco y Pedro, que no son del CIG, pero también tabajamos y charlamos juntos. Levantarme por las mañanas se hacía llevadero sabiendo que estas personas iban a estar conmigo durante el día.

También quiero agradecer la financiación parcial de los siguientes proyectos e instituciones: el proyecto PID2019-109247GB-I00 del Ministerio Español de Ciencia, Innovación y Universidades, el proyecto "Score-based nonstationary temporal Bayesian networks. Applications in climate and neuroscience" (BAYES-CLIMA-NEURO) de la fundación BBVA (2019 call), el proyecto "MadridDataSpace4Pandemics-CM" (REACT-EU) de la Comunidad de Madrid y el proyecto "Outcome prediction and treatment efficiency in patients hospitalized with Covid-19 in Madrid: A Bayesian network approach" de la Fundación BBVA (2020 Call).

Abstract

With the proliferation in recent years of industrial systems monitored by sensors, there has been a drastic increase in the volume of data recovered that characterizes these systems. Specifically, the continuous recovery of data generates multivariate time series, which have specific characteristics and require special treatment. Additionally, industrial systems usually entail complex underlying processes, and these recovered time series offer valuable information about these processes and about the relationships between the variables in our system.

One model that allows us to work with time series data and that is able to offer us insights about the underlying process is dynamic Bayesian networks. However, the dynamic Bayesian network architecture is very restrictive, and cannot be freely applied to a wide range of industrial processes. Time series do not necessarily follow autoregressive order 1 and the relationships between their variables can come from non-linear physical processes. As opposed to this, dynamic Bayesian networks for continuous data usually assume normality and autoregressive order 1. On top of that, they are inherently lineal models in the case of linear Gaussian networks. These restrictions cause that dynamic Bayesian networks incur in severe inaccuracies when applied directly to industrial problems. In this dissertation, we focus on extending the dynamic Bayesian network model so that it can be applied as a general purpose model in industrial problems.

As our first contribution, we explore higher autoregressive orders of dynamic Bayesian networks to apply them for long-term forecasting of the temperature inside an industrial furnace. We create several dynamic Bayesian network models that are able to assimilate the tendency of the time series better as we increase the Markovian order of the networks up to a certain point. Thus, we are able to create a dynamic Bayesian network model that accurately forecasts the temperature inside an industrial furnace over spans of thousands of hours.

To fix the issue of increasing complexity when learning higher order networks, we propose a particle swarm algorithm to move through the space of possible structures. Firstly, we restrict the search to transition networks, which are graphs that only allow arcs to the most recent instant. Secondly, we define a natural number encoding and operators that are invariant to the Markovian order of the network in terms of particle size. Regardless of the desired Markovian order of the network, the size of the vectors that encode the graphs remains constant, increasing the efficiency of the algorithm for higher orders.

To make dynamic Bayesian network models more flexible in the presence of non-linear systems, we propose a hybrid model between model trees and dynamic Bayesian networks. This hybrid approach fits a classification and regression tree to some dataset and then fits a dynamic Bayesian network model in each of its leaf nodes. With this structure, we can use the tree to classify each new instance into one of the leaf dynamic Bayesian network models and

use the most appropriate one to forecast the next instant. Afterwards, the forecasted values are processed again with the tree and a new dynamic Bayesian network model is selected, dynamically choosing the most fitting network for each instant over time. This allows us to perform non-linear inference via this piecewise regression mechanism.

As a last application of dynamic Bayesian networks, we have used them to forecast future states in a classification problem through time of the state of patients infected by the COVID-19, using neural networks as the classifier model.

Finally, we compiled and distributed the dynamic Bayesian network framework and all of our contributions inside an open source R package that allows a streamlined application of dynamic Bayesian networks in any industrial and real world settings.

Resumen

Con la proliferación en estos últimos años de sistemas industriales monitorizados por sensores, ha habido un incremento drástico en el volumen de datos recuperados que caracterizan estos sistemas. En particular, el guardado continuo de datos genera series temporales multivariantes, las cuales tienen características específicas y requieren un tratamiento especial. Además, los sistemas industriales normalmente contienen procesos complejos subyacentes, y estas series temporales ofrecen información valiosa sobre estos procesos y sobre cómo se relacionan las variables de nuestro sistema entre ellas.

Un modelo que nos permite tratar datos de series temporales y que es capaz de ofrecernos información sobre el proceso subyacente son las redes bayesianas dinámicas. Sin embargo, la arquitectura de estas redes es muy restrictiva, y no puede ser aplicada directamente a muchos procesos industriales. Las series temporales no necesariamente siguen un orden autorregresivo unitario y pueden tener variables cuyas relaciones vienen definidas por procesos físicos no lineales. Sin embargo, las redes bayesianas dinámicas para datos continuos suelen asumir normalidad en los datos y orden autorregresivo 1, siendo modelos lineales por definición en el caso de las redes lineales gaussianas. Estas restricciones provocan que la precisión de las redes bayesianas dinámicas disminuya cuando se aplican directamente a problemas industriales. En esta tesis vamos a extender el modelo de redes bayesianas dinámicas para que pueda usarse como un modelo de propósito general en problemas industriales.

En primer lugar, relajamos la restricción del orden autorregresivo de las redes bayesianas dinámicas para aplicarlas a la predicción a largo plazo de la temperatura dentro de un horno industrial. Para ello, creamos varios modelos de redes bayesianas dinámicas que son capaces de modelar mejor la tendencia de las series temporales según aumentamos el orden markoviano de las redes hasta un cierto punto. Con esto, generamos un modelo de redes bayesianas dinámicas que puede hacer predicciones precisas de la temperatura dentro del horno industrial a lo largo de periodos de miles de horas.

Para solucionar el aumento de complejidad del aprendizaje de estructuras de alto orden, proponemos un algoritmo de enjambre de partículas para movernos en el espacio de posibles grafos. Primero, restringimos la búsqueda a grafos que únicamente permiten arcos dirigidos al instante más reciente. Después, definimos unos operadores y una codificación por medio de vectores de números naturales que son independientes del orden markoviano de la red en términos de tamaño de las partículas. Así, el tamaño de los vectores que codifican las estructuras se mantiene constante sin importar el orden markoviano, mejorando la eficiencia del algoritmo para órdenes altos.

Para hacer los modelos de redes bayesinas dinámicas más flexibles en presencia de sistemas no

lineales, proponemos un híbrido entre model trees y redes bayesianas dinámicas. Este híbrido ajusta un árbol de clasificación y regresión a un conjunto de datos y después ajusta un modelo de red bayesiana dinámica en cada una de las hojas del árbol. Con esta estructura, podemos utilizar el árbol para asignar cada nueva instancia a un modelo de una hoja del mismo y así usar el modelo de red bayesiana dinámica apropiado para predecir el siguiente instante. Esto nos permite realizar inferencia no lineal utilizando este mecanismo de regresión a trozos.

Como último caso de uso, hemos aplicado las redes bayesianas dinámicas a un problema de clasificación a lo largo del tiempo del estado de pacientes que sufren infecciones de COVID-19, usando redes neuronales combinadas como modelo clasificador.

Finalmente, hemos compilado y distribuido el modelo base de red bayesiana dinámica y todas nuestras contribuciones en un paquete de código abierto en R que permite una aplicación directa de las redes bayesianas dinámicas en casos industriales.

Contents

Contents xiv

List of figures xx

List of tables xxii

Acronyms xxiv

Notation xxv

I INTRODUCTION 1

1 Introduction 3
- 1.1 Hypotheses and objectives . . . 4
- 1.2 Document organization . . . 4

II BACKGROUND 7

2 Bayesian networks 9
- 2.1 Introduction . . . 9
- 2.2 Bayesian networks . . . 10
  - 2.2.1 Structure . . . 10
  - 2.2.2 Parameters . . . 12
- 2.3 Learning . . . 14
  - 2.3.1 Parameter estimation . . . 15
  - 2.3.2 Structure learning . . . 17
- 2.4 Inference . . . 21
  - 2.4.1 Exact inference . . . 21
  - 2.4.2 Approximate inference . . . 23

3 Dynamic Bayesian networks 25
- 3.1 Introduction . . . 25

3.2 time-series characteristics . . . 26
3.2.1 Trend and seasonality . . . 28
3.3 Dynamic Bayesian network definition . . . 30
3.4 Learning . . . 32
3.4.1 Score-based methods . . . 33
3.4.2 Constraint-based methods . . . 33
3.4.3 Hybrid methods . . . 34
3.5 Inference . . . 34

III CONTRIBUTIONS 37

4 Long-term forecasting of multivariate TS in industrial furnaces with DBNs 39
4.1 Introduction . . . 39
4.2 Forecasting the temperature of an industrial furnace . . . 41
4.3 Preprocessing . . . 42
4.3.1 Characteristics of the data . . . 43
4.3.2 Cleaning and imputation . . . 44
4.3.3 Cycle treatment . . . 45
4.3.4 Feature selection . . . 46
4.4 Methods and results . . . 47
4.4.1 Structure learning . . . 48
4.4.2 DBN models . . . 48
4.4.3 Convolutional recurrent neural networks . . . 49
4.4.4 Results . . . 50
4.5 Conclusions and future work . . . 55

5 Structure learning of high-order DBNs via PSO with order invariant encoding 57
5.1 Introduction . . . 57
5.2 Background . . . 58
5.2.1 High-order dynamic Bayesian networks . . . 58
5.2.2 Particle swarm structure learning . . . 59
5.3 Encoding and operators . . . 60
5.3.1 Natural vector encoding . . . 61
5.3.2 Position and velocity operators . . . 61
5.4 Results . . . 64
5.4.1 Implementation . . . 64
5.4.2 Experimental comparison . . . 64
5.5 Conclusions and future work . . . 67

6 Piecewise forecasting of nonlinear TS with model tree DBNs **69**
6.1 Introduction . . . 69
6.2 Tree-based dynamic Bayesian networks . . . 71
6.2.1 Hybrid definition: mtDBN . . . 71
6.2.2 Forecasting . . . 75
6.3 Experimental results . . . 75
6.3.1 Simulated nonlinear problem: Fouling phenomenon . . . 76
6.3.2 Real data application: Electrical motor . . . 82
6.3.3 Real data application: TSEC stock index . . . 88
6.4 Conclusions and future work . . . 91

7 Classifying the evolution of COVID-19 severity on patients by coupling DBNs and NNs **93**
7.1 Introduction . . . 93
7.2 Combining DBNs and static classifiers . . . 95
7.2.1 Forecasting the state vector . . . 95
7.2.2 Classifying critical values . . . 96
7.3 Experimental results . . . 96
7.3.1 Preprocessing . . . 97
7.3.2 Classification results . . . 99
7.4 Conclusions and future work . . . 103

8 dbnR: Gaussian DBN learning and inference in R **105**
8.1 Introduction . . . 105
8.2 BN definition in dbnR . . . 106
8.3 Structure learning . . . 107
8.3.1 Dynamic max-min hill-climbing . . . 109
8.3.2 Binary encoding PSO . . . 110
8.3.3 Natural number invariant encoding PSO . . . 111
8.3.4 Structure visualization tool . . . 111
8.4 Inference and forecasting . . . 111
8.4.1 Exact inference . . . 112
8.4.2 Forecasting TS . . . 113
8.4.3 Smoothing . . . 113
8.5 Pipeline and main functions . . . 114
8.6 Example application: Sample motor dataset . . . 116
8.7 Conclusions and future work . . . 127

IV CONCLUSIONS **129**

9 Conclusions and future work **131**

9.1 Summary of contributions 131
9.2 List of publications 132
9.3 Future work 133

V APPENDICES **135**

A ODE system for the fouling phenomenon **137**

Bibliography **141**

List of figures

2.1 Examples of d-separation of the sets $\mathbf{X}_1$ and $\mathbf{X}_2$ given $\mathbf{X}_3$. From left to right, there are sequential, divergent and convergent scenarios. In the first two cases, $\mathbf{X}_1$ and $\mathbf{X}_2$ are d-separated given $\mathbf{X}_3$, but they are not in the convergent case because X_b is part of $\mathbf{X}_3$. 11

3.1 Example of a TS reflecting the production of olive oil in Spain from October 2018 to April 2019. In this case, the values of the production in kilotonnes were recorded with a monthly frequency. 26

3.2 Example of a stochastic process $\mathbf{X}$ generating two TS $\mathbf{x}$ and $\mathbf{x}'$. These TS are just instances of a stochastic process where the random variables took specific values. 27

3.3 The TS corresponding to the olive oil production in Spain from 2015 to 2020. Each one of these TS can be considered a particular instance of the underlying stochastic process. 27

3.4 The resulting olive oil production time-series after performing lag-1 and lag-12 operators to remove both the trend and the seasonality in the data. We can see that the obtained residuals are close to stationary, which makes them suitable for being fitted by zero-mean models. 29

3.5 Representation of a DBN in relation with some stochastic process $\mathbf{S}$ and a TS $\mathbf{s}$ generated from it. We can model the relationships of random variables within the same time slice with intra-slice arcs similarly to a static BN, but we need the dotted inter-slice arcs to represent the autoregressive component of TS. 31

3.6 Schematic representation of the sliding window mechanism in an inference step. The new $\hat{\mathbf{x}}^{t+1}$ is predicted with the DBN model and is introduced into the state vector, removing the oldest $\mathbf{x}^0$. 35

4.1 Schematic representation of the fouling effect inside a tube in the furnace. The fouling layer appears over time and has to be removed when the working conditions degrade past a temperature that would melt the tube walls. . . . 42

4.2 Illustration of the layout of the four different sections (S1, S2, S3 and S4) inside the furnace. When one needs to be shut down to perform cleaning operations, the others remain operational. 43

4.3 The heat supplied to the tubes during a cycle in one section. The periods in which cleaning is being performed in another section are shown in red. One of these cleanings severely affects the temperature of the tube. 44

4.4 A segment of the temperature TS on the tube walls. Three different cycles are shown in different colours. The black dotted lines show the limits of each cycle. If we split the multivariate TS into these independent cycles, we obtain several i.i.d. instances of the same underlying process. 46

4.5 Adapting a dataset D to learn a Markovian order one DBN. All columns are shifted with $O = 1$ and added to the dataset. The grey row contains missing values and should be deleted. 47

4.6 One of the clusters obtained from the hierarchical clustering. A comparison between the two most distant series in the cluster is shown. Variables inside the same cluster are mostly redundant TS with some variations in scale. . . 48

4.7 As the Markovian order of the trained networks increases, the error in the forecasts decreases to a point where the networks start overfitting and accumulating error. 51

4.8 Resulting predictions of a Markovian order four Gaussian DBN for two cycles with different characteristics. The black line represents the real temperature TS while the red curve is the estimation of the DBN. Note how the predictions adapt to the observed tendency in the initial evidence provided. The predictions are extended to the full length of the cycles, with the first being an example of a cycle that ends early. 51

4.9 Average absolute error over time of forecasting with the Markovian order four network. The absolute error is low in the first days and then increases due to the cycles individual behaviours. As the ending point of the cycles approaches, the error decreases. 52

4.10 A screenshot of the visualization tool included in our package showing the parent and child nodes of the objective variable (cyan) in the Markovian order four DBN. 53

4.11 Results of giving the CRNN the first 24 hours of a cycle as inputs and forecasting the next 50 hours. The forecasting returns an accurate profile in the short term. As opposed to the DBN, this model is normalized with the mean and variance of the training dataset. 54

5.1 An example of a Markovian order 2 transition network with three nodes per time slice. Only arcs directed to t_0 from earlier time slices are allowed. . . . 59

5.2 Representation of a position natural vector on the left and its equivalent transition network on the right. The transition network has three nodes per time slice and a Markovian order 2 for simplicity and clarity. Notice how increasing the order, thus adding many possible nodes and arcs, only implies bigger natural numbers in the vector, but it does not increase its length. For example, increasing the order up to 3 in the figure would only mean that natural numbers up to 7 can now appear in the vector. 62

5.3 An example of adding a position and a velocity encoding a network with two receiving variables X_0^0 and X_1^0 and maximum Markovian order 2 for simplicity. All the operations are performed bitwise on the natural numbers of all vectors. 62

5.4 Pipeline of the PSO algorithm. The particles move applying the updating rules described in Section 5.2.2 and the operators described in Section 5.3.2. The position with the best fitness is translated into its DBN equivalent and returned as the best solution found. 64

5.5 Execution time in minutes and percentage of recovered arcs of both PSO algorithms with 300 particles, 50 iterations and 20 receiving nodes when learning transition networks as we increase the Markovian order. 66

6.1 Schematic representation of the mtDBN architecture. The tree structure is initially used to divide the original dataset $\mathbf{D}$ into several datasets $\mathbf{D}_{leaf}$. Each one of these datasets is then used to fit a DBN model, which will be stored in the model vector $\mathbf{M}$. Afterwards, the tree structure will be used to classify new instances according to the appropriate DBN model inside $\mathbf{M}$. . . 72

6.2 Representation of the mtDBN forecasting a single instant. The state vector is classified by the tree, and the correspondent DBN leaf and parent models are used to predict the next state vector. 76

6.3 Example of the tree structure of a homogeneous multivariate mtDBN model. The resulting tree has five different leaf DBN nodes, represented in various colours. The splitting rules are shown in the branches, and the average value of the objective temperature T_1 and the number of instances N per node are shown inside the tree nodes from the initial 96,700 instances in the training dataset at the root of the tree. From the tree, we can identify that the m_c variable reduces the sum of squares the most in the different scenarios. From the root node, lower quantities of fuel m_c administered to the furnace will result in slower processes with lower T_1 and vice versa. In the case of extremely high T_1, the most severe cases are defined by a high flow of fluid Q_{in} entering the system. 80

6.4 Example of the DBN structure of the homogeneous multivariate mtDBN model. The objective variable T_1 at the present instant is represented as the blue node, and all the parent nodes in red represent those variables at the previous instant. With these parent nodes, we can see the autoregressive component and the most relevant variables in the inference of T_1, i.e., S_c, T_2, m_c, Q_{in} and vol. 81
6.5 Example of predicting the fluid temperature T_1 of a 100 instant trace using the homogeneous multivariate mtDBN. The model used for forecasting is changed three times during forecasting based on the predicted state of the system at each instant. It is important to note that we only show the fluid temperature in the figure, but all variables in the system are jointly forecasted simultaneously to obtain the state of the system at the next instant. 82
6.6 Example of the objective temperature in two different sessions in our dataset. Session 10 is rather irregular, with several interventions drastically changing the tendency of the series. Session 12 is a more common case in our data, with only one significant intervention after 6000 seconds. 83
6.7 Example of the alignment of two TS sessions of the objective variable in the original and reduced datasets. The alignment being almost a straight diagonal line indicates that no displacement in time is seen and that no drastic jumps in values are present. 84
6.8 Example of the tree structure of the non-homogeneous univariate mtDBN model in the case of the electric motor. In this case, the initial cuts are made around the yoke and tooth temperatures inside the motor. It makes sense that temperatures at other points of the motor are useful in determining our objective temperature pm, and for very high temperatures, the electric current i_d defines the two most extreme groups. 86
6.9 Example of the objective variable pm (in blue) and its parent nodes inside the DBN structure in the non-homogeneous mtDBN model of Markovian order 2. Apart from the autoregressive component of pm on itself, we can see that two variables are also used in the tree structure, *stator tooth* and i_d, and two other temperatures, *stator winding* and *ambient*, are used by the DBN model to predict pm. Intra-slice arcs are not permitted due to the structure learning algorithm used. 87
6.10 Heatmap showing the correlation between the variables in the TSEC stock dataset. Most of the variables have a correlation very close to 1, except for the volume of daily transactions. 89
6.11 Tree structure of the homogeneous univariate mtDBN model. The splits define three cases depending on the highest value that the stock index took that day. From left to right, the leaf nodes define the cases where the stock index was lower than the mean, the cases where it was around the mean and the cases where it was higher than the mean. 91

6.12 Example of the DBN structure of the homogeneous univariate mtDBN model in the stock dataset. The opening index value of the next day is obtained taking into account the lowest values of the last two days and the closing value of the previous day. 92

7.1 Schematic representation of the classifier-DBN framework. After obtaining a state vector $\mathbf{s}_0$ from a patient, we can use it to forecast the next t state vectors with the DBN model and check if they are critical at each time t with our static classifier. 97
7.2 Histogram with the number of instances per patient greater than 1 in the dataset. Inside the last bin we have grouped all the patients with 10 or more instances. A higher number of instances indicates a longer stay in the hospital and as such a more severe case of COVID-19, which was far less common than a mild case in the sixth wave. 98
7.3 Structure of the NN model used in the experiments. 99
7.4 Classification results of the neural network model as we feed it with the state vectors further ahead in time with the DBN model. The classification performance of the neural network improves monotonically by combining it with the DBN forecastings. 101
7.5 Subset of relevant variables to the forecasting of maximum oxygen saturation (*Max_02_sat* in light blue) in the DBN model. The initial and maximum oxygen saturation variables from the last instant (*Initial_02_sat* and *Max_02_sat* in red) affect the calculation of the next 4 hours maximum oxygen saturation value. Other variables like body temperature *Min_body_temp*, systolic *Min_SBP* and diastolic *Min_DBP* blood pressures and heart rate *Min_heart_rate* also influence this value in the forecast. 102

8.1 Example of transforming a dataset with two variables X_1 and X_2 into several variables depending on the desired Markovian order. The rows in the original data are ordered from the oldest recorded values, $X_1 = 3$ and $X_2 = -1$, to the newest. The grey rows contain missing values and should be deleted. . . 108
8.2 Example of the visualization of the structure of both a BN (left) and a DBN (right). The nodes on both networks can be clicked on to be highlighted and dragged to reposition them. 112
8.3 Plot returned by the `forecast_ts` function after forecasting 20 instances of a TS. The *pm* variable represents the magnet temperature inside an electric motor. The black line represents the original values of the TS, and the red line represents the forecasting. Only the values of the variables at the initial instant 0 are known as evidence to the DBN. 114
8.4 Diagram with the main workflow of the **dbnR** package. We start by preparing a dataset; then, we learn the DBN structure, learn its parameters and typically perform forecasting. Optionally, the network structure can also be plotted. . 115

8.5 Plot of the predictions (red) returned by the `predict_dt` function. All the predictions are performed to horizon 1 using the last instant as evidence. They are deceivingly accurate because the last instant is always used as evidence for the next prediction. If looking closely, it can be seen that large changes in the profile of the curve are not properly predicted by the DBN until one instant later when evidence of these changes is provided to the model. 123
8.6 Plot obtained from forecasting 30 instances with a DBN model using only the evidence from the initial point at $t = 0$. 124
8.7 A comparison between the plotted forecasts of fixing the `motor_speed` variable to -1.4 (left) and progressively increasing it from -1.4 to -1.1 (right). The real values of the TS are represented by black lines, and the red lines represent the forecasts. The effects of both actions can be clearly seen in the profile of the predictions. . . . 126

A.1 Example of a 100 time instant trace created by the simulation. Several time series with data and some noise of each variable are generated. 140

List of tables

3.1 Resulting means and standard deviations from the olive oil production TS after performing lag-1 and lag-12 operators. 30

4.1 DBN forecasting results 52
4.2 CRNN forecasting results 53

5.1 Results for low-order networks 65
5.2 Results for high-order networks 66

6.1 Results in terms of the MAE, MAPE, training and execution time of the models for the fouling experiment. 78
6.2 Resulting p-value of the Wilcoxon rank-sum tests for the forecasts of the different hybrid models in comparison with the baseline DBN model for the fouling experiment. 79
6.3 Resulting p-value of the Wilcoxon rank-sum tests in the multiple comparisons between the baseline DBN model, the homogeneous multivariate mtDBN, the LSTM and the HFCM model. All the tests reject the null hypothesis of equal performance in MAE. 79
6.4 Results in terms of the MAE, MAPE, training time and execution time of the models for the motor dataset. 85
6.5 Resulting p-value of the Wilcoxon rank-sum tests for the electric motor dataset with respect to the regular DBN model. 85
6.6 Results of the Wilcoxon rank-sum tests in the multiple comparisons for the motor experiment. Similar to the synthetic case, all the pairwise tests reject the null hypothesis of equal performance in MAE. 85
6.7 Results in terms of the MAE, MAPE, training time and execution time of the models for the stock index dataset. 89
6.8 Resulting p-value of the Wilcoxon rank-sum tests for the stock index dataset with respect to the regular DBN model. 89
6.9 Results of the Wilcoxon rank-sum tests in the multiple comparisons for the stock index experiment. The pairwise tests show that the LSTM and HFCM models do not present significant differences in their results. 90

7.1 Mean results in terms of the accuracy, g-mean score, training and execution time of the models on average for all the experiments. It is worth noting that training time includes optimization of parameters, which involves the creation of multiple models to evaluate different configurations. 100

8.1 Overview of all the exported functions in the **dbnR** package ordered by the type of task they perform. 115

Acronyms

AIC Akaike information criterion

ARIMA Autoregressive integrated moving average

BGe Bayesian Gaussian equivalent

BIC Bayesian information criterion

BN Bayesian network

CART Classification and regression tree

CRAN Comprehensive R archive network

CPD Conditional probability distribution

CPT Conditional probability table

CRNN Convolutional recurrent neural network

DAG Directed acyclic graph

DBN Dynamic Bayesian network

GDBN Gaussian dynamic Bayesian network

DMMHC Dynamic max-min hill-climbing

XGBoost Extreme gradient boosting

g-mean Geometric mean

HODBN High-order dynamic Bayesian network

HFCM High-order fuzzy cognitive maps

HC Hill-climbing

HCSP Hill-climbing super-parent

i.i.d. Independent and identically distributed

I-map Independence map

JPD Joint probability distribution

LL Log-likelihood

LSTM Long short-term memory

MAPE Mean absolute percentage error

MB Markov blanket

MMHC Max-min hill-climbing

MLE Maximum likelihood estimation

MAE Mean absolute error

MI Mutual information

NN Neural network

ODE Ordinary differential equation

PSO Particle swarm optimization

RNN Recurrent neural network

SD Standard deviation

SVM Support vector machine

TSEC Taiwan Stock Exchange Corporation

TS Time-series

Notations

X, Y, Z, ...	random variables
$\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$, ...	sets of random variables
$\mathbf{X}^t$	set of random variables at time slice t
x, y, z, ...	assignments to random variables
$\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, ...	assignments to sets of random variables
$\Omega_{\mathbf{X}}$	set of all possible joint assignments to $\mathbf{X}$
$\mathbf{X} \perp \mathbf{Y}$	independence of $\mathbf{X}$ and $\mathbf{Y}$
$\mathbf{X} \perp \mathbf{Y} \vert \mathbf{Z}$	conditional independence of $\mathbf{X}$ and $\mathbf{Y}$ given $\mathbf{Z}$
$P(\mathbf{X})$	probability distribution of $\mathbf{X}$
$P(\mathbf{x})$	shorthand for $P(\mathbf{X} = \mathbf{x})$
$P(\mathbf{X}, \mathbf{Y})$	joint probability distribution of $\mathbf{X}$ and $\mathbf{Y}$
$P(\mathbf{X} \vert \mathbf{Y})$	conditional probability distribution of $\mathbf{X}$ given $\mathbf{Y}$
$\mathcal{G}$	the graph structure of a Bayesian network
$\boldsymbol{\theta}$	parameters of a Bayesian network
$\hat{\boldsymbol{\theta}}$	maximum likelihood estimate of the parameters of a Bayesian network
$\mathbf{Pa}(X_i)$	parent variables of variable X_i in $\mathcal{G}$
$\mathbf{Pa}^g(X_i)$	continuous parent variables of variable X_i in $\mathcal{G}$
$\beta_0, \ldots, \beta_k$	parameters that define the effect that parent nodes $\mathbf{Pa}^g(X_i)$ have on the mean of X_i in a linear Gaussian Bayesian network $\mathcal{G}$
$\vert\mathbf{Pa}(X_i)\vert$	number of parents of variable X_i in $\mathcal{G}$
$\mathcal{D}$	the training dataset
$\vert\mathbf{Z}\vert$	number of variables in $\mathbf{Z}$
$score(\mathcal{G}, \mathcal{D})$	score of $\mathcal{G}$ given a dataset $\mathcal{D}$

Part I

INTRODUCTION

Chapter 1

Introduction

In recent years, machine learning has become one of the most popular tools for solving problems in all kind of real world situations. We can find applications in industrial scenarios like electrical motor monitoring [Kirchgässner et al., 2021] or refinery furnace modelling [Sundar et al., 2020], in neuroscience [Bielza and Larrañaga, 2014], in natural language generation [Gatt and Krahmer, 2018], in automatic image identification and description [Hossain et al., 2019], in automatic driving [Kiran et al., 2021] and many other fields.

However, behind all these applications there are machine learning models trained with data. These models can be fitted to perform all kinds of tasks, but each one has its own advantages and disadvantages. Some of them are suited to model nonlinear problems where as others are limited to linear problems, some require huge amounts of data and processing power to be trained while others can be trained with smaller datasets and demand less from processors, and some can show how they perform their tasks while others work as black boxes that offer results without showing the reasoning behind. These are just some examples that, usually, no machine learning model is suited for every task [Adam et al., 2019]. Most models have fields where they are popular tools, and they have evolved over time in order to overcome some of their initial drawbacks and extend their capabilities [Wang and Raj, 2017].

If we focus ourselves in industrial systems, the increasing number of sensors monitoring processes in real time has generated huge loads of time-series data, where many readings of these sensors are recorded as time passes. This has given rise to an increased interest in machine learning models that can work with time-series and the specific problems related to them, such as estimating the remaining useful life of components [Ma et al., 2014; Cai et al., 2019] or forecasting possible future values for specific variables, which can also be applied to situations outside industry like stock market predictions [Duan, 2016] or medicine [Ehwerhemuepha et al., 2021].

In this dissertation, we will focus on one specific machine learning model, dynamic Bayesian networks (DBNs). This model allows us to work with time-series data by identifying conditional independence relationships between variables over time. They can be applied as general purpose models, but suffer from some shortcomings that limit their use. When applied to continuous variables, DBNs are linear models that can usually assume au-

toregressive order 1, which limits their accuracy in real world scenarios where systems do not usually follow these assumptions. On top of that, their application is not straightforward because of the lack of software packages that allow fast prototyping of DBN models. Throughout this work, we will define modifications and additions to the DBN framework in order to improve the capabilities of DBNs as general purpose models applied to real world problems. We will also compile all these modifications inside an open source R package that allows a streamlined application of these models

Chapter outline

First, in Section 1.1 we present the hypotheses and objectives of this work. Then, in Section 1.2 we discuss the organization of the rest of the document.

1.1 Hypotheses and objectives

This dissertation revolves around three main hypotheses:

- High-order Gaussian DBNs can be used to accurately model the profile and tendency of multivariate time-series inside industrial furnaces in long-term scenarios.
- The extension of DBN models to higher autoregressive orders and to nonlinear predictions will allow their application as general purpose models that fit industrial data accurately and competitively with other state-of-the-art time-series forecasting models.
- The development of a DBN framework that allows a straightforward application and modelling of data-driven networks will boost the usage and industrial prototyping of DBNs.

In order to prove these hypotheses, our main objectives in this work are:

- To develop a tool that allows seamless and straightforward learning and inference of high-order Gaussian DBNs.
- To develop a particle swarm algorithm that is capable of learning high-order DBN structures and is invariant of the desired Markovian order of the networks.
- To create a new framework that allows Gaussian DBNs to perform nonlinear forecasting.
- To extend the use of DBNs as general purpose time-series forecasting models in industrial settings and compare them with other state-of-the-art models.

1.2 Document organization

This document includes five parts, nine chapters and an appendix organized as follows:

Part I. Introduction

This part provides the preambles of this work.

- Chapter 1 shows the hypotheses and objectives of the thesis. It also presents the document organization.

Part II. Background

- Chapter 2 provides an introductory background to Bayesian network (BN) basic theory. We introduce the formal definition of the joint probability distribution and explain the process of learning both the structure and the parameters of a BN. Finally, we provide methods to perform exact and approximate inference with the model.
- Chapter 3 explains the extension of BNs to the dynamic scenario. We begin with a short introduction to time-series (TS) data and how to work with them. Then, we explain the modifications performed on the baseline BN framework in order to allow it to work with time-series. After presenting learning and inference methods, we end the chapter explaining how to perform forecasting.

Part III. Contributions

This part contains the five chapters corresponding to the contributions of this work.

- Chapter 4 addresses the modelling of an industrial furnace with high-order DBNs. In this chapter we evaluate how increasing the Markovian order of DBNs helps when modelling the trend of TS and improves the accuracy of long term forecasting.
- Chapter 5 provides a new structure learning algorithm specific for high-order DBNs. Our new method uses particle swarm optimization to move through the space of possible DBN structures and is encoded in a way that increasing the Markovian order does not increase the size of the particles.
- Chapter 6 provides a new hybrid model between model trees and DBNs. This new model is able to perform nonlinear forecasting via piecewise regression. This addresses the linearity constraint of Gaussian DBNs, which makes the model incur in innacuracies when modelling nonlinear processes.
- Chapter 7 provides a method to couple Gaussian DBNs with static classifiers to identify whether patients infected with the COVID-19 are going to be in a critical state in the next 40 hours or not. It provides a framework to make Gaussian DBNs able to perform classification tasks over time with the aid of a classifier. In this case, the classifier that provided the best accuracy was a neural network.
- Chapter 8 discusses the open source R package developed throughout all the contributions. This package encapsulates the functionality of creating DBNs, visualizing

them and performing forecasting. The package has gained a level of notoriety by being downloaded by more than 18,000 people worldwide.

Part IV. Conclusions

This part offers the conclusions of this work.

- Chapter 9 summarises the contributions of this work, provides a list with all the publications related to this dissertation and defines some possible future work.

Part V. Appendices

This part offers supplementary information.

- Appendix A defines a system of ordinary differential equations that simulates the fouling phenomenon. This simulator can be used to generate synthetic nonlinear datasets for testing purposes.

Part II

BACKGROUND

Chapter 2

Bayesian networks

2.1 Introduction

When we deal with industrial problems, we have to face several issues regarding the data recovered from them. Industrial systems usually represent processes not perfectly defined and understood by their operators, and tend to be affected by a stochastic component. This situation creates streams of data that do not always follow clear distributions and that can be affected by uncontrollable exogenous agents. In addition, data recovered from these processes comes from sensors, which can suffer from noisy readings and that not always cover all the aspects of the original process. This leaves us with data that defines both the state and the performance of our industrial system, but which has also a great deal of uncertainty.

One of the ways to treat uncertainty in data is through the use of probabilistic models and probability theory. By adjusting a probability distribution to our data and estimating its parameters, we can obtain a model that explains the data and allows us to perform queries that take into account the uncertainty. However, estimating probability distributions in large multivariate spaces can be prohibitive due to the computational complexity.

To approach this dimensionality issue, we can employ probabilistic graphical models. These models offer a way to describe the complex structure of multivariate probability distributions in compact graphs that can be interpreted by human users. In particular, in this dissertation we will focus on BNs, which are a type of probabilistic graphical models that encode the conditional independence relationships between the variables in our system as a directed acyclic graph. This way, they allow a complex probability distribution to be represented in a compact way that can be used to perform inference and to answer probabilistic queries.

Chapter outline

The rest of this chapter is organized as follows. Section 2.2 introduces the BN model and differentiates between the types of BN depending on the nature of the variables. Section 2.3 revolves around the learning of both the graphical structure and the parameters associated

with each node in the graph inside a BN. Lastly, Section 2.4 describes how to perform inference in the BN frameworks.

2.2 Bayesian networks

A BN is a representation of a joint probability distribution (JPD) P over a set of random variables $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$. This JPD is defined by independence relationships between the random variables. In the most elemental case, we can define the JPD of a set of independent variables as:

$$P(\mathbf{X}) = P(X_1)P(X_2)\cdots P(X_n), \tag{2.1}$$

where we say that for any disjoints sets of nodes $\mathbf{Y}_1$ and $\mathbf{Y}_2$ where $\mathbf{Y}_1, \mathbf{Y}_2 \in \mathbf{X}$, $P(\mathbf{Y}_1, \mathbf{Y}_2) = P(\mathbf{Y}_1)P(\mathbf{Y}_2)$, denoted as $\mathbf{Y}_1 \perp \mathbf{Y}_2$. However, the assumption that all the variables in our system are independent from one another is not usually valid in most scenarios. A more natural relationship between variables is that of conditional independence. We say that two variables X_1 and X_2 are conditionally independent from one another given another variable X_3 if:

$$P(X_1, X_2|X_3) = P(X_1|X_3)P(X_2|X_3), \tag{2.2}$$

which we denote as $X_1 \perp X_2|X_3$. Conditional independence is similar to that of total independence, but in this case we need to know the value of the variable X_3 for X_1 and X_2 to be independent from one another.

2.2.1 Structure

We can define a BN as a tuple $(\mathcal{G}, \boldsymbol{\theta})$, where $\mathcal{G}$ is a directed acyclic graph (DAG) and $\boldsymbol{\theta}$ is the set of parameters that define the conditional probability distribution (CPD) of each variable in the BN. Inside $\mathcal{G}$, each node corresponds to a random variable and the arcs define the conditional independence relationships between triplets of random variables. These relationships can be explained by using the concept of local independences.

Let $\mathbf{Pa}(X_i)$ denote the set of parents of the node X_i in $\mathcal{G}$ and let $\mathbf{NonDescendants}(X_i)$ be the set nodes that do not descend from X_i. The local independences imply that a node X_i is independent of its non descendants in the graph $\mathcal{G}$ given its parents, which is formally defined as:

$$\forall X_i : X_i \perp \mathbf{NonDescendants}(X_i)|\mathbf{Pa}(X_i) \tag{2.3}$$

This means that the conditional independence relationships between the variables in the BN are encoded via the arcs in the graph and the local independences. We can then factorize the JPD in a BN using the set of parents of each variable in $\mathcal{G}$:

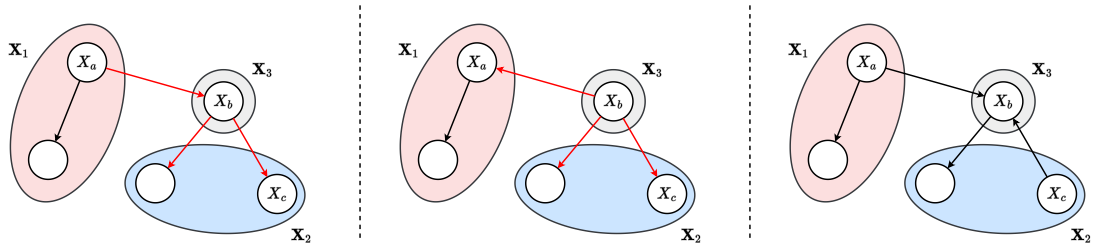


Figure 2.1: Examples of d-separation of the sets $\mathbf{X}_1$ and $\mathbf{X}_2$ given $\mathbf{X}_3$. From left to right, there are sequential, divergent and convergent scenarios. In the first two cases, $\mathbf{X}_1$ and $\mathbf{X}_2$ are d-separated given $\mathbf{X}_3$, but they are not in the convergent case because X_b is part of $\mathbf{X}_3$.

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i)). \tag{2.4}$$

2.2.1.1 D-separation

Based on local independences, we can derive the set of conditional independence statements encoded in $\mathcal{G}$ by using the concept of d-separation [Geiger et al., 1990]. Given three disjoints sets of nodes $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$, to know if $\mathbf{X}_1 \perp \mathbf{X}_2 | \mathbf{X}_3$ holds in $\mathcal{G}$ we have to check all the sequences of nodes from $\mathbf{X}_1$ to $\mathbf{X}_2$ without taking into account the direction of arcs, that is, we have to check all the trails from $\mathbf{X}_1$ to $\mathbf{X}_2$ in $\mathcal{G}$. Let $\mathbf{T}$ be the set of all possible trails from any node $X_a \in \mathbf{X}_1$ to any node $X_b \in \mathbf{X_2}$. Then $\mathbf{X_3}$ blocks a trail $T \in \mathbf{T}$ if:

- T contains a sequential chain with the form $X_a \rightarrow X_b \rightarrow X_c$ or $X_a \leftarrow X_b \leftarrow X_c$ where $X_c \in \mathbf{X}_3$
- T contains a divergent chain with the form $X_a \leftarrow X_b \rightarrow X_c$ where $X_c \in \mathbf{X}_3$
- T contains a convergent chain with the form $X_a \rightarrow X_b \leftarrow X_c$ such that X_c and any of its descendants do not belong to $\mathbf{X}_3$

If all the trails in $\mathbf{T}$ are blocked by $\mathbf{X}_3$, then $\mathbf{X}_3$ d-separates $\mathbf{X}_1$ and $\mathbf{X}_2$, which we denote d-sep$(\mathbf{X}_1, \mathbf{X}_2|\mathbf{X}_3)_{\mathcal{G}}$. We illustrate the sequential, divergent and convergent concepts of d-separation in Figure 2.1.

2.2.1.2 I-maps

In the case of BNs, we need to ensure that there is a correspondence between the set of conditional independences encoded in $\mathcal{G}$ and the set of conditional independences of the JPD P. This correspondence is called a mapping, and in particular we are interested in I-maps. We say that a DAG $\mathcal{G}$ is an I-map of a probability distribution P if the set of all conditional independence relationships encoded in $\mathcal{G}$ are also true for P. We can use the concept of d-separation to formally define I-maps:

$$\text{d-sep}(\mathbf{X}_1, \mathbf{X}_2|\mathbf{X}_3)_{\mathcal{G}} \implies (\mathbf{X}_1 \perp \mathbf{X}_2|\mathbf{X}_3)_P, \tag{2.5}$$

where we specify that any sets of nodes $\mathbf{X}_1$ and $\mathbf{X}_2$ d-separated in $\mathcal{G}$ by a third set $\mathbf{X}_3$ have an implied conditional independence relationship in P.

In a BN, the DAG $\mathcal{G}$ needs to be an I-map of the JPD P to allow the BN to compactly factorize P in the unique way defined by Equation (2.4).

2.2.2 Parameters

Apart from the graph $\mathcal{G}$, the other component that forms a BN are the parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$. These parameters θ_i define the conditional probabilities assigned to each node X_i given its parents $\mathbf{Pa}(X_i)$. When a variable does not have parents, its parameters define the marginal distribution associated to its corresponding node. On the other hand, non-orphan variables in $\mathcal{G}$ have parameters that define the effects of its parents on its conditional probability distribution.

The form of these parameters depends on whether our variables are discrete, continuous or a mixture of both.

2.2.2.1 Discrete variables

The most extended representation is that of BNs composed solely of discrete variables, that is, variables with a numerable set of states that they can take. This includes both qualitative variables that have a finite number of possible categories, such as blood type or hair color, and variables with a finite number of numerical values they can take, such as number of siblings or star rating of a hotel.

For these kinds of variables, the most popular representation for their parameters is that of a table. The idea behind this tabular form is that we need to represent the CPD of a variable depending on every single value configuration that its parents can take. This generates local conditional probability tables (CPTs) that dictate the probability $P(X_i = x|\mathbf{pa}(X_i))$ for every value x of X_i and every configuration of $\mathbf{Pa}(X_i)$. In addition, these CPTs require that $P(X_i = x_i|\mathbf{pa}(X_i)) \geq 0$ for every value configuration and that $\sum_{x_i} P(x_i|\mathbf{Pa}(X_i)) = 1$. This ensures that the underlying CPD of the model is well defined.

Given that the parameters are explicitly represented as probabilities, the tabular representation is easily interpretable, but it presents the heavy drawback of an exponential number of parameters. As the number of parents of a variable grows, the CPTs become exponentially large in turn. This presents a limitation to both parameter learning and inference complexity. To overcome this issue, the two most common alternatives are canonical models and graphical CPTs [Sucar, 2015]. Canonical models reduce the number of parameters per parent node when the probability of a random variable in a BN presents specific deterministic characteristics, such as those present in logical gates. Examples of this representation are the noisy-OR [Pearl, 1988] and the noisy-MAX [Diez, 1993] models. On the other hand, a more general solution for more compact CPTs is that of graphical substructures like decision trees [Friedman and Goldszmidt, 1998; Talvitie et al., 2018]. These solutions improve the computational costs of the CPT representation, but ultimately the only feasible way to limit

the exponential explosion of parameters is by limiting the number of parents of the variables in the graph.

2.2.2.2 Continuous variables

In the presence of continuous variables, we face a situation where variables can take values in a real interval. As such, our previous tabular representation is not able to encode the CPDs of these variables. We need to define $f(X_i|\mathbf{pa}(X_i))$, which is the conditional density function of X_i, in a way that all the infinite values that the parent variables can take have an established effect on X_i.

The first typical solution to this issue is to go back to the discrete scenario by discretizing all the variables. However, this approach inevitably incurs in a loss of information by simplifying a continuous space into a finite set of bins. In addition, a balance has to be found between a very high computational cost due to a high number of possible values for the discretized variables, and a severe information loss.

If we opt for working with the continuous variables, the most common way to define the dependency of a continuous node X_i on a continuous parent is to make the mean of X_i a linear function of $\mathbf{Pa}(X_i)$. This way, the value of X_i is well defined based on all the possible values of $\mathbf{Pa}(X_i)$ given the probability distribution that we assume for X_i.

The most popular representation for this conditional density function is the Gaussian distribution. By assuming that all the variables in our system follow a normal distribution and that their variances are independent from one another, we obtain the linear Gaussian CPDs [Shachter and Kenley, 1989]. In a linear Gaussian CPD we have that:

$$f(X_i|\mathbf{pa}(X_i)) \equiv f(X_i|x_1,\dots,x_k) = \mathcal{N}(\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k; \sigma_i^2), \tag{2.6}$$

where $\mathcal{N}(\mu;\sigma)$ denotes a normal distribution with mean μ and variance σ, $\beta_0,\dots,\beta_k$ are the parameters that define the effect of the values of $\mathbf{Pa}(X_i)$ on the mean of X_i and σ_i^2 is the unconditional variance of X_i. It is important to note that when all the nodes in a BN have linear Gaussian CPDs, the BN serves as an alternative representation of a multivariate Gaussian distribution. With linear Gaussian CPDs, inference and parameter learning are greatly simplified from the discrete case and from other more complex continuous representations, but we apply strong assumptions on the variable distribution in turn. All in all, this representation acts as a good approximation for real-world problems [Kotz et al., 2004].

Other representations for BNs with continuous variables are also seen in the literature. We can find approaches similar to the linear Gaussian CPDs that allow more complex relationships between the variables or different distributions. These include distributions like the inverse-Gaussian or the gamma distributions [Masmoudi and Masmoudi, 2019], kernel density estimation CPDs [Atienza et al., 2022] and Gaussian processes surrogate models [Zhu et al., 2019], among others. These kind of representations allow for more complex relationships between the variables than the linear Gaussian model, but they loose interpretability due to the more complex parameters and most of them have to resort to simulation techniques for

inference and parameter learning. In this work, we will focus our attention on linear Gaussian CPDs.

2.2.2.3 Hybrid networks

When we encounter both discrete and continuous variables at the same time, we need to define new conditional independence relationships that cover all the possible values of the variables. So far, we have discussed the case of a discrete variable with discrete parents and a continuous variable with continuous parents. We will now go over the case of continuous variables with discrete parents and discuss some possible approximations to the case of discrete variables with continuous parents.

The most extended approach to continuous variables with discrete parents is that of conditional linear Gaussian CPDs [Lauritzen and Wermuth, 1989]. They assume linear Gaussian CPDs and do not allow continuous parents for discrete variables. This representation stores several linear Gaussian CPDs, one for each different configuration that the discrete parents of a continuous variable can take. For each different configuration c of the discrete parents $\mathbf{Pa}^d(X_i)$, we have that:

$$f(X_i|\mathbf{pa}^g(X_i), c) = \mathcal{N}(\beta_{0c} + \beta_{1c}x_1 + \cdots + \beta_{kc}x_k; \sigma^2_{ic}), \tag{2.7}$$

where $\mathbf{Pa}^g(X_i)$ are the continuous parents of X_i. This encoding has the advantages and simplicity of linear Gaussian CPDs while mitigating the linearity issue of the model, but it forces strong restrictions on the graph structure and a high number of interconnected discrete variables incurs in the same exponential explosion of tabular CPDs as the purely discrete case.

In the case of discrete nodes with continuous parents, the conditional linear Gaussian CPD cannot be applied. An extended version of this CPD was proposed by Lerner et al. [2001] which allows arcs from continuous variables to discrete ones through the use of the soft-max function. Other authors opted for approaches related to dividing the continuous space of the parent variables into several subspaces, like mixtures of truncated exponentials [Moral et al., 2001], mixtures of polynomials [Shenoy and West, 2011], and the generalization of both with mixtures of truncated basis functions [Langseth et al., 2012; Pérez-Bernabé et al., 2020]. These approximations use the sum of a set of basis functions to divide the space of continuous variables into hyper-cubes. Afterwards, the effect of the continuous variable over the discrete one is defined depending on the hyper-cube that the continuous variable belongs to.

2.3 Learning

The problem of fitting BNs requires learning both the structure of the graph $\mathcal{G}$ and the parameters $\boldsymbol{\theta}$. To learn the structure of the network we need to find the underlying conditional independence relationships between our variables. This in turn fixes the number of parameters of our model and, after making some assumptions on the distributions of our variables, allows

us to find the most appropriate values for $\boldsymbol{\theta}$.

Traditionally, BNs have been learned either manually from expert knowledge or automatically from data. In this work, we will focus on learning Gaussian BNs from complete datasets, i.e., without missing values.

2.3.1 Parameter estimation

Let us assume that we already have a graph structure $\mathcal{G}$ and a dataset $\mathcal{D} = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m\}$, where $\mathbf{d}_i = (x_1^i, x_2^i, \ldots, x_n^i)$ is a row in our dataset with the values of all variables $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ in our graph. In this scenario, we want to learn the network parameters $\boldsymbol{\theta}$ defined by $\mathcal{G}$ that best fit our data $\mathcal{D}$. We can typically perform this automatically from data via maximum likelihood estimation or Bayesian estimation.

2.3.1.1 Maximum likelihood estimation

One of the most common methods for parameter estimation is maximum likelihood estimation (MLE). With this method, our aim is to obtain the parameter set $\boldsymbol{\theta}$ by using a point estimate based on all instances of $\mathcal{D}$. To know whether a set $\boldsymbol{\theta}$ fits some data properly we use the likelihood function:

$$P(\mathcal{D}|\mathcal{G}, \boldsymbol{\theta}) = \prod_{i=1}^{m} P(\mathbf{d}_i|\mathcal{G}, \boldsymbol{\theta}) = \prod_{i=1}^{m} \prod_{j=1}^{n} P(x_j^i|\mathbf{pa}(x_j^i)), \tag{2.8}$$

where $\mathbf{pa}(x_j^i)$ represents the values that the parents of X_j took in the $\mathbf{d_i}$ instance. To obtain the set of parameters $\hat{\boldsymbol{\theta}}$ that best fit $\mathcal{D}$ via MLE, we have to maximize Equation (2.8) with respect to $\boldsymbol{\theta}$. Given that the likelihood and the log-likelihood obtain the same set of parameters, we maximize over the log-likelihood for convenience. This log-likelihood is defined as:

$$L(\mathcal{G}, \boldsymbol{\theta} : \mathcal{D}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \log P(x_j^i|\mathbf{pa}(x_j^i)) = \sum_{j=1}^{n} L(X_j|\mathbf{Pa}(X_j), \boldsymbol{\theta}_j : \mathcal{D}), \tag{2.9}$$

where $L(X_i|\mathbf{Pa}(X_i), \boldsymbol{\theta}_i : \mathcal{D})$ is the local log-likelihood of variable X_i and $\boldsymbol{\theta}_i$ are the set of parameters of the $P(X_i|\mathbf{Pa}(X_i))$ CPD. Given that the likelihood is locally decomposable, we sum over the local log-likelihood of each variable X_j and further simplify Equation (2.9):

$$L(\mathcal{G}, \boldsymbol{\theta} : \mathcal{D}) = \sum_{i=1}^{m} \sum_{j=1}^{n} L(x_j^i|\mathbf{pa}(x_j^i))). \tag{2.10}$$

In this scenario, the maximization of $\hat{\boldsymbol{\theta}}$ is defined as:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} L(\mathcal{G}, \boldsymbol{\theta}|\mathcal{D}) = \{ \arg\max_{\boldsymbol{\theta}_1} \sum_{i=1}^{m} L(x_1^i|\mathbf{pa}(x_1^i), \boldsymbol{\theta}_1), \ldots, \arg\max_{\boldsymbol{\theta}_n} \sum_{i=1}^{m} L(x_n^i|\mathbf{pa}(x_n^i), \boldsymbol{\theta}_n)\}, \tag{2.11}$$

and due to the CPDs not sharing any parameters, we can find the MLE parameters independently for each node X_j. Depending on the CPD of each node, the local log-likelihood functions can take different forms. We will go over the case of linear Gaussian CPDs for BNs with only continuous variables.

2.3.1.2 Estimation of Gaussian networks

Following on Equation (2.6), we can see that the parameters of a node with a linear Gaussian CPD are a set of linear regression coefficients $\boldsymbol{\theta}_j = (\beta_{0j}, \beta_{1j}, \ldots, \beta_{kj})$ and a standard deviation σ_j^2. Given that the local likelihood corresponds to a CPD of linear regressors, the most popular method to estimate the local $\beta_0, \ldots, \beta_k$ parameters is using linear algebra through ordinary least squares regression. For a specific node X_j we can write:

$$\mathbf{x}_j = \mathbf{pa}(x_j)\hat{\boldsymbol{\theta}}_j + \mathbf{e}_j, \tag{2.12}$$

where $\mathbf{x}_j$ is the vector of dimension m with all the values that X_j took in our dataset $\mathcal{D}$, $\mathbf{pa}(\mathbf{x}_j)$ is the matrix of dimension $m \times |\mathbf{Pa}(X_j)| + 1$ with all the values that the parent nodes of X_i took in $\mathcal{D}$ and $\mathbf{e}_j$ is the residuals vector of dimension m obtained. Additionally to the values contained in $\mathbf{pa}(\mathbf{x}_j)$, we have to concatenate a unitary column vector to $\mathbf{pa}(\mathbf{x}_j)$ to account for the effect of β_0, which is the intercept and is not ligated to any parent of X_j. Now, the least square estimator is that whose values of $\hat{\boldsymbol{\theta}}_j$ minimize the sum of squares of the residuals $\mathbf{e}_j$:

$$\begin{aligned} sum(\mathbf{e}_j) &= \sum_{i=1}^{m} \mathbf{e}_j^{i\,2} = \mathbf{e}_j^{i\,T}\mathbf{e}_j^i = (\mathbf{x_j} - \mathbf{pa}(x_j)\hat{\boldsymbol{\theta}}_j)^T(\mathbf{x}_j - \mathbf{pa}(x_j)\hat{\boldsymbol{\theta}}_j) \\ &= \mathbf{x}_j^T\mathbf{x}_j - \mathbf{x}_j^T\mathbf{pa}(x_j)\hat{\boldsymbol{\theta}}_j - \hat{\boldsymbol{\theta}}_j{}^T\mathbf{pa}(x_j)^T\mathbf{x}_j + \hat{\boldsymbol{\theta}}_j{}^T\mathbf{pa}(x_j)^T\mathbf{pa}(x_j)\hat{\boldsymbol{\theta}}_j \end{aligned} \tag{2.13}$$

To obtain the maximum likelihood estimation of $\hat{\boldsymbol{\theta}}_j$, we need to minimize the sum of squares of the residuals. To accomplish this, we differentiate with respect to the parameters and equate to 0:

$$\begin{aligned} \frac{\partial sum(\mathbf{e}_j)}{\partial \hat{\boldsymbol{\theta}}_j} &= -2\mathbf{pa}(x_j)^T\mathbf{x}_j + 2\mathbf{pa}(x_j)^T\mathbf{pa}(x_j)\hat{\boldsymbol{\theta}}_j = 0 \\ \hat{\boldsymbol{\theta}}_j &= (\mathbf{pa}(x_j)^T\mathbf{pa}(x_j))^{-1}\mathbf{pa}(x_j)^T\mathbf{x}_j \end{aligned} \tag{2.14}$$

Now we are only missing the standard deviation σ_j^2. In this case, we use the sum of squared residuals to calculate its value:

$$\hat{\sigma}_j^2 = \frac{sum(\mathbf{e}_j)}{m - |\hat{\boldsymbol{\theta}}_j|}, \tag{2.15}$$

where m was the total number of instances in $\mathcal{D}$ and $|\hat{\boldsymbol{\theta}}_j|$ is the total number of parameters of node X_j, which is always equal to the number of parents of the node plus one.

In the case a node X_j with no parents, the only parameter β_0 that the node has is the most likely value in our dataset $\mathcal{D}$, that is, the arithmetic mean of the components of vector $\mathbf{x}_j$. Afterwards, its parameter σ_j^2 is calculated as usual with Equation (2.15), but in this case $\mathbf{e}_j$ are the residuals obtained from subtracting the arithmetic mean of $\mathbf{x}_j$ to each of the values of $\mathbf{x}_j$.

2.3.2 Structure learning

The task of learning the graph structure of a BN is a complex problem where the search space of possible DAGs grows super-exponentially with the number of nodes [Robinson, 1977]. In order to find appropriate structures, many authors have proposed different structure learning algorithms. Typically, these have been categorized as score-based, constraint-based or hybrid depending on how they navigate the solution space of possible graphs. We will cover each category and focus on some specific algorithms that are relevant to this work. For a more in-depth review of existing algorithms, we refer the readers to Scanagatta et al. [2019] and to Kitson et al. [2023].

2.3.2.1 Score-based structure learning

Score-based algorithms define the task of finding the best graph structure as an optimization problem. If we define a score that identifies how well does a graph fit some data, then we can maximize this score to find an optimal structure. However, the problem of finding the optimal network is NP-hard [Chickering et al., 2004] and algorithms have to resort to heuristics in the search phase to find the best possible structure instead of the optimal one.

Score phase

Let $\mathcal{D}$ be our training dataset and let $\mathcal{G}$ be a DAG structure in the space of all possible valid DAGs g^{BN}. Our objective can now be defined as:

$$\arg\max_{\mathcal{G} \in g^{BN}} score(\mathcal{G}, \mathcal{D}). \tag{2.16}$$

A very direct choice for this score would be the log-likelihood score that we previously used in Equation (2.9) for parameter learning. This score would determine how likely is it that a graph structure $\mathcal{G}$ and its MLE parameters $\hat{\boldsymbol{\theta}}$ generated our dataset $\mathcal{D}$. This can be written as $L(\mathcal{G}, \hat{\boldsymbol{\theta}}|\mathcal{D})$, but this method tends to generate fully connected graphs. Instead, penalized

versions of the log-likelihood like the Akaike information criterion (AIC) and the Bayesian information criterion (BIC) [Geiger and Heckerman, 1994] are more used in practice. These scores use a penalizing factor for the number of parameters that a model has, generating sparser graphs. They can be both expressed with the same general formula:

$$score(\mathcal{G}, \mathcal{D}) = L(\mathcal{G}, \hat{\boldsymbol{\theta}}|\mathcal{D}) - pen(\mathcal{G}, \mathcal{D}), \tag{2.17}$$

where $pen(\mathcal{G}, \mathcal{D})$ is a penalization function. In the case of AIC, $pen(\mathcal{G}, \mathcal{D})$ is just the number of parameters of $\mathcal{G}$ ($nparams(\mathcal{G})$, and for the BIC score the penalization is $pen(\mathcal{G}, \mathcal{D}) = \frac{nparams(\mathcal{G})}{2} log(m)$.

Search phase

With a scoring function, we can now perform the search through the space of possible networks as the optimization problem defined in Equation (2.16). However, given that finding the optimal BN structure is an NP-hard problem and that the search space is super-exponential on the number of nodes, the most common procedure is to apply some heuristic to search for good candidate networks rather than the optimal one.

A very popular search procedure in this context is the greedy hill-climbing algorithm. This method starts from an initial structure $\mathcal{G}_0$ and applies local modifications to improve its overall score. These operations usually comprise the addition, deletion or reversal of arcs. Given that the likelihood-based scores are decomposable and can be calculated independently for each variable as shown in Equation (2.10), local changes to specific nodes only affect a small part of the final score. By initially precalculating the resulting scores for each operation in every node, the algorithm can easily find the one that increases the global score the most and then apply it. Afterwards, the precalculated scores affected by the arc modification performed are recalculated and a new modification is selected in this greedy manner until no further improvements to the score can be found. Some variations of this algorithm that try to avoid local optima include the use of random restarts or the addition of TABU lists [Tsamardinos et al., 2006], where a TABU list is used to avoid repeating steps in the greedy search, and the K2 algorithm [Cooper and Herskovits, 1992], where only additions of parents to nodes are allowed from an initially empty graph $\mathcal{G}_0$. For reference, we include in Algorithm 2.1 the pseudo-code of the greedy hill-climbing algorithm with the addition of random restarts.

Another extended approach is the use of evolutionary or swarm intelligence search algorithms to move through the solution space. By encoding the graph structure of the network as an individual in the optimization process and adapting the addition, deletion and reversal of arcs operators, among other possible ones, the application of these kinds of algorithms can be very natural given the immense solution space. We can find examples of genetic algorithms [Larranaga et al., 1996; Sun and Zhou, 2022], estimation of distribution algorithms [Blanco et al., 2003], particle swarm optimization [Gheisari and Meybodi, 2016] or ant colony optimization [Wu et al., 2010], among others. For more examples of these kinds of algorithms, we refer the readers to the works of Larrañaga et al. [2013] and Sun and Zhou [2022].

Algorithm 2.1: Greedy hill-climbing algorithm with random restarts.

Input: Initial graph $\mathcal{G}_0$, number of random restarts r_{max}, score S, set of available operators **Op**
Output: The best graph found $\mathcal{G}_{best}$
Data: Training set $\mathcal{D}$

$end = False$;
$r = 0$;
$score = S(\mathcal{G}_0, \mathcal{D})$;
$score_{best} = score$;
$pre\_scores = precalculate\_scores(\mathcal{G}_0, \mathcal{D}, S, \textbf{Op})$; `// Score all possible operators`
$\mathcal{G} = \mathcal{G}_0$;
$\mathcal{G}_{best} = \mathcal{G}_0$;
while *not end* **do**
    $op = find\_best\_operator(pre\_scores)$; `//` $op \in \textbf{Op}$
    $\mathcal{G}_{mod} = apply\_operator(\mathcal{G}, op)$;
    $score_{mod} = S(\mathcal{G}_{mod}, \mathcal{D})$;
    if $score_{mod} > score$ **then**
        $\mathcal{G} = \mathcal{G}_{mod}$;
        $score = score_{mod}$;
        $pre\_scores = update\_scores(\mathcal{G}, \mathcal{D}, S, op)$; `// Update only the scores related to` op
    if $score > score_{best}$ **then**
        $score_{best} = score$;
        $\mathcal{G}_{best} = \mathcal{G}$;
    if $score \geq max(pre\_scores)$ **then**
        if $r < r_{max}$ **then**
            $r = r + 1$;
            $\mathcal{G} = randomize(\mathcal{G})$; `// Apply a random restart`
        else
            $end = True$; `// Convergence at no improvement and max restarts`
return $\mathcal{G}_{best}$;

2.3.2.2 Constraint-based structure learning

In the case of constraint-based algorithms, the main idea is to generate an initial network structure based on the underlying conditional independences between variables found in our data. If we manage to identify these conditional independences, then we can model them with the arcs of the graph. For this task, we can use conditional independence tests such as the χ^2 test for discrete variables and the t-test for Pearson's correlation coefficient [Edwards, 2012] for continuous variables:

$$t(X, Y|\mathbf{Z}) = \rho_{X,Y|\mathbf{Z}} \sqrt{\frac{m - |\mathbf{Z}| - 2}{1 - \rho^2_{X,Y|\mathbf{Z}}}}, \tag{2.18}$$

where m is the total number of instances in our dataset, $\rho_{X,Y|\mathbf{Z}}$ is the partial correlation coefficient of X and Y given the set of variables $\mathbf{Z}$ and $|\mathbf{Z}|$ is the number of variables in the set $\mathbf{Z}$. Other tests based on measures like mutual information are also popular in the literature. These kinds of tests can be used to define the local structures of the network: if we do not find sufficient evidence to reject the null hypothesis of conditional independence of X and Y given Z, then we keep the arc between X and Y. In practice, if no $\mathbf{Z}$ is found for two variables X and Y that defines them as independent given $\mathbf{Z}$, then there should be an arc $X - Y$ in some direction. By applying this procedure to all the variables in our graph, we can find all the local structures of our nodes. Afterwards, we merge them into a single skeleton graph with possibly undirected arcs. The last step in these kinds of algorithms usually entails refining the skeleton graph and orienting any undirected arc in order to obtain a DAG. Theoretically, if the conditional independence tests do not fail and are able to identify all conditional independence relationships in our data, this kind of algorithms should return the optimal network, but these tests tend to fail on real world datasets. Some examples of these kinds of methods are the PC algorithm [Spirtes et al., 2000; Tsagris, 2019], the grow-shrink algorithm [Margaritis, 2003] and the max-min parents and children algorithm [Tsamardinos et al., 2003].

2.3.2.3 Hybrid structure learning

In the case of hybrid algorithms, they try to incorporate the advantages of both score-based and constraint-based methods. They usually try to limit the search space by taking into account the underlying conditional independence relationships between variables present in the dataset $\mathcal{D}$ and afterwards use the heuristics of score-based algorithms in the reduced solution space.

One particular hybrid algorithm that is of relevance to this work is the max-min hill-climbing (MMHC) algorithm [Tsamardinos et al., 2006]. This algorithm starts by defining an initial skeleton graph through the use of conditional independence tests to find the possible parents of nodes. This step is performed with the constraint-based max-min parents and children algorithm [Tsamardinos et al., 2003], which finds a set of undirected edges between the variables. Once a set of undirected edges has been found for every variable in the graph, a hill-climbing algorithm similar to the one shown in Algorithm 2.1 is run, but only arcs present in the set returned by the parents and children algorithm can be added. This way, the algorithm combines the use of the conditional independence relationships in the dataset with the heuristic search of the hill-climbing algorithm to orient the undirected edges obtained with the conditional independence tests.

Other examples of hybrid structure learning algorithms include the works of De Campos et al. [2003], Dash and Druzdzel [1999] and the RSMAX2 algorithm [Scutari et al., 2014]. Recently, hybrid algorithms have also seen active development, as shown by the works of Dai et al. [2020] or Sun and Zhou [2022].

For further comparison of the performance of score-based, constraint-based and hybrid structure learning algorithms we refer the readers to Scutari et al. [2019].

2.4 Inference

Once that we have learned the structure and the parameters of a BN, we can use our model to perform probabilistic inference. The main idea of this process is that given the values of some evidence variables $\mathbf{E} = \mathbf{e}$, we want to know the most likely values that another set of unobserved variables $\mathbf{X}$ is going to take:

$$P(\mathbf{X}|\mathbf{E} = \mathbf{e}) = \frac{P(\mathbf{X}, \mathbf{e})}{P(\mathbf{e})} \tag{2.19}$$

This procedure allows BNs to perform similar tasks to other general purpose machine learning models, like classification or regression. Moreover, any variable in the BN can be the objective of inference and it allows for more complex probabilistic queries. However, inference in BNs is an NP-hard problem no matter if we perform exact [Cooper, 1990] or approximate [Dagum and Luby, 1993] inference. As such, this can create scenarios where inference is prohibitively slow in complex BNs. Fortunately, we can bypass this issue in some specific cases like BNs with linear Gaussian CPDs. We will briefly go over some common exact and approximate inference methodologies and explain the exact inference procedure with the Gaussian multivariate equivalent on Gaussian BNs, which is of relevance to this work.

2.4.1 Exact inference

When we want to obtain exactly conditional probabilities as in Equation (2.19), we say that we perform exact inference. One general purpose popular approach is the variable elimination algorithm. This algorithm can be applied to BNs with categorical CPDs, with linear Gaussian CPDs or with a mixture of both.

Variable elimination

The main idea of this algorithm is to take advantage of the independence relationships inside a BN to marginalize the JPD and calculate probabilities more efficiently. Let us have a set of variables $\mathbf{X} = \{\mathbf{X}_1 \dot{\cup} \mathbf{X}_2 \dot{\cup} \mathbf{X}_r\}$ composed of unions of disjoint sets, where $\mathbf{X}_1$ are our objective variables for the inference, $\mathbf{X}_2$ are our evidence variables and $\mathbf{X}_r$ are the rest of unobserved variables which are not our objective. Following Equation (2.19), our inference has the form:

$$P(\mathbf{X}_1|\mathbf{X}_2) = \frac{P(\mathbf{X}_1, \mathbf{X}_2)}{P(\mathbf{X}_2)}, \tag{2.20}$$

where both terms of the fraction can be defined as:

$$P(\mathbf{X}_1, \mathbf{X}_2) = \sum_{X_i \in \mathbf{X}_r} P(X_i) \tag{2.21}$$

$$P(\mathbf{X}_2) = \sum_{X_i \in \mathbf{X}_1} P(X_i, \mathbf{X}_2) \tag{2.22}$$

The problem arises in the summations of products $\sum_{X_i \in \mathbf{X}_r}$ and $\sum_{X_i \in \mathbf{X}_1}$, because these generate a high number of summations of local probabilities, many of which are repeated throughout the whole computation. This problem gets even more complicated if the variables are not binary and have large CPTs. In the variable elimination algorithm, we try to use the BN representation to eliminate those calculations that are not needed to obtain our objective probabilities. We call the different terms that are generated during the calculations factors, and they are functions over a subset of variables. Each factor maps a specific instantiation of these variables to non-negative numbers that can be reused throughout the inference process. To simplify these calculations, we can rearrange these operations by taking advantage of the properties of summation and multiplication. If we arrange the variables in an optimal order, we can drastically reduce the number of operations needed to obtain these probabilities by generating less factors and reusing already calculated ones. However, finding this optimal order is an NP-hard problem, and many times we have to resort to heuristics to find good orderings in a reasonable time frame.

Multivariate Gaussian equivalent

An interesting case specific to linear Gaussian CPDs is the possibility to alternate between a BN and its equivalent multivariate Gaussian distribution [Koller and Friedman, 2009]. We can transform the structure and parameters of a Gaussian BN into a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$ so that we are able to perform fast inference with the multivariate Gaussian distribution defined by them.

To obtain the mean vector, we have to calculate the most likely values for each of the variables, that is, their mean given the parameters in the network:

$$\mu_{X_i} = \beta_0 + \sum_{j=1}^{|\mathbf{Pa}(X_i)|} \beta_j \mu_{X_j}, \tag{2.23}$$

where $|\mathbf{Pa}(X_i)|$ is the number of parents of X_i, and so the summation iterates through the means of all the parents of X_i. Note that in the case of variables with no parents, their mean is defined only by their own mean parameter, which when learned by MLE it is equivalent to the mean of the variable in the dataset.

On the other hand, we can calculate the covariance matrix with two steps. First, to calculate the variances in the diagonal of the $\boldsymbol{\Sigma}$ matrix, we use the parameters of each variable and the variance of the parent variables:

$$\Sigma_{ii} = \sigma_i^2 + \sum_{j=1}^{|\mathbf{Pa}(X_i)|} \beta_j^2 \sigma_j^2. \tag{2.24}$$

Similarly to the mean calculation, variables without parents will simply use their own variance. After calculating the diagonal, we can compute the covariances Σ_{ij} between variables:

$$cov(X_i, X_j) = \Sigma_{ij} = \Sigma_{ji} = \sum_{k=1}^{|\mathbf{Pa}(X_j)|} \beta_k \Sigma_{ik}. \tag{2.25}$$

When we complete this process, we obtain a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ that allows us to perform inference over a set of unobserved variables $\mathbf{X}_1$ using another set of observed variables $\mathbf{X}_2$ as evidence. By splitting between unobserved and observed variables the mean vector $\boldsymbol{\mu} = \binom{\boldsymbol{\mu}_1}{\boldsymbol{\mu}_2}$ and the covariance matrix $\boldsymbol{\Sigma} = \binom{\boldsymbol{\Sigma}_{11}\boldsymbol{\Sigma}_{12}}{\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{22}}$, we can calculate the means $\boldsymbol{\mu}_{1|2}$ and covariance matrix $\boldsymbol{\Sigma}_{1|2}$ of the unobserved variables given the observed ones [Murphy, 2012]:

$$f(\mathbf{x}_1|\mathbf{x}_2) = \mathcal{N}(\boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}) \tag{2.26}$$

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2) \tag{2.27}$$

$$\boldsymbol{\Sigma}_{1|2} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} \tag{2.28}$$

This inference process allows us to perform fast inference on linear Gaussian networks, where the only costly operation is the inversion of the $\boldsymbol{\Sigma}_{22}$ covariance submatrix of the observed variables.

2.4.2 Approximate inference

In case we are not interested in obtaining the exact solution when performing inference, we can resort to obtaining approximate results instead. Through approximate inference, we can define the inference process as a sampling procedure. Even though approximate inference in BNs is also an NP-hard problem [Dagum and Luby, 1993], we can find improvements in terms of execution time when compared to exact inference methods for categorical or hybrid BNs.

One popular method for approximate inference in BNs is importance sampling [Yuan and Druzdzel, 2006]. Generating the values of variables via sampling is a straightforward procedure by forward sampling. This method begins by obtaining a topological ordering of the variables in a BN, that is, ordering them in such way that variables without parents are listed first and then we list variables whose parents have already been listed. This way, if we sample all the variables in a BN following such ordering, when we sample a node we make sure that its parents have already been sampled. In the case that we have observed variables, we use their observed values instead of sampling them. Then, we define a particle

as a complete sampling of all the nodes of a BN, where all the variables have a value. If we generate a set of N particles, by using the relative frequency in the categorical case or by averaging the particles values in the continuous case we can approximately estimate the values of a set of unobserved variables $\mathbf{X}_1$ given another set of observed variables $\mathbf{X}_2$ in the BN. However, this method only relies on the size N of randomly generated samples which do not take into account how likely each of them is to occur. Importance sampling is a variation that takes into account the likelihood of each particle given the parameters of the BN, so that we can take this likelihood when we average a final value. This way, we obtain more realistic predictions in the presence of unlikely evidence.

Other sampling methods, like Markov chain Monte Carlo approaches [York, 1992], try to generate samples from the posterior distribution of the network in the way of Bayesian statistics. This can offer better accuracy when we have an idea of the shape of the prior distribution, but in turn can be slower in terms of execution time.

Chapter 3

Dynamic Bayesian networks

3.1 Introduction

In real world scenarios, it is very common for industrial systems to operate for prolonged periods of time. Due to the monitoring of sensors, data is recorded and stored over time. This generates datasets where instances are ordered in time from the oldest one to the newest, generating time-series. As time progresses, we can see the evolution of all the variables in the system inside these time-series.

In order to make use of this additional information, we need to take into account the effect of this time component in our models. In the case of BNs, one of the ways available to deal with time-series is by extending them to dynamic models able to work variables over time. A dynamic Bayesian network will model the same independence relationships as a static BN, and it will also reflect the effect that past values have on the current and future instants.

By being able to deal with the additional dimension that time represents, dynamic Bayesian networks allow a much richer representation of industrial systems than that of static models. In addition, new operations like forecasting, estimating the remaining useful life of components or simulating the behaviour of processes will become available. This offers new takes on relevant industrial problems.

Chapter outline

The rest of this chapter is organized as follows. Section 3.2 introduces the concept of time-series and some of its basic characteristics that differentiate it from static data. Section 3.3 explains how the extension from BNs to dynamic Bayesian networks is performed and the resulting structure of dynamic Bayesian networks. Section 3.4 introduces how is the structure learning procedure of dynamic Bayesian networks. Lastly, Section 3.5 introduces some methods to perform inference in dynamic Bayesian networks and some specific time-series operations like forecasting.

3.2 time-series characteristics

A time-series (TS) is a collection ordered in time of instances of a single (univariate) or multiple (multivariate) random variables. We can refer to discrete TS when instances are separated at equal intervals from each other, or at least recorded at known distances from each other, or we can refer to continuous TS, when instances are recorded continuously in a given interval of time [Brockwell et al., 2002]. In this work, we will focus exclusively on discrete TS. An example of a univariate TS can be seen in Figure 3.1[1].

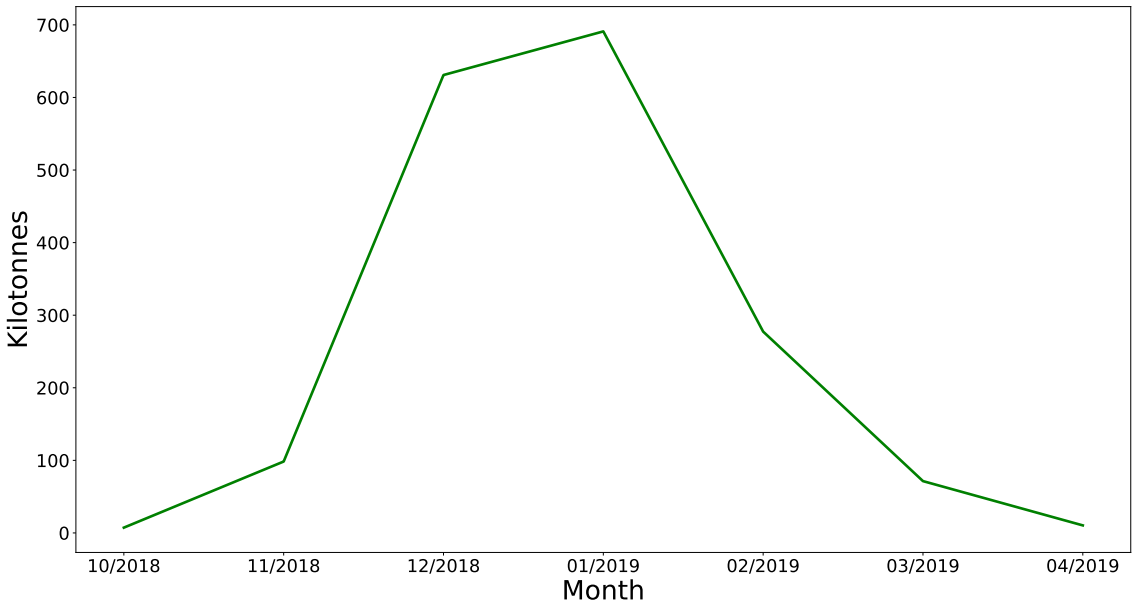


Figure 3.1: Example of a TS reflecting the production of olive oil in Spain from October 2018 to April 2019. In this case, the values of the production in kilotonnes were recorded with a monthly frequency.

When we analyse TS, we are interested in extracting information from the original system that generated them. We refer to this system as a stochastic process. A stochastic process is a collection of single or multiple random variables ordered in time, in our case at known time intervals from each other. When the random variables X_t in a stochastic process take specific values x_t, we obtain a time-series. A time-series is a specific realization of certain stochastic process, and by analysing several realizations of this process we can model the random variables in each instant and extract relevant information from the underlying process [Mauricio, 2007]. This relationship between TS and stochastic processes is shown in Figure 3.2.

We can further illustrate this by extending our previous olive oil production example. If we suppose that our stochastic process is the yearly olive oil production in Spain, then in Figure 3.1 we are observing a single instance of this process in the form of the TS. In this

[1]Source of the olive oil production data: https://www.mapa.gob.es/es/agricultura/temas/producciones-agricolas/aceite-oliva-y-aceituna-mesa/Datos_produccion_movimiento_existencias_AICA.aspx

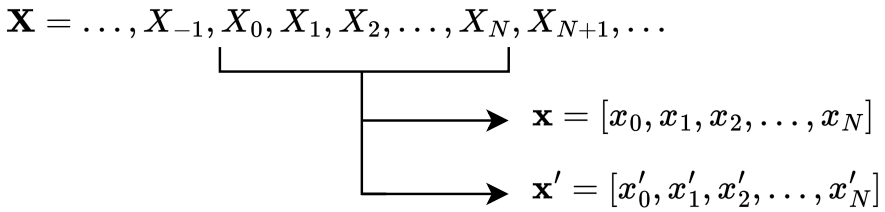


Figure 3.2: Example of a stochastic process $\mathbf{X}$ generating two TS $\mathbf{x}$ and $\mathbf{x}'$. These TS are just instances of a stochastic process where the random variables took specific values.

case, we can have several TS of this particular process that instantiate the random variables X_t corresponding to each month of the year, as shown in Figure 3.3.

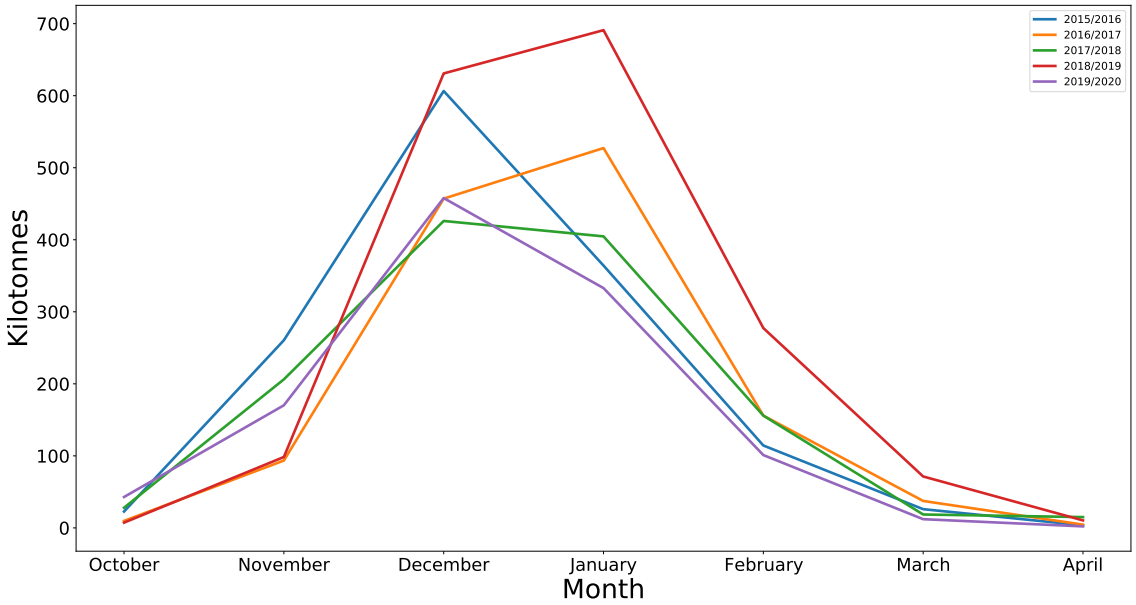


Figure 3.3: The TS corresponding to the olive oil production in Spain from 2015 to 2020. Each one of these TS can be considered a particular instance of the underlying stochastic process.

In an industrial environment, if we record the values of some sensors inside a system over a period of time we obtain a TS with the values of all the variables ordered from the oldest recorded to the most recent ones. This is a particular instance of the underlying stochastic process defined by that industrial system. A single TS can generate a dataset suitable for fitting some machine learning model, but ideally we would prefer several time-series from the same process. Usually, the variables inside stochastic processes have some kind of relationship between the previous instants and the next ones, although this is not always the case. For example, in the case of white noise where all the random variables of the process are sampled from a $\mathcal{N}(0, 1)$ distribution, they are independent and identically distributed (i.i.d.). This means that the values of some variable from previous instants are not relevant when forecasting that same variable in the future. Luckily, on real world scenarios

this is rarely the case due to the autoregressive component that most processes have. We say that the value of a variable in a process depends on the previous value of that same variable, or that it is first-order autoregressive, if:

$$X^t = \beta X^{t-1} + \epsilon_t, \tag{3.1}$$

where β is the autoregressive parameter that determines the relationship between X^t and X^{t-1} and ϵ_t is white noise. Additionally, defining a process as first-order autoregressive means that we assume that the next instant is calculated directly with the previous one, and older values prior to $t-1$ have no effect on the value of t. This assumption can be too strict for some processes, and it can be relaxed with the general formula of p-order autoregression:

$$X^t = \sum_{i}^{p} \beta_i X^{t-i} + \epsilon_t \tag{3.2}$$

This higher order autoregressive component allows modelling more complex behaviours in TS and will be relevant later on in this work. Along with this autoregression, we also find some of the most relevant features of TS analysis: trend and seasonality. These features define the behaviour of a TS and need to be addressed to properly model processes. We can represent a classical decomposition model for time-series analysis as:

$$X^t = m_t + s_t + Y^t + \epsilon_t, \tag{3.3}$$

where m_t is a slowly changing function that defines the trend, s_t is a periodic seasonal component that affects the TS and Y_t are the stationary residuals that we want to model. The stationary residuals have mean and covariance independent from time and contain either the relevant autoregressive information to model X_t in the manner of Equation (3.2) or, in the case of no autoregressive component, the mean and variance information that represents the distribution of the random variable X_t.

3.2.1 Trend and seasonality

The aforementioned slowly changing function m_t modifies the local mean of a time-series over time in an increasing or decreasing manner, and the seasonal component s_t is a function with a certain period that modifies our TS over specific spans of time. By removing both the trend and seasonality components in a TS, we obtain stationary residuals that can be used to fit zero-mean models. This procedure is very common in general approaches to modelling time-series, given that with this transformation we obtain a scaled dataset with apparently constant mean and computable covariances between the variables at different times. Some models require both components to be removed, but the modelling of the trend component will be a point of study on a later chapter of this work.

There are two popular approaches to identify and remove trends and seasonality in TS: to find an estimation of these components and remove them from the series or to differentiate

the series with lag-p operators. We can see the estimation in two different ways. The first one consists in assuming that both components can be estimated with a filter operator, and so applying, for example, linear smoothing with a finite moving average filter or some more complex filter like exponential smoothing can remove local trends and seasonality from TS. The problem with these filters is that choosing the appropriate one for a specific TS is not a trivial matter, and oftentimes requires previous intuition and testing several different ones to evaluate which filter performs better. On the other hand, we can assume that these trend and seasonality functions have some specific form, for example polynomial, and fit models for them through least squares regression. After we fit them, we subtract from the original series the trend and seasonality estimations obtained.

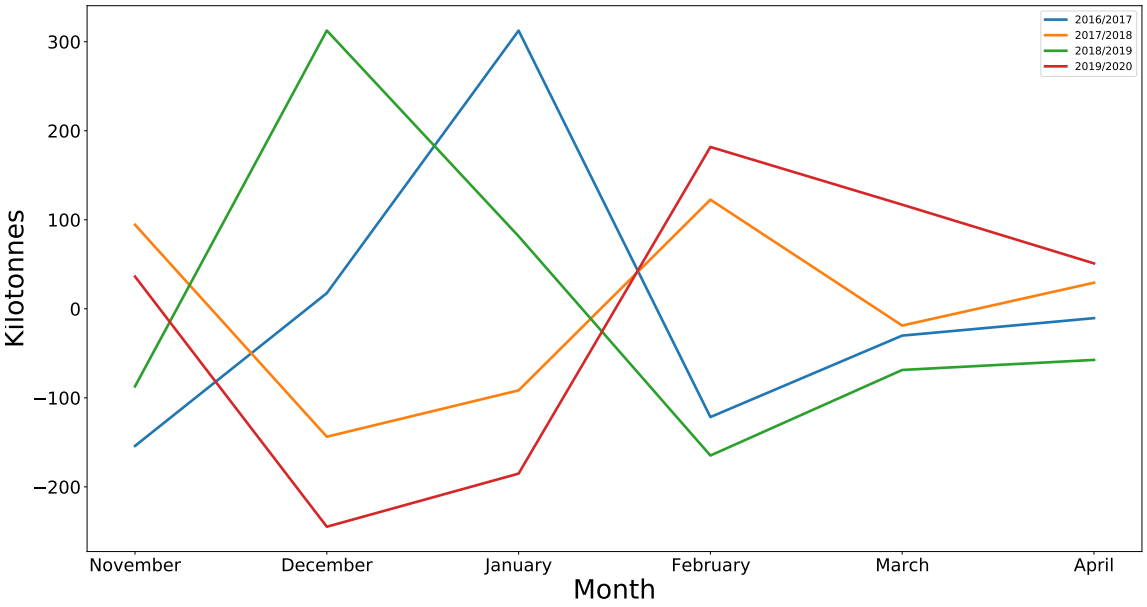


Figure 3.4: The resulting olive oil production time-series after performing lag-1 and lag-12 operators to remove both the trend and the seasonality in the data. We can see that the obtained residuals are close to stationary, which makes them suitable for being fitted by zero-mean models.

In the case of differentiation, we use the lag-p operator ∇_p to subtract the value at the last $t-p$ instant to the values of X^t:

$$\nabla_p X^t = X^t - X^{t-p} \tag{3.4}$$

Usually, lag-1 or lag-2 operators suffice to remove the trend component of a TS. In the case of the seasonality component, differentiation is also a very effective method. In that scenario, higher lag-p operators are useful to remove yearly effects in series spanning over several years. If we apply two consecutive lag-1 and lag-12 operators to the TS shown in Figure 3.3, we observe in Figure 3.4 and in Table 3.1 that the means and variances remain very similar between all years, which points to the residuals obtained being stationary or close to it. If we wanted to be completely certain, we would have to perform statistical tests

Year	Mean	Standard deviation
2016/2017	2.32	151.36
2017/2018	-1.30	95.03
2018/2019	2.68	156.57
2019/2020	-7.32	155.20

Table 3.1: Resulting means and standard deviations from the olive oil production TS after performing lag-1 and lag-12 operators.

to see if the differences in mean and variance are significant from one year to another, but it is worth noting that perfectly stationary residuals are seldom achieved with real world data. The differentiation method has the advantage that no prior knowledge of specific filters is needed and that it is easier to implement and apply.

3.3 Dynamic Bayesian network definition

In the presence of TS data, we need to make adjustments to the base BN framework in order to be able to use this data to fit a model. A BN is capable of modelling a number of random variables in a static manner, but a TS offers us the values that the different random variables of an underlying stochastic process took at specific points in time. For this purpose, we need to define dynamic Bayesian networks (DBNs) [Murphy, 2002; Koller and Friedman, 2009] capable of representing the effect that past instants have on the future.

Given that we are capable of representing JPDs over a set of random variables with static BNs, a first approach to modelling TS with BNs would be to partition time into instances defined by the frequency of the TS, which we will refer to as time slices, and then model each time slice with a static BN. This procedure takes into account the effects that the random variables have on each other inside a time slice. However, this representation disregards the temporal component by having disconnected time slices and it would only be useful for modelling processes that are not autoregressive. We illustrate this procedure in Fig. 3.5. As we can see, the random variables found in the stochastic process $\mathbf{S}$ can be modelled with a static BN in each time slice by using the data found in the TS $\mathbf{s}$ to fit the distributions. However, unless we introduce inter-slice arcs (shown as dotted arrows in the figure), past values will not have any effect on future ones. It is worth noting that in the example we kept the same BN structure for all time slices. We refer to this as homogeneous DBNs, and is not necessarily always the case, but for the sake of simplicity we will use homogeneous DBNs in the examples.

The inter-slice arcs can go from some previous time slice $t - i$, $i > 0, i \in \mathbb{N}$, to t. This allows any autoregressive order to be represented inside the DBN model. A benefit that this kind of arcs have is that, because they are only allowed from older to newer time slices, the addition of new inter-slice arcs cannot introduce cycles and invalidate the DBN structure. This concept will be useful later on in this work. To define the JPD of a DBN model, we now take into account all previous time slices up to some horizon T:

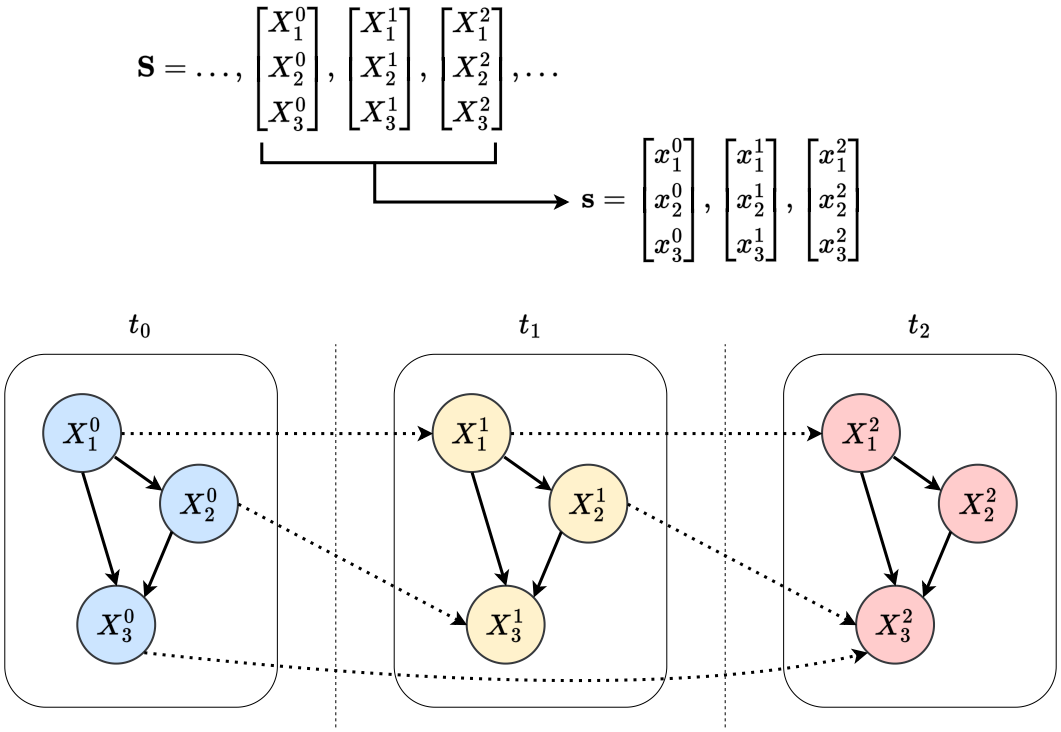


Figure 3.5: Representation of a DBN in relation with some stochastic process $\mathbf{S}$ and a TS $\mathbf{s}$ generated from it. We can model the relationships of random variables within the same time slice with intra-slice arcs similarly to a static BN, but we need the dotted inter-slice arcs to represent the autoregressive component of TS.

$$p(\mathbf{X}^0, \mathbf{X}^1, \ldots, \mathbf{X}^T) \equiv p(\mathbf{X}^{0:T}) = p(\mathbf{X}^0) \prod_{t=0}^{T-1} p(\mathbf{X}^{t+1} | \mathbf{X}^{0:t}), \tag{3.5}$$

where $\mathbf{X}^t = (X_1^t, X_2^t, \ldots, X_n^t)$ is the set of nodes at time slice t. In theory, this representation has all the needed components to model stochastic processes through TS data, but in practice it is computationally prohibitive to model all time slices from 0 to T. Two issues arise: an indefinite number of time slices depending on how long the TS are and a super exponential space of possible DBN structures. A stochastic process can contain an indefinite number of time instants and, subsequently, random variables. As we process longer TS, the number of time slices needed increases, and the space of possible DBN structures increases super exponentially with respect to the number of nodes [Robinson, 1977]. On a similar note, as the number of time slices grows, if we do not limit the autoregressive component, we can find ourselves with an intractable amount of arcs going from previous time slices to future ones.

In order to fix both issues, we need to introduce a simplifying assumption called the Markovian order. The Markovian order of the DBN defines that future time slices are independent of the past given k previous time slices. This puts a limit to the number of time slices that we model inside a DBN, and it simplifies the JPD computation from Equation (3.5) greatly:

$$p(\mathbf{X}^{0:T}) = p(\mathbf{X}^0) \prod_{t=0}^{k-1} p(\mathbf{X}^{t+1}|\mathbf{X}^{0:t}) \prod_{t=k}^{T-1} p(\mathbf{X}^{t+1}|\mathbf{X}^{(t-k+1):t}), \tag{3.6}$$

where m is the Markovian order of the network. Traditionally, the most common Markovian order for DBNs would be order 1, which is equivalent to modelling first order autoregressive stochastic processes. In that specific scenario, the computation of the JPD is simplified even further:

$$p(\mathbf{X}^{0:T}) = p(\mathbf{X}^0) \prod_{t=0}^{T-1} p(\mathbf{X}^{t+1}|\mathbf{X}^t) \tag{3.7}$$

We usually refer to DBNs with a Markovian order greater than 1 as high-order DBNs. Even though they are able to represent higher autoregressive orders than order 1 DBNs, learning their structure and performing inference on them poses a greater challenge. Additionally, one important advantage that the Markovian order offers is that it allows us to use a temporal window of size $m+1$ when processing TS data to fit our DBN models. This means that by fixing a Markovian order m, we need to fit $m+1$ time slices with their corresponding nodes, inter and intra-slice arcs. If we have one or several TS sequences $\mathbf{s}$ of length T, $T > m$, we can partition this sequence into several $m+1$ sized batches of consecutive values. These batches can then be used to fit our DBN models. In practice, this procedure allows DBNs to be fitted with several unequal length TS. This characteristic is of importance when applying DBNs to real-world problems, where the length of data from industrial processes can vary depending on circumstances outside of the system.

3.4 Learning

The issue of learning a DBN structure from data is similar to that of a static BN, but on a much bigger search space. In essence, each DBN structure has one or more static BN structures per time slice and the additional arcs from the inter-slice transition structure. Additionally, some authors learn an initial static BN structure (prior model) that represents the initial JPD of the process [Trabelsi, 2013]. Depending on the characteristics that we assume for our DBN model, we have different possible structures:

- We say that a DBN structure is homogeneous when all time slices t have the same BN static structure.
- We say that a DBN structure is stationary when the inter-slice arcs from time slice t to $t+1$ are always the same and do not depend on t.
- We say that a DBN is high-order when we fix a Markovian order higher than 1.

Depending on the specific characteristics of our desired DBN model some structure learning algorithms will be suitable or not. Broadly speaking, most DBN structure learning algorithms consist of adaptations of static BN learning methods to the dynamic case. Many can

be adapted by performing a two-step pass, learning first the intra-slice arcs of the network and then the inter-slice arcs. For this reason, we will maintain the score-based, constraint-based and hybrid categorization from Chapter 2.3.2.

3.4.1 Score-based methods

The score-based methods are a popular option in learning DBN structures. In essence, evaluating a network with some dataset via a score can be transferred directly from BNs to DBNs, and so the biggest issue raises from the huge increase in search space that comes with DBNs.

We can find adapted versions of the greedy hill-climbing algorithm that we covered in Algorithm 2.1 in works like Tucker and Liu [2003], van Berlo et al. [2003] and Wu and Liu [2008]. These modifications need to include in the set of available operators the addition or deletion of inter-slice arcs to model the dynamic structure. For scoring, we can use the same log-likelihood scores as for BNs, but we can find specific scores made for DBNs, like the mutual information score from Vinh et al. [2011] or loss functions derived around acyclicity constraints [Pamfil et al., 2020]. Some authors also opt to perform the search via sampling with Markov chain Monte Carlo methods [Wu and Liu, 2008; Robinson et al., 2010; Grzegorczyk and Husmeier, 2011].

In contrast with the local search procedure of the greedy hill-climbing, we have global search evolutionary algorithms. Methods like genetic algorithms [Tucker et al., 2001; Ross and Zuviria, 2007; Ashrafi, 2021] or particle swarm optimization [Xing-Chen et al., 2007; Santos and Maciel, 2014] have seen much interest due to their ability to move through vast search spaces. This characteristic works well with the super-exponential search space of possible DBN structures. A defining characteristic of these kinds of methods is that they need to define an encoding for their individuals, and this plays a key role in the performance of each algorithm. While some authors simply opt to use the binary adjacency matrices of DBNs as an encoding, which is prone to generating invalid individuals if the matrices are not checked for acyclicity in the equivalent DAG, other authors define specific data structures that, by definition, cannot represent invalid networks [Santos and Maciel, 2014].

3.4.2 Constraint-based methods

In the case of DBNs, pure constraint-based methods tend to not be as popular as other approaches. This is probably due to the fact that the amount of statistical tests that need to be performed can become a prohibitive issue for high numbers of variables or higher Markovian order. Even so, we can still find extensions of classic BN structure learning algorithms to the dynamic scenario, like the PC algorithm [Liu et al., 2021], the grow-shrink algorithm [Li et al., 2011; Naili et al., 2019] or the incremental association Markov blanket algorithm [Li et al., 2011; Haque et al., 2022].

3.4.3 Hybrid methods

In the case of hybrid methods that combine score-based and constraint-based characteristics, the dynamic max-min hill-climbing (DMMHC) algorithm from Trabelsi et al. [2013] is of particular relevance to this work. This method extends the max-min hill-climbing algorithm from Tsamardinos et al. [2006] to the case of DBNs. It performs two search phases, one for the initial network at t_0 and another one for the transition network in a first order Markovian fashion. It also proposes an extension to high-order DBNs, but the increase in complexity that comes with this kind of structures can degrade the execution time of the algorithm greatly.

3.5 Inference

In contrast with static BNs, DBNs need to be able to perform operations unique to TS data such as forecasting values up to a certain point T in the future based on some initial data. To be able to do this, we first need to make use of an inference method from Section 2.4. In our case, we will be using Gaussian DBNs, and so we will apply the exact multivariate Gaussian equivalent method in this work.

With a Markovian order k DBN, we can only represent k past instants of time and a future instant of unknown values. By providing the values of the variables in previous time slices as evidence in the DBN, we can predict the values of the future time slice with our multivariate Gaussian equivalent inference method. However, this procedure only allows us to predict a single instant into the future. In order to forecast up to horizon T, we need to either unroll the DBN into T future time slices by assuming homogeneity of the network and replicating the structure of the network, or use a temporal window. The drawback of unrolling the network is that we can end up performing inference on hundreds of unknown nodes all at once if we need to perform long term inference. On the other hand, using a sliding window approach will allow us to perform inference only on one time slice at a time, which is less time consuming and can be extended to any arbitrary horizon T.

In the baseline case of Markovian order 1, the procedure followed to forecast the multivariate TS with a sliding window is to provide it with some initial evidence of the nodes at time slice $t-1$ and predict the state of the nodes that conform the system at t. In the next iteration, the forecasted values of the nodes at t will be used as evidence for $t+1$. In the general case of arbitrarily large temporal windows, evidence is provided for all time slices except the present one. First, given the initial state vector $\mathsf{s}_0 = ((x_1^0, \ldots, x_n^0), \ldots, (x_1^t, \ldots, x_n^t))$ with values of the variables in our system observed at instant t and at many previous instants as defined by the Markovian order of the network, we perform inference to obtain the values $\hat{\mathbf{x}}^{t+1} = (\hat{x}_1^{t+1}, \ldots, \hat{x}_n^{t+1})$ that the variables are predicted to take at the next instant. After this, we move all the previous evidence forward in time. We forget the oldest evidence $\mathbf{x}^0$ from the system and introduce $\hat{\mathbf{x}}^{t+1}$ as the new evidence of the last instant to create the new state vector $\mathbf{s}' = ((x_1^1, \ldots, x_n^1), \ldots, (\hat{x}_1^{t+1}, \ldots, \hat{x}_n^{t+1}))$. This completes the current inference step, while $\mathbf{s}'$ is used as the initial state vector of the next inference step, predicting the

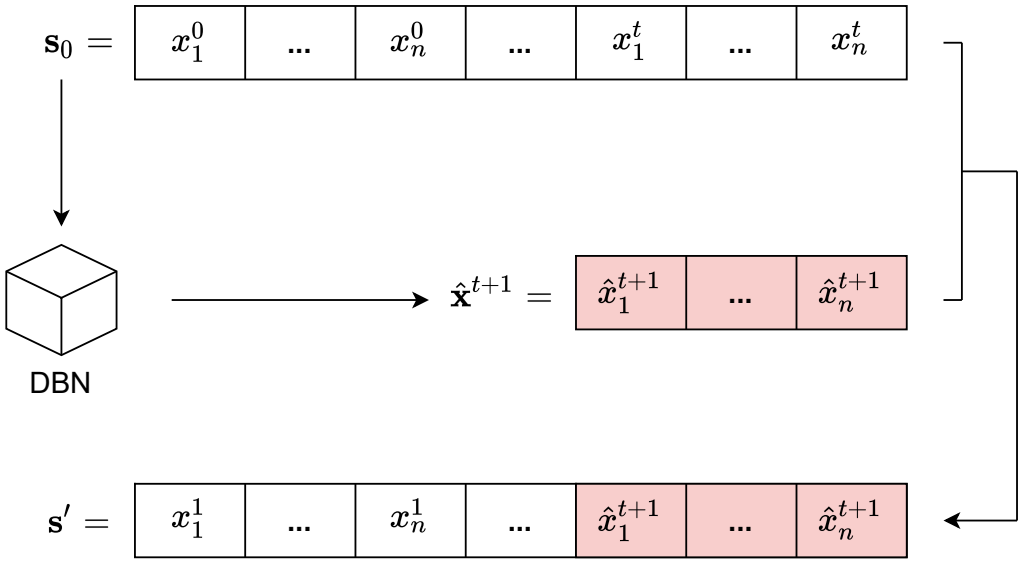


Figure 3.6: Schematic representation of the sliding window mechanism in an inference step. The new $\hat{\mathbf{x}}^{t+1}$ is predicted with the DBN model and is introduced into the state vector, removing the oldest $\mathbf{x}^0$.

next state of the system. This process is illustrated in Figure 3.6. We can perform as many inference steps as needed to reach the desired horizon T.

One advantage of DBNs is that forecasting can be performed with some kind of intervention in mind by fixing some of the initial evidence to some values that are to be tested. This way, we can use the DBN model as a simulator to test how modifying specific variable values will affect the future state of some system.

Another operation that can be performed with DBNs is what we call smoothing. This is an opposite function to forecasting and consists of, given data of the current state of the system, predicting backwards in time in order to see where could this data come from. This operation can be useful in situations with missing data, where predicting backwards can help filling gaps, or as noise removal to perform smoothing of TS curve profiles.

Part III

CONTRIBUTIONS

Chapter 4

Long-term forecasting of multivariate TS in industrial furnaces with DBNs

4.1 Introduction

As the first step in our work, we need to asses the effectiveness of DBNs in complex industrial environments. This is also an opportunity to propose the use of high Markovian order DBNs to model the tendency of TS better.

A common objective in industrial environments of this kind is to be able to forecast one or more of the variables that compose the system in order to optimize some of its aspects. The difference from the univariate case is that we must take into account not only the autoregressive component but also the influence that the TS can have on the other TS.

An additional problem that TS can have is the presence of seasonality and non-stationarity [Cheng et al., 2015]. In real-world situations, TS tend to be non-stationary because they have different trends or scaling variances as time t increases. This non-stationary component typically has to be identified and addressed in order to apply models such as the autoregressive integrated moving average (ARIMA) [Peña et al., 2011] to the data. The seasonal component, on the other hand, requires specific treatment depending on the kind of cycles present and their characteristics.

Inside an industrial furnace, one of the processes that costs a great deal of money in terms of efficiency loss is the deposition of solidified impurities of the fluid being processed in the tubes where it is preheated before entering the furnace [Pogiatzis et al., 2012]. This process is called *fouling*, and it forces the furnace tubes to be cleaned periodically. Fouling has been treated extensively in the literature with different techniques, ranging from physical models defining the process to more data-driven approaches. Some of the more classical methods focus on developing physical equation models that simulate the fouling effect inside the tubes. In Diaz-Bejarano et al. [2019], the authors simulate the growth of the fouling layer and its

thermal resistance with a physical model over long spans of time. A similar approach is used in both Pogiatzis et al. [2012] and Santamaria and Macchietto [2019], where the authors simulate the fouling effect with a physical model, but they reformulate it afterwards into a non-linear programming problem which they use to optimize the cleaning schedule that must be performed to remove the fouling layer over time.

An alternative approach to this optimization is proposed by Diaby et al. [2016], where instead of non-linear programming they opt to use a genetic algorithm to find the optimal cleaning schedule while simulating the fouling effect. A very popular alternative to these kind of models is a data-driven approach, where data from simulations or from operating furnaces is used to fit machine learning models to capture the fouling effect. In particular, neural networks (NNs) have found a lot of use in this area. A feed-forward network with three hidden layers is used in Radhakrishnan et al. [2007], where they forecast the temperature of the fluid inside the tubes over a span of two weeks. This temperature is also an indicative of the fouling effect over time, given that it will be harder to increase it as the fouling layer grows if no countermeasure is taken. Lalot et al. [2007] identify drifts in the temperature TS that are a result of the fouling effect and use a mixed approach where they either recommend forecasting with a recurrent two hidden layer NN when this drift is sudden and with a Kalman filter if the drift appears slowly over time. A simpler approach is proposed by Davoudi and Vaferi [2018] with a feed-forward dense network with 10 hidden neurons distributed in two hidden layers. Instead of forecasting the temperatures over time, they use the network to obtain an index that approximates the thermal resistance of the current fouling layer in a single instant. Sundar et al. [2020] propose a similar approach, but they apply an ensemble of NNs with two hidden layers to predict the fouling resistance based on the state of the system. A review of similar methods of fouling prediction in industrial furnaces is discussed in Wang et al. [2015]. In our case, this problem can be seen as one of multivariate TS that have a non-stationary component as they follow an evolving trend over time and a seasonal component of non-homogeneous cycles recovered from the furnace. We will focus on the approach of forecasting the temperature inside the tubes to represent the fouling effect over time.

Fouling causes a decrease in the thermal conductivity of the tube walls in the furnace. As a result, as the fouling layer grows, we have to increase the heat provided to maintain the temperature of the fluid constant inside the tubes. When the temperature of these tube walls rises above a certain threshold, the efficiency loses become too severe and a cleaning must be performed to remove the fouling layer. If we are able to predict the temperature that we need to have the tube walls at in the long term, we can estimate the number of hours left until the next cleaning must be performed. In addition, given that we want to gain some insight on the fouling phenomenon and how the variables in our system affect each other, we propose the use of Gaussian DBNs [Murphy, 2002] for long term forecasting of the system evolution. We will treat the variables inside the TS recovered from the sensors of the furnace as different nodes in our network and aim to model the conditional probabilistic independence relationships among them. Once we learn the structure of the network and its parameters from the data,

the dynamic part of the DBN will model the temporal component of the system. We can then forecast the state of the system and the temperature of the tube walls up to a certain horizon. With the resulting structure of the DBN, we are able to understand which sensors in the system are more relevant and how the variables interact with each other. As opposed to a black-box model, we have a clear picture of which variables influence future predictions. In addition, DBNs are a good tool for making long-term forecasts that take into account the evolution of the tendencies in TS, which is key in the problem at hand. To provide a comparison with another state-of-the-art model, we will also train a convolutional recurrent neural network (CRNN). NNs are very popular in the literature, and a CRNN will allow us to perform forecasting over different spans of time.

This chapter includes the content of Quesada et al. [2021b]. The data used is not made public due to privacy reasons, but the code used was compiled initially into the first version of the **dbnR** package.

Chapter outline

The rest of the chapter is organized as follows. Section 4.2 introduces the fouling problem. Section 4.3 explains the preprocessing performed on the data recovered from an industrial furnace. Section 4.4 is devoted to the learning and inference methods used for the DBN, the results obtained and the comparison with the CRNN. Section 4.5 concludes the chapter and describes future work.

4.2 Forecasting the temperature of an industrial furnace

Inside an industrial furnace, fluids are circulated through tubes and heated to some desired temperature to make them chemically reactive. In our case, the fouling effect degrades the heat capacity of the tubes over time by creating an insulating layer on their walls, as depicted in Fig. 4.1. The fouling degradation forces the preheat train to use more energy to attain the same temperature inside the tube, up to a limit point where the melting temperature of the tube walls is nearly reached. At this stage, the heating costs are very high as a result of the degenerated heat transfer coefficient, and the desired temperature of the fluid cannot be obtained because of this physical limit. To solve this problem, some cleaning of the insulating deposits has to be performed, either by physical or chemical means, after which the tube walls revert to their initial state and the fouling process begins again. This is what gives rise to the seasonality in the corresponding TS. In addition, because the fouling process depends on many different factors, the limit point is not reached after the same number of hours every time, which creates non-homogeneous cycles in the data.

Inside the furnace, there are four different sections of tubes being heated. These sections are grouped in pairs according to proximity, which means that their conditions are similar and they have some influence in their paired section. We show an illustration of the furnace layout in Fig. 4.2. To estimate the state of the system, sensors inside the furnace record hourly operational data of the temperature at different points, the pressure inside the tubes,

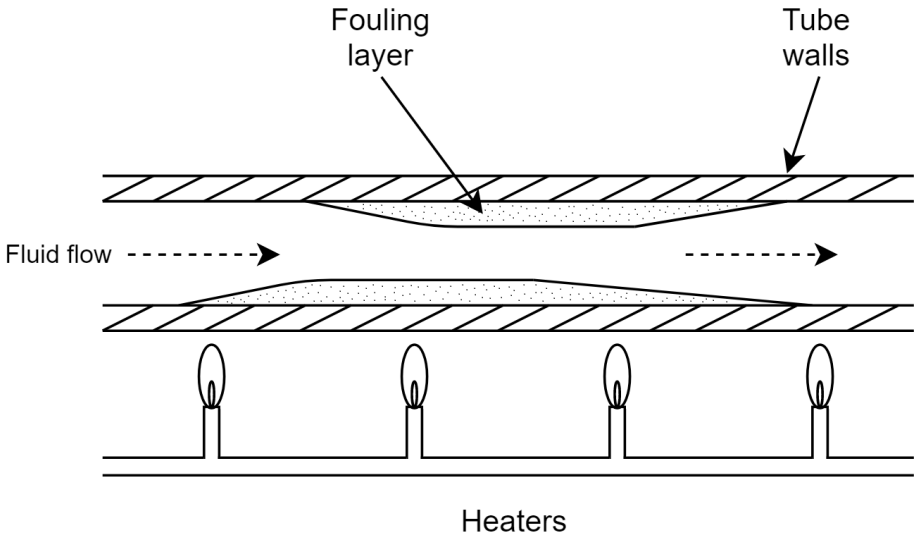


Figure 4.1: Schematic representation of the fouling effect inside a tube in the furnace. The fouling layer appears over time and has to be removed when the working conditions degrade past a temperature that would melt the tube walls.

the feed rate and the state of the heaters. Although fouling also depends on the composition of the fluid being processed, we do not have information about it. In this scenario, we take a data-driven approach by approximating the state of the fouling effect inside the furnace over time with only data related to the physical properties and then modelling the gradual degradation the furnace undergoes.

At section level, the cleaning of the tubes greatly affects the measurements of their sensors. It is important to note that while one section is in a cleaning period, the others are working normally. The main problem with this strategy is that cleaning a section may affect the state of its paired section, which is reflected as severe interventions on the TS as shown in Fig. 4.3. Given that the cleanings are planned in such a way that at most one section is not operational at any one time, the effects of this phenomenon are visible in nearly all cycles. These spontaneous interventions, and the usual noise and outliers typical in industrial data, have to be taken into account in the preprocessing of the data.

In this scenario, our objective is to be able to forecast the evolution of the temperature we have to provide to the tube walls during a cycle in a time window of 2000 hours. The end of a cycle is not only based on a threshold for the temperature of the tube walls but can also be cut short by the operators to avoid two or more sections undergoing cleaning operations at the same time.

4.3 Preprocessing

The multivariate TS input data were composed of the hourly recordings of the sensors that describe the furnace state over a time span of five years. This data came from several years of sensor readings inside an industrial furnace heating a fluid prone to fouling. This TS

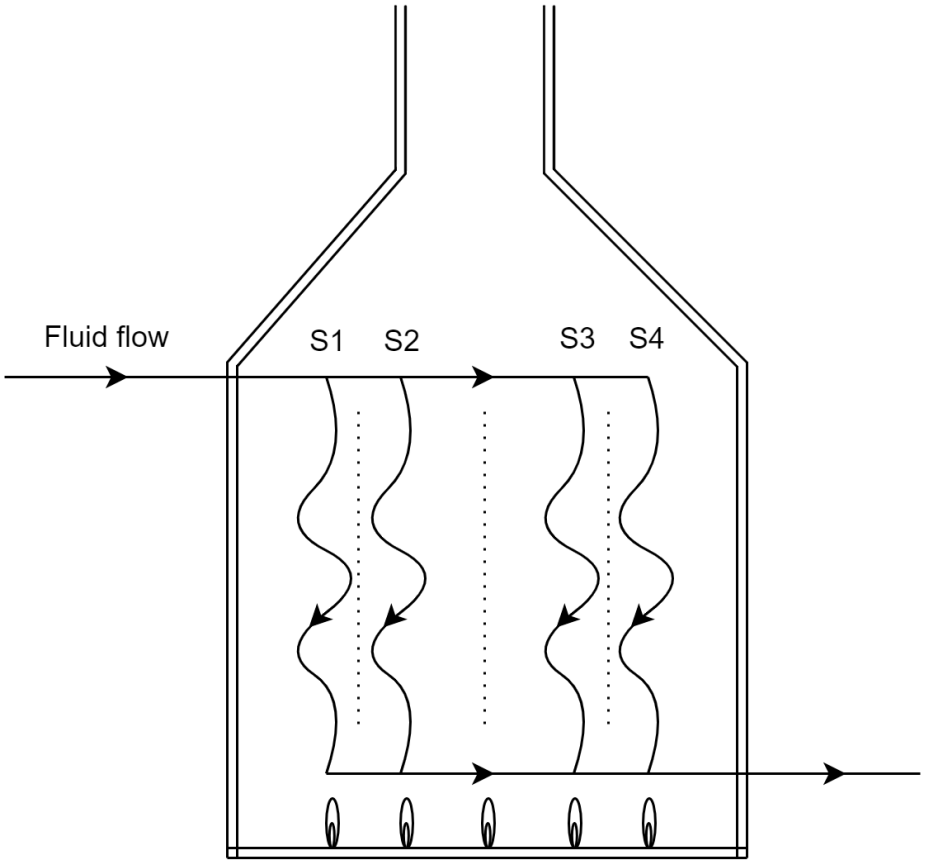


Figure 4.2: Illustration of the layout of the four different sections (S1, S2, S3 and S4) inside the furnace. When one needs to be shut down to perform cleaning operations, the others remain operational.

exhibited the typical characteristics of industrial generated data, such as noise, outliers and missing values. Prior to the modelling phase, the dataset had to be preprocessed to reduce its dimensionality and to address some of its irregularities, such as the effect of cleanings in other sections and noise in the signals.

4.3.1 Characteristics of the data

The input dataset consisted of 226 variables recovered from the same number of sensors inside the furnace. These sensors record data that describe the heating process, such as the flow pressure inside the tubes, temperatures at specific points and the state of the heaters. Some of these variables show seasonality caused by the cleanings performed on the sections, and others are unaffected by them. In addition, there are many variables with redundant information that come from sensors that record the same characteristic and are placed next to each other. This results in very similar TS or even copies of the same TS shifted in time.

Regarding the rows in the dataset, there are a total of 43,415 with a time difference of one hour between two consecutive instances. The sensors had different periods, and the shortest period was hourly. Thus, we reduced all of them to the hourly dimension in the recording

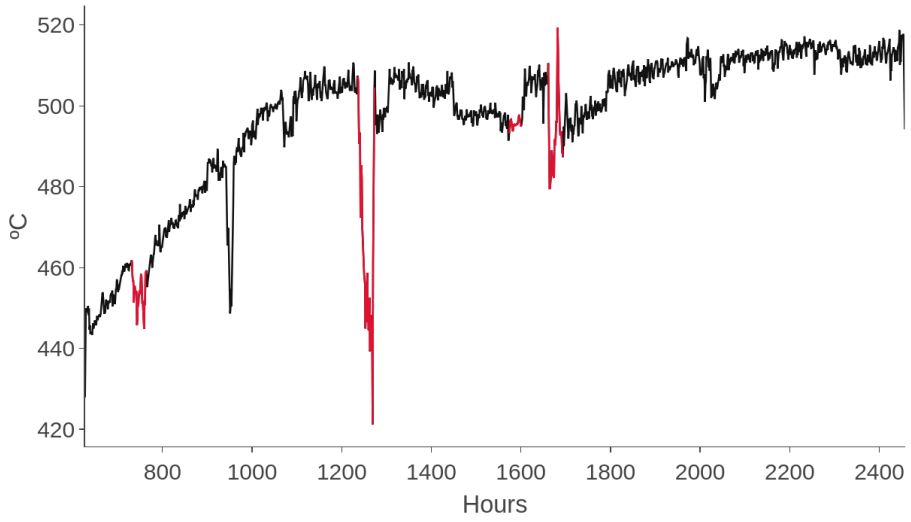


Figure 4.3: The heat supplied to the tubes during a cycle in one section. The periods in which cleaning is being performed in another section are shown in red. One of these cleanings severely affects the temperature of the tube.

process to avoid the problem of having mixed sampling frequencies in the TS [Andreou et al., 2010].

4.3.2 Cleaning and imputation

Many of the TS in the dataset had missing values due to sensor malfunctions or system shutdowns. In the span of five years, this is a common scenario that produces information loss. Before treating the TS, these missing values had to be processed either to accept the information loss or to try to impute the missing values. In our dataset, we encountered three different situations:

1. Sporadic missing values of less than ten hours. These cases were imputed using linear interpolation, because such short periods in a time span of 2000 hours should not create severe fluctuations, and they should have a mild impact on the subsequent training of the model.

2. Prolonged periods of approximately 100 hours of missing data. These are more severe cases with a greater impact on the information that a cycle can provide, given that cycles last 2000 hours on average. In this case, linear or polynomial interpolation could introduce much undesired noise into the model, so we decided to use ARIMA interpolation [Moritz et al., 2015]. This procedure fits an ARIMA model to the data immediately before the missing block and then tries to forecast the missing values.

3. Extreme cases where thousands of hours are missing. This happened in two columns of the dataset. In these cases, we opted to drop these variables entirely, as we could not

afford to impute such long periods and then use these synthetic data to fit a model and forecast from it.

Interpolations were performed in R software with the package **ImputeTS** [Moritz and Bartz-Beielstein, 2017].

In this case, seasonal-trend decomposition [Cleveland et al., 1990] could not be applied properly because the seasonal component was very irregular and did not show a clear period. This was aggravated by the fact that some cycles were stopped early by the operators of the furnace to avoid having more than one section on a cleaning period at the same time. When they expected that at the end of a cycle two sections would need cleaning, they stopped one of them early to keep at least three out of the four sections operational at all times.

Once the data was complete, some signal denoising and outlier smoothing were required, as some of the sensors corresponding to temperatures inside the furnace had clear trends but very noisy readings. In addition, we treated the effect of the cleanings in nearby sections as if they were outliers and smoothed them. For this purpose, we used Friedman's smoother implementation in the **forecast** R package [Hyndman et al., 2007]. This method consists of locally fitting linear least-squares regressions to outlier points of the TS with a temporal window around them to obtain a smoothed outcome.

4.3.3 Cycle treatment

In this case, the seasonal component is not *innate* to the process, but rather forced via human intervention to revert the system to a prior state. The information to model is the behaviour of a cycle over time without being conditioned by its ending point. A cycle always ends with a cleaning, and with this approach we can forecast when this cleaning will be mandatory due to the limit temperature being reached. Instead of treating the TS as a whole, we use the points where the cleaning interventions occur and divide the series into independent and identically distributed cycles, as shown in Fig. 4.4. The objective is to predict the temperature to be provided to the tube walls as the fouling process worsens over time. This information is important to the operator of the furnace to estimate when a section will need to be cleaned.

To achieve this, the cycles have to be identified and the dataset indexed by them. In total, there were 22 cycles.

For a DBN, given that the network has some Markovian order, we can adapt the full training dataset to learn the structure and parameters by dividing it into a set of i.i.d. cycles. Once it is divided into different cycles, we learn the parameters and the structure globally from all of them.

To learn the dynamic relationships among variables, we need to adapt our dataset so that each row contains the values of all the variables in an instant and in all of the previous instants that are needed for the desired Markovian order. To perform this operation, we use the following definition:

Definition 4.1 *Given a dataset D with M rows, we define a shift function that takes as input a column of D and an order O and returns its last $M - O$ rows.*

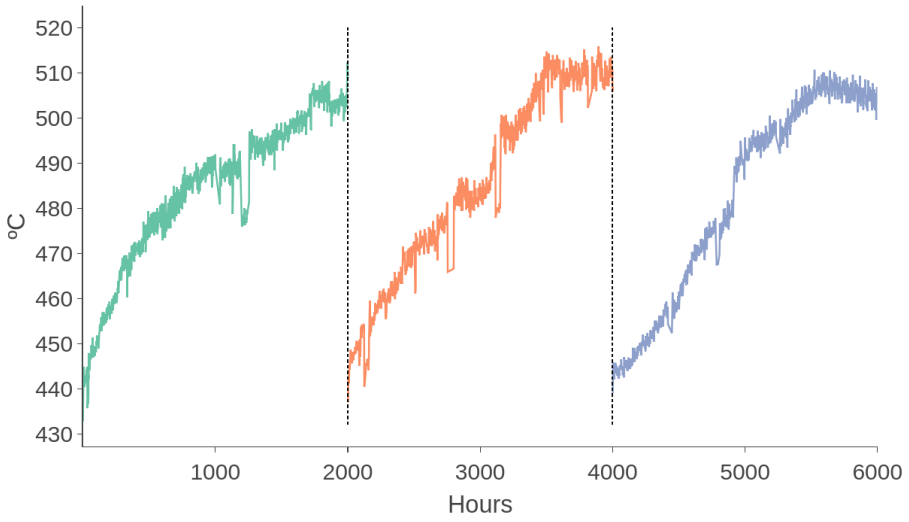


Figure 4.4: A segment of the temperature TS on the tube walls. Three different cycles are shown in different colours. The black dotted lines show the limits of each cycle. If we split the multivariate TS into these independent cycles, we obtain several i.i.d. instances of the same underlying process.

In practice, shifting a column in a dataset D with $O = 1$ means that all but the first row will be returned. When we combine a column C with its shifted version for $O = 1$, we obtain the value in each row at $t = 0$ and $t = 1$, effectively preparing that column to learn the influence of the previous time slice on the present. The shift function can also be used with higher orders to see the influence of older time slices. To obtain a dataset D with the columns shifted to the desired Markovian order k for a DBN, we need to apply the *shift* function to all the columns in D for all orders O in $1, \ldots, k$. An example of adapting a dataset to learn a Markovian order one DBN is shown in Fig. 4.5.

4.3.4 Feature selection

The motivation for selecting relevant features is that it is a means to reduce the dimensionality, and it is a way to detect which sensors are not providing sufficiently relevant information and which are providing redundant information. A posterior evaluation of the model can then decide whether the irrelevant sensors are worth keeping.

With 226 variables for each section, the resulting Markovian order one DBN would have twice the number of nodes. This situation is feasible for Gaussian DBNs, but the amount of redundant information is expected to be high. Many of the sensors record data about the same specific characteristic but at relatively close points in the furnace.

In this situation, TS clustering [Montero et al., 2014] was performed to group variables based on their behaviour over time. We used hierarchical clustering with the adaptive dissimilarity index distance [Chouakria and Nagabhushan, 2007]. This metric takes into account the proximity of the values in two TS and similar changes in the behaviour of both of them. To address the effect of having TS of different orders of magnitude, we performed max-min

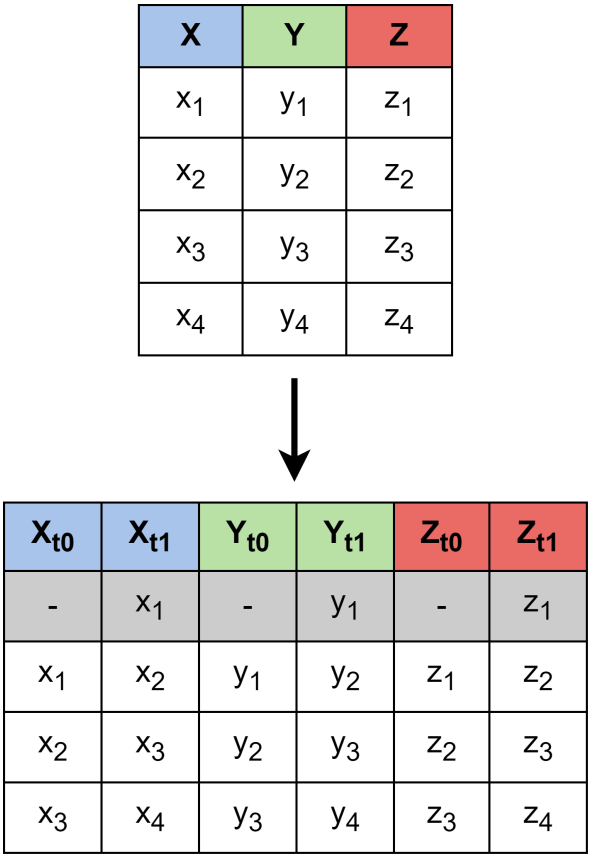


Figure 4.5: Adapting a dataset D to learn a Markovian order one DBN. All columns are shifted with $O = 1$ and added to the dataset. The grey row contains missing values and should be deleted.

normalization. Given that our objective was to filter out redundant variables, we established a conservative cut-off point for the clusters, as shown in Fig. 4.6. In this way, we make sure that selecting one variable as the representative of the cluster will not lead us to discard useful information.

As a result, we obtained 35 variables for each section. As expected, the redundancy among the sensors was very high, and there were clusters of five or more sensors that yielded TS with minimal differences between them.

4.4 Methods and results

To apply a DBN model, we must first define the structure learning algorithm and the inference method to follow. We will train multiple DBNs to compare the behaviour and accuracy of their forecasts. Afterwards, we will also train a CRNN to compare its performance with the DBN model.

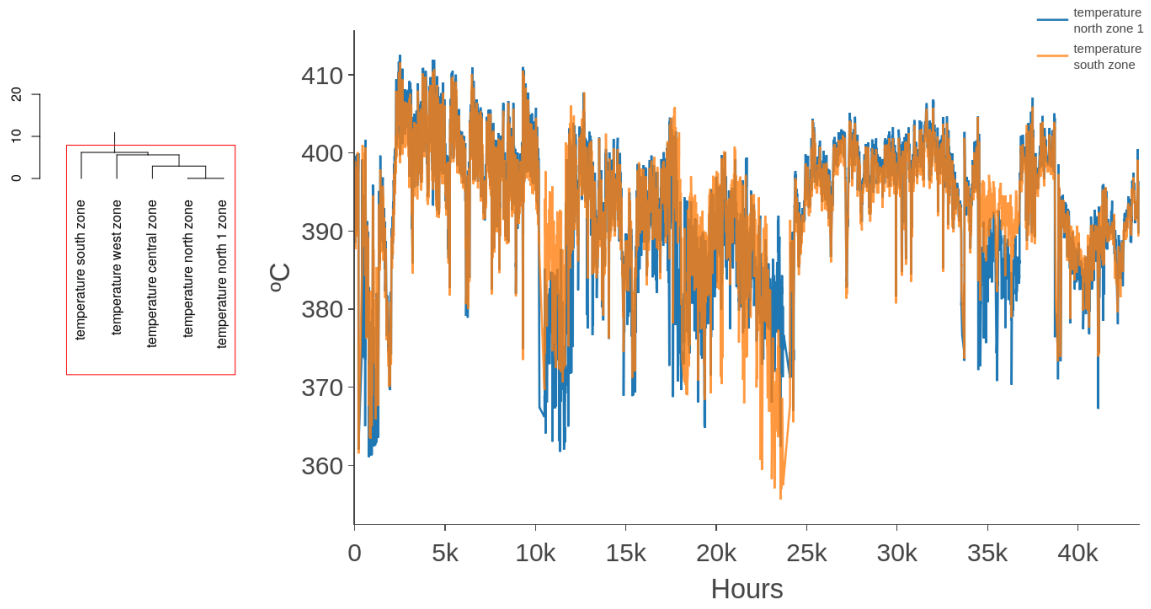


Figure 4.6: One of the clusters obtained from the hierarchical clustering. A comparison between the two most distant series in the cluster is shown. Variables inside the same cluster are mostly redundant TS with some variations in scale.

4.4.1 Structure learning

The structure of the DBN was learned with the adapted dataset and an adapted version of the DMMHC algorithm as explained in Trabelsi [2013]. To create Gaussian DBNs and perform forecasting for TS, we created an R package called **dbnR**[1], freely available in the CRAN repository, with which we implement all the phases of structure learning and inference and the visualization methods required. This package began as an extension of the **bnlearn** package [Scutari, 2010] to dynamic scenarios. To learn the structure of a DBN, we use bnlearn's MMHC implementation and partially adapt it by following the DMMHC algorithm. This is done in two steps: first, we learn the static structure of the network with the MMHC algorithm and then we learn the inter-time slice arcs of the net. Moreover, we added the option to modify the Markovian order of the network arbitrarily to adapt the autoregressive order empirically as needed. To our knowledge, there are no R packages that cover all these aspects of Gaussian DBN learning and inference.

4.4.2 DBN models

The differences among the networks we trained were mostly structural, relative to the size of the temporal window considered and the amount of initial evidence provided. In particular, we focused on creating DBNs with increasing Markovian orders, where we allowed arcs from a time slice to any more recent one. By this means, we can take into account longer temporal

[1] https://CRAN.R-project.org/package=dbnR

lags, where variables in earlier states of the system may be able to add relevant information to the present state.

Increasing the Markovian order results in more complex models in terms of the number of nodes and arcs, and we aim to trade off the efficiency costs with better forecasting accuracy. If we overestimate the Markovian order, the learning of the structure and parameters from the data will take too long and the predictions will degrade due to overfitting.

We also added a variable representing the time since the beginning of the cycle. This gave the model an idea of the state that the system should be in based on how long it has been running. As a result, the tendency was better captured.

It is important to note that we decided not to normalize the TS before using them to fit the DBN model. The only visible difference when normalizing was in the values of the weights inside the linear Gaussian CPDs, given that those coefficients had to account for the differences in scales between the parent nodes and their children. On this note, we also did not standardize the TS to remove their tendencies. The reason is that we aim to model these tendencies with the Markovian order of the networks. As we provide the model with more initial evidence, we find that the future tendency of the TS is captured better in the forecasts.

The initial resulting Markovian order one network has 70 nodes and 466 arcs without limiting the number of parents of each node in the DMMHC algorithm. It is a dense network, but feasible in terms of exact inference in a DBN. Each order increase adds 35 new nodes and all the new arcs that appear from the new time slices to more recent time slices.

Once the structure of the DBN and its parameters are learned from the training data, we need to make inference on it. In inferring, we need to provide the DBN with some evidence to forecast the most likely state of the system over the coming hours. In the dynamic case, the evidence from past time slices should be used to predict the next time slices. Depending on the Markovian order of the DBN, the amount of evidence needed for the past time slices will vary. To perform inference, we implemented the exact inference method by calculating conditional probabilities in the equivalent multivariate Gaussian.

4.4.3 Convolutional recurrent neural networks

In order to compare the results of the Gaussian DBN with another typical model in the literature, we will fit a CRNN. NNs have been widely applied in similar problems, and the are shown to be a powerful tool for predicting the fouling phenomenon. In our case, we need our model to be recurrent in order to be able to forecast TS of variable length.

We will take a similar approach to the case of the DBN model by providing the network with a temporal window of the 24 previous hours of operation and predicting the next 24 hours. Afterwards, the predictions will become the inputs of the network to predict the next window. Similarly to the DBN case, we will predict the whole state of the system rather than only the temperature to be able to use the predicted state as input. To train the model, we will use the same preprocessed dataset, but in this case we will normalize the data. Normalization is a very common procedure with neural networks, and we saw a clear improvement of the training loss after we applied it. This model is coded in Python 3.8 using

TensorFlow and **Keras**, and the code is available in a GitHub repository [2].

4.4.4 Results

Among the 22 cycles available in the dataset, two of them were degraded and had to be removed. The first degraded cycle lasted only 598 hours, while the average cycle length was 2073.05 hours. This is because the sensors only started collecting data at the end of this cycle. The other degraded cycle had a length of 787 hours and was cut short by the operators of the furnace to avoid having two sections undergo a cleaning period at the same time. After removing those cycles the 20 remaining cycles ranged from 1046 to 3635 hours.

With these 20 cycles, we prepared a 5-fold cross-validation experiment, leaving 16 cycles in each execution for training and 4 for testing. The model is trained and tested for all the folds and the mean absolute error (MAE) results are then averaged. Given that the cycles have different time spans, the forecast length is the same as that of the test cycle each time, up to a maximum of 2000 hours. The resulting errors of the different models are shown in Table 4.1. For the error metric, we calculated the MAE as:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |x_i - \hat{x}_i|}{s} \tag{4.1}$$

where x_i are the real values of the TS, $\hat{x}_i$ are the predictions and s is the total number of samples. The MAE measures how much the forecasted curve differs from the real profile of the cycle. A high MAE indicates that the forecasts differ greatly from the real behaviour of the furnace, either by over- or underestimating the temperature rise. If we miss the ending point temperature by too many degrees, this could mean that we estimate that a cleaning will occur several days or even weeks away from the real date on which it has to be performed. For this reason, the MAE of the model should be below 20, as a larger error could mean estimating the ending point of a cycle some days later than it should be.

The results show that the error decreases as the Markovian order increases up to order 4, as shown in Fig. 4.7, but the training time of the networks increases sharply and the BIC score of the networks decreases due to the increase in the number of nodes and arcs. Up to a Markovian order of 4, the accuracy of the forecasts continues to improve due to the better performance of the DBN in learning the different tendencies that can appear in the data. By giving the network more past time slices and having the present time slice depend on them, the initial evidence given to the network helps it decide whether the tendency of the series will grow more sharply, as shown in Fig. 4.8. At Markovian order four and greater, increasing the Markovian order decreases the forecast accuracy due to the accumulated noise and overfitting, which increases because the network is extended backwards by many time slices.

Once we chose the Markovian order four DBN, we assessed how the average error in the forecast varies as time increases. The majority of the cycles exhibited a low MAE in the first days of predictions and then an increase. Then, the error diminished progressively as

[2] https://github.com/dkesada/kTsnn

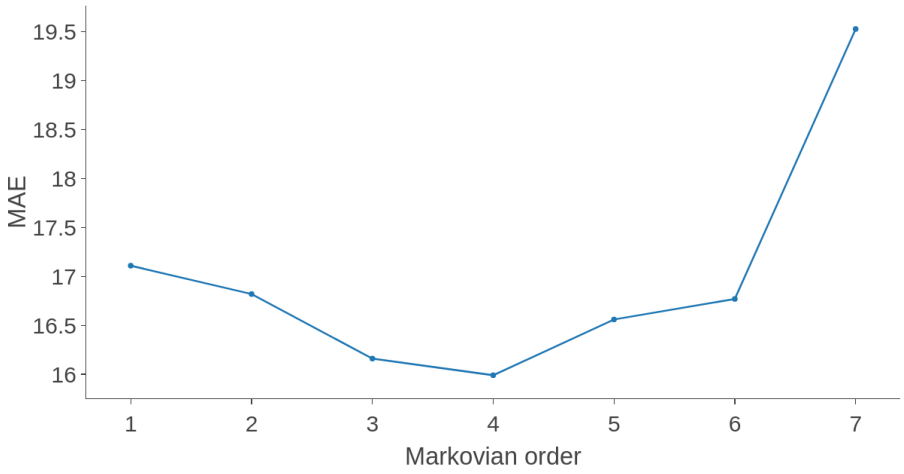


Figure 4.7: As the Markovian order of the trained networks increases, the error in the forecasts decreases to a point where the networks start overfitting and accumulating error.

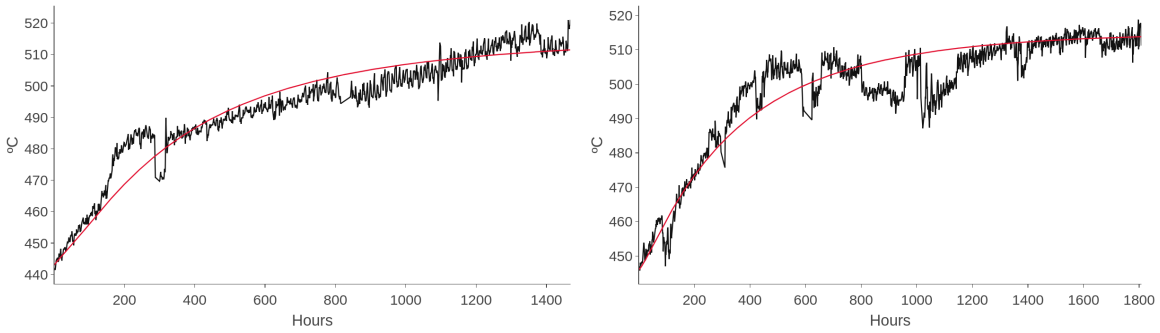


Figure 4.8: Resulting predictions of a Markovian order four Gaussian DBN for two cycles with different characteristics. The black line represents the real temperature TS while the red curve is the estimation of the DBN. Note how the predictions adapt to the observed tendency in the initial evidence provided. The predictions are extended to the full length of the cycles, with the first being an example of a cycle that ends early.

the cycle reached its ending point. This behaviour can be seen in Fig. 4.9. This indicates that the model is appropriate for short-term predictions as well as for predicting the end of a cycle's life, which was our original objective. On the other hand, the predictions are less accurate around the 400-hour mark, where many of the predictions behave uniquely due to the interventions of the furnace operators. Our objective of predicting the evolution of the series in the long term is fulfilled by predicting what the initial growth of the series will be in the short term and what temperature the series will have by the end of the forecast.

The inference process maintains a good execution time overall, and the bottleneck is the learning of the structure. It is important to note that, rather than an exact prediction, we obtain a probability distribution that defines the estimate of the variable, and in our case we take the mean as the most probable value in calculating the metrics. In Fig. 4.8, we plot the

Table 4.1: DBN forecasting results

Order	BIC	Training time	Exec. time	MAE
1	-3.56e6	4.72m	4.6s	17.11
2	-4.2e6	9.35m	7.63s	16.82
3	-4.83e6	20.31m	9.84s	16.16
4	-5.46e6	43.88m	12.8s	15.99
5	-6.09e6	1h 32m	16.54s	16.56
6	-6.71e6	3h 7m	20.81s	16.77
7	-7.33e6	6h 39m	25.62s	19.53

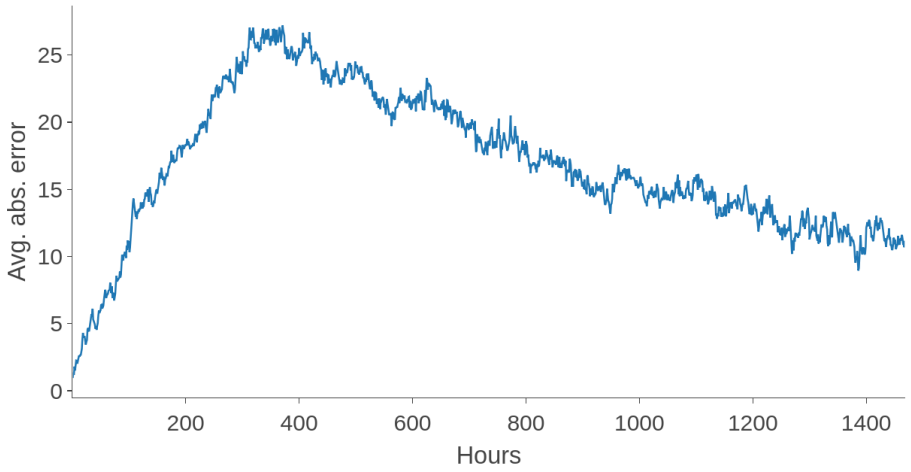


Figure 4.9: Average absolute error over time of forecasting with the Markovian order four network. The absolute error is low in the first days and then increases due to the cycles individual behaviours. As the ending point of the cycles approaches, the error decreases.

mean of the distribution in each time slice.

If we examine the network, we can see which variables directly influence the value of the average oven temperature. In Fig. 4.10, we can see an example of our visualization tool with a Markovian order four DBN. We can see that the time since the last cleaning and the pressure of the gas in the heaters, among other variables, appear as parents of the temperature, as well as the temperature in the last four temporal instants. With the time since the start of the current cycle we model the state of the fouling effect in the system, and the gas pressure directly influences the heat transferred to the tube walls. The appearance of the flow of fuel administered to the oven heaters four hours previously as a direct parent of the temperature indicates possible past interventions on a four hour-basis. With this structure, we can also simulate scenarios to see how the variation of certain temperature variables will affect the system in the future.

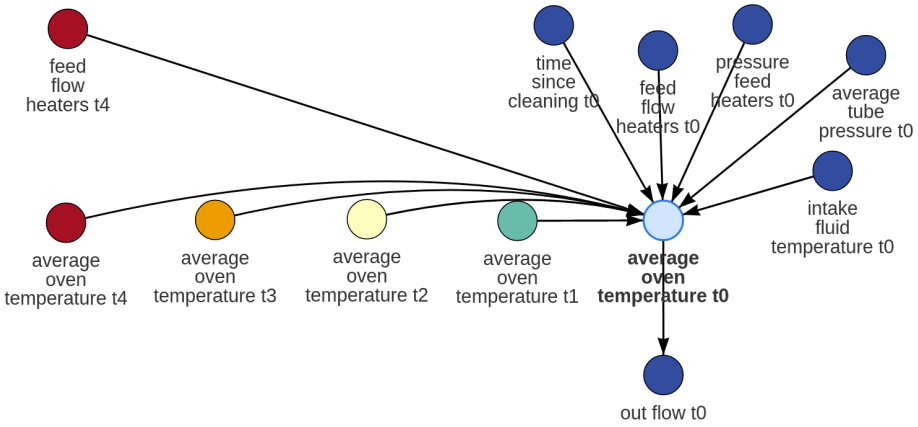


Figure 4.10: A screenshot of the visualization tool included in our package showing the parent and child nodes of the objective variable (cyan) in the Markovian order four DBN.

The results of the Markovian order four network prediction over the test cycles show it to be a powerful tool in forecasting the profile that the temperature curve will have over the coming months. By providing it with four hours of previous evidence of the behaviour of the furnace, the operators of the plant will be able to gain information on the life expectancy of the current cycle and plan the cleaning of the tubes accordingly.

In contrast, we can see the MAE results when forecasting in short, mid and long-term with the CRNN model in Table 4.2. The network averaged 44.7 minutes of training time on 300 epochs. It shows exceptionally good results on short and mid-term forecasting, but it degrades rapidly on the long-term. One interesting contrast that can be seen in Fig. 4.11 in comparison to forecasting with DBNs is that the profile of the curve it predicts is less smooth, providing a prediction that looks closer to real data. This is due to the DBN predicting the expected mean over time, not the exact value. The real value is expected to fall close to the mean in an interval defined by the variance. The results for short and mid term forecasting with the CRNN are better than the DBN model, but the long-term forecasting degrades rapidly. In the scenario of finding the expected remaining useful life of each cycle, the DBN model is able to give us a good estimate, while the CRNN is able to predict the near future more accurately.

Table 4.2: CRNN forecasting results

Time span	Exec. time	MAE
50h	0.98s	4.63
100h	1.6s	7.05
250h	3.4s	14.01
500h	6.57s	47.43

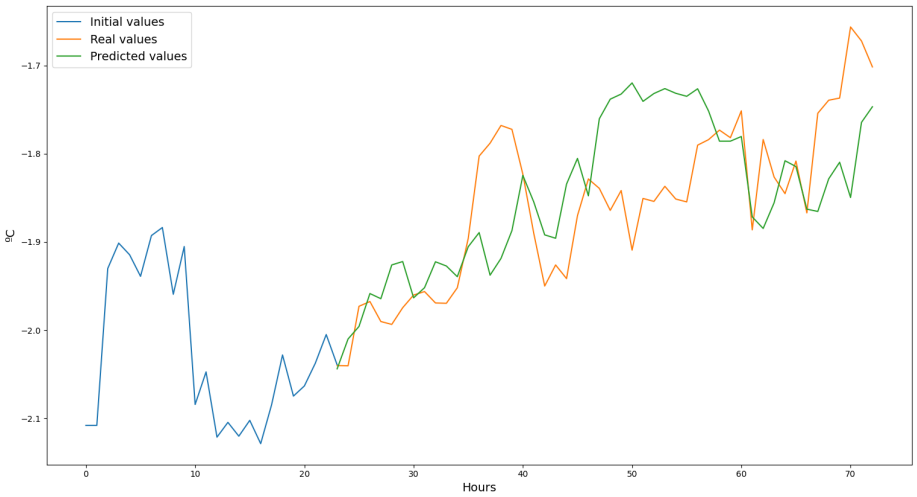


Figure 4.11: Results of giving the CRNN the first 24 hours of a cycle as inputs and forecasting the next 50 hours. The forecasting returns an accurate profile in the short term. As opposed to the DBN, this model is normalized with the mean and variance of the training dataset.

The decrease in accuracy on the long-term is very likely due to lack of data to fit CRNNs that are able to predict a longer span of time. This type of neural network models are very powerful for either static predictions or forecastings, but require enough previous samples to be able to converge to a stable state. In our case, generating new data takes months of continuous operation, and due to the typical noise or unforeseen problems that can arise in industrial applications, not all of this data recovered from sensors can be salvaged afterwards. From our point of view, a model for short or mid-term forecasting with CRNNs that can be fitted with reasonable sized datasets is a very useful and powerful tool for black-box forecasting of this kind of TS to make decisions that affect the near future. Moreover, instead of seeing DBNs and CRNNs as exclusive choices of fouling models, it can be argued that they can complement each other in different scenarios. By fitting a DBN model that is specifically tailored to perform long-term forecastings, we can combine it with a CRNN that excels at shorter forecastings and obtain the benefits of both. We can combine the interpretability of the DBN to improve both models at the same time and the efficacy of the CRNN to perform accurate forecastings and decisions about the near future. Forecasting the temperature over a long span of time with the DBN can give us an idea of the remaining useful life of our current cycle, and performing short-term forecasts with the CRNN as time passes can give us a clearer idea of how the profile of the curve is going to change in the near future and can help us adjust our decisions within the expected time frame that the long-term forecasting provided us.

4.5 Conclusions and future work

Although only the physical properties of the process of interest were available, our DBNs were capable of making long-term predictions with an acceptable MAE while our CRNNs were able to make short and mid-term predictions with high accuracy. The strong outliers of the cleanings present in the dataset did not have a severe impact on the forecasting, and the Markovian assumption for the network helped in modelling the tendency of the series. However, increasing the order of the DBN drastically increases the learning time of the structure and decreases the BIC score, so a compromise must be found between the desired accuracy and the complexity of the model.

The reduced number of cycles resulted in the Markovian order one DBN models learning the most common tendency. Given that the DBN did not have enough evidence to discern between tendencies, all forecasts tended to the same curve profile, incurring in greater error. On the other hand, when we increased the Markovian order of the network, we allowed the model to identify the tendency in the previous time slices. This resulted in much better forecasts and greater robustness in cases with seasonality, where not all the cycles present the same characteristics. In comparison, the CRNN model used was able to provide very accurate predictions of the near future, but failed to predict spans of thousands of hours. We proposed a combination of both models, where we could take advantage of the strengths of both of them without having to resort to bigger datasets for fitting more complex models.

The structure of the network helps us understand the interactions among the variables inside the furnace. Inspecting the structure, we can see which variables make the objective temperature conditionally independent of the rest of the variables of the oven. This can help us decide which of the sensors are more relevant, and it allows the operator to see which variables affect the forecasting directly and the influence of each of them. In contrast to a black-box model, the structure of the network offers an explanation of how the model makes forecasts. The individual effect of a certain variable on the objective temperature can be checked in its corresponding node in the network, which helps in discovering new information on the relationships between the variables inside the furnace.

We have shown that DBNs are useful models when treating TS in industrial environments, and we have provided the tools to apply them to specific scenarios and visualize the results. The comparison with the popular NN models in the literature also showed some interesting interactions between both of them and the possibility of combining the two models. In the future, we would like to address the issue of automatically finding the optimal Markovian order of a DBN to make the model easier to deploy. To accomplish this, we want to explore the relationship between the Markovian order, the autoregressive order in the ARIMA family of models and the tendency of the TS. We would also like to apply DBNs in cases where the chemical properties of the fluid are available in each cycle, as we would be able to extract more relevant conclusions related to the needed temperature given a specific composition. The last issue we would like to tackle is a more in-depth hybrid model between Gaussian DBNs and CRNNs, where the initial forecasting is performed with the CRNN model and its

outputs are fed to the Gaussian DBN model for long-term forecasting. This hybrid model could potentially produce better results than the use as separate tools.

Chapter 5

Structure learning of high-order DBNs via PSO with order invariant encoding

5.1 Introduction

After applying high-order DBNs to the industrial problem of fouling, the issue of increasing execution times for structure learning algorithms became highly apparent. The base DMMHC algorithm is very effective for Markovian order 1 networks, but as the number of variables and time slices increase, it becomes unfeasible to learn a graph with it. To tackle this issue, we will propose our own structure learning algorithm specific for high Markovian orders.

Meta-heuristic optimization algorithms can be applied to structure learning by searching over the space of possible network structures and assessing the fitness of each solution with some score. In the case of particle swarm optimization (PSO) algorithms [Kennedy and Eberhart, 1995], they offer a powerful solution for big search spaces but require them to be continuous and real numbered, and the space of possible BN structures does not fulfill these requirements. To fix this issue, some authors have translated the space of possible BN graphs into a continuous one over which PSO can be applied [Liu and Liu, 2018]. Many other authors [Du et al., 2005; Gheisari and Meybodi, 2016; Santos and Maciel, 2014; Xing-Chen et al., 2007] have opted for translating the operations of the PSO algorithm to perform discrete movements and then be able to apply the framework of PSO to BN structure learning. In particular, Du et al. [2005] propose to encode particles as binary adjacency matrices that are modified as the particle moves to represent additions and deletions of arcs. Santos and Maciel [2014] extend this concept by defining particles as lists with the parents of each node, and so a particle moving means adding or deleting parent nodes. However, these approaches become less efficient as the number of nodes and the Markovian order increase, and the correct order is assumed to be known beforehand. We will expand on both of those methods by defining an encoding for particles that is unaffected by the Markovian order and allows for more efficient

searches in larger spaces.

This chapter includes the content of Quesada et al. [2021a]. In addition, all code, data, and results are available at https://github.com/dkesada/natPSOHO.

Chapter outline

The rest of the chapter is organized as follows. In Section 5.2 we explain how to deal with the concept of high-order DBNs in PSO. Section 5.3 contains the details of our order invariant DBN structure encoding into PSO particles and their operators. Section 5.4 shows the empirical results. Finally, Section 5.5 concludes the chapter and gives some final remarks.

5.2 Background

In the field of BN structure learning, one family of methods is dedicated to applying PSO to move through the space of possible structures to find an optimal solution. Depending on the type of BN that we want to model, encoding the individuals and moving through the solution space can be done in different ways. In our case, we will center our attention in DBN models and translating the PSO operations to the discrete space defined by our encoding of a particle.

5.2.1 High-order dynamic Bayesian networks

We call a high-order dynamic Bayesian network (HODBN) a DBN model where the first-order Markovian assumption is relaxed and the network is represented with more than two time slices. To leverage the increase in complexity of this kind of model, we can restrict the arcs in the network so that they can only be directed to nodes in the most recent time slice, in our case $\mathbf{X}^0$. This kind of DBN structures are called transition networks [Santos and Maciel, 2014] and they avoid by definition any kind of cycles, which simplifies the search. The space of possible structures that transition networks allow is also much smaller than that of regular DBN models. To prove this, let $\mathcal{G}$ be a DBN network structure and $\mathcal{G}'$ be a transition network, both with the same number of nodes n_0 per time slice, total number of nodes n and Markovian order k. Let $\mathbf{D}$ be the set of all possible inter-slice arcs in $\mathcal{G}$ and $\mathbf{T}$ be the set of all possible arcs in $\mathcal{G}'$. We can calculate the number of possible DBN structures g^{DBN} as the different combinations of the elements in $\mathbf{D}$, that is, the different combinations of all the possible arcs in the network:

$$g^{DBN} = \sum_{i=0}^{|\mathbf{D}|} \binom{|\mathbf{D}|}{i} = \sum_{i=0}^{|\mathbf{D}|} \frac{|\mathbf{D}|!}{i!(|\mathbf{D}| - i)!}. \tag{5.1}$$

By definition, $|\mathbf{T}| \leq |\mathbf{D}|$ because even though both have the same number of nodes, the arcs in $\mathbf{T}$ are restricted in a way that satisfies $\mathbf{T} \subseteq \mathbf{D}$. If we apply Equation (5.1) to calculate the number of possible transition networks g^{TN}, we can see that:

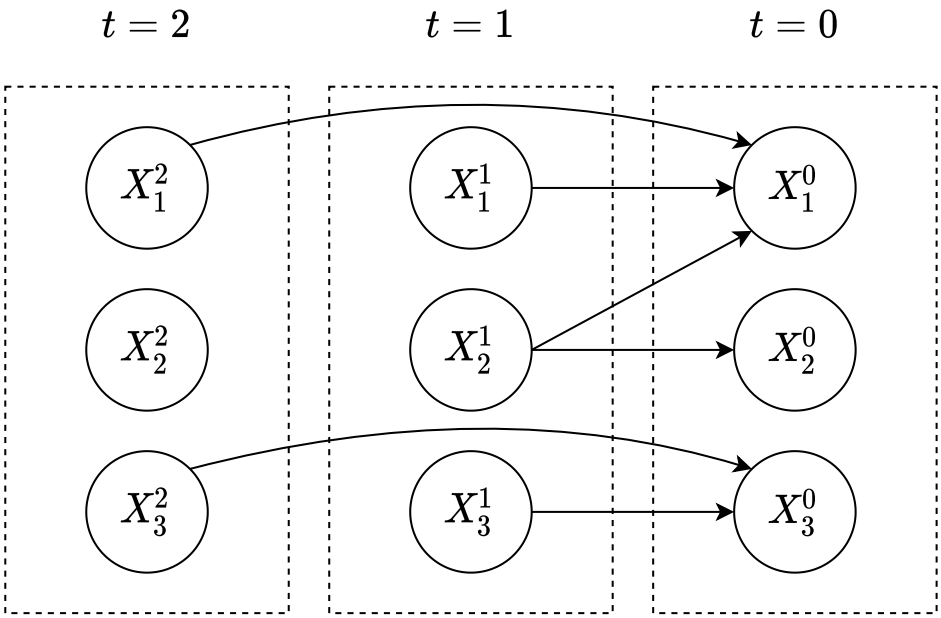


Figure 5.1: An example of a Markovian order 2 transition network with three nodes per time slice. Only arcs directed to t_0 from earlier time slices are allowed.

$$|\mathbf{T}| < |\mathbf{D}| \implies g^{TN} << g^{DBN}. \tag{5.2}$$

If we would also take into account the possible intra-slice arcs in $\mathbf{D}$, the inequality in Equation (5.2) would be super-exponentially bigger. Moreover, increasing k by one translates into adding only n_0^2 arcs in the case of the transition network, as we only allow arcs from the new n_0 nodes to the nodes in $\mathbf{X}^0$. This means that increasing k by one increases $|\mathbf{T}|$ by the constant n_0^2. On the other hand, this operation means adding $n_0 * n$ new inter-slice arcs in the case of a regular DBN, which increases $|\mathbf{D}|$ exponentially with each increase in k. This shows that not only $|\mathbf{T}|$ is always smaller than $|\mathbf{D}|$, but it also increases drastically slower when we increase the Markovian order of the network. For both the lack of cycles in transition networks and their reduced, although still vast, space of possible structures we will be using this kind of network in the rest of the chapter. An example of a HODBN with the restrictions of a transition network can be seen in Figure 5.1.

5.2.2 Particle swarm structure learning

The PSO algorithm [Kennedy and Eberhart, 1995] is a meta-heuristic technique that simulates a swarm consisting of n particles moving in a k-dimensional space to find the optimal solution. Particles can be defined as $\mathbf{Pr}_i = \{P_i, V_i, P_l, P_g\}$, where P_i is its current position, V_i is its current velocity, P_l represents the best position found by the particle so far and P_g is the best position found by the whole swarm. A position represents one specific solution of the optimization problem, and the velocities modify these positions, allowing the particles to move in the solution space. To calculate in each iteration t the next position P_i^{t+1} and the

next velocity V_i^{t+1} of each particle, the following updating rules are applied:

$$V_i^{t+1} = wV_i^t + c_1 r_1(P_l - P_i^t) + c_2 r_2(P_g - P_i^t), \tag{5.3}$$
$$P_i^{t+1} = P_i^t + V_i^{t+1}, \tag{5.4}$$

where $w, c_1, c_2 \in \mathbb{R}$, w is the inertia factor of the last velocity, c_1 is the factor that weighs the importance of the local best position, c_2 weighs the global best position and r_1 and r_2 are two real numbers sampled uniformly from the interval $[0, 1]$. One of the key factors in PSO is the pondered effects that the global and local best positions have on the current velocity of a particle, which can change in each iteration if a particle finds a position that has a better score than P_l or P_g. The inertia factor defines how much importance is given to the random search of each particle, increasing or decreasing the exploratory capabilities of the swarm. A higher inertia factor will mean that the $t + 1$ velocity of the particle will be very similar to the one it had in the previous instant t.

Although PSO was originally designed for continuous and real valued spaces, there are adaptations to discrete scenarios. In particular, the approach established by Du et al. [2005] defines positions and velocities as the binary adjacency matrix of a BN. In this case, velocities matrices can take any value from the set $\{-1, 0, 1\}$, representing deletions, non modifications or additions of arcs respectively. This same approach is taken by Santos and Maciel [2014], but instead of adjacency matrices they define a structure called *causality list* that establishes positions and velocities as sets of parent nodes.

To be able to apply PSO to the problem of learning DBN structures, we need a score that measures how likely it is that a network structure fits some data. Let $\mathcal{D}$ be our training data and let $\mathcal{G}$ be the network structure represented by a position. Our objective can now be defined as:

$$\arg\max_{\mathcal{G} \in g^{DBN}} score(\mathcal{G}, \mathcal{D}). \tag{5.5}$$

This score of fitness is necessary to assess which particles in the solution space fit the training data better and guide the exploration towards them. There are many examples of scores in the literature, such as the BIC [Schwarz, 1978] and the AIC [Akaike, 1998] scores for discrete networks, or the Bayesian Gaussian equivalent (BGe) [Geiger and Heckerman, 1994] score and the adapted BIC and AIC scores for Gaussian Bayesian networks. Depending on the type of score used, the same algorithms can be used to learn both discrete and Gaussian Bayesian networks.

5.3 Encoding and operators

One of the crucial elements of meta-heuristic optimization algorithms is the encoding used. A suboptimal encoding could generate losses in efficiency by having redundant solutions, having

to fix invalid individuals each iteration or by not allowing an even exploration of the solution space.

Our proposed encoding maps each possible transition network structure to a vector of natural numbers. This mapping is bijective in both sets: a transition network structure can only be represented with one specific vector, and a vector only represents one specific transition network structure.

5.3.1 Natural vector encoding

In a particle swarm scenario, each particle has a position and a velocity. In our case, positions represent specific transition network structures and velocities represent additions and deletions of arcs.

In a transition network, there are several nodes representing the same variable in different instants of time. We define the concept of a temporal family of nodes $\mathbf{X}_i^f = \{X_i^0, X_i^1, \ldots, X_i^T\}$ as the set of nodes representing a single variable X_i in all existing time slices of the network. Inside a temporal family, we call a *receiving node* the node X_i^0 in the present time slice. This node is the only one in a temporal family that can have arcs pointing to it from any other node in earlier time slices. In our encoding, we will divide a vector in as many sections as receiving nodes X_i^0 there are in the network. Each of these sections is further subdivided into a subsection consisting of a single natural number for each existing temporal family $\mathbf{X}_i^f$ in the network. This number defines with its binary representation the existing arcs from a certain temporal family to a specific receiving node. Each 1-bit encodes an arc from a specific member of the temporal family to the receiving node. By definition, this encoding does not allow invalid individuals because only receiving nodes can have arcs pointing to them and no cycles can appear. Furthermore, the length of the encoded vectors only depends on the number of existing receiving nodes, and not on the Markovian order. Higher orders will only mean bigger natural numbers in the vector. To clarify this explanation, an example of a position and the network it encodes can be seen in Figure 5.2.

The velocities follow the same encoding, but represent arc additions or deletions instead of the presence or absence of an arc. Each velocity is composed of two vectors, $\mathbf{V}_p$ and $\mathbf{V}_n$, defining additions and deletions of arcs respectively.

5.3.2 Position and velocity operators

To perform the operations of the PSO, we will adapt them to the discrete space defined by our encoding. In essence, we will need to be able to add positions and velocities, subtract positions and multiply velocities by real numbers. All operators shown are supposed to be bitwise logical operations. Both positions and velocities have the same vector length, so when operated together this bitwise operations will be performed throughout both vectors to each pair of natural numbers.

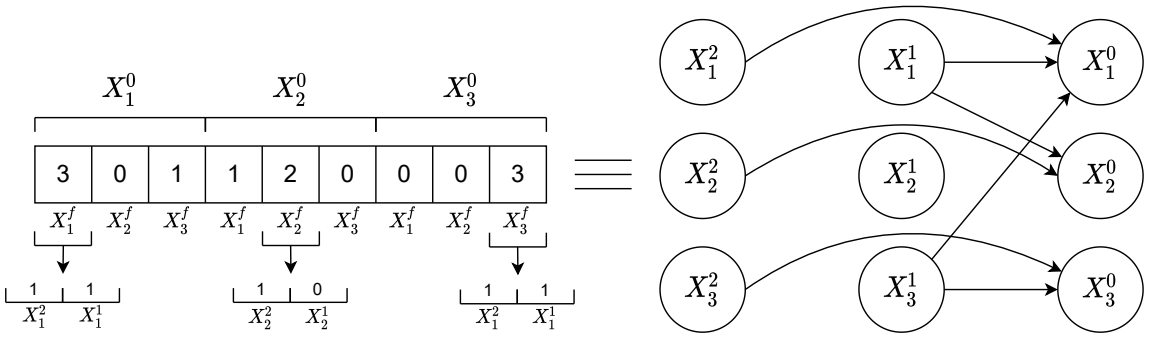


Figure 5.2: Representation of a position natural vector on the left and its equivalent transition network on the right. The transition network has three nodes per time slice and a Markovian order 2 for simplicity and clarity. Notice how increasing the order, thus adding many possible nodes and arcs, only implies bigger natural numbers in the vector, but it does not increase its length. For example, increasing the order up to 3 in the figure would only mean that natural numbers up to 7 can now appear in the vector.

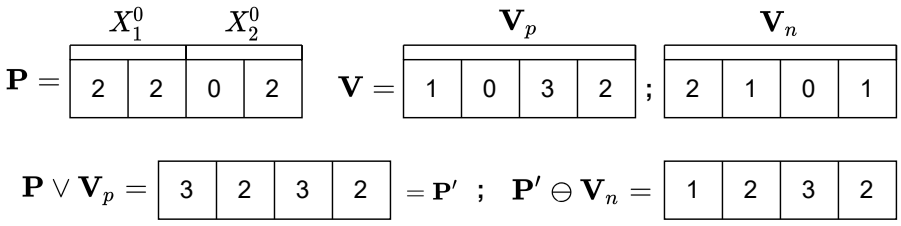


Figure 5.3: An example of adding a position and a velocity encoding a network with two receiving variables X_0^0 and X_1^0 and maximum Markovian order 2 for simplicity. All the operations are performed bitwise on the natural numbers of all vectors.

5.3.2.1 Position plus velocity

To add a velocity to a position, first we add all the arcs in $\mathbf{V}_p$ by performing a logical 'or' in the form of $\mathbf{P}' = \mathbf{P} \vee \mathbf{V}_p$. As for the negative part, we define the $\ominus$ operator as:

$$x_1 \ominus x_2 = x_1 \wedge \neg x_2. \tag{5.6}$$

This operator is equivalent to a 1-bit subtractor without borrow. By performing $\mathbf{P}' \ominus \mathbf{V}_n$, we remove the 1-bits that are present in both the position and negative velocity temporal families and maintain the rest unaffected. The consecutive positive and negative operations will add and remove the marked arcs of the velocity in the original position. An example of this operation can be seen in Figure 5.3.

5.3.2.2 Addition of velocities

Given two velocities $\mathbf{V}^1$ and $\mathbf{V}^2$, to add them we first need to combine their positive and

negative parts. For this, we operate $\mathbf{V}'_p = \mathbf{V}^1_p \vee \mathbf{V}^2_p$ and $\mathbf{V}'_n = \mathbf{V}^1_n \vee \mathbf{V}^2_n$. Afterwards, we perform a bitwise logical 'and' operation to identify redundancies in the form of $\mathbf{Rd} = \mathbf{V}'_p \wedge \mathbf{V}'_n$. Any 1-bit present in $\mathbf{Rd}$ means that an arc is being added and deleted at the same time in the resulting velocity, and it has to be set to 0 in both positive and negative vectors of $\mathbf{V}'$ with the 'xor' operator by performing $\mathbf{V}'_p \oplus \mathbf{Rd}$ and $\mathbf{V}'_n \oplus \mathbf{Rd}$ respectively. An example of this operation is shown in the following lines:

$$\mathbf{V}^1 = [1,0,2,0];[2,0,0,3], \mathbf{V}^2 = [0,1,2,1];[0,2,0,2],$$
$$\mathbf{V}^1 \vee \mathbf{V}^2 = [1,1,2,1];[2,2,0,3] = \mathbf{V}',$$
$$\mathbf{V}'_p \wedge \mathbf{V}'_n = [0,0,0,1] = \mathbf{Rd},$$
$$\mathbf{V}' \oplus \mathbf{Rd} = [1,1,2,0];[2,2,0,2].$$

5.3.2.3 Subtraction of positions

Given two positions $\mathbf{P}_1$ and $\mathbf{P}_2$, the operation $\mathbf{P}_1 - \mathbf{P}_2 = \mathbf{V}'$ returns the velocity $\mathbf{V}'$ such that $\mathbf{P}_1 + \mathbf{V}' = \mathbf{P}_2$. This effect is obtained by using the operator $\ominus$ defined in Equation (5.6) to calculate both the positive part $\mathbf{V}'_p = \mathbf{P}_2 \ominus \mathbf{P}_1$ and the negative part $\mathbf{V}'_n = \mathbf{P}_1 \ominus \mathbf{P}_2$ of the velocity. Notice that the $\ominus$ operator is not commutative, and so the inverted positions in each operation give different results. The $\ominus$ operator can be used to get the bits that need to be added to transform a position into another. An example to clarify this operation is shown in the following lines:

$$\mathbf{P}_1 = [1,0,2,1], \mathbf{P}_2 = [1,1,0,3],$$
$$\mathbf{V}'_p = \mathbf{P}_2 \ominus \mathbf{P}_1 = [0,1,0,2],$$
$$\mathbf{V}'_n = \mathbf{P}_1 \ominus \mathbf{P}_2 = [0,0,2,0],$$
$$\mathbf{V}' = [0,1,0,2];[0,0,2,0].$$

5.3.2.4 Multiplication of velocities by real numbers

Let $|\mathbf{V}|$ be the total population count in a velocity, that is, the total number of 1-bits in both positive and negative vectors. As proposed by Santos and Maciel [2014], multiplying a velocity by a real number increases or decreases $|\mathbf{V}|$ in the form of $\lfloor \alpha * |\mathbf{V}| \rfloor = |\mathbf{V}|'$. This means that we will randomly add or delete 1-bits in the velocity until the new total number of operations is obtained. We will follow a uniform distribution when sampling a temporal family in the vector and the open bits in the natural numbers. If $\alpha < 0$, $\mathbf{V}_p$ and $\mathbf{V}_n$ will be swapped with each other to invert all additions and deletions of arcs in the velocity and the absolute value of α will be used.

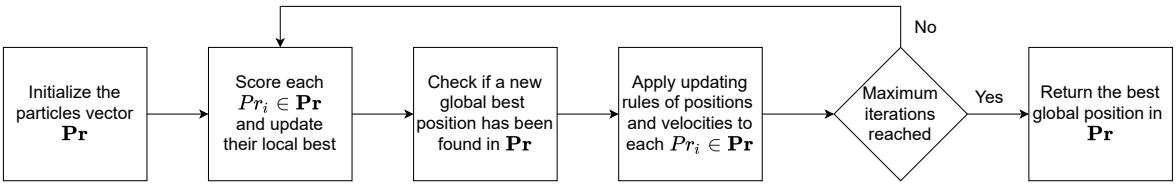


Figure 5.4: Pipeline of the PSO algorithm. The particles move applying the updating rules described in Section 5.2.2 and the operators described in Section 5.3.2. The position with the best fitness is translated into its DBN equivalent and returned as the best solution found.

5.4 Results

In this section we will first discuss the implementation of our PSO structure learning algorithm (natPSOHO) and compare it with two other algorithms: the PSO for HODBNs proposed by Santos and Maciel [2014] and our variation of the dynamic max-min hill climbing algorithm [Trabelsi et al., 2013]. This comparison will consist of recovering several synthetic randomly generated networks from sampled datasets, evaluating the execution time and how many of the original arcs are recovered from the data. The Markovian order and the number of receiving nodes will vary, to assess the efficiency and precision of the algorithms as both these factors increase. The number of iterations of the PSO algorithms is set to 50 with populations of 300 particles, and all the datasets will consist of 10.000 instances. These parameters remain constant through all the experiments.

5.4.1 Implementation

All the algorithms have been implemented in R and C++. The code of the natPSOHO algorithm, the experiments and the generation of the synthetic datasets have been combined into an R package that is publicly available in a *GitHub* repository[1]. All experiments were conducted on an Ubuntu 18 machine with an Intel i7-4790K processor and 16 Gb of RAM.

Due to the translation of the position and velocity operations to our specific encoding, we can use the normal pipeline of the PSO algorithm shown in Figure 5.4 without any change. For the evaluation of the positions based on the dataset, we will use the BGe score.

As recommended in other PSO works [Liu and Liu, 2018], the inertia value w is set high at the beginning and slowly decreases as the iterations advance to favour exploration at first, and c_1 is set high and decreases over time while c_2 is set low and increases over time, so that the positions close to the global optimum are properly explored prior to finishing the execution.

5.4.2 Experimental comparison

The results for Markovian orders 1 and 2 networks can be found in Table 5.1. We can see that for smaller networks with few nodes and low Markovian order, the DMMHC algorithm is much faster than the particle swarm ones, but its execution time scales rapidly with the

[1] https://github.com/dkesada/natPSOHO

number of variables. We will not be testing the DMMHC algorithm on higher orders, given its poor scalability and the similar performance to the other two algorithms in terms of number of real arcs recovered. On the other hand, the natPSOHO algorithm is less efficient in time than the binary PSOHO for low-order networks and many receiving nodes, but it consistently outperforms the other two algorithms in terms of real recovered arcs.

Table 5.1: Results for low-order networks

[Order, n_0,Arcs]	**Algorithm**	**Rec. arcs**	**Exec. time**
[1, 10, 47]	natPSOHO	47	1.49 m
	PSOHO	42	1.55 m
	DMMHC	36	0.22 s
[1, 15, 111]	natPSOHO	96	3.24 m
	PSOHO	78	2.81 m
	DMMHC	69	2.54 s
[1, 20, 201]	natPSOHO	152	5.81 m
	PSOHO	118	4.22 m
	DMMHC	100	42.21 s
[2, 10, 96]	natPSOHO	86	3.37 m
	PSOHO	68	2.88 m
	DMMHC	71	24.1 s
[2, 15, 234]	natPSOHO	175	7.1 m
	PSOHO	139	12.68 m
	DMMHC	132	40.73 m
[2, 20, 398]	natPSOHO	288	14.24 m
	PSOHO	215	23.15 m
	DMMHC	233	19.1 h

The results for recovering high-order networks from data can be seen in Table 5.2 and in Figure 5.5. We can see how the execution time for the natPSOHO algorithm scales better than the other method as we increase the Markovian order of the networks. We can also see that the number of recovered arcs is consistently higher in the case of the natPSOHO algorithm. In order to evaluate the networks, the BGe score is used to find the fitness of the particles. We have enhanced this score by omitting the scoring of nodes outside of t_0, due to the fact that in transition networks these nodes never have parents and their structure remains constant. Regardless of this, the score takes longer to compute for networks with a higher number of arcs. Given that the natPSOHO algorithm consistently recovers more arcs from the real networks that generated the synthetic data, the execution time of the score is also consistently higher. This makes it impossible to achieve a constant execution time on the evaluation of the networks, but the constant execution time in the operations of the particles is shown in the overall better performance of the algorithm.

The percentage of recovered arcs decreases in both algorithms as we increase the order

Table 5.2: Results for high-order networks

[Order, n_0,Arcs]	**Algorithm**	**Rec. arcs**	**Exec. time**
[3, 10, 147]	natPSOHO	125	4.93 m
	PSOHO	85	7.56 m
[3, 20, 616]	natPSOHO	398	23.8 m
	PSOHO	308	37.06 m
[4, 10, 208]	natPSOHO	157	6.87 m
	PSOHO	110	11.06 m
[4, 20, 825]	natPSOHO	533	38.02 m
	PSOHO	423	1.04 h
[5, 10, 247]	natPSOHO	181	9.23 m
	PSOHO	148	16.87 m
[5, 20, 982]	natPSOHO	622	57.45 m
	PSOHO	425	1.3 h
[6, 10, 294]	natPSOHO	208	12.43 m
	PSOHO	170	21.6 m
[6, 20, 1171]	natPSOHO	739	1.4 h
	PSOHO	517	2.04 h

due to the PSO not being close to convergence. The solutions obtained are the best ones found after 50 iterations, but it is expected that the algorithms did not yet converge to a solution due to the number of iterations being constant in all experiments. As the size of the search space increases, the number of iterations should also increase accordingly.

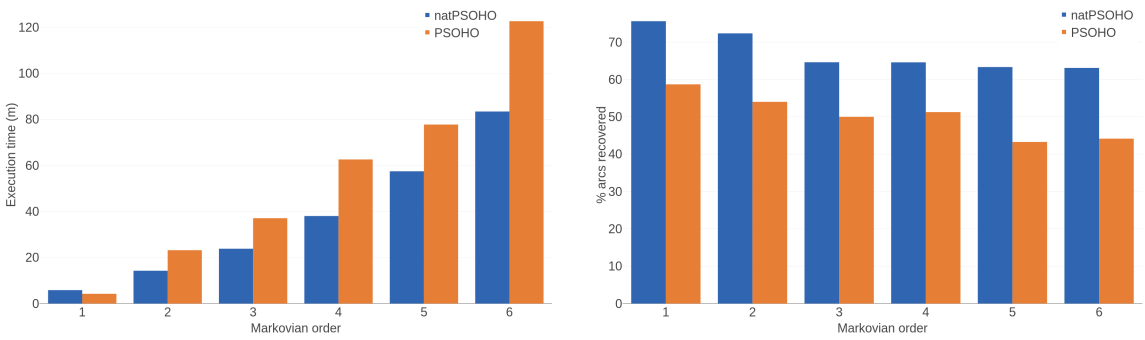


Figure 5.5: Execution time in minutes and percentage of recovered arcs of both PSO algorithms with 300 particles, 50 iterations and 20 receiving nodes when learning transition networks as we increase the Markovian order.

5.5 Conclusions and future work

We have presented a new high-order DBN structure learning algorithm that employs PSO to find the network structure that best fits the training data provided. Its order invariant encoding allows a good scalability to bigger networks and high Markovian orders. It also offers the possibility to search up to a maximum desired order rather than having to specify it beforehand.

When learning high-order networks, the search space becomes huge rapidly. Algorithms that rely on independence tests, like the DMMHC algorithm, can become unfeasible in terms of execution time due to the high number of variables and the datasets with thousands of instances. On the other hand, PSO algorithms can scale well to finding solutions in bigger search spaces, but suffer slightly in smaller ones.

The execution time of the proposed natPSOHO algorithm scales much better in high-orders due to the underlying data structures being constant in size and the operations being performed bitwise. It is shown to be an effective algorithm when dealing with processes with big search spaces generated by high-order networks. In situations where the exact Markovian order is not known beforehand, a higher number of iterations and a maximum desired order can be provided and the algorithm will search for a fitting network in that scenario.

In future work, we would like to refine our encoding and generalize it to be used with any meta-heuristic algorithm, not only with PSO. The main issue that it presents is that even though it relies on vectors of natural numbers, the operators must be bitwise. This would require a transition to a different encoding that is able to take advantage of the natural numbers, like the Gray code, or a generalization of the bitwise treatment before being able to extend it to other frameworks.

Chapter 6

Piecewise forecasting of nonlinear TS with model tree DBNs

6.1 Introduction

After applying HODBNs to industrial furnaces and defining a structure learning algorithm specific for this kind of networks, we now switch to another of the shortcomings of DBNs. By definition, Gaussian BNs are linear models, and so are Gaussian DBNs. This means that when we try to use Gaussian DBNs as general purpose models in a wide range of applications, we may find that the linearity constraint produces severe inaccuracies when modelling nonlinear processes. In this chapter, we are going to tackle the nonlinearity issue and apply our DBN models to a variety of different systems to evaluate their general use capability.

In a multivariate setting, we must take into account the effects that fluctuations in a variable have on the rest of the system. This means that we face a set of coupled univariate TS where variations in one series can generate linear or nonlinear changes in the others. As a result, modelling a multivariate process over time requires updating the state of the system and forecasting the evolution of all variables simultaneously.

In recent years, DBN models have seen a lot of use in industrial settings due to their characteristics as TS forecasting models and their interpretability, which has changed DBNs into more general purpose models. They have been applied to stock market forecasting [Duan, 2016], to ecosystem changes prediction based on climate variations [Trifonova et al., 2019], to topic-sentiment evolution analysis over time [Liang et al., 2020], to assess the remaining useful life of structures [Zhu et al., 2019; Cai et al., 2019], to monitor aircraft wing cracks evolution over time [Li et al., 2017] and to identify abnormal events during cybersecurity threats [Vaddi et al., 2020], among others. However, in a continuous case, where Gaussianity is typically assumed, DBNs present some drawbacks: DBN models are inherently linear models, and they do not allow the insertion of discrete variables without the introduction of additional constraints [Koller and Friedman, 2009]. Over time, industrial processes commonly follow complex nonlinear relationships or have changing distributions of the variables as

the system evolves, incurring in a phenomenon called *concept drift* [Gama et al., 2014]. In both scenarios, a traditional DBN presents weaknesses inherent to the model that will result in severe inaccuracies when modelling and forecasting this kind of pattern. In the aforementioned applications, Gaussian DBNs are used to fit nonlinear problems. In those cases, the authors have to either assume Gaussianity in their experiments, discretize the variables to be able to apply discrete DBNs or modify the DBN architecture with nonlinear conditional probability distributions that do not allow exact inference and need Markov Chain Monte Carlo sampling [Liang et al., 2020].

To address the linearity issue of Gaussian DBNs, we propose a new hybrid model called model tree dynamic Bayesian network (mtDBN). This model combines a classification and regression tree (CART) [Breiman et al., 1984] with DBNs in a manner similar to that of model trees [Quinlan, 1992]. First, a tree is fitted over the data with the desired variables, with the possibility of adding the elapsed time as a variable too. This tree structure is used to classify all instances of the dataset into different subpopulations depending on which leaf node they correspond to. Finally, a DBN model is fitted to each of the subsets of data in the leaf nodes. This way, we obtain several DBN models tuned specifically for certain contexts of the feature space defined by the tree splits instead of a unique global network. When performing forecasting, we first classify the current state of the system with the tree structure, and then we perform inference with the appropriate DBN model. This results in a piecewise regression [Vanli and Kozat, 2014], which is closer than a linear model to the real behaviour of a nonlinear or non-stationary system. Another advantage of this model is that in the presence of discrete variables, one can use all or several of them in the construction of the tree structure and then train Gaussian DBNs on each leaf node, thus avoiding switching to DBN models with discrete and continuous variables and their limiting constraints.

To compare the results obtained with DBN and mtDBN models with another state-of-the-art TS forecasting model, we use long short-term memory (LSTM) neural networks [Schmidhuber, 2015] and high-order fuzzy cognitive maps (HFCM) [Kosko, 1986]. LSTM models have found much use and popularity in recent years [Van Houdt et al., 2020], and they have been applied to a wide range of TS forecasting problems. On the other hand, fuzzy cognitive maps are graph-based TS forecasting models. They share similarities with RNN and use fuzzy sets to represent the relationships between the variables in real world problems. They have seen much use in recent decades and many studies have been presented introducing new variations of this framework [Orang et al., 2022].

The main contribution of this chapter is the introduction of a hybrid model between model trees and DBNs, which allows nonlinearity in the forecasting via piecewise regression. Our results in three different datasets show that the mtDBN model outperforms DBN models and is competitive with other state-of-the-art TS forecasting models. The mtDBN model learning and inference procedures are also distributed as an open source R package to facilitate its future possible applications.

This chapter includes the content of Quesada et al. [2022]. All code, data, and results are available at `https://github.com/dkesada/mtDBN`. Additionally, the code of the LSTM models

can be found at https://github.com/dkesada/kTsnn, and the code of the HFCM models is at https://github.com/dkesada/HFCM.

Chapter outline

The rest of the chapter is organized as follows. Section 6.2 defines our proposed mtDBN model. In Section 6.3 we explain the experimental methods and compare the results of the mtDBN and state-of-the-art methods. Finally, in Section 6.4, we give some final remarks and conclusions for the chapter.

6.2 Tree-based dynamic Bayesian networks

6.2.1 Hybrid definition: mtDBN

Our proposed mtDBN model consists of a hybrid between model trees [Quinlan, 1992] and DBNs in a similar fashion to Bayesian multinets [Bilmes, 2000] in a dynamic scenario. The objective of the tree is to divide instances from the original dataset into different contexts leading to each leaf node. Afterwards, we train several DBN models using the different instances attached to each leaf node. Similar to a multinet architecture, all the networks in the leaf nodes are related to each other, given that they model different situations from the same process, and the tree structure serves as a switch that selects the appropriate model for some new instance that we want to forecast. We chose model trees as the switch in our hybrid because it allows the DBN model to retain its interpretability. The leaf nodes in a tree are clearly differentiated by each branch split, and one can easily see the characteristics of the instances in each leaf node. This way, we can evaluate the specific DBN fitted to a leaf node knowing the context of that leaf node.

Typically, model trees require a response variable to be predicted, although in DBN models any variable can be the objective of inference. Thus, depending on the problem, we can choose any variable in the system as the objective variable for both growing the tree structure and forecasting in the network. Moreover, there is also the possibility of growing a multivariate regression tree [Larsen and Speckman, 2004] and performing inference over several variables at the same time with multi-output regression.

Similar to a model tree, we may have a Gaussian DBN model in each leaf node instead of a linear regression. Given a dataset $\mathbf{D}$, we can fit either a univariate or multivariate tree model and then evaluate which leaf node each of the instances belongs to. With these subdatasets, we can train each of the aforementioned DBNs and store them in a vector $\mathbf{M}$ of models and link each leaf node with one position of this vector. Then, while performing forecasting, a new instance is sorted down the tree to find its corresponding leaf node, and then we use the network inside $\mathbf{M}$ that is attached to that leaf node to forecast. Afterwards, if we are predicting up to a certain horizon in the future, the results of the forecasting are processed again with the tree to select a new DBN model to continue the forecasting. This way, we select an appropriate network based on the state of the system at each instant. Given that

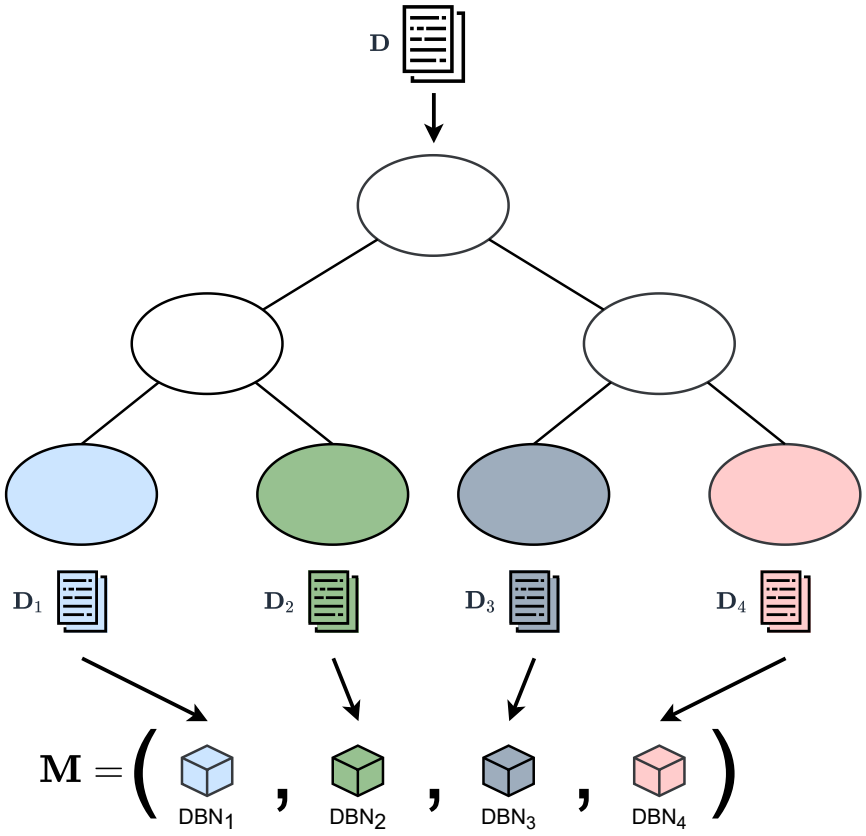


Figure 6.1: Schematic representation of the mtDBN architecture. The tree structure is initially used to divide the original dataset $\mathbf{D}$ into several datasets $\mathbf{D}_{leaf}$. Each one of these datasets is then used to fit a DBN model, which will be stored in the model vector $\mathbf{M}$. Afterwards, the tree structure will be used to classify new instances according to the appropriate DBN model inside $\mathbf{M}$.

the inference performed by Gaussian DBNs is linear, with this hybrid we can augment it to perform piecewise linear regression to achieve pseudo nonlinear forecasting with DBN models. The architecture of the hybrid mtDBN model is depicted in Fig. 6.1.

In our case, we opted to use a CART model [Breiman et al., 1984] with reduced variance splitting criterion for the tree structure that classifies instances into different leaf nodes. We chose this model due to its generality and simplicity as a starting point for the hybrid. CART models are well established and extended, and variance reduction splitting can be applied as a general method for continuous data. The model that defines the multinet, in our case the CART model, can potentially be swapped for more appropriate ones in different applications, such as a clustering method or another type of tree-based model. The splitting criterion can also be defined differently to prioritize specific behaviours, for example, using information gain [Sharma and Kumar, 2016] or change point detection methods on TS [Aminikhanghahi and Cook, 2017]. In this first version, we want to establish a more general approach that can be specialized to different scenarios. The pseudocode of the learning phase of our hybrid model is shown in Algorithm 6.1. We will refer to specific lines of this algorithm during the remainder of this section.

Algorithm 6.1: Learning the mtDBN model from data

Input: Parameters α, inc_cte, min_inst, $homogen$, mv
Output: The hybrid model $\mathbf{M}_{mtDBN}$
Data: Training set $\mathbf{D}$

```
valid = False;
full_tree = growCART(D, α, mv);
while not valid do
    pruned_tree = pruneCART(full_tree, α);
    D_class = classifyDataWithTree(D, pruned_tree);
    valid = checkInstancesPerLeafNode(D_class, min_inst);
    α = α + inc_cte
if homogen then
    dbn_struc = learnDBNStruct(D);
M_mtDBN = [ ]
for unique tree node data D_leaf in D_class do
    if not homogen then
        dbn_struc = learnDBNStruct(D_leaf);
    M^i_mtDBN = fitDBNParameters(D_leaf, dbn_struc);
return M_mtDBN;
```

Our dataset $\mathbf{D}$ contains all instances of our regressor variables X_j inside a table structure:

$$\mathbf{D} = [\mathbf{X}_0, \ldots, \mathbf{X}_n] = \begin{bmatrix} x_{00} & \cdots & x_{n0} \\ \vdots & \ddots & \vdots \\ x_{0l} & \cdots & x_{nl} \end{bmatrix}, \tag{6.1}$$

where l denotes the total length of the dataset. The specific value of variable X_j in instance k is thus defined by x_{jk}. In Algorithm 6.1, we provide $\mathbf{D}$ as input data.

We begin by defining an objective response variable to grow the tree. In our implementation, we allow both univariate and multivariate model trees, but in this section, we focus on univariate trees for simplicity. Note that, no matter the kind of model tree used, the mtDBN model will always perform multivariate inference. Once we set the response variable Y, which is one of our X_j variables, we grow the tree defining cut-offs on some regressor variable X_j based on the sum of squares of Y:

$$SS = \sum_i (y_i - \bar{y})^2, \tag{6.2}$$

where y_i is the value of Y in an instance of our dataset and $\bar{y}$ is the sample mean. The only difference with multivariate trees is that this sum of squares would be calculated for a vector $\mathbf{Y}$ of response variables instead of a single response variable. We calculate the cut-off point that generates two new tree nodes by maximizing the reduction of SS on both branches rooted at X_j compared to the parent node:

$$\delta_{tree} = SS_p - (SS_l + SS_r), \tag{6.3}$$

where δ_{tree} is the total improvement of a split, SS_p is the sum of squares of the parent tree node, SS_l is the sum of squares of the left branch and SS_r is the sum of squares of the right branch. We calculate the best δ_{tree} for all X_j variables available and then choose the one that maximizes it. Given that our regressor variables are continuous, there are potentially infinite possible cut-off points. To solve this issue, we only check the cut-off points defined by the values of X_j in our data D; that is, we evaluate Equation (6.3) for $X_j \geq x_{jk}$ in the left branch and $X_j < x_{jk}$ in the right branch. During the growing phase, we grow a univariate or multivariate tree until either none of the possible new splits obtain a positive δ_{tree} or we reach a maximum tree depth defined by the user in line 2 of Algorithm 6.1. The input parameters *homogen* and *mv* in Algorithm 6.1 are used to determine whether a homogeneous or non-homogeneous and univariate or multivariate tree is built. Once we grow the tree, we prune it by fixing an $\alpha \in (0, 1]$ parameter that defines a threshold over δ_{tree}. If a split does not satisfy $\delta_{tree} \geq \alpha\delta^P_{tree}$, where δ^P_{tree} is the total improvement of the previous tree node, then that subtree is pruned. The growing and pruning of the tree is encapsulated from line 2 to line 7 of Algorithm 6.1. In addition, we want to focus some attention on line 5 because it labels each row in $\mathbf{D}$ to its corresponding leaf node in the pruned tree and generates the $\mathbf{D}_{class}$ dataset. This will be key in the next steps of the algorithm when we use specific instances to learn each DBN model.

It is important to note that we apply the splitting criterion to the TS data in a static fashion, without taking into account the time difference between instances. This is because we want to be able to select the appropriate DBN model based on the current state of the system in each instant. If one wants to add the temporal component to the tree construction, the time passed since the beginning of the process may be added as a variable. This could potentially give the tree an idea of how long the system has been running, and it allows splits based on how much time has passed.

When we obtain an adequate tree model, we use the splitting rules defined by the tree to process the original dataset and assign each instance to its corresponding leaf node, generating several subdatasets. Afterwards, we use these subdatasets to fit a different DBN model for each leaf node. This way, each of the DBNs is tuned to its specific context as defined by the tree structure. This process is shown in Algorithm 6.1 from line 8 to line 14. These models can be homogeneous if they all share the same network structure learned from the whole dataset in lines 8 and 9, but each has different parameters fitted in line 14, or non-homogeneous if both the structure and the parameters of each network are learned from the local instances of each leaf node in lines 12 to 14. The loop in line 11 iterates over the different subdatasets $\mathbf{D}_{leaf}$ generated by the classification in line 5. Each $\mathbf{D}_{leaf}$ contains only those instances corresponding to a specific leaf node to fit the DBN with that specific part of the complete dataset $\mathbf{D}$.

As a reliability measure, we added the *min_inst* parameter that sets a minimum number of instances per leaf node. This avoids training DBN models with few data. We perform this

by pruning the tree structure and checking the number of instances per leaf node with that structure. If it has fewer instances than the minimum allowed, we increase the parameter α by a constant defined by the user with the *inc_cte* parameter and repeat the pruning process to generate shallower trees from line 5 to line 7. In particular, in line 6, we declare a pruned tree valid or invalid by recounting the number of instances in each leaf inside $\mathbf{D}_{class}$ and checking that they are all above the minimum.

6.2.2 Forecasting

Once a tree is obtained and all necessary DBN models are fitted, they can be used simultaneously to perform forecasting over some time horizon T. We begin the forecasting procedure with an initial evidence vector $\mathbf{s}_0$ defined as:

$$\mathbf{s}_0 = (\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^t) = ((x_1^0, x_2^0, \ldots, x_n^0), \ldots, (x_1^t, x_2^t, \ldots, x_n^t)), \tag{6.4}$$

where n is the number of variables per time slice and $t+1$ is the number of time slices in the network. This vector $\mathbf{s}_0$ defines the state of the system at the initial point of the forecast. The t-th time slice represents the variables in the present, and only $(x_1^t, \ldots, x_n^t)$ will be processed by the tree structure to determine which of the DBN models in the mtDBN will be the most suitable to forecast the next instant. Inspired by model trees, we also fit a DBN model on the tree nodes just before the leaf nodes. When we perform the forecast, we use both the correspondent DBN model attached to the leaf node of the tree and the DBN attached to the parent node of that leaf. Afterwards, we average the forecasting results of both DBN models to obtain a final forecast of the next state of the system. This step helps the hybrid model obtain smoother transitions between the DBNs of each leaf node. Once we obtain the new forecasted vector $\hat{\mathbf{x}}^{t+1} = (\hat{x}_1^{t+1}, \hat{x}_2^{t+1}, \ldots, \hat{x}_n^{t+1})$ from the inference of the two DBN models, we delete the oldest $\mathbf{x}^0$ and insert $\hat{\mathbf{x}}^{t+1}$ as the new $\mathbf{x}^t$, moving all the evidence for the remaining time slices forward in a sliding window fashion. The whole process is illustrated in Fig. 6.2.

The new $\mathbf{s}'$ vector obtained after all the evidence is moved forward is used in the next forecasting step and processed by the tree prior to the next forecast. This way, when forecasting with our hybrid model to some horizon, a DBN will be dynamically selected in each forecasting step based on the current state of the system. This provides the desired nonlinear behaviour with piecewise forecasts and allows the model to adapt to both nonlinear systems and sudden interventions or drifts.

6.3 Experimental results

To test the effectiveness of our proposed model, we set up several experiments with both synthetic and real-world data from nonlinear processes. By analysing the forecasting results, we can assess how the mtDBN model fits nonlinear processes in comparison with classical DBN models, LSTM neural networks and HFCMs, which are well behaving and popular models.

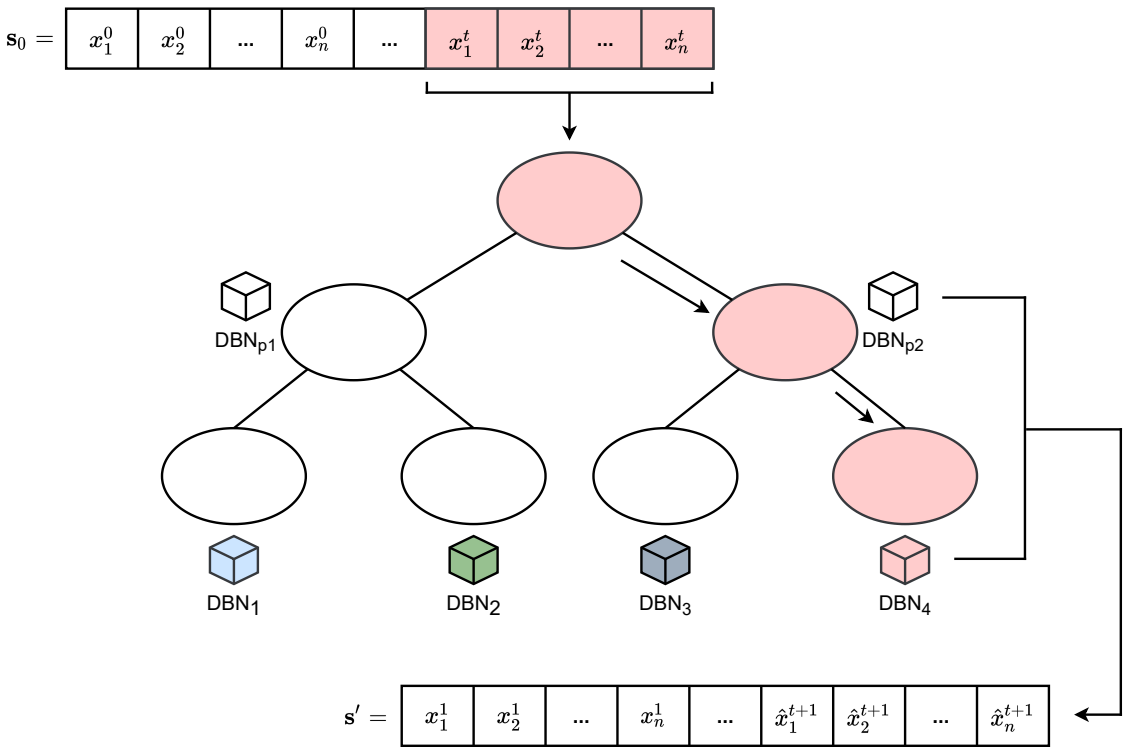


Figure 6.2: Representation of the mtDBN forecasting a single instant. The state vector is classified by the tree, and the correspondent DBN leaf and parent models are used to predict the next state vector.

All the programming code of the models, experiments and datasets can be found online in a GitHub repository[1]. The DBN and mtDBN models are written in both R and C++, the latter used for expensive computations, and distributed as R packages on CRAN and GitHub. All experiments were performed on an Ubuntu 18 machine with an Intel i5-4790k processor and 16 GB RAM.

6.3.1 Simulated nonlinear problem: Fouling phenomenon

To obtain performance results in a controlled environment, for our first experiment we generated a synthetic dataset from a nonlinear process defined by a system of ordinary differential equations (ODEs). Our scenario defines the fouling chemical reaction that occurs inside industrial furnaces that we covered in Chapter 4. As we heat a chemically reactive fluid to some desired temperature, this fluid can be prone to depositing solidified impurities on the inside of the pipes. As time passes, these impurities can generate an insulating layer that will force us to progressively increase the furnace temperature to keep the fluid temperature inside the pipes constant, which is the cause of severe costs in terms of efficiency loss. Due to the inherently nonlinear nature of the physical relationships that define this phenomenon, it can be used as an example to test the effectiveness of our hybrid model.

[1] https://github.com/dkesada/mtDBN

However, the previous dataset used was private industrial data and we did not have access to it anymore. To generate a new dataset, we used a system of ODEs that define a simplified fouling phenomenon without spatial dimensions. We modelled the following 10 variables: the fluid temperature T_1, the tube wall temperature T_2, the thickness of the fouling layer S_c, the concentration C_a of particles in the fluid prone to depositing, the amount of fuel m_c administered to the furnace heaters, the density ρ_1 of the fluid and its thermal capacity C_{p1}, the flow inside the tubes Q_{in}, the volume vol and concentration of particles C_{ain} in the fluid prone to depositing that is renewed at each instant. The details about the simulation construction and definition are further discussed in Appendix A.

Once the simulation was created, we generated independent and identically distributed multivariate TS of 100 time instants running a cycle. Each cycle represents the case where the furnace is reverted to an initial state after cleaning the fouling layer inside the tube, and we performed 100 time instants of operation as the time horizon. For each cycle, we set random starting values for the fluid properties and several different behaviours for the temperature and flow variations so that different fluids and furnace configurations were seen in the data. In total, we generated a dataset of 1000 cycles of 100 time instants each, that is, 100,000 instances and 10 variables. Then, we performed a 30-fold cross validation where 967 cycles were used in training and 33 were used for testing in each of the 30 hold-out processes. In our experiments, we used the first instance of each cycle as evidence, and we forecast T_1 until the last time instant. The mean absolute error and mean absolute percentage error (MAPE) were estimated in all instants and then averaged in all cycles to obtain the accuracy of each model. The DBN structures were learned using our particle swarm algorithm [Quesada et al., 2021a].

To assess our proposed model, we compare mtDBN variations that employ multivariate and univariate trees and homogeneous and non-homogeneous DBN structures for the models attached to the leaf nodes. Univariate trees can be better suited for problems where we are only interested in forecasting one variable. With our DBN models, we always forecast the complete multivariate state vector, but the tree structure can focus on a single objective variable that is of interest. With multivariate trees, we can cover situations where we may have several objective variables that we are interested in forecasting. On the other hand, non-homogeneous models can perform better when the process that we are trying to model is non-stationary, given that the conditional independence relationships defined by the DBN structures can differ from one leaf model to another. Homogeneous models are expected to work better in modelling processes that are nonlinear in nature but do not drift over time because the dependence between variables will always have the same structures but different parameters underneath. We set the Markovian order of the DBNs to 1, given that we know beforehand that our ODE system is autoregressive of order 1 and only uses the last time instant to generate the next one.

We also tune and fit LSTM and HFCM models in the experiment for the sake of comparing the performance of DBN-based models with other state-of-the-art models for TS forecasting. LSTM models are specifically tailored to forecast both univariate and multivariate TS, and

their recurrent structure can be used to perform forecasts of arbitrary length, similar to the case of DBN models. The code of these models and the experiments can also be found online in a GitHub repository[2]. HFCM are also graphical models that can be used for multivariate TS forecasting, and serve as a middle ground between DBNs and LSTMs because they share characteristics with both models. Our version of the HFCM comes originally from a project[3] that applies HFCMs to the uWave dataset, which is a dataset used to train gesture recognizers. An autoregressive version of HFCMs has been programmed by using a sliding window that transforms the predictions of the model into inputs for the next instant. This was necessary to perform the long-term and mid-term forecasting in two of the experiments. The code of both the model and the experiments can also be found online in a GitHub repository[4].

	Homogen?	**Tree**	**MAE**	**MAPE**	**Train (m)**	**Exec (s)**
DBN	-	-	80.23	13.42	7.99	0.58
mtDBN	Yes	Univariate	60.33	9.72	8.08	1.388
mtDBN	Yes	Multivariate	49.60	8.07	8.29	1.387
mtDBN	No	Univariate	79.52	12.83	32.91	1.388
mtDBN	No	Multivariate	54.15	8.90	32.04	1.387
LSTM	-	-	59.81	10.20	7.09	0.08
HFCM	-	-	177.32	27.29	59.27	1e-3

Table 6.1: Results in terms of the MAE, MAPE, training and execution time of the models for the fouling experiment.

The results in Table 6.1 show that the mtDBN models obtain better MAE and MAPE results than the baseline DBN model at the cost of longer training and execution times. The MAE result of the best hybrid model is almost half the baseline MAE of the regular DBN model. In the case of non-homogeneous models, where each leaf node has a different DBN structure, the training time increases due to having to run the structure learning algorithm for all the different contexts. In this experiment, we can see that the model that excels is the homogeneous and multivariate mtDBN. This can be explained from the point of view of the process that we are modelling. Our simulation generates traces from a nonlinear system of ODEs that does not change over time. This means that the relationships established between the variables remain constant throughout, and this is best represented by homogeneous models. Multivariate trees also seem to outperform their univariate counterparts, likely because the DBN models perform multivariate inference themselves. The execution time in the mtDBNs is consistently higher due to having to classify the state of the system in each forecasting step with the tree structure to select the appropriate DBN model and having to forecast with both the leaf node model and the parent node model to smooth the results. However, given that the forecasting process of all the mtDBN models is the same, all their execution times are equivalent. When we compare the best resulting mtDBN model with the

[2] https://github.com/dkesada/kTsnn
[3] https://github.com/julzerinos/python-fuzzy-cognitve-maps
[4] https://github.com/dkesada/HFCM

baseline unique DBN, we can see that we do not sacrifice too much training and execution time in exchange for a significant reduction in the MAE and MAPE.

Given that our proposed mtDBN model combines DBN models with model trees, we want to ensure that the results obtained with the hybrid are statistically significant when compared with the baseline model. For this purpose, we performed pairwise Wilcoxon rank-sum tests to check that the mean accuracies obtained from the mtDBN models are statistically significantly better in comparison with the baseline model. We chose this non-parametric test because the samples do not follow Gaussian distributions. The results shown in Table 6.2 show that all mtDBN models reject the null hypothesis of equal performance in MAE. We also performed this test between the baseline model, the best mtDBN model and the LSTM model with multiple comparisons and observed from the p-values that they all reject the null hypothesis of equal performance in MAE, as shown in Table 6.3. Given that we have more than two data samples from all the tested models, we also performed the Kruskal-Wallis test on our results. This allows us to perform a non-parametric test that does not rely on pair wise tests between the models. The test obtains a p-value of 2.2e-16, which indicates that at least one of the sample results from the models is better and statistically significant from the others.

Homogen?	**Tree**	**p-value**
Yes	Univariate	2.200e-16
Yes	Multivariate	2.200e-16
No	Univariate	1.471e-11
No	Multivariate	2.200e-16

Table 6.2: Resulting p-value of the Wilcoxon rank-sum tests for the forecasts of the different hybrid models in comparison with the baseline DBN model for the fouling experiment.

	DBN	**mtDBN**	**LSTM**	**HFCM**
DBN	-	2.20e-16	1.82e-08	2.20e-16
mtDBN	2.20e-16	-	2.20e-16	2.20e-16
LSTM	1.82e-08	2.20e-16	-	2.20e-16
HFCM	2.20e-16	2.20e-16	2.20e-16	-

Table 6.3: Resulting p-value of the Wilcoxon rank-sum tests in the multiple comparisons between the baseline DBN model, the homogeneous multivariate mtDBN, the LSTM and the HFCM model. All the tests reject the null hypothesis of equal performance in MAE.

In comparison with the LSTM model, the MAE results are much better than the baseline and similar to the hybrid models. In terms of time costs, training times of the LSTM model are similar to the DBN and homogeneous mtDBN models, and its execution time is the second best, only behind the HFCM model. They are very powerful models in this kind of settings, and their only remarkable downsides are that they are black box models and that the tuning process can be very time consuming due to the large number of parameters and

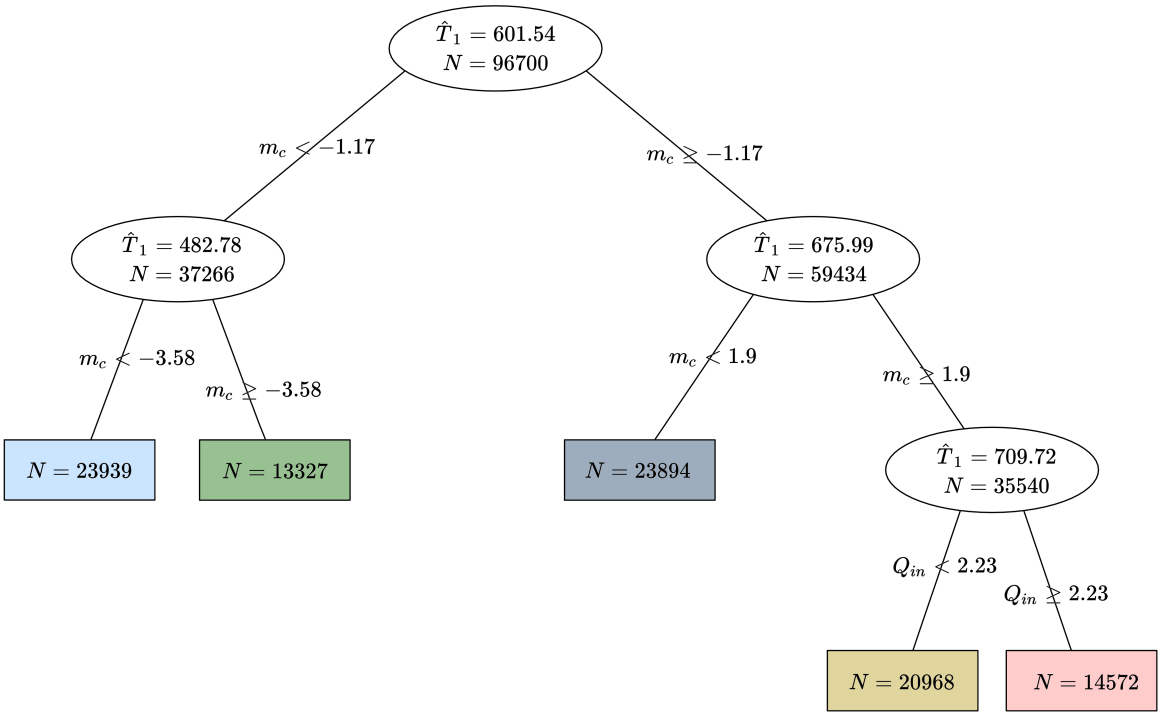


Figure 6.3: Example of the tree structure of a homogeneous multivariate mtDBN model. The resulting tree has five different leaf DBN nodes, represented in various colours. The splitting rules are shown in the branches, and the average value of the objective temperature T_1 and the number of instances N per node are shown inside the tree nodes from the initial 96,700 instances in the training dataset at the root of the tree. From the tree, we can identify that the m_c variable reduces the sum of squares the most in the different scenarios. From the root node, lower quantities of fuel m_c administered to the furnace will result in slower processes with lower T_1 and vice versa. In the case of extremely high T_1, the most severe cases are defined by a high flow of fluid Q_{in} entering the system.

network architectures that may be evaluated before finding some optimal configuration being guided blindly. In contrast, the HFCM model works noticeably worse than the rest of the models in this synthetic long-term case. This is due to the fact that fuzzy cognitive map models perform well on short-term predictions, but worse on long-term predictions [Feng et al., 2021]. The training time ramps up due to the internal optimization of the weight matrix having to work with long cycles of 100 instances each, and the accuracy of the model is almost twice as worse than the baseline DBN model. In terms of execution time though, it is by far the fastest model.

Fig. 6.3 shows an example of the tree structure from an mtDBN model. This tree structure is interpretable and can garner valuable insights into the problem at hand. For example, in this particular case, we know that the amount of fuel m_c administered to the furnace is an external parameter decided by a human operator. Higher m_c on the right branch of the tree generates higher fluid temperatures T_1 and promotes a faster creation of the insulating fouling layer S_c. This increase in S_c means that increasingly higher values of m_c will be needed to maintain a high T_1. Additionally, the reaction is further characterized by the flow rate Q_{in} on the rightmost branch with the highest T_1 cases. A fast flow rate can only be

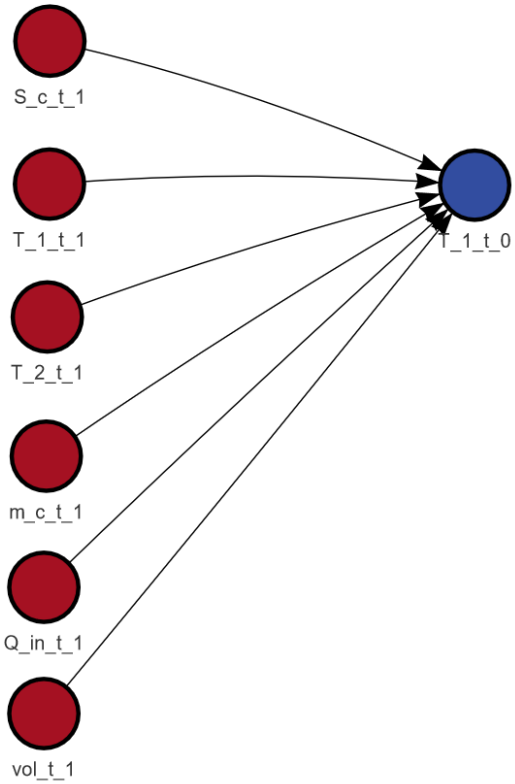


Figure 6.4: Example of the DBN structure of the homogeneous multivariate mtDBN model. The objective variable T_1 at the present instant is represented as the blue node, and all the parent nodes in red represent those variables at the previous instant. With these parent nodes, we can see the autoregressive component and the most relevant variables in the inference of T_1, i.e., S_c, T_2, m_c, Q_{in} and vol.

coupled with high T_1 if the furnace is near its maximum temperature and the process has not been running for long, because it is very unlikely to achieve a high T_1 along with a fast flow rate and an already thick insulating layer. This differentiation with S_c of whether the furnace is working at different capacities improves the overall performance of the mtDBN compared to the baseline DBN by allowing the hybrid to switch between models in the middle of forecasting based on the current state of the system. It is important to note that we can explain the cuts performed by the tree structure more in-depth because we already know the underlying process that we are modelling. In a real case scenario, the tree structure offers valuable information on how this process operates, but we will likely not reach the level of interpretation shown in this example without the insight of an expert on the problem.

For illustration purposes, we also interpret the structure of a DBN model in the homogeneous multivariate mtDBN in Fig. 6.4. We see how the T_1 and T_2 values at the previous instant affect the predictions of the current value of T_1, which is directly defined in the underlying system of equations. Both m_c and Q_{in} affect the objective temperature, as hinted at by the tree structure, and finally, both S_c and the volume are also relevant to the variation

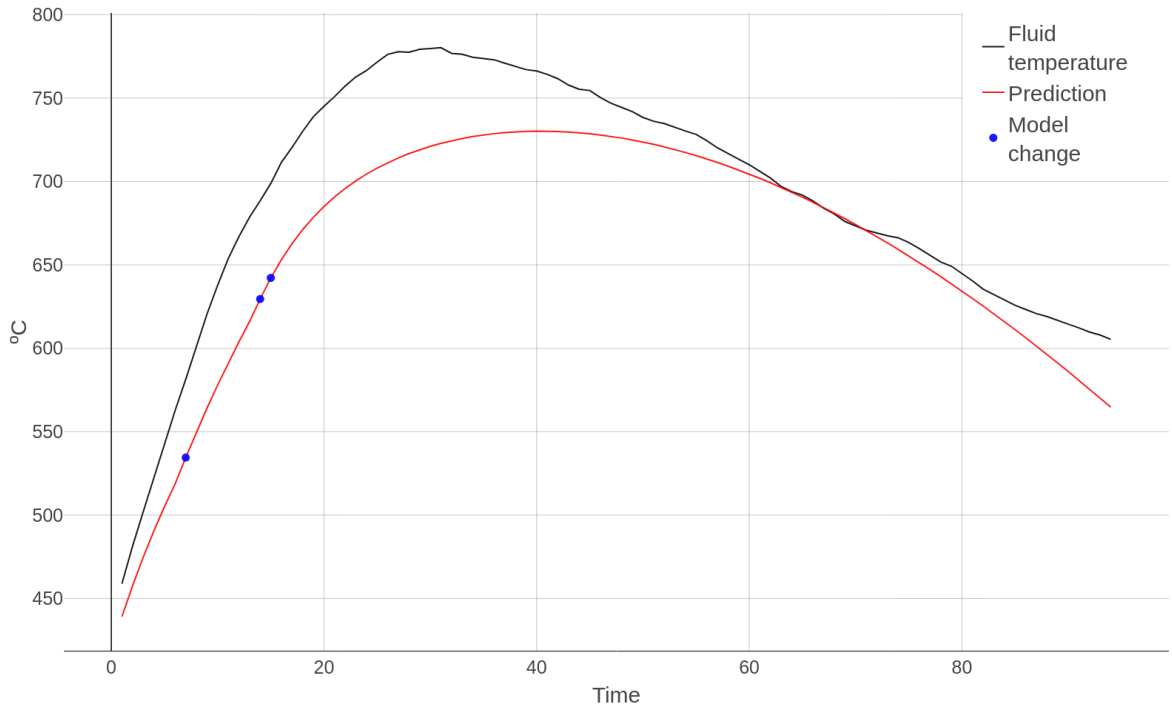


Figure 6.5: Example of predicting the fluid temperature T_1 of a 100 instant trace using the homogeneous multivariate mtDBN. The model used for forecasting is changed three times during forecasting based on the predicted state of the system at each instant. It is important to note that we only show the fluid temperature in the figure, but all variables in the system are jointly forecasted simultaneously to obtain the state of the system at the next instant.

in T_1. By using both the tree and DBN structures, we can identify the key variables that intervene in the estimation of our objective variable T_1 in the underlying process, and we can check the specific effect that these variables have in the inference performed by the DBN.

To put the profile of the forecasted temperature curves into context and to illustrate what the predictions look like, an example of forecasting a cycle with the best mtDBN model is shown in Fig. 6.5. In total, four different leaf node models are used in that specific forecasting to obtain the desired piecewise prediction of the temperature.

6.3.2 Real data application: Electrical motor

To observe the effectiveness of our hybrid model using real-world data, we set up a similar exercise using public experimental data collected from an electrical motor [Kirchgässner et al., 2021]. The motor speed, temperature in different sections, voltages and torque were recovered each instant from sensors, generating a multivariate TS dataset of several recording sessions with a total of 11 variables. The objective is to forecast the rotor temperature of the motor, given that this feature is not easily monitored inside electric motors and obtaining accurate predictions can increase the efficiency due to being able to predict overheating situations ahead of time and prevent them.

The dataset consists of 69 different recording sessions. A session consists of recordings

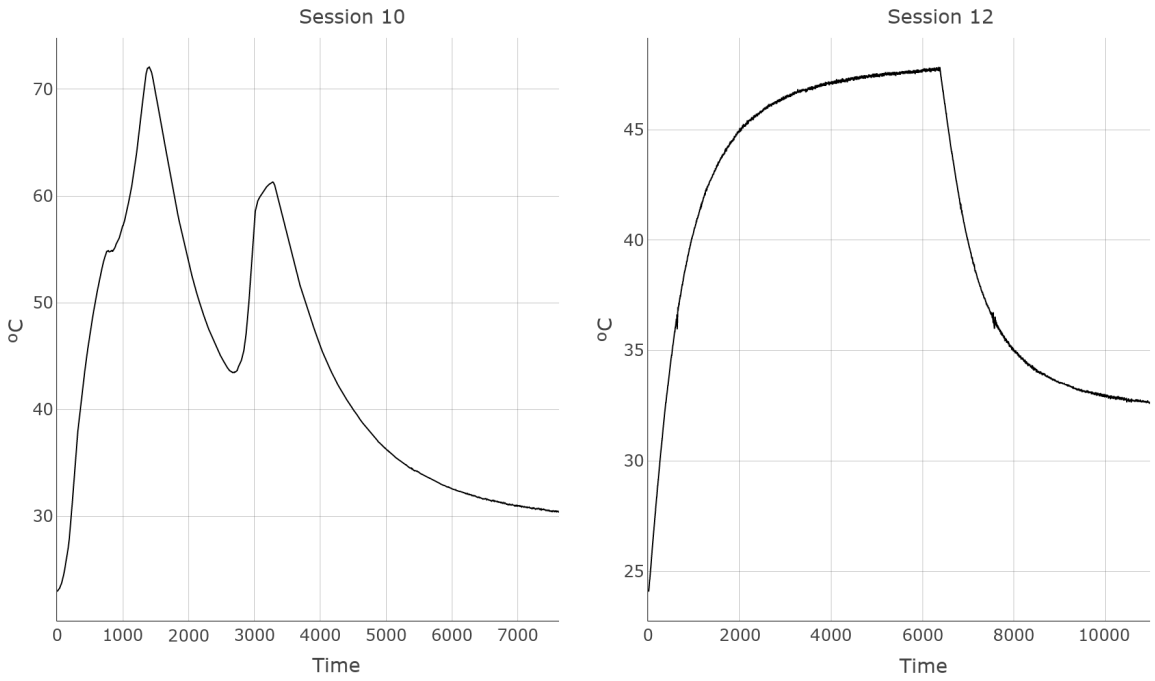


Figure 6.6: Example of the objective temperature in two different sessions in our dataset. Session 10 is rather irregular, with several interventions drastically changing the tendency of the series. Session 12 is a more common case in our data, with only one significant intervention after 6000 seconds.

of the active motor from an initial idle state up to some point in time. The values of the sensors were recorded with a frequency of 2 Hz, that is, each row is separated by 0.5 seconds. There are a total of 1,330,816 instances in the dataset, with the shortest session spanning 2,176 instances (18.13 minutes) and the longest 43,971 (6.12 hours). In this scenario, we face two problems: high dimensionality in terms of the number of instances and sessions of different lengths in the data. Performing long-term forecasting of complete cycles from a single starting point seems unreasonable due to the length of the TS. To aggravate this problem, the profile of the curves is very irregular, and interventions in the system affect the tendency during sessions, as seen in Fig. 6.6.

To approach the dimensionality issue, we reduce the frequency of the data to $0.0\dot{3}$ Hz so that rows are separated by 30 seconds each. When reducing the frequency, the mean of each 60-instance bin is returned as the new value. After this process, the dataset is reduced to 22,247 instances total, with sessions ranging from 37 to 734 rows. To check that we are not losing too much information with this reduction of frequency, we compute the average distance between the original TS and the reduced ones using the dynamic time warping distance implemented in the **dtw** R package [Giorgino, 2009]. Given that the TS are not too noisy, as seen in Fig. 6.6, and that we are averaging binwise, we do not expect the reduced versions to be very distant from the original ones. The total average normalized distance between the series is 0.1291, and this coupled with the alignments depicted in Fig. 6.7 indicates that there was not a severe information loss during the frequency reduction step.

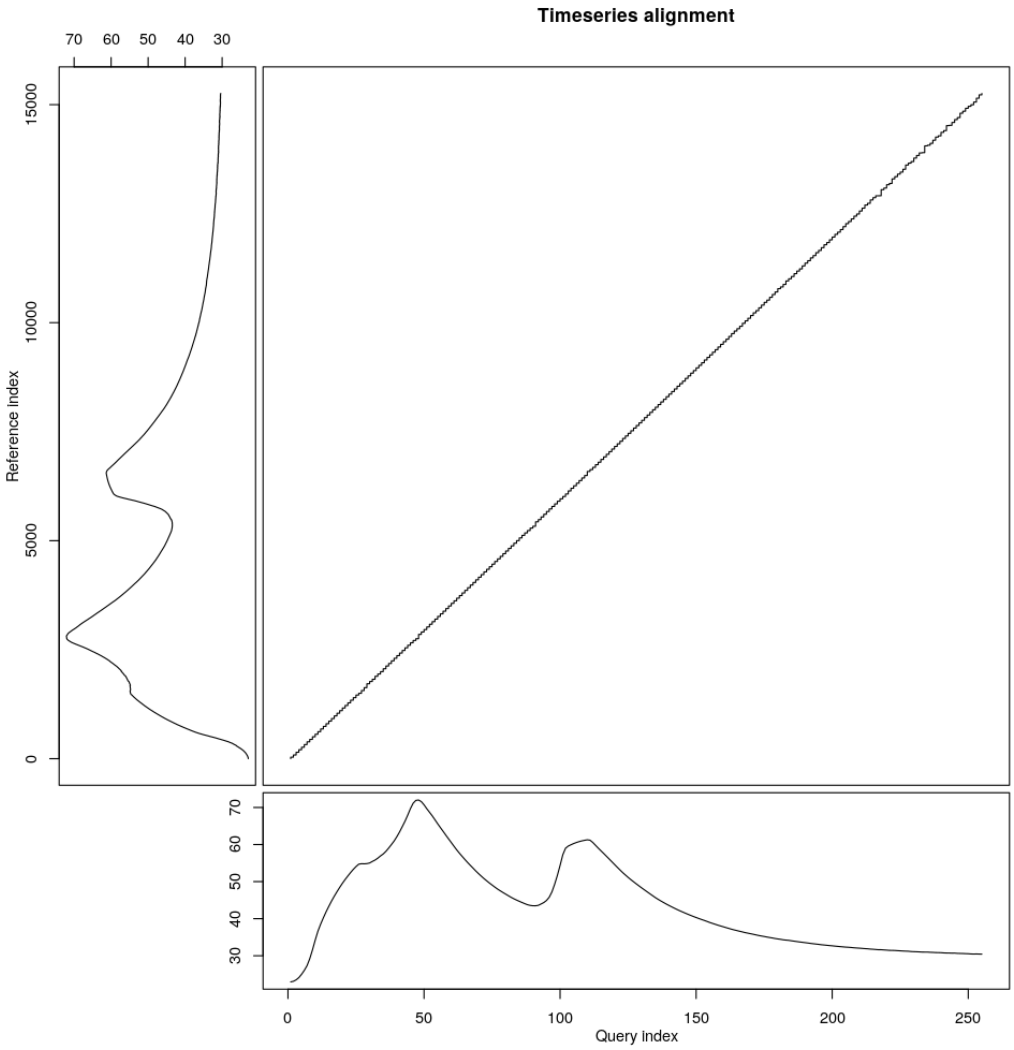


Figure 6.7: Example of the alignment of two TS sessions of the objective variable in the original and reduced datasets. The alignment being almost a straight diagonal line indicates that no displacement in time is seen and that no drastic jumps in values are present.

To address cycles of different lengths, we forecast up to fixed intervals of time instead of the whole sessions. We perform several consecutive forecasts of 20 instants. In the DBN models, we fixed the Markovian order to 2 for the structure of the networks because this real case scenario does not necessarily have an autoregressive order of one. The MAE result of these predictions were calculated, and the total MAE of a model is the global mean value of all the forecasts. To divide the dataset into training and test partitions, we used 3-fold cross-validation to form groups of 66 sessions for training and three sessions for testing. This way, the groups were formed randomly and all cycles appear in the test once.

The results of the experiments show that the mtDBN models also improve the MAE and MAPE results of the baseline DBN model in this experiment, as seen in Table 6.4. In this

	Homogen?	Tree	MAE	MAPE	Train (m)	Exec (s)
DBN	-	-	1.616	2.970	8.53	0.102
mtDBN	Yes	Univariate	1.499	2.602	8.73	0.215
mtDBN	Yes	Multivariate	1.574	2.807	8.74	0.214
mtDBN	No	Univariate	1.484	2.558	36.45	0.214
mtDBN	No	Multivariate	1.553	2.806	36.59	0.215
LSTM	-	-	2.289	3.937	2.21	0.083
HFCM	-	-	2.958	5.150	37.07	2e-4

Table 6.4: Results in terms of the MAE, MAPE, training time and execution time of the models for the motor dataset.

Homogen?	Tree	p-value
Yes	Univariate	1.540e-03
Yes	Multivariate	1.633e-03
No	Univariate	1.764e-06
No	Multivariate	2.079e-03

Table 6.5: Resulting p-value of the Wilcoxon rank-sum tests for the electric motor dataset with respect to the regular DBN model.

	DBN	mtDBN	LSTM	HFCM
DBN	-	1.764e-06	8.981e-16	2.200e-16
mtDBN	1.764e-06	-	2.200e-16	2.200e-16
LSTM	8.981e-16	2.200e-16	-	1.894e-05
HFCM	2.200e-16	2.200e-16	1.894e-05	-

Table 6.6: Results of the Wilcoxon rank-sum tests in the multiple comparisons for the motor experiment. Similar to the synthetic case, all the pairwise tests reject the null hypothesis of equal performance in MAE.

scenario of mid-term forecasting, the different DBN models are able to fit the contexts defined by the tree better than a global linear DBN model. The execution and training times of the hybrid models present the expected characteristics, taking longer to compute, similar to the synthetic case. The best resulting model was the non-homogeneous univariate mtDBN, which indicates that the underlying process is probably non-stationary and thus better modelled with different DBN structures, although not by a large margin. The univariate trees obtaining better results than the multivariate trees could be due to defining less significant subsets of data in the leaf nodes for the objective of predicting a single temperature. In this mid-term scenario, focusing on forecasting the objective variable with univariate trees can lead to a better accuracy if the error on the prediction of the rest of the variables does not add up. In this case, all models perform better than in the synthetic one, which can be seen in the MAPE being lower in Table 6.4 than in Table 6.1. The MAPE indicates that the predictions in this experiment are more accurate than in the previous experiment, which is due to performing

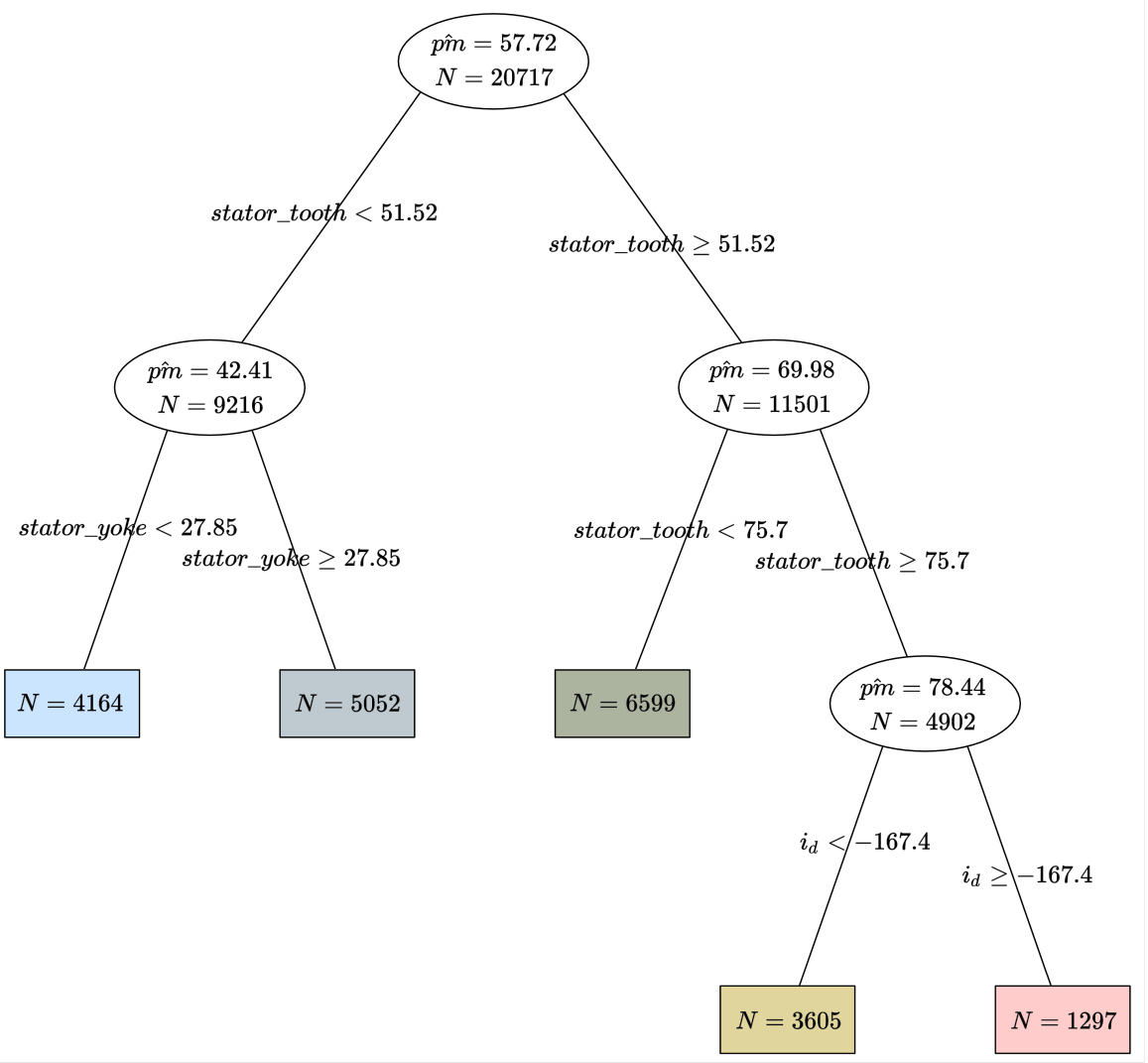


Figure 6.8: Example of the tree structure of the non-homogeneous univariate mtDBN model in the case of the electric motor. In this case, the initial cuts are made around the yoke and tooth temperatures inside the motor. It makes sense that temperatures at other points of the motor are useful in determining our objective temperature pm, and for very high temperatures, the electric current i_d defines the two most extreme groups.

shorter term forecastings.

We also performed statistical significance tests in this case. The p-values shown in Table 6.5 indicate that we reject the null hypothesis of equal performance with respect to the unique DBN model for all mtDBN models. As in the previous experiment, we also performed pairwise Wilcoxon rank-sum tests between the models to check for statistically significant differences in performance in terms of the MAE, and all the models rejected the null hypothesis, which can be seen in Table 6.6. The Kruskal-Wallis test also obtained a p-value of 2.2e-16, which shows that at least one of the models obtains better, statistically significant results from the others.

The tree structure shown in Fig. 6.8 identifies the two temperatures inside the motor,

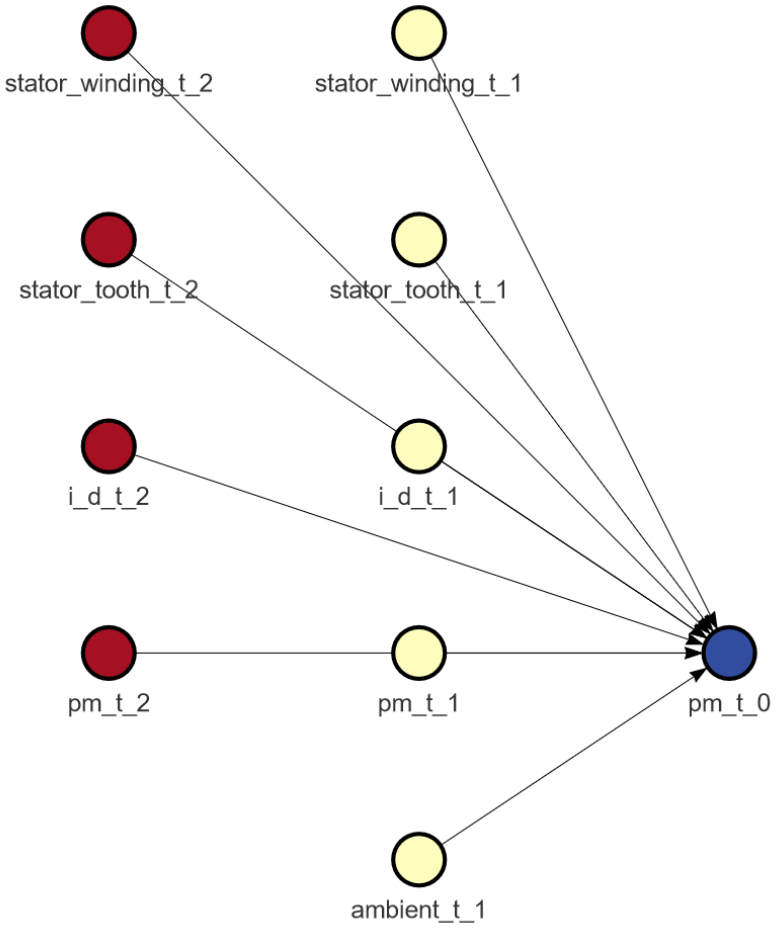


Figure 6.9: Example of the objective variable *pm* (in blue) and its parent nodes inside the DBN structure in the non-homogeneous mtDBN model of Markovian order 2. Apart from the autoregressive component of *pm* on itself, we can see that two variables are also used in the tree structure, *stator tooth* and i_d, and two other temperatures, *stator winding* and *ambient*, are used by the DBN model to predict *pm*. Intra-slice arcs are not permitted due to the structure learning algorithm used.

stator_tooth and *stator_yoke*, as those that better differentiate the different contexts of the objective temperature, and in the case of very high temperatures, the electric current i_d helps determine whether we are in the case of the highest temperatures of the dataset. In the case of temperatures, the relationships between the variables do not increase linearly as the temperature increases. The baseline DBN model approximates the global behaviour of the system, while the mtDBN model defines different splits based on these temperatures. The hybrid structure allows the model to switch between the leaf node models depending on the current temperatures of the system, both at the beginning of the prediction and while performing forecasting. This helps improving the accuracy of the mtDBN models.

We can also look at the DBN structure shown in Fig. 6.9 for further insight. Given that the model is non-homogeneous, this structure corresponds to the third leaf node, the one with the higher number of instances in the training dataset. We can see that, apart from the

variables used in the tree, the DBN also uses the *ambient* and the *stator_winding* temperatures to perform inference. This information could help an expert in the field understand the relationships of the variables in the problem and identify the most important physical relationships inside the system. In our case, we cannot extract a level of information as deep as in the synthetic case, but it can help us understand which are the most important variables in the system model and subsequently which interventions will affect it the most and what variables the model bases its forecasts on.

The LSTM model results remain in a similar range and are specially good in terms of training and execution time. The performance in terms of the MAE does not differ much from the rest of the models, given that this kind of model excels when trained with a large number of instances and the dimensionality is drastically reduced in the preprocessing. In this scenario, the biggest drawback we could find is in terms of interpretability of the model. The forecasts and times of the LSTM model are very competitive, but it offers no insight into how the system works and what the model bases its predictions on. The HFCM model obtains much better results in this experiment compared to the long-term scenario of the synthetic data. The MAE and MAPE results obtained are comparable to the LSTM results but slightly worse, because it still suffers from the mid-term predictions. On the other hand, the execution time is the best of all the models once again, but the training time is the slowest, even slower than the homogeneous mtDBN models.

6.3.3 Real data application: TSEC stock index

One of the most popular real world applications of TS forecasting models is in financial data from the stock market [Singh and Huang, 2019]. In this experiment, we have chosen the Taiwan Stock Exchange Corporation (TSEC) weighted index, which aggregates the stock values of almost 900 Taiwanese companies, as our objective index. The original data was obtained from yahoo! finance[5].

We extracted 16 months worth of daily values from the 1st of January 2021 to the 29th of April 2022. The dataset is composed of five variables: the opening value of the index ($Open$), the closing value ($Close$), the maximum value during that day ($High$), the minimum value (Low) and the volume of transactions ($Volume$). In contrast with the other experiments, we have a reduced dataset of 319 instances in total, given that the stock market closes during the weekends and on specific holidays, with four out of five variables which are deeply correlated due to being specific values that the index took during a single day. This correlation can be seen in Fig. 6.10. In this scenario, our objective will be to forecast the opening value that the index will take on the next day. This case offers a clear contrast with the previous experiments and can be used to evaluate the performance of the mtDBN model with reduced training data and very short-term predictions. Our objective variable has a mean value of 17205, with a maximum value of 18620 and a minimum of 14937. We also set a minimum of 50 instances per leaf node as a safety measure, given that the number of instances in the

[5] https://finance.yahoo.com/quote/^TWII/history?p=^TWII

Figure 6.10: Heatmap showing the correlation between the variables in the TSEC stock dataset. Most of the variables have a correlation very close to 1, except for the volume of daily transactions.

dataset is very low.

	Homogen?	**Tree**	**MAE**	**MAPE**	**Train (m)**	**Exec (s)**
DBN	-	-	64.55	0.385	21.08	0.009
mtDBN	Yes	Univ	55.18	0.327	20.68	0.011
mtDBN	Yes	Multiv	61.12	0.362	20.88	0.008
mtDBN	No	Univ	56.88	0.336	11.73	0.011
mtDBN	No	Multiv	61.47	0.364	13.69	0.008
LSTM	-	-	77.48	0.460	0.96	0.112
HFCM	-	-	84.87	0.504	6.58	3e-5

Table 6.7: Results in terms of the MAE, MAPE, training time and execution time of the models for the stock index dataset.

Homogen?	**Tree**	p**-value**
Yes	Univariate	0.033
Yes	Multivariate	0.542
No	Univariate	0.193
No	Multivariate	0.120

Table 6.8: Resulting p-value of the Wilcoxon rank-sum tests for the stock index dataset with respect to the regular DBN model.

	DBN	mtDBN	LSTM	HFCM
DBN	-	0.033	1.165e-3	3.280e-3
mtDBN	0.033	-	1.430e-4	4.636e-5
LSTM	1.165e-3	1.430e-4	-	0.405
HFCM	3.280e-3	4.636e-5	0.405	-

Table 6.9: Results of the Wilcoxon rank-sum tests in the multiple comparisons for the stock index experiment. The pairwise tests show that the LSTM and HFCM models do not present significant differences in their results.

The results of the experiment in Table 6.7 show that all the hybrid mtDBN models obtain better accuracy results than the baseline DBN model in both MAE and MAPE. The MAPE results show that this is the most accurate scenario for all the fitted models, largely due to the fact that we are performing single-step forecasting. In this case, given that we are predicting only the next instant, it is not possible to perform piecewise regression, because only a single prediction with a single model is performed. As a result, the only difference between the baseline and the mtDBN models is in which DBN model is used to perform forecasting. This is clearly evidenced in Table 6.8, where only the homogeneous univariate mtDBN obtains statistically significant results when compared with the baseline. A univariate tree is the most effective in this experiment due to most of the variables being deeply correlated with one another. There is little improvement from adding another variable as the possible objective of the tree when they all encode redundant information. Ultimately, the univariate homogeneous mtDBN model is able to obtain statistically significantly better results than the baseline because the tree structure is able to differentiate appropriate scenarios where a specifically trained DBN model can perform a single prediction more accurately than a global DBN model.

The tree structure of the model can be seen in Fig. 6.11. We can appreciate that the tree structure separates the values of the variable $High$ into three groups: one for the instances under the mean value of the stock index, one for the instances around that value and one for the instances above. With the information at hand, this kind of division is the most interesting one, given that stock indexes values behave differently while on their highest and on their lowest based on the effects of market operations of selling and buying. Another interesting result from the tree structure is that the number of instances in each leaf node is so low that it takes less training time to learn three networks in the non-homogeneous models than a single network with all the instances in the homogeneous ones in Table 6.7. This result is not seen in the other experiments because they both have thousands of instances per leaf node, which ramps up the structure learning time.

The DBN structure in Fig. 6.12 shows that the value of the *Open* variable is mainly decided by the last two opening values of the stock index and the closing value of the last day. It makes sense that the opening value of the next day is directly influenced by the closing value of the previous day, and using the lowest values of the index in the two previous days can help mitigating the cases where the index tanked its value one day, but it recovered on

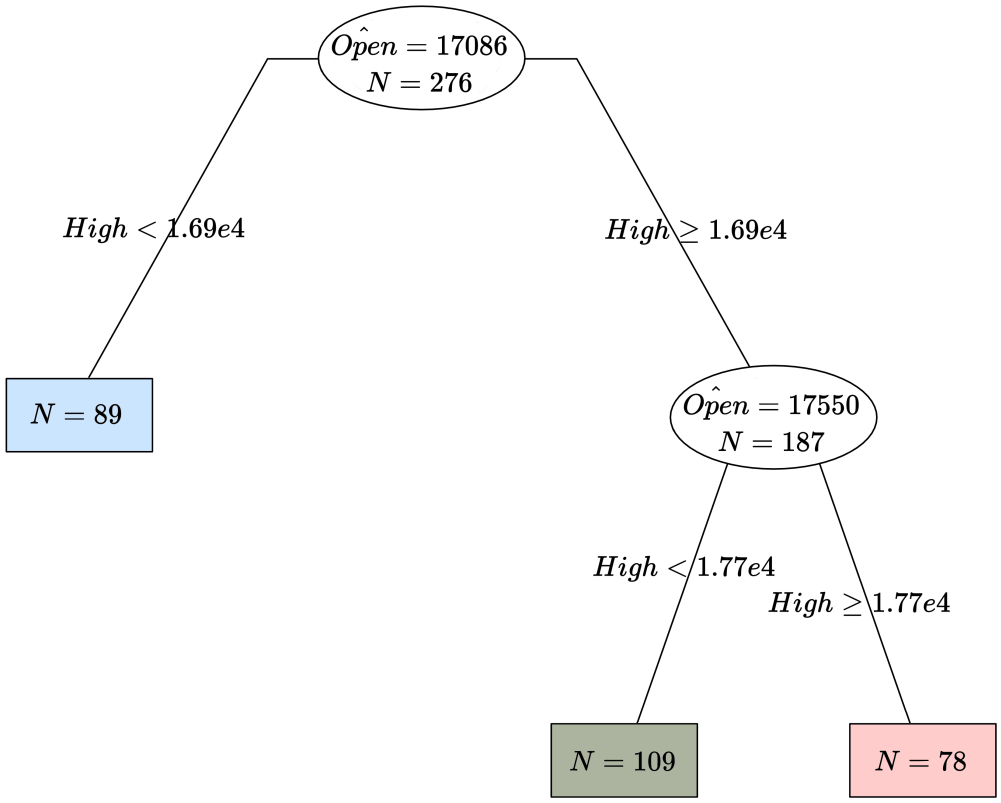


Figure 6.11: Tree structure of the homogeneous univariate mtDBN model. The splits define three cases depending on the highest value that the stock index took that day. From left to right, the leaf nodes define the cases where the stock index was lower than the mean, the cases where it was around the mean and the cases where it was higher than the mean.

the other one.

In this case, both the LSTM and HFCM models obtain comparable accuracies to the DBN based models. The LSTM model has the best training time, while the HFCM has the fastest execution time. The tests in Table 6.9 show that the results of the LSTM and the HFCM are equivalent in terms of accuracy, given that their differences are not statistically significant. When comparing all the algorithms with the Kruskal-Wallis test, we obtained a p-value of 3.12e-05. Ultimately, the results obtained by the homogeneous univariate mtDBN are statistically significantly better than the other models for this specific scenario.

6.4 Conclusions and future work

In this chapter, we proposed a hybrid model, mtDBN, that performs piecewise forecasting of nonlinear multivariate TS combining a model regression tree and DBN models. The tree model divides the contexts represented in the training dataset, and different DBN models are fitted to each. Then, the tree structure is used as a model selector in a multinet architecture

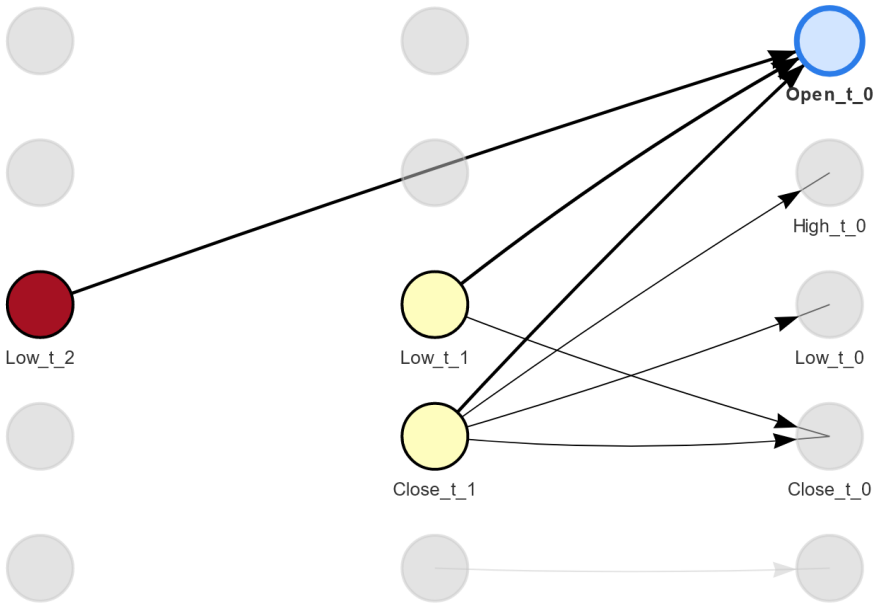


Figure 6.12: Example of the DBN structure of the homogeneous univariate mtDBN model in the stock dataset. The opening index value of the next day is obtained taking into account the lowest values of the last two days and the closing value of the previous day.

for deciding which DBN model to use in each instant of the forecast. Thus, we overcome one of the shortcomings of Gaussian DBN models when applied to real data, that they can only model linear processes. Our results on both synthetic and real data show that the hybrid mtDBN model effectively reduces the error of a baseline DBN when applied to nonlinear problems at the cost of a higher training time and a similar execution time. The hybrid model proved to be a viable alternative when applying DBN models in this kind of setting. To make the experiments replicable, we made all code available online, we created a simulation for the first experiment that is publicly available and used public data from real-world problems for the second and third experiments. We also offer the hybrid mtDBN model on a public repository inside an R package that makes it ready to use and deploy.

We presented the most elemental version of the mtDBN model. In future work, we would like to try different splitting criteria for growing the tree model. In particular, given that we are working with TS data, it could be interesting to define a tree model that separates TS based on their similarity with a metric such as dynamic time warping [Berndt and Clifford, 1994]. Another open option is the use of a different partition method, for example clustering or more complex tree structures such as random forests, to perform the initial dataset division and classification of new instances during forecasting.

Chapter 7

Classifying the evolution of COVID-19 severity on patients by coupling DBNs and NNs

7.1 Introduction

In the previous chapters we have dealt with extending Gaussian DBNs to higher orders (Chapter 5) and to nonlinear scenarios (Chapter 6) in order to use them as general purpose models in a wide range of applications. However, classification is also a very important objective in many different settings. While DBNs are able to perform classification in their discrete and hybrid variants, purely Gaussian DBNs cannot perform this operation. In this chapter, we will pair DBNs with static classifiers in order to apply them to a medical scenario, where we need to both forecast into the future and make predictions on the state of patients.

Throughout the COVID-19 pandemic, healthcare systems all around the world have suffered a staggering pressure due to the high number of infected patients that arrived at medical centres. The nature of this pandemic was such that patients could range from completely asymptomatic patients to those with severe, life-threatening disease. As a result, and given that the amount of resources in medical centres is limited, it was a crucial task to discern whether or not a patient presented with symptomatology that could turn out to be a critical condition or stayed only in the mild range of clinical expression.

The issue of predicting the clinical outcome of COVID-19 patients has raised much interest in recent years. Some authors opted for discerning the severity of the illness depending on certain comorbidities like heart failure [Arévalo-Lorido et al., 2022], neurodegenerative diseases [Yu et al., 2021], cardiovascular diseases [Ehwerhemuepha et al., 2021], or chronic pulmonary diseases [Momeni-Boroujeni et al., 2021]. These studies have shown that comorbidities related to COVID-19 increase the risk of death of a patient. For this reason, many efforts have been dedicated to preprocessing clinical data and selecting an appropriate set of variables that can predict the effect of the illness.

From the point of view of predicting the outcome from data, many machine learning approaches have been tested in the literature. Some authors opted for performing a statistical analysis and applying logistic regression for classifying mortality [Yu et al., 2021; Ehwerhemuepha et al., 2021; Wu et al., 2020; Xiong et al., 2020; Berenguer et al., 2021]. Another popular approach consists of training simple perceptron or multilayer neural network models to approximate a function that relates the variables in the system and classifies patient instances [Pinter et al., 2020; Dhamodharavadhani et al., 2020; Aznar-Gimeno et al., 2021; Kianfar et al., 2022]. Tree-based models like random forests [Ehwerhemuepha et al., 2021; Aznar-Gimeno et al., 2021; Cornelius et al., 2021; Pourhomayoun and Shakibi, 2021; Tezza et al., 2021] or XGBoost [Ehwerhemuepha et al., 2021; Aznar-Gimeno et al., 2021; Bertsimas et al., 2020; Vaid et al., 2020; Yadaw et al., 2020] are also some of the most popular and best performing tools for this task. In the case of graphical models, BNs have also been applied to predicting the severity of COVID-19 on patients while also trying to gain some insights into the problem at hand [Fenton et al., 2021; Vepa et al., 2021].

Another possible approach is to view the problem as a TS forecasting issue. Each patient that arrives at a hospital has their vital signs measured and has blood analysis performed. Afterwards, if the patient is not discharged and requires further care, new parameters are registered on a semi-regular basis. This generates TS data for each patient, where measurements are taken over a period of several hours each until either the patient improves or passes away. In this scenario, TS models can be applied to forecast the state of a patient and predict whether they will be suffering from severe symptoms in the near future or not. This approach has also been explored in the literature with models like DBNs [Pezoulas et al., 2021], RNNs [Villegas et al., 2021] and dynamic Markov processes [Momeni-Boroujeni et al., 2021].

In this chapter, we take a hybrid approach between static and dynamic models. We use pseudonymized data from a cohort of SarsCov2 patients admitted to the 4 hospitals belonging to Fundación Jimenez Díaz Health Research Institute, in Madrid, during the sixth wave of the COVID-19 pandemic. After preprocessing this data consisting of 15,858 patients and 532 variables and choosing an appropriate variable set with machine learning feature selection methods, we trained hybrid models between DBNs as forecasting models and several classifier models, with NNs having the best performance among them. The main idea of our proposal is to obtain the first vital signs and blood analysis from a patient and then perform forecasting of these variables with the DBN model up to 40 hours in the future. Afterwards, the classifier model can use the forecasted values to predict the patient as critical or not in this interval. Applying classifier models directly to the data recovered from a patient in its current state lacks the effect of the temporal component. By modelling the evolution of patients with the DBN, we provide information to the classifier models about the expected state of a patient in the near future. This procedure can help identifying whether a patient that just arrived at triage in a medical centre is going to worsen significantly in the following days.

This chapter includes the content of Quesada et al. [2023]. The dataset used is not made public due to privacy reasons, but all code and results are available at `https://github.com/dkesada/Class-DBN`.

Chapter outline

The rest of this chapter is organized as follows. Section 7.2 explains the architecture of the hybrid model with the neural network, where this classifying model is interchangeable with any other static classifier. Section 7.3 presents the data and the experimental results of the tested models. Finally, Section 7.4 gives some conclusions and future work.

7.2 Combining DBNs and static classifiers

When we predict COVID-19 severity on patients in the near future, we face several issues. On the one hand, we only have the data of their vital signs and blood analysis when the patient first arrives at the hospital. As we are interested on their state on the following days, we need to forecast the evolution of these variables over time. On the other hand, we need a mechanism that identifies given a state vector of a patient whether they are in a critical state or not.

7.2.1 Forecasting the state vector

When a patient afflicted with COVID-19 stays in intensive care for a prolonged period of time, they are monitored and new readings of their vital signs and blood analysis are recorded on a semi-regular basis of several hours or days. All the variables in these instances form a state vector $\mathbf{s}^t = \left[s_0^t, s_1^t, \ldots, s_n^t\right]$ at each point in time t, and the final data recovered from a patient k is a vector of instances $\mathbf{p}_k^{0:T} = \left[\mathbf{s}^0, \mathbf{s}^1, \ldots, \mathbf{s}^T\right]$ ordered in time from the oldest vital sign readings and blood analysis to the most recent ones. When we combine data from several patients, it generates a TS dataset that can be used to train a TS forecasting model. It is worth noting that the length T of the data from each patient depends on the time they spent in the hospital. If a patient is discharged with only one vital sign reading and blood analysis, then we do not have data with a time component. In this situation, this patient could not be used for training our temporal model.

Given that in our case all the variables in a state vector $\mathbf{s}^t$ are continuous, we will use a Gaussian DBN to model the dependencies and to perform forecasting. A DBN model can help us gain some insight on which variables have a greater impact on the evolution of a patient. Furthermore, the ability of DBNs to be trained with different length TS after deciding a Markovian order is also relevant in this problem, given that the number of instances per patient varies greatly. By setting a Markovian order 1, we will be able to use the data from all patients except the aforementioned ones with a single reading, where no temporal data at all can be used. As we increase the Markovian order, the sample size to learn the DBN diminishes.

After training the DBN model, we can use it to forecast the state vector of a patient up to a certain point in the future. This forecasting represents an estimate of the evolution that the patient will undergo, and it can be used to assess whether it will lead to severe symptoms or not. This process effectively gives an estimate of the future vital signs and blood analysis

of a patient without spending additional resources and time on it.

7.2.2 Classifying critical values

The task of evaluating whether a patient is in a critical state of the COVID-19 infection has been performed in the literature mainly through some kind of medical score [Fan et al., 2020] or by labelling instances due to some external indicator, for example being transferred or not to the intensive care unit. If we obtain a labelled dataset of patients through any of these methods, we can then take a machine learning approach by training classifier models that identify whether a patient is in a critical state given their state vector $\mathbf{s}$.

If we combine this approach with the forecasting of the state vector, we get a hybrid model between static classifiers and TS models that is capable of evaluating the present and near future condition of a person suffering from COVID-19. When a patient arrives at a hospital and gets their vital signs and blood analysis recorded, we obtain the state vector $\mathbf{s}^0$ of the very first instant of time. Then we can feed $\mathbf{s}^0$ to a trained classifier model to evaluate whether this patient is already in a critical state or not. If this is not the case, we can then use $\mathbf{s}^0$ as the starting point for our DBN to perform forecasting. This will return the values of $\hat{\mathbf{s}}^1, \hat{\mathbf{s}}^2, \ldots, \hat{\mathbf{s}}^t$ up to a certain point t in time. All these state vectors can in turn be classified to evaluate the expected severity of the symptoms in that patient. With this method, we can see if a patient is expected to end up suffering from critical COVID-19 and when approximately will this situation occur. To illustrate this whole process, a schematic representation of this framework can be seen in Fig. 7.1.

Our proposed framework supports any kind of classifier that is able to produce a discrete prediction given a continuous state vector $\mathbf{s}^t$. We used a modular implementation where the classifier used can be a support vector machine, an XGBoost, a neural network and a Bayesian classifier. All these classifiers been used in the literature and applications where one is more effective than the others can be found. Due to this architecture, any other classifier model could potentially be introduced as a new module if needed.

7.3 Experimental results

For our experiments, we used a dataset consisting of pseudonymized data from the 27th of October 2021 to the 23rd of March 2022 recovered from 4 different Spanish hospitals from the Fundación Jiménez Díaz in Madrid. and the study protocol was approved by their local Ethics Committee. This data covered patients that had confirmed cases of COVID-19 via a positive PCR test. After preprocessing it, we used this data to fit our proposed model and evaluate its capabilities to predict the future critical status of patients suffering from the COVID-19 disease.

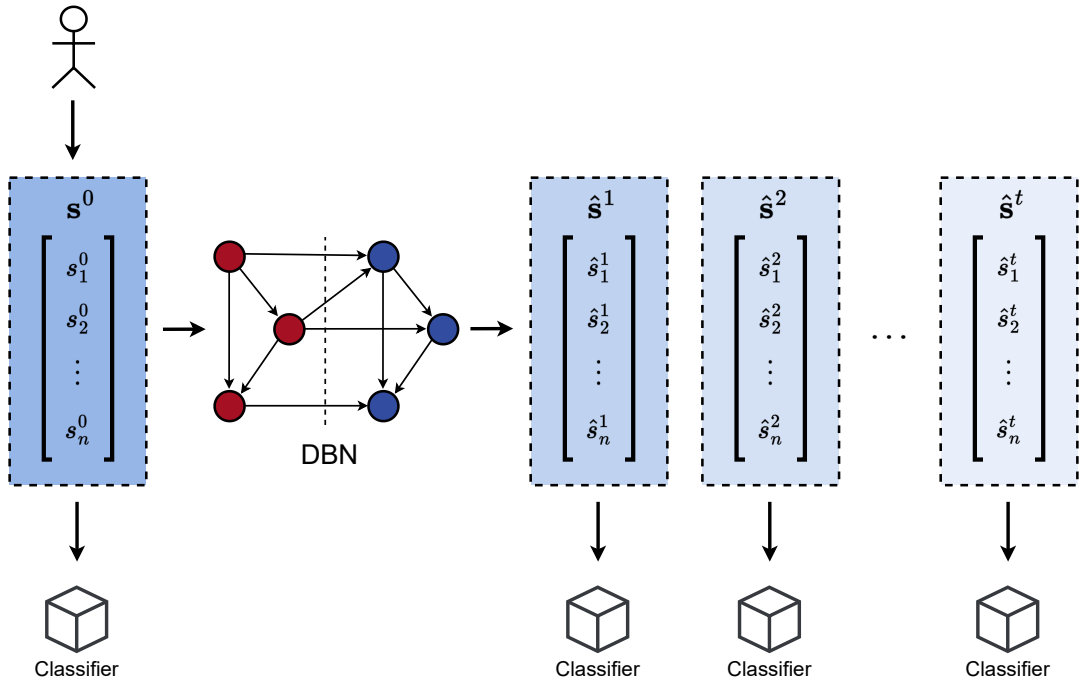


Figure 7.1: Schematic representation of the classifier-DBN framework. After obtaining a state vector $\mathbf{s}_0$ from a patient, we can use it to forecast the next t state vectors with the DBN model and check if they are critical at each time t with our static classifier.

7.3.1 Preprocessing

In our raw dataset, there are a total of 21,032 rows with incomplete data from 15,858 patients and 532 variables, most of which had missing values for the majority of patients. This is a common occurrence in a medical dataset of these characteristics, given that not the same tests are performed to all the patients and some of the results have to be recorded manually.

The consecutive rows in the dataset that correspond to a same patient are ordered chronologically resulting in TS sequences. However, the frequency at which the instances were recorded is uneven. This is due to the fact that performing blood analysis from patients and obtaining the results does not take a fixed amount of time and is not always performed after fixed intervals. To tackle this issue, we established a period of 4 hours between each row and formed batches of instances where missing data was filled with the average values of the rest of instances in the same batch. This 4-hour period was chosen because usually new tests were performed on average roughly after every 4 hours in our dataset.

From the 21,032 rows, 13,971 were from patients that appear only in a single instance, since in the sixth pandemic wave in Madrid the vast majority were discharged from the hospital afterwards due to mild symptomatology and only 48 of these patients passed away. This data cannot be used to train the DBN models, given that a single register is not enough to form a TS sequence. However, it will be used to train the classifier models. From the remaining patients with more than a single instance, the majority of them have either two or three rows of recorded values. To illustrate this, we show a histogram with the distribution

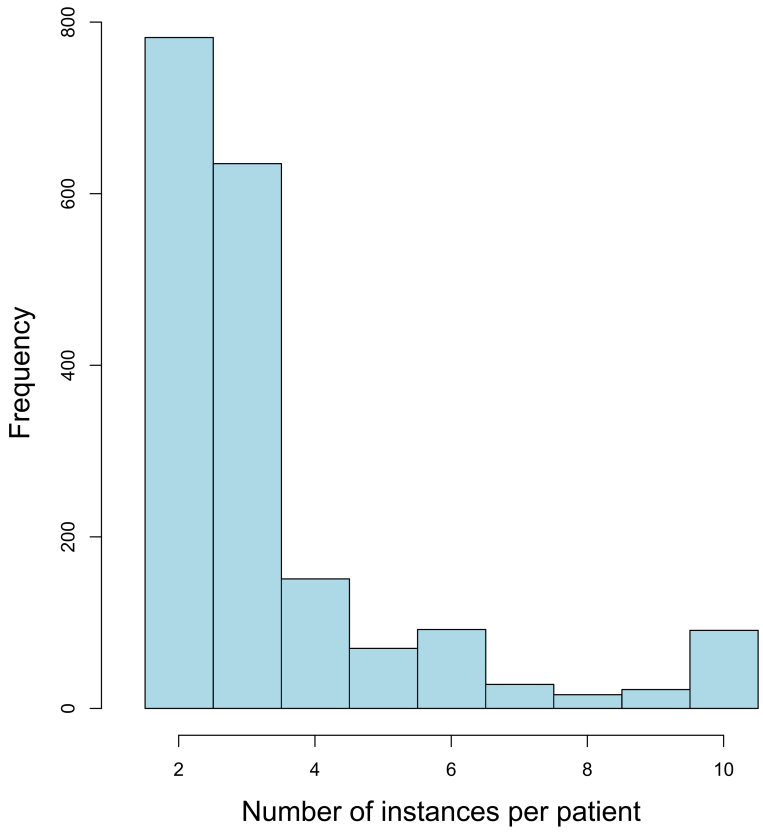


Figure 7.2: Histogram with the number of instances per patient greater than 1 in the dataset. Inside the last bin we have grouped all the patients with 10 or more instances. A higher number of instances indicates a longer stay in the hospital and as such a more severe case of COVID-19, which was far less common than a mild case in the sixth wave.

of the number of instances per patient in Fig. 7.2.

Regarding the 532 variables in our dataset, most of them correspond to specific values in uncommon tests and analysis, and they have over 70% of missing values across all instances. In our case, we have opted for reducing the number of variables to only those that are obtained from the vital signs of a patient, like their body temperature and their heart rate, their descriptive characteristics like age, gender and body mass index, and the variables from a regular blood analysis like the albumin and D-dimer values. All these variables are routinely taken when a patient arrives at the emergency room and obtaining them does not pose a severe expense of resources. This reduced the number of variables to 62, and from those we chose to retain all vital sign readings and descriptive characteristics, while applying feature subset selection on the blood analysis related variables, resulting in a total of 38 variables. This subset selection was performed via random forest importance on classification on our objective variable, which will be whether or not a patient was admitted to the intensive care unit or passed away. This is what defines our critical cases of COVID-19, which are only a

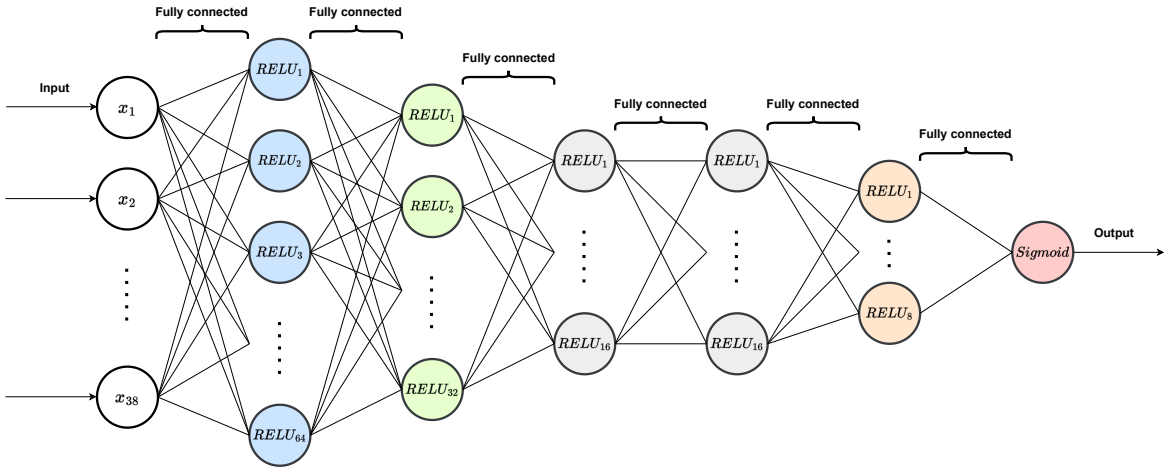


Figure 7.3: Structure of the NN model used in the experiments.

18.8% of the total number of patients in our dataset.

7.3.2 Classification results

In this section we show the experimental results obtained with different combinations of classifier-DBN models. For our experiments, we used an XGBoost, a support vector machine, a neural network and a Bayesian classifier. In particular, this Bayesian classifier is a tree-augmented naive Bayes built following the hill-climbing super-parent (HCSP) algorithm [Keogh and Pazzani, 2002]. The NN had an internal structure of 5 hidden dense layers with 64, 32, 16, 16 and 8 neurons each. They all used RELU activation functions and had their weights initialized with the identity. The last layer used a single neuron with a sigmoid activation function for binary classification. A result greater than 0.5 is equated to predicting a critical status for a patient, and a result lesser or equal to 0.5 predicts a non-critical scenario. A representation of this structure can be seen in Fig. 7.3. All the project was coded in R and is publicly available online in a GitHub repository[1]. The dataset used is not made public due to privacy and legal reasons.

Regarding the software we used in our experiments, the DBN models were trained using our own public package **dbnR** [Quesada, 2021], the XGBoost models were trained with the **xgboost** package [Chen et al., 2022], the support vector machines (SVMs) were trained with the **e1071** package [Meyer et al., 2022], the neural networks with the **keras** R interface [Allaire and Chollet, 2022] and the Bayesian classifiers with the **bnclassify** package [Mihaljević et al., 2018]. The parameters of each classifier were optimized using differential evolution with the R package **DEoptim** [Mullen et al., 2011] guided by the geometric mean (g-mean) [Tharwat, 2021] of the models. This metric is defined as $g_m = \sqrt{recall * specificity}$, and uses all values in the resulting confusion matrix when calculating the final score. Using both the recall and specificity of the predictions ensures that the imbalance between critical cases and non-

[1] https://github.com/dkesada/Class-DBN

critical cases is taken into account when optimizing the parameters. We do not want a model optimized solely with respect to accuracy because it would lead to models that only predict the majority class of non-critical for all patients.

To alleviate the issue of imbalanced data, we also applied SMOTE oversampling with the **DMwR** package [Chawla et al., 2002; Torgo, 2010] to synthetically generate instances of both critical and non-critical cases. This is a common practice that creates synthetic data to offset the difference between the number of instances of the majority and minority classes. In our case, we will use SMOTE to create modified datasets for training our classifiers. This will help the models to avoid getting stuck on predicting the majority non-critical class for almost all instances.

To test our hybrid models, we take the state vector of a patient in an instance and forecast up to 10 instants into the future with the DBN model. Then, we use the classifier model to identify each of these forecasts as critical or not and we compare the predicted label with the true label of the instance. Given that each instance is separated from the next one by 4 hours, in total we forecast 40 hours into the future with the DBN model. With this method, we will be able to see the behaviour of the classifiers and the changes in accuracy and g-mean as we use state vectors from further into the future. The average results obtained across all forecasts of the models can be seen in Table 7.1. Additionally, we performed the Kruskal-Wallis test on our results to evaluate whether the results obtained from the different models are statistically significant from one another or not. The Kruskal-Wallis test allows us to perform a non-parametric statistical test on the samples of all of our 4 models that does not assume normality on the samples and allows multiple models to be tested simultaneously. The test obtains a p-value of 9.71e-5, which indicates that at least one of the sample results from the models is better and has statistically significant differences from the others.

	g-mean	**Accuracy**	**Train (h)**	**Exec (s)**
XGBoost	0.455 ± 0.032	0.698 ± 0.012	1.950	9.634
SVM	0.522 ± 0.056	0.735 ± 0.015	1.145	9.654
NN	0.541 ± 0.041	0.771 ± 0.016	1.384	9.863
HCSP	0.468 ± 0.072	0.736 ± 0.018	1.046	9.878

Table 7.1: Mean results in terms of the accuracy, g-mean score, training and execution time of the models on average for all the experiments. It is worth noting that training time includes optimization of parameters, which involves the creation of multiple models to evaluate different configurations.

The results in Table 7.1 show that, on average, the most accurate model is the neural network in both accuracy and g-mean. The performance of both the SVM and the HCSP are very similar in terms of accuracy, but the difference in g-mean score of the SVM shows that it is able to discern better the more uncommon critical instances. For this particular case, although the XGBoost model is very popular in the literature, it obtains worse overall results than the rest of the classifiers. In our experiments, due to the imbalance between classes we had to find a compromise between the global accuracy and the accuracy of the

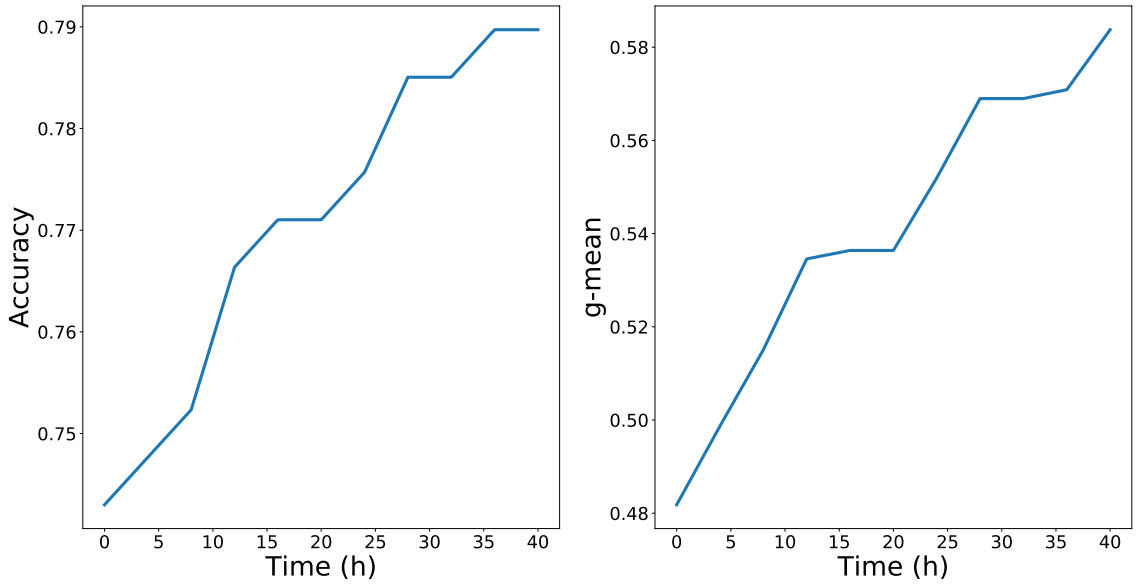


Figure 7.4: Classification results of the neural network model as we feed it with the state vectors further ahead in time with the DBN model. The classification performance of the neural network improves monotonically by combining it with the DBN forecastings.

minority class. If left unchecked, the models would become biased to the majority class and predict almost unanimously every single instance as non-critical, invalidating the use of the model while obtaining accuracies close to 90%. By using the g-mean as optimization metric in combination with the SMOTE oversampling, we were able to alleviate this problem. A high accuracy on the majority class of non-critical patients will be able to help reduce the oversaturation of ICU resources, given that all models can evaluate whether a patient will reach a critical state of the COVID-19 infection or not in less than 10 seconds. On the other hand, being able to discern the few critical cases that arise is also needed to help doctors determine which patients need more specific care to try to reduce the mortality rate. On the topic of training time, training and tuning the models takes on average between one and two hours. Given that this kind of models should not need to be retrained until some significant issue happens with the disease, like a new variant or new specific symptoms appear on patients that differ from the training data used, these training times should be reasonable to be performed once. Another possible approach in a real world scenario could be that of periodic retrainings on a weekly or monthly basis, where a lower volume of data would obtain faster training times.

Given that the model with the NN obtains the best average results, we show in Fig. 7.4 the details of its performance depending on the time horizon. The first instant at 0 hours is equivalent to performing classification with the NN model directly to the state vector obtained from the patient. From there on, we perform forecasting up to 40 hours with the DBN model of this state vector and use the results as input for the NN model. We can see that the NN model performs considerably better if we couple it with the DBN to classify

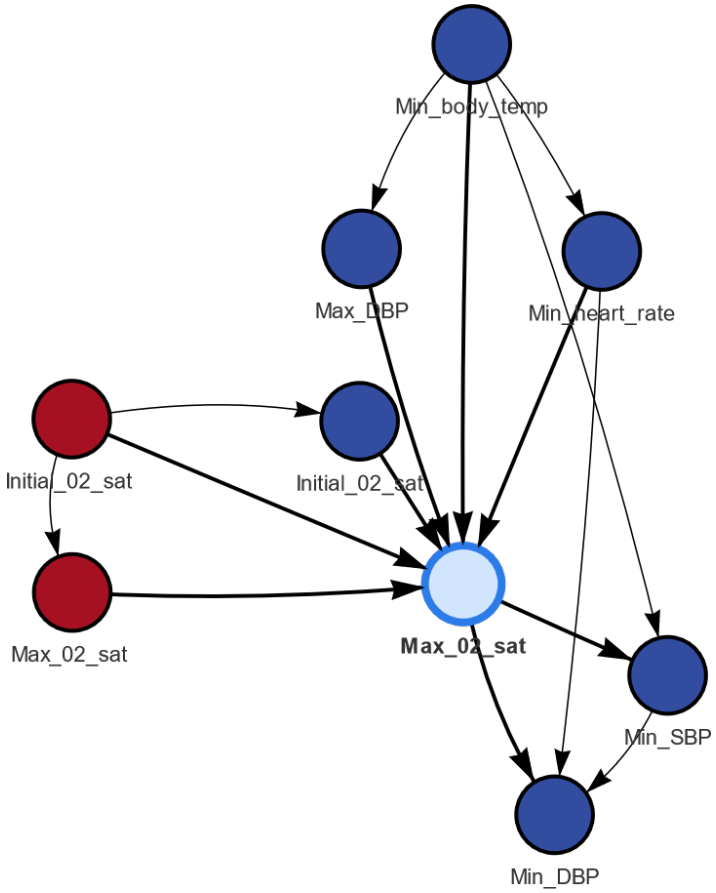


Figure 7.5: Subset of relevant variables to the forecasting of maximum oxygen saturation (*Max_02_sat* in light blue) in the DBN model. The initial and maximum oxygen saturation variables from the last instant (*Initial_02_sat* and *Max_02_sat* in red) affect the calculation of the next 4 hours maximum oxygen saturation value. Other variables like body temperature *Min_body_temp*, systolic *Min_SBP* and diastolic *Min_DBP* blood pressures and heart rate *Min_heart_rate* also influence this value in the forecast.

the forecasted state of the patients rather than their initial state. As we forecast the state vector of patients further into the future, the NN improves its classification performance monotonically. However, we cannot extend this forecasting indefinitely. Trying to forecast longer term than the proposed 40 hours resulted in too much accumulated noise and error, which degraded the performance of all classifiers.

In addition, DBNs perform multivariate inference and are interpretable models. This allows them to offer doctors the forecasted values of any variable in the system as well as the underlying relationships between the rest of variables that led to those results. In the case of relevant values like the oxygen saturation of a patient, which is a good indicator of the state of a patient suffering from respiratory issues, we show an example of the relationships present in the DBN model in Fig. 7.5. This subgraph shows the variables directly related to the maximum oxygen saturation registered in a 4-hour interval. We can see the previous maximum value of oxygen saturation from the last instant, which is to be expected due to

the autoregressive component of TS. On a similar note, the initial values of oxygen saturation registered serve the model to define the range of the maximum value: lower initial oxygen saturation will likely lead to lower maxima and vice versa. Additionally, we also find that body temperature leading to fever, maximum diastolic blood pressure and minimum heart rate are also direct indicators of maximum oxygen saturation and play an important role in its forecasting. Lastly, minimum systolic and diastolic blood pressures are also affected. A lower level of oxygen saturation will cause higher systolic and diastolic blood pressures, increasing both minima. This situation is reflected in the fact that these values are child nodes that depend on the current value of oxygen saturation.

7.4 Conclusions and future work

In this chapter we have presented a hybrid model between DBNs and static classifiers where the state vector recovered from a patient suffering from COVID-19 is used to forecast their future state. This information is then used to assess how severe will their infection be in the following 40 hours based on their current vital signs and basic blood tests. This method shows the best performance when combining DBN and NN models. While the NN is capable of discerning whether or not a patient will reach a critical state with better accuracy and g-mean than the other classifiers, the DBN adds an explainable layer regarding the variables extracted from the patient. This model could help doctors decide whether or not a patient needs further specialized care and allow for a better organization of the resources available in medical centres. Additionally, we offer the code of all our models online for future reference and use.

For future work, this model could be applied in different environments that require forecasting TS and classifying the state of a system. The combination of a generative model that forecasts the state of a system with a classifier model that evaluates this expected future state is a promising framework that could prove useful in several applications like remaining useful life estimations for industry 4.0. Another possible improvement could be the potential use of the DBN model as a simulator, introducing interventions in the forecasting in order to see the effects that possible actions can have in the expected future. In the medical case, the effects of specific drugs or treatments could potentially be reflected in the DBN predictions, and in other industrial cases this could lead to optimizing the expected future based on possible interventions in the initial state.

Chapter 8

dbnR: Gaussian DBN learning and inference in R

8.1 Introduction

As we progressed through this thesis, we developed **dbnR**, an R package that encapsulates all the process of generating a DBN model, performing inference with it and visualizing the network. This code served as a framework where we could compile all of our new developments and easily deploy our models on different datasets. In this last chapter, we will go into detail on how **dbnR** is coded and how does it work. One of the most important parts of this thesis was for it to be relevant and useful to researchers and professionals all around the world, and the best way to make it so was to generate a useful open source tool that can be easily applied to some data.

In recent years, the use of DBNs has gained popularity in several fields, as shown in previous chapters. Most of the time, authors resort to implementing the models themselves. Other times, researchers opt for adapting existing static BN packages, such as **bnlearn** [Scutari, 2010] in R or **pgmpy** [Ankan and Panda, 2015] in Python. Both of these options can be very time-consuming and result in *ad hoc* implementations that, for the most part, are not extendable to new applications.

It is uncommon to find software packages specifically designed for DBNs. In R, we can find **dbnlearn** [Fernandes, 2020], a package that allows creating univariate DBNs and making predictions of the next instant with them. These univariate DBNs only have a single variable repeated in several instances of time and always have the same fixed structure, where nodes are connected to the node in the next instant and to the objective variable. Underneath, the package uses the parameter learning and inference offered by **bnlearn**. In Python, there is the **pyAgrum** package [Ducamp et al., 2020], which is a wrapper of the C++ library **aGruM**. It offers learning, inference and visualization of BNs and supports DBNs, but it only allows the use of discrete variables and discretizes continuous ones. If we opt for adapting existing BN software for a dynamic scenario, there are several possibilities. The most versatile one is the

use of **bnlearn** with certain restrictions and pre-processing steps for learning DBN structures with BN methods. Afterwards, one can use either **gRain** [Højsgaard, 2012] for inference with discrete variables or **rbmn** [Denis and Scutari, 2021] for inference with continuous ones. In Python, one can use either **pgmpy** or **bnlearn** [Taskesen, 2020], which has the same name but a different author from the original R package by Scutari, to potentially fit a DBN model. However, both packages only support discrete variables, and only **pgmpy** has the skeleton of the classes for a potential extension to DBNs. In the case of discrete DBNs, these options could offer a solution after the user adapts them, but neither offers an option for continuous variables.

The process of training a DBN model from data and forecasting has several intermediate steps [Koller and Friedman, 2009]: adapting the dataset for TS learning, applying a structure learning algorithm, visualising the network, using an exact or approximate inference method and running a forecasting motor. With **dbnR**, our objective is to create a simple pipeline where upon providing some data, we can obtain a model that we can visualize and use to perform inference. All the intermediate steps are encapsulated and parametrized inside the package to allow both a simple deployment and the possibility of tuning the learning and inference process to the user needs. Throughout this chapter, we will show several examples of code executed directly on the R prompt with the `R>` symbol at the beginning. These are basically lines of code and what would they return if run in a real prompt.

This chapter includes the content of an article submitted to the Journal of Statistical Software. The stable **dbnR** [Quesada, 2021] source code and binaries can be found in the Comprehensive R Archive Network (CRAN)[1], while active development is underway in GitHub[2].

Chapter outline

The rest of the chapter is organized as follows. Section 8.2 covers the definition of the DBN model in the package. Section 8.3 discusses the structure learning algorithms available in **dbnR** and the visualization tool. Section 8.4 presents the inference and forecasting methods. Section 8.5 describes the main pipeline, functions and the classes of the package. Section 8.6 showcases a complete example of applying DBNs for forecasting some data. Finally, Section 8.7 offers some final remarks and conclusions.

8.2 BN definition in dbnR

In the literature, the most well-known package that offers support for all types of BNs is **bnlearn**. It allows training and inference with all of them, but it does not include DBNs. In our package, we opted to focus on Gaussian DBNs for two reasons. First, real-world data recovered from sensors are usually real valued. Second, exact inference with Gaussian BNs is much faster to compute than with their discrete counterparts due to the CPD of their nodes.

[1] https://cran.r-project.org/package=dbnR
[2] https://github.com/dkesada/dbnR

In Gaussian BNs, we assume that all the variables in our system follow a normal distribution represented as a linear Gaussian model, as in Equation (2.6).

Given the popularity of **bnlearn**, we opted to extend some of the functionality of this package to the case of Gaussian DBNs. As a result, all **dbnR** networks extend the S3 classes 'bn' for the graph structure and 'bn.fit' for the fitted networks offered by **bnlearn**. The new resulting 'dbn' and 'dbn.fit' classes enable all the graph operations and the score functions coded in **bnlearn** to work with the DBN models in **dbnR**. The network structure in **bnlearn** uses an R 'data.frame' as the main data structure for both the BN graph and the fitted model. This 'data.frame' stores the information of each node parent and children as string vectors and the values of the parameters $\beta_{0i}, \ldots, \beta_{ki}$ and σ_i^2 as numeric vectors. To allow the previously mentioned compatibility with **bnlearn**, we maintained this structure and added two new modifications; we switched the use of 'data.frame' to 'data.table' [Dowle and Srinivasan, 2021], and we added the multivariate Gaussian equivalent of the network as a new attribute of the S3 'dbn.fit' object. The switch to 'data.table' was motivated by the efficiency of this data structure compared to 'data.frame' in terms of making queries, operating with the data and passing it down between functions. On the other hand, to allow fast exact inference, we switched to C++ using **rcpp** [Eddelbuettel and François, 2011] to calculate the mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, and we store them as attributes of our S3 object so that they can be used in place of the graph structure when forecasting with the network. We use C++ several times in our package, especially when we need to perform heavy computations or matrix operations inside structure learning algorithms.

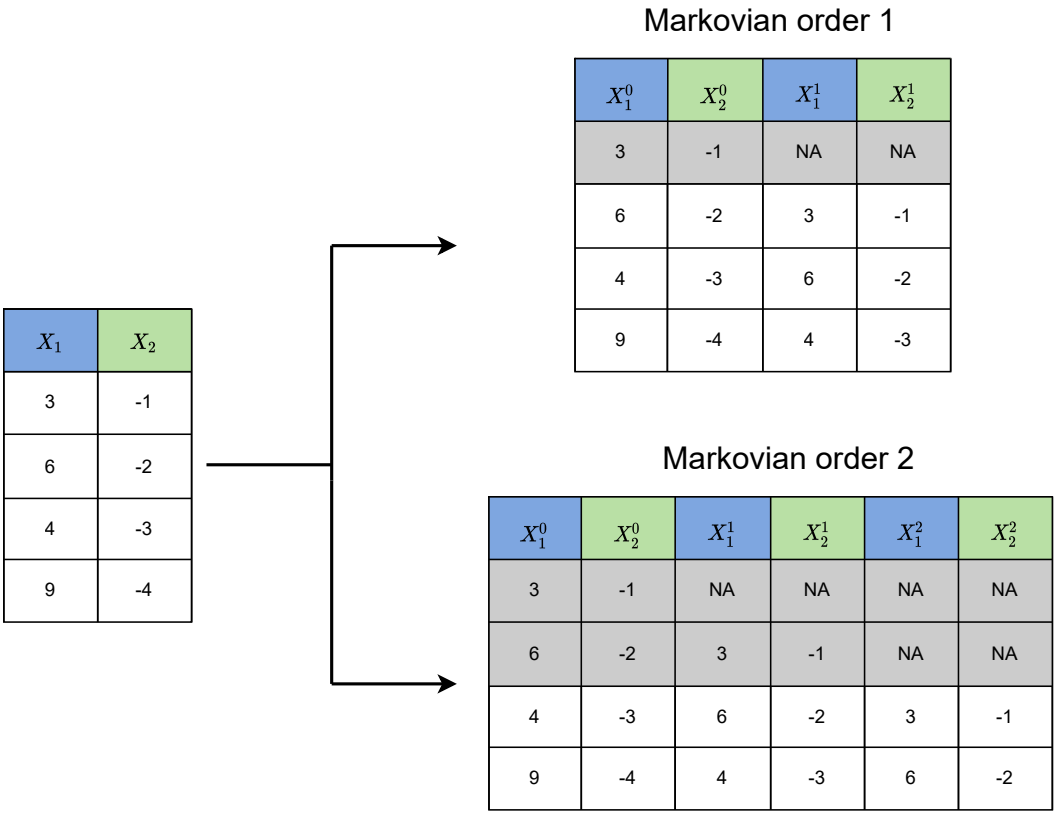
After extending the 'bn' class from **bnlearn** to the DBN scenario, we opted for taking a different route in terms of the structure learning algorithms and the inference motor inside **dbnR**.

8.3 Structure learning

To learn the effect that past values of the variables have on the present, the first step we need to take is to adapt our datasets to the time discretization of DBNs. In TS data, instances are arranged in chronological order, where the oldest instance is usually the first row. Depending on the desired Markovian order, we need to shift the rows in our dataset to ensure that the values of several rows are grouped into a single row. To illustrate this process, we show an example in Figure 8.1. Unlike in (3.5), we set $t = 0$ as the most recent time slice instead of the oldest. This is merely a convention change motivated by an easier implementation of the package architecture. Given that we allow an arbitrary Markovian order, it is more convenient to have the most recent time slice always named t_0 during inference regardless of the order.

In **dbnR**, the function `fold_dt` performs this dataset modification automatically. To begin the learning process, we call this function to adapt our data to the desired format. The example in Figure 8.1 can be replicated with the following code:

```
R> df <- data.frame(X1 = c(3, 6, 4, 9), X2 = c(-1, -2, -3, -4))
```



X_1	X_2
3	-1
6	-2
4	-3
9	-4

X_1^0	X_2^0	X_1^1	X_2^1
3	-1	NA	NA
6	-2	3	-1
4	-3	6	-2
9	-4	4	-3

X_1^0	X_2^0	X_1^1	X_2^1	X_1^2	X_2^2
3	-1	NA	NA	NA	NA
6	-2	3	-1	NA	NA
4	-3	6	-2	3	-1
9	-4	4	-3	6	-2

Figure 8.1: Example of transforming a dataset with two variables X_1 and X_2 into several variables depending on the desired Markovian order. The rows in the original data are ordered from the oldest recorded values, $X_1 = 3$ and $X_2 = -1$, to the newest. The grey rows contain missing values and should be deleted.

```
R> df

  X1 X2
1  3 -1
2  6 -2
3  4 -3
4  9 -4

R> dbnR::fold_dt(df, size = 2)

   X1_t_0 X2_t_0 X1_t_1 X2_t_1
1:      6     -2      3     -1
2:      4     -3      6     -2
3:      9     -4      4     -3

R> dbnR::fold_dt(df, size = 3)

   X1_t_0 X2_t_0 X1_t_1 X2_t_1 X1_t_2 X2_t_2
```

```
1:      4     -3      6     -2      3     -1
2:      9     -4      4     -3      6     -2
```

Note that due to restrictions in variable names, the time slice that each variable corresponds to is written as `t_x`. The `size` argument determines the total number of time slices in the network, that is, the Markovian order plus one.

In the most recent **dbnR** version (version 0.7.9), three structure learning algorithms are available: a version of Trabelsi et al. [2013] dynamic max-min hill-climbing, Santos and Maciel [2014] binary particle swarm optimization algorithm and Quesada et al. [2021a] natural number order invariant encoding PSO algorithm. There is only a single function that handles the calls to all the different algorithms:

```
learn_dbn_struc(dt, size, method, f_dt, ...)
```

The `learn_dbn_struc` function requires the training dataset and a desired size, and depending on the `method` argument (`"dmmhc"`, `"psoho"` or `"natPsoho"`, respectively, for the three aforementioned methods), it will call the appropriate non-exported function of the specific structure learning algorithm inside the package. The `f_dt` argument allows the user to pass down a dataset shifted manually or with the `fold_dt` function in case it is needed, and the ellipsis can be used to pass down further algorithm-specific arguments. It will return an S3 'dbn' object that extends the 'bn' object from **bnlearn**.

Once we have the structure of the DBN, we need to fit its parameters to our data with the following function:

```
fit_dbn_params(net, f_dt, ...)
```

The `fit_db_params` function takes a 'dbn' object and a dataset shifted with the `fold_dt` function to learn the parameters of the DBN and return a 'dbn.fit' object. The function allows us to use the maximum likelihood estimation implemented in **bnlearn** to learn the parameters of the provided network from the shifted dataset. The $\boldsymbol{\mu}$ vector and the $\boldsymbol{\Sigma}$ matrix are estimated from the fitted DBN and added along with the `size` of the network as attributes of the 'dbn.fit' object automatically in the `fit_dbn_params` function for future inference and forecasting.

8.3.1 Dynamic max-min hill-climbing

This was the first algorithm implemented in **dbnR** mainly as an extension of the offered methods in **bnlearn** for the case of DBNs. The DMMHC algorithm builds a DBN structure in two steps: learn the static structure and learn the transition network. The static structure is the BN structure of the first time slice, with only intra-slice arcs, and it represents the effect of the variables in the same time instant. The transition network is the structure that represents only the inter-slice arcs in the DBN and the effects that the past has on the present. The basic max-min hill-climbing algorithm is used to learn both of these structures separately.

In our case, we used the max-min hill-climbing implementation available in **bnlearn** to learn both of these networks, and then combined them into a single structure. To force the additional constraints on arcs imposed by DBNs, we make use of the `blacklist` argument that allows the user to introduce a matrix of forbidden arcs that will not appear in the final network. This `blacklist` argument will be used to ban inter-slice arcs backwards in time while learning the transition network. The construction of the `blacklist` matrix is performed automatically with regular expression operations based on the names of the variables. The user will not have to worry about this process, and additional arcs might be added to the `blacklist` if needed.

Once both networks are built, we combine all their arcs to generate the final network structure that can be used in the `fit_dbn_params` function to fit its parameters. The original algorithm in Trabelsi et al. [2013] is defined for Markovian order 1 DBNs, and extending it to higher orders only implies learning a larger transition network with more than two time slices. It performs well for Markovian order 1 or 2 networks, but due to the super-exponential nature of the number of possible BN structures depending on the number of nodes [Robinson, 1977], it scales poorly to higher orders.

8.3.2 Binary encoding PSO

The first alternative to the DMMCH is the PSO algorithm presented by Santos and Maciel [2014]. In this case, the problem of finding an optimal DBN structure is transformed into an optimization one, where the quality of each network is evaluated with a score. The graph structure is encoded into a list of lists, where the parents of each node are defined in a binary representation. An arc from one node to another is defined by a `1` in this list, and its absence is defined by a `0`. To drastically reduce the space of possible DBN structures, only inter-slice arcs from older time slices to the most recent one are allowed.

This algorithm has been implemented from scratch in **dbnR** using **R6** classes [Chang, 2021] and follows an object-oriented programming paradigm. The 'Particle' class in the framework contains a 'Position', which encodes the binary list of lists, and a 'Velocity', which contains arc additions or deletions with the same binary representation. The custom operations defined for these positions and velocities can be found in Santos and Maciel [2014] and are encapsulated inside the **R6** classes. These operations also switch to C++ when needed. The particles are evaluated by calculating the BIC score [Schwarz, 1978] or the BGe score [Geiger and Heckerman, 1994] of the graph encoded in each particle. We defined a 'PsoCtrl' class that initializes and contains all the particles and controls the execution of the search by evaluating the positions of the particles, obtaining the local and global optima, calculating new velocities and moving the particles. The best position found at the end of the process is transformed into its equivalent DBN form and is returned as the solution of the search.

8.3.3 Natural number invariant encoding PSO

The implementation of this algorithm is in many ways similar to the binary PSO algorithm, as they share the same pipeline. The main differences between them are the encoding of the DBN structures and the operations between positions and velocities. In this case, the networks are encoded in vectors of natural numbers of constant length regardless of the Markovian order desired. Each particle consists only of a 'numeric' vector, where each number corresponds to a single node of the network in t_0. This number encodes the information about which arcs from previous nodes point to that specific node. The binary bitwise representation of the natural number indicates which arcs are present in the DBN and which arcs are not. With this encoding, a higher Markovian order only generates larger natural numbers, but it does not increase the size of the 'numeric' vectors. The operations of additions and deletions of arcs are now performed bitwise with custom operators on the natural numbers representing the existing arcs, making this encoding scalable to high orders. Similar **R6** classes 'natParticle', 'natPosition' and 'natVelocity' were generated for this algorithm to follow similar developing procedures.

8.3.4 Structure visualization tool

To offer the possibility of visualizing the graph structure of both the BNs learned with **bnlearn** and the DBNs learned with **dbnR**, we implemented a tool using the **visNetwork** package [Almende et al., 2019]. This tool is included in **dbnR**, but the **visNetwork** package is listed as suggested and will only be downloaded in case the user needs to plot some network. By using **visNetwork**, we plot the graph structures as HTML widgets that the user can interact with by highlighting arcs and nodes and by clicking on and dragging the nodes. The tool is intended only for visualization purposes, and any changes to the graph structure have to be made programmatically.

Plotting a BN or a DBN structure can be done with a single function call:

```
plot_network(structure, ...)
```

The `structure` argument can be a 'bn', a 'bn.fit', a 'dbn' or a 'dbn.fit' object obtained after learning a network structure with either **bnlearn** or **dbnR**. The ellipsis argument is used to provide two additional parameters for the DBN case: the `offset` argument, which modifies the size of the blank space between time slices in the plot, and the `subset_nodes` argument, which allows the user to plot only a certain subgraph from the whole network by providing a vector with the names of the desired nodes. An example of two network structures plotted with this tool can be seen in Figure 8.2.

8.4 Inference and forecasting

Once we have the structure and the parameters of a DBN, we can use the model to perform inference over some data. In **dbnR**, we offer an approximate inference method and

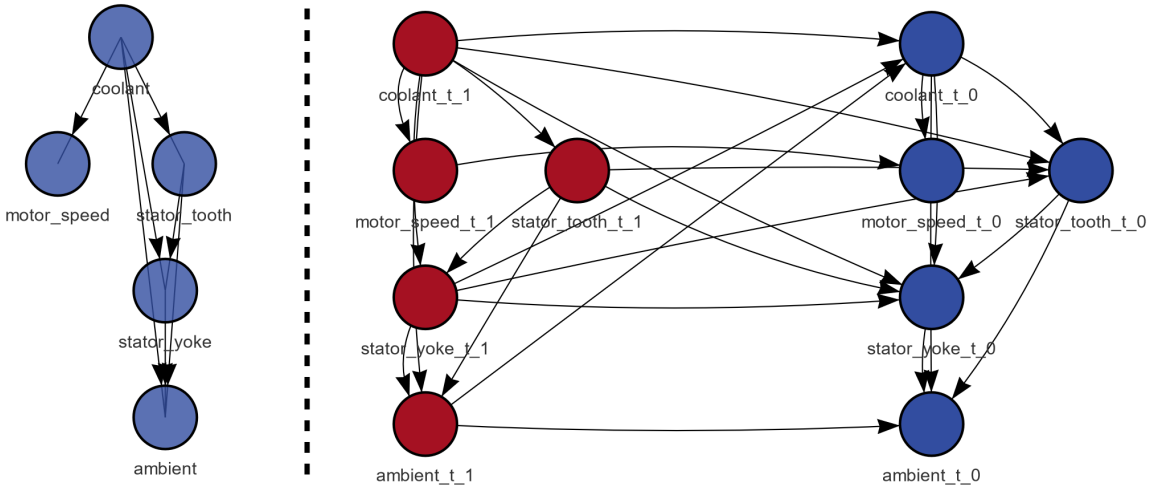


Figure 8.2: Example of the visualization of the structure of both a BN (left) and a DBN (right). The nodes on both networks can be clicked on to be highlighted and dragged to reposition them.

an exact one. The approximate inference is performed via the likelihood weighting [Korb and Nicholson, 2010] based on Monte Carlo sampling offered in **bnlearn**. The exact inference algorithm has been implemented in **dbnR** specifically for Gaussian BNs by using the equivalent multivariate joint distribution.

8.4.1 Exact inference

One way to perform exact inference in a Gaussian BN is to use the equivalent multivariate joint distribution of the network. To do this, we can use the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ that we calculated when learning the network structure. When we perform inference, the values of some of the variables are known beforehand and used to predict the most likely values for the rest of the variables in the system. Note that in the degenerated case where no evidence whatsoever is provided, the values predicted for the variables are the marginal means in $\boldsymbol{\mu}$. In the dynamic scenario, usually the variables in the past are observed and will be used to perform inference over the variables in the present. Multivariate Gaussian inference as shown in Equations (2.26), (2.27) and (2.28) can be performed in **dbnR** with the function

```
mvn_inference(mu, sigma, evidence)
```

The `mu` and `sigma` arguments, which correspond to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ respectively, are stored in the 'dbn.fit' as attributes, and the `evidence` argument is a named vector with the values of the observed variables in $\mathbf{X}_2$. This function returns a list with both the calculated $\boldsymbol{\mu}_{1|2}$ from (2.27) and $\boldsymbol{\Sigma}_{1|2}$ from (2.28). Typically, the `mvn_inference` function is not used outside of **dbnR** because it is already encapsulated in other exported methods for prediction and forecasting with DBN models, but we exported it too in case the user needs to perform inference over

only a specific subset of nodes or wants to perform inference over a multivariate Gaussian distribution outside of the scope of the package.

8.4.2 Forecasting TS

When using DBNs to deal with TS data, one of the most common operations that we can perform is forecasting up to some horizon T. In **dbnR**, we perform forecasting with DBN models using a sliding window procedure. This process was explained in Chapter 3.5 and illustrated in Figure 3.6. We perform as many inference steps as needed with this procedure in order to reach the desired horizon T. Finally, when the forecasting is completed, the values of the target variables at each instant are returned, and the mean absolute error is calculated with the original TS if some test data are provided. If the user is making predictions without knowing the future values of the TS, as in a real-world application, no metrics will be calculated.

The whole process of forecasting up to some horizon T is encapsulated in the function

```
forecast_ts(dt, fit, obj_vars, ini, len, mode, print_res, plot_res)
```

By providing a 'data.frame' and a 'dbn.fit' with the `dt` and the `fit` arguments, the **dbnR** package handles the moving window procedure underneath. The forecasting can be tuned by defining the target variables with `obj_vars`, setting the initial instance of the forecasting in the 'data.frame' with `ini`, defining the horizon T regarding how long the forecasting should be with `len` and if either exact or approximate inference should be used with the `mode` parameter. If test data are provided in `dt` and the `print_res` argument is set to `True`, the MAE of the forecasting compared to the original TS will be printed. The `plot_res` argument shows a plot of both the original and the predicted TS. An example of forecasting a TS and plotting the results can be seen in Figure 8.3.

An additional argument `prov_ev` can be provided to the `forecast_ts` function, which allows the user to give specific future evidence to the DBN in each forecasting step. This can be useful when employing a DBN as a simulator to make interventions in the forecasting and see the effects that they have in the prediction profiles.

8.4.3 Smoothing

Additionally, DBN models can perform smoothing operations over TS [Koller and Friedman, 2009]. In this context, smoothing refers to, given some initial evidence from instants 1 to t, performing inference over $p(\mathbf{X}^0|\mathbf{X}^{1:t})$. Essentially, we predict the past given the current value of the variables in our system. Afterwards, we move all evidence backwards in the same manner as the sliding window from the forecasting case, but in the opposite direction in time. If we repeat this process up to horizon T in the past, we will obtain a TS that reflects the predicted state of the system along several instants in the past. Typically, this operation is performed when we do not know the past state of the system, for example, due to missing data or when we want to check how much our past data differ from the smoothed values to check for faulty sensor recordings.

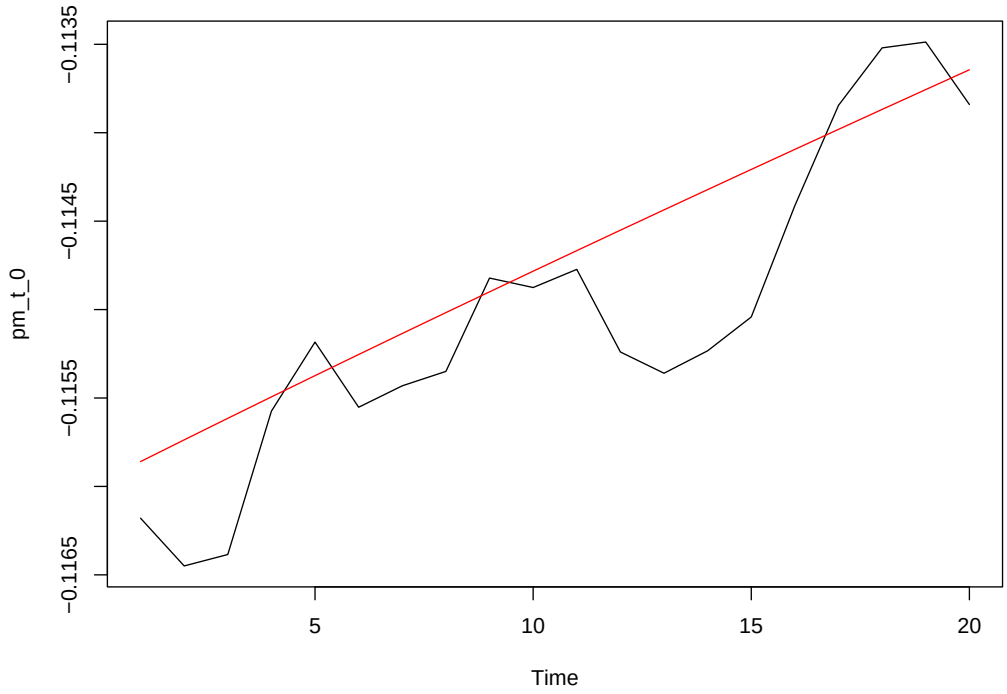


Figure 8.3: Plot returned by the `forecast_ts` function after forecasting 20 instances of a TS. The *pm* variable represents the magnet temperature inside an electric motor. The black line represents the original values of the TS, and the red line represents the forecasting. Only the values of the variables at the initial instant 0 are known as evidence to the DBN.

Smoothing is called in a similar way to forecasting by using the function

```
smooth_ts(dt, fit, obj_vars, ini, len, mode, print_res, plot_res)
```

8.5 Pipeline and main functions

The **dbnR** package is focused on making DBN models readily available for application. As such, the main pipeline from some dataset to a fitted DBN model is kept as simple as possible. A diagram of this pipeline can be seen in Figure 8.4, where with four function calls, we can obtain a fully functional DBN model and perform inference with it.

Apart from the essential functionality, **dbnR** offers other utilities. In Table 8.1, we show all the exported functions of the package. All three structure learning algorithms are encapsulated and parametrized inside the `learn_dbn_struc()` function.

Additionally, all the functions for the addition or deletion of arcs and nodes offered by **bnlearn** also work for 'dbn' objects. To show this compatibility, we use the code below to obtain a random DBN structure and a simulated dataset with the `generate_random_network_exp` function, and then apply some graph modification functions.

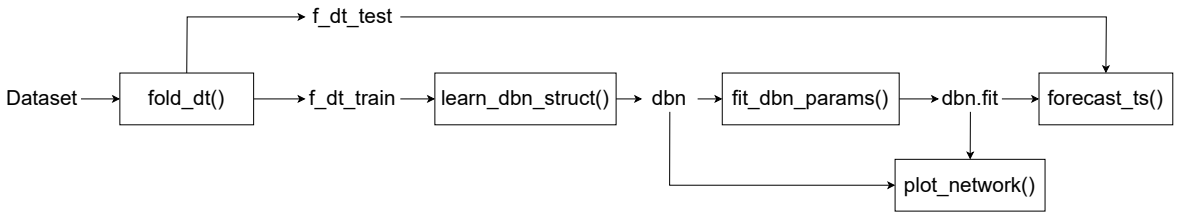


Figure 8.4: Diagram with the main workflow of the **dbnR** package. We start by preparing a dataset; then, we learn the DBN structure, learn its parameters and typically perform forecasting. Optionally, the network structure can also be plotted.

Task	Function	Output
Learning		
Learn DBN structure	`learn_dbn_struc`	'dbn'
Fit DBN parameters	`fit_dbn_params`	'dbn.fit'
Calculate $\boldsymbol{\mu}$ from 'dbn.fit'	`calc_mu`	'numeric'
Calculate $\boldsymbol{\Sigma}$ from 'dbn.fit'	`calc_sigma`	'matrix'
Visualization		
Plot BN or DBN	`plot_network`	HTML
Plot DBN	`plot_dynamic_network`	HTML
Plot BN	`plot_static_network`	HTML
Inference		
Forecast a TS	`forecast_ts`	'list'
Smooth a TS	`smooth_ts`	'list'
Multivariate Gaussian inference	`mvn_inference`	'list'
Inference over 'data.table'	`predict_dt`	'data.table'
Inference over 'data.table'	`predict_bn`	'data.table'
Utilities		
Rename variables into `t_x`	`time_rename`	'data.table'
Fold dataset	`fold_dt`	'data.table'
Fold dataset based on index	`filter_fold_dt`	'data.table'
Dataset folding utility	`filter_same_cycle`	'data.table'
Create a random DBN	`generate_random_network_exp`	'list'
Reduce frequency of TS	`reduce_freq`	'data.table'

Table 8.1: Overview of all the exported functions in the **dbnR** package ordered by the type of task they perform.

```
R> dbn_ex <- dbnR::generate_random_network_exp(n_vars = 3, size = 2,
+                           min_mu = -5, max_mu = 5, min_sd = 0.5,
+                           max_sd = 2, min_coef = -1, max_coef = 1,
+                           seed = 42)
R> names(dbn_ex$net$nodes)

[1] "X0_t_0" "X1_t_0" "X2_t_0" "X0_t_1" "X1_t_1" "X2_t_1"

R> dbn_ex$net$arcs
```

```
     from     to
[1,] "X0_t_1" "X0_t_0"
[2,] "X1_t_1" "X0_t_0"
[3,] "X0_t_1" "X1_t_0"
[4,] "X1_t_1" "X1_t_0"
[5,] "X2_t_1" "X1_t_0"
[6,] "X0_t_1" "X2_t_0"
[7,] "X2_t_1" "X2_t_0"
```

We generated a random DBN with two time slices and three variables per time slice. In total, the network structure has six nodes and seven arcs. If we want to delete the first arc, we can do so with the `drop.arc` function from **bnlearn**, and if we want to delete a node entirely, we can use the `remove.node` function.

```
R> dbn_ex$net <- bnlearn::drop.arc(dbn_ex$net, "X0_t_1", "X0_t_0")
R> dbn_ex$net$arcs

     from     to
[1,] "X1_t_1" "X0_t_0"
[2,] "X0_t_1" "X1_t_0"
[3,] "X1_t_1" "X1_t_0"
[4,] "X2_t_1" "X1_t_0"
[5,] "X0_t_1" "X2_t_0"
[6,] "X2_t_1" "X2_t_0"

R> dbn_ex$net <- bnlearn::remove.node(dbn_ex$net, "X0_t_0")
R> names(dbn_ex$net$nodes)

[1] "X1_t_0" "X2_t_0" "X0_t_1" "X1_t_1" "X2_t_1"
```

Several other auxiliary functions from **bnlearn**, such as obtaining the node ordering of a network with the `node.ordering` function or getting the Markov blanket of a node by calling `mb` can be used in a similar manner.

8.6 Example application: Sample motor dataset

To show a practical full example of using the **dbnR** package, we use a dataset to learn the structure of three different DBNs, fit their parameters and perform forecasting with them. We use the sample dataset included in the package. The data come from the *electric motor temperature* dataset in Kaggle [Kirchgässner et al., 2021], from which we selected a sample of the first 3000 instances intended only for testing purposes of the package utilities. For the complete dataset, we refer the readers to the original source[3].

[3] https://www.kaggle.com/wkirgsn/electric-motor-temperature

```
R> dt <- dbnR::motor
R> summary(dt)

    ambient            coolant              u_d
 Min.   :-0.79598   Min.   :-0.07434   Min.   :-1.6415
 1st Qu.: 0.01516   1st Qu.: 0.05816   1st Qu.: 0.3122
 Median : 0.05754   Median : 0.09546   Median : 0.3137
 Mean   : 0.05927   Mean   : 0.89587   Mean   : 0.3006
 3rd Qu.: 0.10300   3rd Qu.: 2.15739   3rd Qu.: 0.3157
 Max.   : 0.20956   Max.   : 2.27659   Max.   : 2.2359
      u_q             motor_speed            i_d
 Min.   :-1.332770   Min.   :-1.22243   Min.   :-2.4526
 1st Qu.:-1.328685   1st Qu.:-1.22243   1st Qu.: 0.2333
 Median :-1.326736   Median :-1.22243   Median : 1.0291
 Mean   :-0.601467   Mean   :-0.58853   Mean   : 0.4785
 3rd Qu.: 0.008286   3rd Qu.: 0.02407   3rd Qu.: 1.0291
 Max.   : 1.729171   Max.   : 1.87129   Max.   : 1.0292
      i_q                pm             stator_yoke
 Min.   :-2.9470   Min.   :-0.14291   Min.   :-0.0564
 1st Qu.:-0.2524   1st Qu.:-0.11738   1st Qu.: 0.1337
 Median :-0.2457   Median : 0.04524   Median : 0.2650
 Mean   :-0.2941   Mean   : 0.01934   Mean   : 0.5307
 3rd Qu.:-0.2457   3rd Qu.: 0.12920   3rd Qu.: 0.9512
 Max.   : 2.2931   Max.   : 0.25813   Max.   : 1.5442
  stator_tooth      stator_winding
 Min.   :-0.31658   Min.   :-0.53658
 1st Qu.: 0.01911   1st Qu.:-0.22645
 Median : 0.31257   Median : 0.13088
 Mean   : 0.27072   Mean   : 0.08419
 3rd Qu.: 0.46184   3rd Qu.: 0.39245
 Max.   : 0.92338   Max.   : 0.71988
```

The sample dataset consists of 11 continuous variables that correspond to different temperatures, voltages and currents inside an electrical motor. Our aim is to use the data to fit the DBN models and showcase the whole process of training a DBN with some dataset and perform forecasting.

Initially, we split our data into training and test sets, and then use the `fold_dt` function to generate the necessary temporal variables in each row.

```
R> dt_train <- dt[1:2800]
R> dt_test <- dt[2801:3000]
R> size <- 2
```

```
R> f_dt_train <- dbnR::fold_dt(dt_train, size)
R> f_dt_test <- dbnR::fold_dt(dt_test, size)
R> print(names(f_dt_train))

 [1] "ambient_t_0"        "coolant_t_0"        "u_d_t_0"
 [4] "u_q_t_0"            "motor_speed_t_0"    "i_d_t_0"
 [7] "i_q_t_0"            "pm_t_0"             "stator_yoke_t_0"
[10] "stator_tooth_t_0"   "stator_winding_t_0" "ambient_t_1"
[13] "coolant_t_1"        "u_d_t_1"            "u_q_t_1"
[16] "motor_speed_t_1"    "i_d_t_1"            "i_q_t_1"
[19] "pm_t_1"             "stator_yoke_t_1"    "stator_tooth_t_1"
[22] "stator_winding_t_1"
```

For the sake of simplicity, we fix the size of the folding to 2 so that we learn Markovian order 1 DBNs. After splitting and folding, we create the necessary variables for the desired size and obtain a dataset that can be used for learning the structure and the parameters of the DBN models.

```
R> t <- Sys.time()
R> net_dmmhc <- dbnR::learn_dbn_struc(dt_train, size, method = "dmmhc",
+                                     f_dt = f_dt_train)
R> Sys.time() - t

Time difference of 0.348067 secs

R> set.seed(42)
R> t <- Sys.time()
R> net_psoho <- dbnR::learn_dbn_struc(dt_train, size, method = "psoho",
+                                     f_dt = f_dt_train, n_it = 10)

  |======================================================================
   ======================================================================
   ============================================================| 100%

R> Sys.time() - t

Time difference of 8.868454 secs

R> t <- Sys.time()
R> net_nat <- dbnR::learn_dbn_struc(dt_train, size, method = "natPsoho",
+                                   f_dt = f_dt_train, n_it = 10)

  |======================================================================
   ======================================================================
   ============================================================| 100%
```

```
R> Sys.time() - t

Time difference of 3.020932 secs
```

We used the three available structure learning algorithms and printed the execution time spent by each one. For small DBNs, the execution time of the DMMHC algorithm is unrivalled by the particle swarm algorithms, but it scales poorly to a greater number of nodes and higher Markovian orders. Note that the particle swarm algorithms are not deterministic, and we had to set a seed number to obtain reproducible results. They also print a progress bar that shows how much of the process is done in terms of the number of iterations finished from the total number of iterations allowed with the `n_it` parameter.

With the network structures, we can now learn their parameters from the folded dataset with the `fit_dbn_params` function. This returns a 'dbn.fit' object that can be used to perform inference.

```
R> fit_dmmhc <- dbnR::fit_dbn_params(net_dmmhc, f_dt_train)
R> fit_psoho <- dbnR::fit_dbn_params(net_psoho, f_dt_train)
R> fit_nat <- dbnR::fit_dbn_params(net_nat, f_dt_train)
R> fit_dmmhc$pm_t_0

  Parameters of node pm_t_0 (Gaussian distribution)

Conditional density: pm_t_0 | coolant_t_0 + stator_winding_t_0 +
                              motor_speed_t_1 + pm_t_1
Coefficients:
       (Intercept)         coolant_t_0  stator_winding_t_0
      0.0017474119       -0.0014122266        0.0011111532
   motor_speed_t_1              pm_t_1
      0.0005594395        0.9849412445
Standard deviation of the residuals: 0.004961973
```

We can inspect a fitted model by checking specific nodes. In the previous code chunk, we printed the parameters of the `pm_t_0` node. This variable represents the temperature of the permanent magnet in the rotor of the motor and can be used to predict overheating. By checking its parameters, we can see all its parent nodes, as well as the effects that each one has on it. Note that all variables are normalized, so the scales of the parameters are similar and can be compared. We can see that the values of the last instant `pm_t_1` have a parameter of approximately 0.985, which hints at the fact that the previous value in the TS is a very good prediction of the next one and has high correlation. If the variables were not normalized, the parameters would need to account for the difference in magnitude of the variables, and they would be much harder to interpret.

In our DBN models, the DMMHC algorithm allows intra-slice arcs, as shown by the `pm_t_0` variable having as parents other variables in t_0. The PSO algorithms, however, do not allow this kind of arc and only learn inter-slice arcs directed to t_0.

```
R> fit_nat$pm_t_1

  Parameters of node pm_t_1 (Gaussian distribution)

Conditional density: pm_t_1
Coefficients:
(Intercept)
 0.02872801
Standard deviation of the residuals: 0.123039

R> summary(f_dt_train[, pm_t_1])

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.14291 -0.11971  0.06236  0.02873  0.13503  0.25813
```

In the case of variables with no parents, it can be seen that their intercept is equal to the mean of the TS in the original training dataset. This can be seen in the case of the PSO algorithms in each variable outside t_0, given that only variables in t_0 are allowed to have parent nodes in those algorithms.

With the fitted models, we can now perform inference over the test dataset. First, we use the `mvn_inference` function directly to perform a single inference step. We use the values from previous time slices as evidence and perform inference over the variables at t_0.

```
R> ev_vars <- names(f_dt_test)[grepl("t_1$", names(f_dt_test))]
R> ev <- f_dt_test[1, .SD, .SDcols = ev_vars]
R> ev

   ambient_t_1 coolant_t_1   u_d_t_1   u_q_t_1 motor_speed_t_1
1:   0.1377959    2.177947 0.3127252 -1.329747       -1.222431
    i_d_t_1    i_q_t_1     pm_t_1 stator_yoke_t_1 stator_tooth_t_1
1: 1.029135 -0.2457216 -0.1207832        1.471376        0.8375071
   stator_winding_t_1
1:          0.3224269
```

First, we extract the values of all the variables whose name finishes with `"t_1"`, that is, all the variables that are at the t_1 time slice from the first row of the folded test dataset. We can now feed the data to any of the 'dbn.fit' objects that we trained earlier.

```
R> res <- dbnR::mvn_inference(attr(fit_dmmhc, "mu"),
+                             attr(fit_dmmhc, "sigma"), evidence = ev)
R> res

$mu_p
                          [,1]
```

```
coolant_t_0         2.1771681
stator_tooth_t_0    0.8386566
stator_yoke_t_0     1.4720731
stator_winding_t_0  0.3229435
u_d_t_0             0.3156188
pm_t_0             -0.1206166
ambient_t_0         0.1371033
u_q_t_0            -1.3303784
i_d_t_0             1.0286580
motor_speed_t_0    -1.2240255
i_q_t_0            -0.2442530

$sigma_p
                     coolant_t_0 stator_tooth_t_0 stator_yoke_t_0
coolant_t_0         3.730297e-03    -9.341215e-05   -6.949001e-05
stator_tooth_t_0   -9.341215e-05     2.389378e-03    1.943209e-05
stator_yoke_t_0    -6.949001e-05     1.943209e-05   -4.629102e-03
stator_winding_t_0  2.617418e-07    -1.718153e-05   -2.775256e-05
u_d_t_0             6.505213e-18    -1.870926e-17   -6.396793e-18
pm_t_0             -5.267734e-06     1.128278e-07    6.729829e-08
ambient_t_0        -7.032338e-07     1.722317e-07   -4.054521e-05
u_q_t_0            -1.899465e-07     4.068401e-09    2.426676e-09
i_d_t_0             5.745827e-07    -1.480856e-05   -4.137123e-07
motor_speed_t_0    -8.185877e-09     1.694358e-07    4.753232e-09
i_q_t_0            -3.539884e-08     8.669667e-09   -2.040933e-06
                   stator_winding_t_0       u_d_t_0        pm_t_0
coolant_t_0              2.617418e-07 -8.673617e-18 -5.267734e-06
stator_tooth_t_0        -1.718153e-05  4.370690e-18  1.128278e-07
stator_yoke_t_0         -2.775256e-05  1.864828e-17  6.729829e-08
stator_winding_t_0      -2.205694e-03 -1.019150e-17 -2.451233e-06
u_d_t_0                 -4.835542e-17  2.375250e-03 -3.361027e-18
pm_t_0                  -2.451233e-06  6.505213e-19 -4.675682e-04
ambient_t_0             -2.870911e-07 -1.169272e-06 -8.393435e-06
u_q_t_0                 -8.838777e-08  1.821460e-17 -1.685981e-05
i_d_t_0                 -2.332173e-05  1.573537e-05 -2.672546e-08
motor_speed_t_0          2.660363e-07 -1.800034e-07 -1.428645e-07
i_q_t_0                 -1.445137e-08 -2.165141e-05 -4.225023e-07
                     ambient_t_0       u_q_t_0       i_d_t_0
coolant_t_0        -7.032338e-07 -1.899465e-07  5.745827e-07
stator_tooth_t_0    1.722317e-07  4.068400e-09 -1.480856e-05
stator_yoke_t_0    -4.054521e-05  2.426675e-09 -4.137123e-07
```

```
stator_winding_t_0 -2.870911e-07 -8.838777e-08 -2.332173e-05
u_d_t_0            -1.169272e-06 -2.645453e-17  1.573537e-05
pm_t_0             -8.393435e-06 -1.685981e-05 -2.672546e-08
ambient_t_0         2.150180e-04 -3.026546e-07 -1.184961e-08
u_q_t_0            -3.026546e-07 -4.193088e-04 -9.636789e-10
i_d_t_0            -1.184961e-08 -9.636806e-10  1.887835e-02
motor_speed_t_0    -2.434532e-09 -3.560678e-06 -2.159572e-04
i_q_t_0             1.083404e-05 -1.523479e-08 -1.436411e-07
                   motor_speed_t_0       i_q_t_0
coolant_t_0          -8.185875e-09 -3.539884e-08
stator_tooth_t_0      1.694358e-07  8.669667e-09
stator_yoke_t_0       4.753232e-09 -2.040933e-06
stator_winding_t_0    2.660363e-07 -1.445137e-08
u_d_t_0              -1.800034e-07 -2.165141e-05
pm_t_0               -1.428645e-07 -4.225023e-07
ambient_t_0          -2.434532e-09  1.083404e-05
u_q_t_0              -3.560678e-06 -1.523479e-08
i_d_t_0              -2.159572e-04 -1.436411e-07
motor_speed_t_0      -5.066376e-02  1.513799e-09
i_q_t_0               1.513799e-09 -9.822611e-03
```

This returns both the $\boldsymbol{\mu}_{1|2}$ vector and $\boldsymbol{\Sigma}_{1|2}$ matrix calculated in Equation (2.27) and (2.28), respectively. The $\boldsymbol{\mu}_{1|2}$ vector is used as the resulting value from the exact inference, that is, the most likely value for our predicted variables given the provided evidence. The $\boldsymbol{\Sigma}_{1|2}$ matrix is also returned, but it is less interesting in the case of Gaussian DBNs given that it remains constant no matter the evidence we provide, as shown by Equation (2.28), where only the constant values of the covariance matrix are used in its calculation.

The mean vector obtained only corresponds to a single instant prediction. We can automate this process for all the rows in a dataset with the `predict_dt` function in case our objective is to predict only the next time instant.

```
R> res <- dbnR::predict_dt(fit_dmmhc, f_dt_test, obj_nodes = "pm_t_0")
```

```
[1] MAE:
      pm_t_0
0.0005821966
[1] SD:
      pm_t_0
0.0007559006
```

Along with the predictions, the `predict_dt` function prints the average MAE and the standard deviation of the residuals and plots the predictions, as shown in Figure 8.5. Although the inference to horizon 1 obtains a seemingly low MAE and the plot seems to be a good

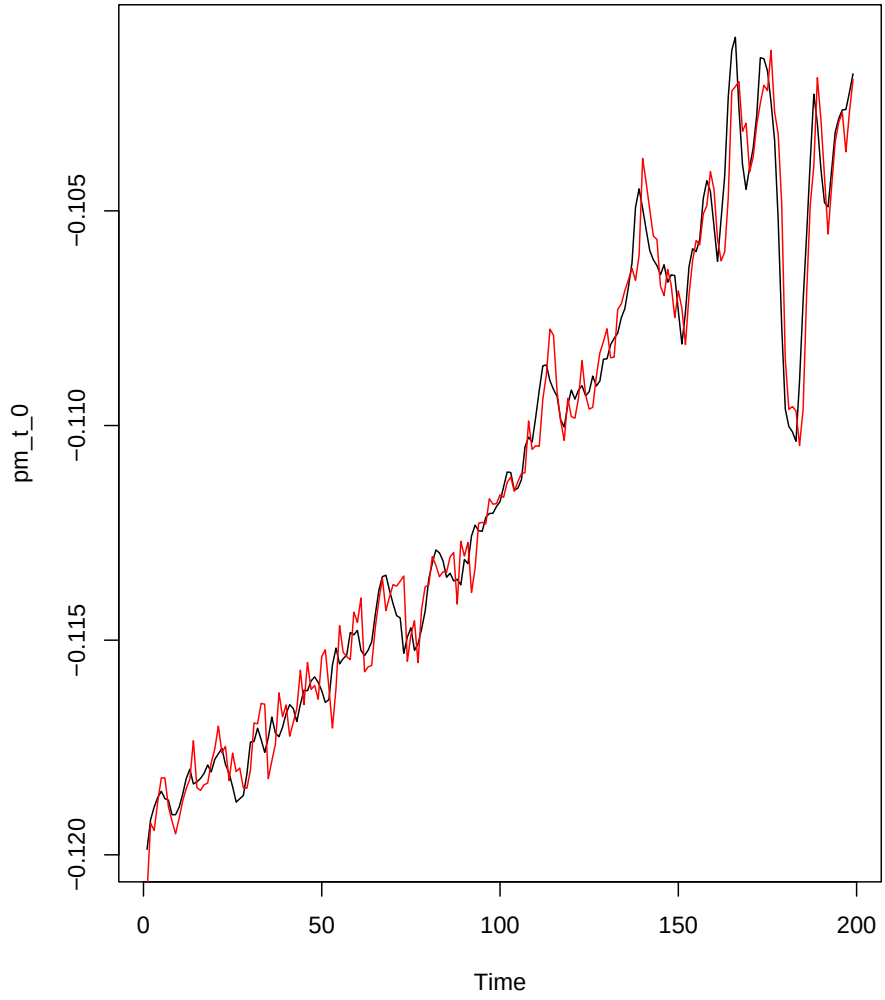


Figure 8.5: Plot of the predictions (red) returned by the `predict_dt` function. All the predictions are performed to horizon 1 using the last instant as evidence. They are deceivingly accurate because the last instant is always used as evidence for the next prediction. If looking closely, it can be seen that large changes in the profile of the curve are not properly predicted by the DBN until one instant later when evidence of these changes is provided to the model.

result, single-step predictions can be misleading. As shown earlier by the parameters, a good prediction of the next instant of a TS is the previous one. Sometimes, a TS model can just be passing forward the values of the variables, incurring good predictions for $T = 1$ but obtaining worse results when forecasting.

For forecasting, we use the `forecast_ts` function, which allows us to forecast up to an arbitrary time horizon. We use the `pm_t_0` variable as our target variable, but more than one variable can be selected simultaneously as target variables.

```
R> res <- dbnR::forecast_ts(f_dt_test, fit_dmmhc, obj_vars = "pm_t_0",
```

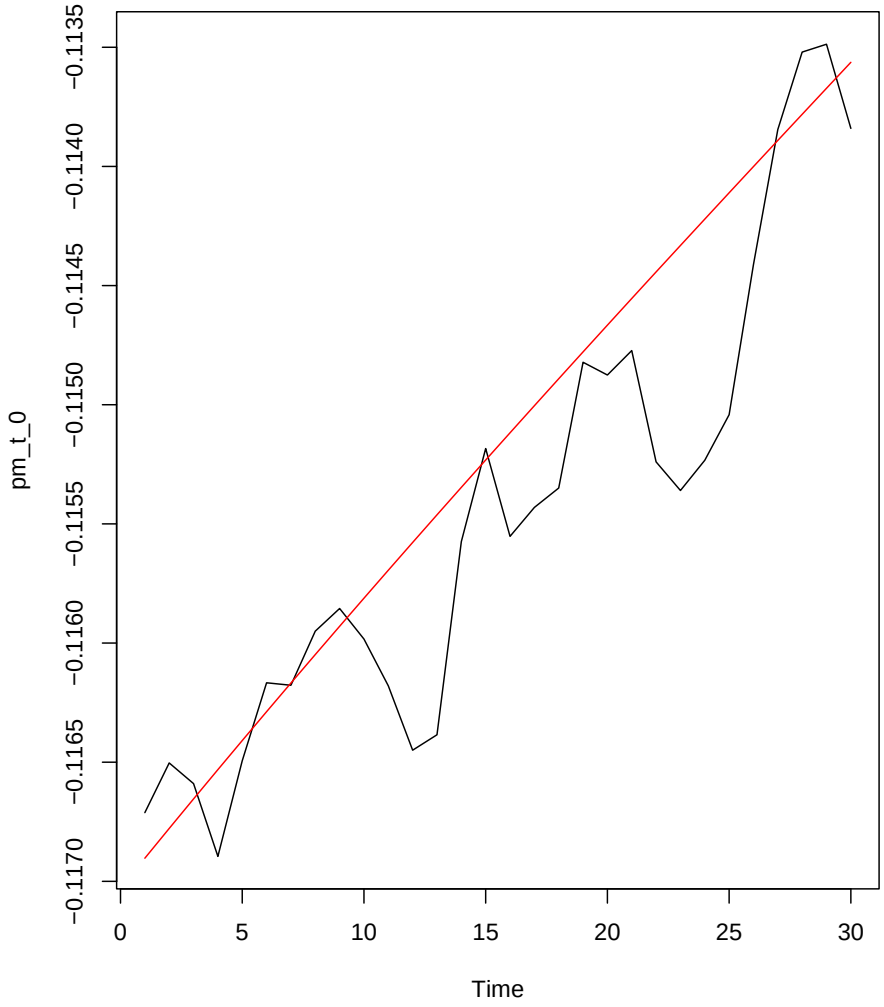


Figure 8.6: Plot obtained from forecasting 30 instances with a DBN model using only the evidence from the initial point at $t = 0$.

```
+                              ini = 40, len = 30, mode = "exact")

Time difference of -0.1097059 secs
[1] The average MAE per execution is:
[1] pm_t_0: 4e-04
```

The obtained forecast shown in Figure 8.6 follows the tendency of the TS, but it is much smoother than the real values. This is because the exact inference in DBN models returns the most likely value in each instant, which is the mean value of the conditional multivariate Gaussian distribution. As such, the variance shown in $\mathbf{\Sigma}_{1|2}$ is expected to take place, but it remains constant throughout the forecast. It is also important to note that the model only

sees the evidence at the first instant of time, which means that any future intervention in the TS will not be seen by the DBN.

If we are applying a DBN model in real time, we do not have the values of the full TS when performing forecasting. In this case, we can provide the model with the evidence of the initial time point, and it will return the forecast without performing MAE calculations.

```
R> res <- dbnR::forecast_ts(f_dt_test[40], fit_dmmhc, obj_vars = "pm_t_0",
+                           ini = 1, len = 5, mode = "exact",
+                           plot_res = FALSE, print_res = FALSE)
R> res$pred$pm_t_0

[1] -0.1169029 -0.1167780 -0.1166541 -0.1165312 -0.1164092
```

We can also use the DBN model as a simulator by providing specific evidence over time during forecasting. Thus, we can test the effects of specific values on some key variables or see how certain interventions affect the system. In our example data, we may want to test how the increasing or decreasing revolutions per minute of the motor represented by the `motor_speed` variable affects our objective temperature. Our first scenario is to fix this value at a very low rate, lower than the real values of the TS. In the second scenario that we propose, we progressively increase the revolutions per minute over time from a very low starting point to see the effects that accelerating a car would have on the permanent magnet temperature.

```
f_dt_i <- f_dt_test[40:70]
summary(f_dt_i$motor_speed_t_0)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -1.222  -1.222  -1.222  -1.222  -1.222  -1.222

f_dt_i[1:5, motor_speed_t_0]

[1] -1.222428 -1.222428 -1.222430 -1.222432 -1.222431
```

We first extract the 30 instances of our previous forecasting to test both interventions because we already know how the model behaves for that period of time. The `motor_speed` variable is fairly constant at -1.22 on this interval. Note that the variables are normalized, and it does not mean negative revolutions per minute. We will modify this subset of data by fixing the values of `motor_speed_t_0` to a lower value of -1.4, and then to a sequence of values increasing from -1.4 to -1.1 to simulate motor acceleration.

```
R> f_dt_i[, motor_speed_t_0 := -1.4]
R> res <- dbnR::forecast_ts(f_dt_i, fit_dmmhc, obj_vars = "pm_t_0",
+                           ini = 1, len = 30, mode = "exact",
+                           prov_ev = "motor_speed_t_0")
```

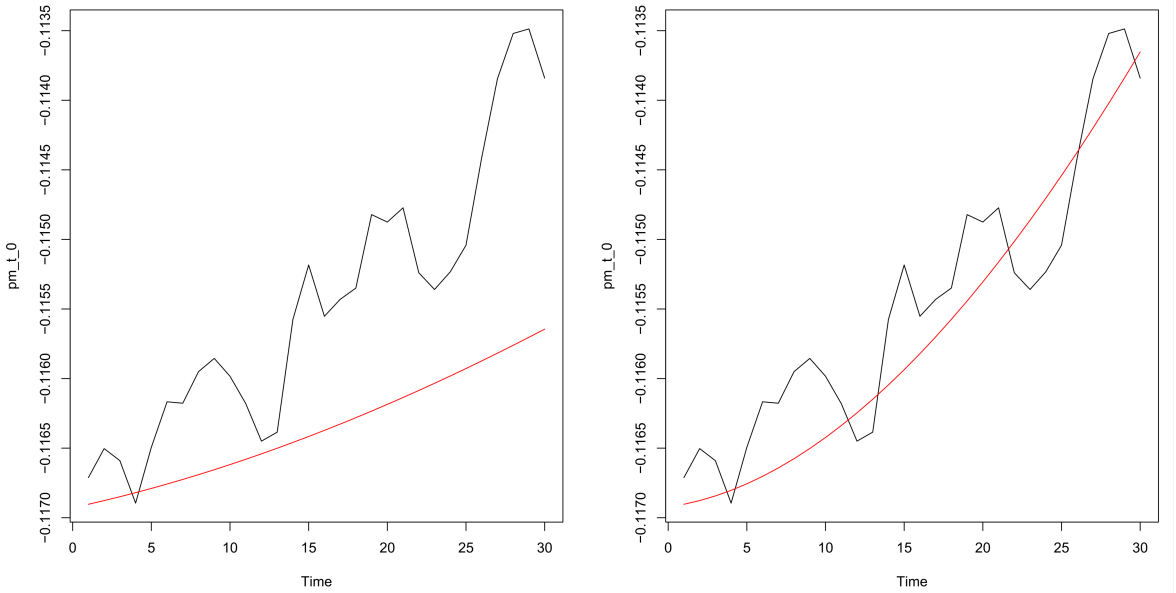


Figure 8.7: A comparison between the plotted forecasts of fixing the `motor_speed` variable to -1.4 (left) and progressively increasing it from -1.4 to -1.1 (right). The real values of the TS are represented by black lines, and the red lines represent the forecasts. The effects of both actions can be clearly seen in the profile of the predictions.

```
Time difference of -0.2303832 secs
[1] The average MAE per execution is:
[1] pm_t_0: 9e-04
```

```
R> f_dt_i[, motor_speed_t_0 := seq(from = -1.4, to = -1.1, by = 0.3 / 30)]
R> res <- dbnR::forecast_ts(f_dt_i, fit_dmmhc, obj_vars = "pm_t_0",
+                           ini = 1, len = 30, mode = "exact",
+                           prov_ev = "motor_speed_t_0")
```

```
Time difference of -0.232743 secs
[1] The average MAE per execution is:
[1] pm_t_0: 4e-04
```

The plots resulting from both scenarios can be seen in Figure 8.7. If we set a low `motor_speed`, the temperature of the magnet increases much more slowly than in the real case. However, we can see the effect that accelerating the motor would have on the temperature, which ramps up when we start increasing the revolutions per minute. This behaviour as simulators of the real world is a powerful tool that can turn DBNs into generative models that offer insight into industrial processes before making an intervention in a system.

8.7 Conclusions and future work

In this chapter we have presented the **dbnR** package for Gaussian DBN learning, inference and visualization. The package covers the whole process from learning both the structure and parameters of a DBN model to performing inference and forecasting with it. It also extends the functionality of the most popular BN package in R, **bnlearn**, to the case of DBNs. The intermediate steps of the learning and inference process, including the structure learning algorithms, are presented and discussed regarding both their definitions and their implementations. The **dbnR** package has managed to achieve a level of notoriety and diffusion, with over 18,000 downloads worldwide.

For future work, we would like to add an option to show an automatically generated user interface with **shiny** [Chang et al., 2021]. This would give the simulator of DBN models more capacity to interact with the user, as well as generating tools readily available for a data scientist to present prototypes to an expert on a specific problem. This can be especially useful in the case of BNs and DBNs due to their capacity to incorporate expert knowledge into the model itself and to show the results of inference clearly and directly to the model end users.

Part IV

CONCLUSIONS

Chapter 9

Conclusions and future work

In this chapter, we review the most important contributions of this work and discuss possible future research options. We also include the publications and submissions produced during this research.

Chapter outline

In Section 9.1, we review the main contributions of this work. Section 9.2 provides a list of works published and submitted during this research. Finally, Section 9.3 revolves around possible future work.

9.1 Summary of contributions

The third part of this document compiles all the contributions done over the years.

- Chapter 4 proposes the use of high order DBNs in the industrial setting of furnaces suffering from the fouling phenomenon. This was a new approach in the field of fouling prediction, and varying the Markovian order of the network provided and increase of the accuracy of the DBNs when modelling the tendency of the TS up to a certain order. The experiments on a real world dataset showed that we can accurately make long term predictions of the profile of the temperature curves with high order DBNs.
- Chapter 5 offers a new structure learning algorithm tailored specifically for high order DBNs. After applying DBNs to the fouling problem and realizing that the DMMHC algorithm scales poorly with the Markovian order, we proposed a new algorithm based on PSO to be able to explore the space of possible networks more efficiently. Our order invariant encoding proved to be more efficient in the experimentation than the baseline DMMHC algorithm and than another PSO structure learning algorithm from the literature.
- Chapter 6 presents a hybrid between DBNs and model trees that is capable of performing piecewise regression. This in turn allows us to overcome the linearity constraint

of Gaussian BNs, and we can use this hybrid approach to model nonlinear problems. The results on a synthetic problem and two real world problems showed the accuracy improvement of our proposed hybrid approach in comparison with the baseline DBN model, and it behaves well against other state-of-the-art forecasting models like LSTMs and HFCMs.

- Chapter 7 proposes the coupling of Gaussian DBNs and neural networks to solve classification problems over time. In particular, we addressed the problem of, given a patient infected with the COVID-19, deciding whether this patient is going to reach a critical state or not in the next 40 hours. This coupling is shown to improve the performance of baseline classification on the experimentation.

- Chapter 8 discusses our developed open source R package, **dbnR**. This package encapsulates all the process of learning, visualizing and performing inference with Gaussian DBN models in order to make their application to new problems as straightforward as possible. Great portions of our contributions are contained in this package to be accessible to the broadest number of users as possible. The package managed to get a level of attention in the DBN ambit, with over 18,000 downloads worldwide and having been used in several scientific publications.

9.2 List of publications

Peer-reviewed JCR journals

- D. Quesada, G. Valverde, P. Larrañaga, and C. Bielza. Long-term forecasting of multivariate time series in industrial furnaces with dynamic Gaussian Bayesian networks. *Engineering Applications of Artificial Intelligence*, 103:104301, 2021.

- D. Quesada, C. Bielza, P. Fontan, and P. Larrañaga. Piecewise forecasting of nonlinear time series with model tree dynamic Bayesian networks. *International Journal of Intelligent Systems*, 37(11):9108–9137, 2022.

- D. Quesada, P. Larrañaga, and C. Bielza. Classifying the evolution of COVID-19 severity on patients with combined dynamic Bayesian networks and neural networks. *Submitted*, 2023.

- D. Quesada, P. Larrañaga, and C. Bielza. dbnR: Gaussian dynamic bayesian network learning and inference in R. *Submitted*, 2023.

Peer-reviewed conferences

- D. Quesada, C. Bielza, and P. Larrañaga. Structure learning of high-order dynamic Bayesian networks via particle swarm optimization with order invariant encoding. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 158–171. Springer, 2021.

Technical reports

- D. Quesada, P. Larrañaga, C. Bielza. Forecasting symptom severity of COVID-19 patients with dynamic Bayesian network hybrid classifiers. TR:UPM-ETSIINF/DIA/2022-2: Universidad Politécnica de Madrid, 2022.

9.3 Future work

In this section we propose possible future research lines for some of the subjects of this dissertation.

- Further exploration of the relationship between the Markovian order and the autorregresive order from Chapter 4 could result in an adaptation of automatic autoregression estimation like the Box-Jenkins model to the case of DBNs.
- The use of PSO in Chapter 5 proved to be intuitive to implement, but there are many other evolutionary algorithms available in the literature. It could be interesting to evaluate the proposed encoding in the different environments provided by these other algorithms.
- In Chapter 6 we presented the most basic version of the mtDBN framework, where the DBNs were embedded in the leaves of a CART. This model can be further explored by defining different partition methods, as in the case of clustering, or by defining context specific metrics for splitting criteria of populations, like mutual information.
- At the moment, the only available parameter learning method inside **dbnR** is MLE. It could be interesting to add a Bayesian estimation method that could provide different results in exchange of a longer execution time.
- To make the use of DBNs readily available to a greater number of data scientists, **dbnR** could be adapted into a graphical interface setting, where there would be no need to have any knowledge of R coding.

Part V

APPENDICES

Appendix A

ODE system for the fouling phenomenon

ODE system definition

The main idea of our simulation is to generate an environment where we have a fluid flowing through a tube and a heat source heating the walls of said tube. Meanwhile, as this fluid is heated, it precipitates materials that create an insulating layer inside the tube over time. This is a process called fouling [Quesada et al., 2021b].

Our aim is to define a simplified version of this phenomenon, one that reflects the non-linear relationships of the variables and respects the underlying physical process. Our first simplification is that the simulation does not have a spatial component, so heat will transfer from the heat source to the tube walls by radiation from a single point to another and from the tube walls to the fluid by convection. This generates a heating component that increases the temperature of the fluid. As the fouling process occurs, the thermal conductivity of the system decreases, and the heat transferred from the tube walls to the fluid is reduced. To dissipate heat from the system, we have a flow component that renews fresh cooler fluid at each instant. All fluid volumes are heated as a whole and move in and out of the system as a singular unit. Initially, the fluid heats easily, but as the fouling layer grows, it becomes increasingly harder to keep the fluid temperature high.

The first component that we define is the growth of the insulating layer over time:

$$\frac{\partial S_c}{\partial t} = A_1 k_1 C_a, \tag{A.1}$$

where S_c is the thickness of the insulating layer, t is the time, $A_1 > 0$ is a control constant, k_1 is the reaction speed at which particles prone to fouling precipitate and C_a is the concentration of particles prone to fouling in the fluid. Equation (A.1) controls the rate at which the fouling layer grows and is dependent on the value of C_a, which is a property of the fluid, and on k_1. The value of k_1 is defined by:

$$k_1 = A_2 e^{\frac{-A_3}{RT_1}}, \tag{A.2}$$

where $A_2 > 0$ is a constant pre-exponential factor, $A_3 > 0$ is an activation energy constant, R is the ideal gas constant and T_1 is the temperature of the fluid. A higher fluid temperature will accelerate the growth of the insulating layer.

The second component is the evolution of the concentration of particles prone to fouling C_a:

$$\frac{\partial C_a}{\partial t} = -A_4 k_2 C_a, \tag{A.3}$$

$$k_2 = A_2' e^{\frac{-A_3'}{RT_1}} \tag{A.4}$$

The process is very similar to the previous one, but in Equation (A.3), the concentration diminishes each instant. In this case, a high T_1 increases the consumption rate of C_a, but it also affects the growth rate of S_c.

Once we have modelled the fouling layer and the concentration of particles that generate it, we can define the evolution of the fluid temperature T_1 and the flow component:

$$\rho_1 C_{p1}(\frac{\partial T_1}{\partial t} - A_5 Q_{in} \Delta T) = f_1(T_1, T_2) \tag{A.5}$$

$$f_1(T_1, T_2) = \frac{A_6}{S_c}(T_2 - T_1) \tag{A.6}$$

$$Q_{in} = vol \frac{\pi (2r)^2}{4} \tag{A.7}$$

In Equations (A.5) and (A.6), ρ_1 is the density of the fluid, C_{p1} is the thermal capacity of the fluid, Q_{in} represents the flow of new fluid inside the system each instant, $\Delta T = T_{in} - T_1$ is the difference of temperature between the fresh fluid entering the system and the fluid currently in the system, T_2 is the temperature of the tube wall and A_5 and A_6 are control constants. In essence, the convective component of the equations will transfer heat from the tube wall to the fluid, and as S_c from Equation (A.1) increases, this convective component will degrade. The flow component Q_{in} in the system adds the effect of cold fluid entering the system each instant through the area of the tube.

In Equation (A.7), vol is the volume of fluid going through the system each instant, and r is the radius of the tube. The effect of this component is written in Equation (A.5) as the product $A_5 Q_{in} \Delta T$ of the flow rate and the difference in the fluid temperature ΔT. The cold fluid entering the system translates into a loss of temperature in the fluid from the previous instant. This is of vital importance because it adds a mechanism to reduce the temperature of the system. Without it, the simulation will only increase its temperature monotonically. Now, with both S_c and the new flow, there can be situations where the heat transfer between T_2 and T_1 is so low that the fluid loses temperature over time due to the effect of the colder

flow. It also adds the volume inside the tube as a variable that can be manipulated by an agent to perform interventions in the system.

The last equations define the temperature of the tube walls T_2 and the temperature of the furnace T_3:

$$\rho_2 C_{p2}(\frac{\partial T_2}{\partial t}) = f_2(T_1, T_2, T_3) \tag{A.8}$$

$$f_2(T_1, T_2, T_3) = \frac{A_6}{S_c}(T_1 - T_2) + A_7(T_3^4 - T_2^4) \tag{A.9}$$

$$T_3 = T_{min} + \frac{1}{1 + e^{-m_c}}(T_{max} - T_{min}) \tag{A.10}$$

In Equations (A.8) and (A.9), T_2 is defined by its interactions with T_1 and T_3, the latter being the main factor that defines the value of T_2 because T_3 represents the temperature inside the furnace. In the previous equations, ρ_2 is the density of the tube wall alloy material, C_{p2} is its thermal capacity and A_7 is a control constant. The tube wall temperature is raised by radiation from T_3 and is decreased by induction due to the colder fluid inside, but this effect is more insignificant than the radiation.

In Equation (A.10), we define how T_3 is calculated. The furnace temperature changes inside a range defined by a minimum temperature T_{min} and a maximum T_{max} depending on the amount of fuel m_c administered to the furnace heaters. This simulates the behaviour of a valve with a sigmoid function, where an operator can modify the heat inside the furnace by varying m_c.

The variables that can be manipulated to generate traces from the same process are the fluid properties such as the initial values of S_c and C_a, the values of ρ_2 and C_{p2}, the furnace state variables such as m_c and the volume vol. These can be modified to obtain different behaviours in the traces generated, but the other parameters remain constant. The idea is to circulate different types of fluids through the same furnace and increase or decrease the temperatures inside the furnace through interventions on m_c and vol.

With this ODE system, we can generate a synthetic dataset with an arbitrary number of traces from the fouling process and use it to train our models. The code of the simulation and the functions to generate the seeded datasets are readily available in a public repository online[1]. An example of the kind of traces it generates is shown in Fig. A.1.

[1] https://github.com/dkesada/mtDBN

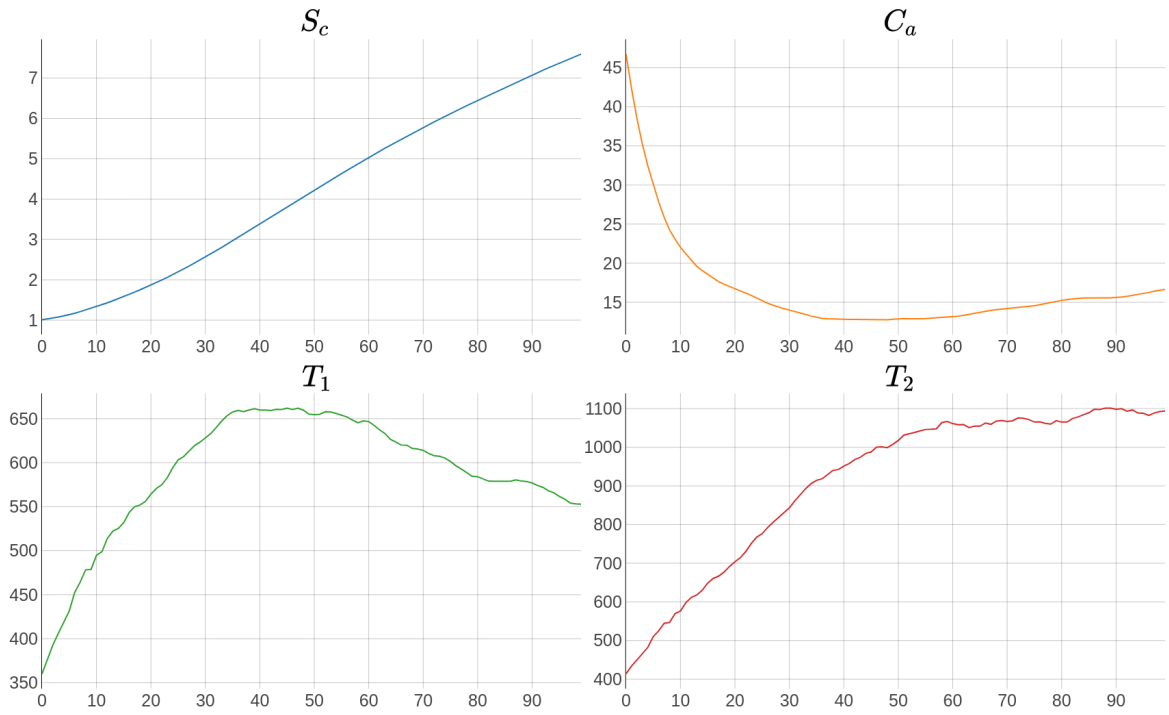


Figure A.1: Example of a 100 time instant trace created by the simulation. Several time series with data and some noise of each variable are generated.

Bibliography

S. P. Adam, S.-A. N. Alexandropoulos, P. M. Pardalos, and M. N. Vrahatis. No free lunch theorem: A review. *Approximation and Optimization: Algorithms, Complexity and Applications*, pages 57–82, 2019.

H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected Papers of Hirotugu Akaike*, pages 199–213. Springer, 1998.

J. Allaire and F. Chollet. *keras: R Interface to 'Keras'*, 2022. URL https://CRAN.R-project.org/package=keras. R package version 2.9.0.

B. Almende, B. Thieurmel, and T. Robert. *visNetwork: Network visualization using vis.js library*, 2019. URL https://CRAN.R-project.org/package=visNetwork. R package version 2.0.9.

S. Aminikhanghahi and D. J. Cook. A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51(2):339–367, 2017.

E. Andreou, E. Ghysels, and A. Kourtellos. Regression models with mixed sampling frequencies. *Journal of Econometrics*, 158(2):246–261, 2010.

A. Ankan and A. Panda. pgmpy: Probabilistic graphical models using Python. In *Proceedings of the 14th Python in Science Conference*, volume 10. Citeseer, 2015.

J. C. Arévalo-Lorido, J. Carretero-Gómez, J. M. Casas-Rojo, J. M. Antón-Santos, J. A. Melero-Bermejo, M. D. López-Carmona, L. C. Palacios, J. Sanz-Cánovas, P. M. Pesqueira-Fontán, A. A. de la Peña-Fernández, et al. The importance of association of comorbidities on COVID-19 outcomes: A machine learning approach. *Current Medical Research and Opinion*, 38(4):501–510, 2022.

M. Ashrafi. Forward and backward risk assessment throughout a system life cycle using dynamic Bayesian networks: A case in a petroleum refinery. *Quality and Reliability Engineering International*, 37(1):309–334, 2021.

D. Atienza, C. Bielza, and P. Larrañaga. Semiparametric Bayesian networks. *Information Sciences*, 584:564–582, 2022.

R. Aznar-Gimeno, L. M. Esteban, G. Labata-Lezaun, R. del Hoyo-Alonso, D. Abadia-Gallego, J. R. Paño-Pardo, M. J. Esquillor-Rodrigo, Á. Lanas, and M. T. Serrano. A clinical decision web to predict ICU admission or death for patients hospitalised with COVID-19 using machine learning algorithms. *International Journal of Environmental Research and Public Health*, 18(16):8677, 2021.

J. Berenguer, A. M. Borobia, P. Ryan, J. Rodríguez-Baño, J. M. Bellón, I. Jarrín, J. Carratalà, J. Pachón, A. J. Carcas, M. Yllescas, and J. R. Arribas. Development and validation of a prediction model for 30-day mortality in hospitalised patients with COVID-19: The COVID-19 SEIMC score. *Thorax*, 76(9):920–929, 2021.

D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Workshop on Knowledge Discovery in Databases*, volume 10, pages 359–370, 1994.

D. Bertsimas, G. Lukin, L. Mingardi, O. Nohadani, A. Orfanoudaki, B. Stellato, H. Wiberg, S. Gonzalez-Garcia, C. L. Parra-Calderón, K. Robinson, et al. COVID-19 mortality risk assessment: An international multi-center study. *PLOS ONE*, 15(12):e0243262, 2020.

C. Bielza and P. Larrañaga. Bayesian networks in neuroscience: A survey. *Frontiers in Computational Neuroscience*, 8:131, 2014. doi: 10.3389/fncom.2014.00131.

J. A. Bilmes. Dynamic Bayesian multinets. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 38–45, 2000.

R. Blanco, I. Inza, and P. Larrañaga. Learning Bayesian networks in the space of structures by estimation of distribution algorithms. *International Journal of Intelligent Systems*, 18 (2):205–220, 2003.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

P. J. Brockwell, R. A. Davis, and M. V. Calder. *Introduction to Time Series and Forecasting*. Springer, 2002.

B. Cai, X. Shao, Y. Liu, X. Kong, H. Wang, H. Xu, and W. Ge. Remaining useful life estimation of structure systems under the influence of multiple causes: Subsea pipelines as a case study. *IEEE Transactions on Industrial Electronics*, 67(7):5737–5747, 2019.

W. Chang. *R6: Encapsulated classes with reference semantics*, 2021. URL https://CRAN.R-project.org/package=R6. R package version 2.5.1.

W. Chang, J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. *shiny: Web application framework for R*, 2021. URL https://CRAN.R-project.org/package=shiny. R package version 1.7.1.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, M. Li, J. Xie, M. Lin, Y. Geng, Y. Li, and J. Yuan. *xgboost: Extreme gradient boosting*, 2022. URL https://CRAN.R-project.org/package=xgboost. R package version 1.6.0.1.

C. Cheng, A. Sa-Ngasoongsong, O. Beyca, T. Le, H. Yang, Z. Kong, and S. T. Bukkapatnam. Time series forecasting for nonlinear and non-stationary processes: A review and comparative study. *IIE Transactions*, 47(10):1053–1071, 2015.

M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

A. D. Chouakria and P. N. Nagabhushan. Adaptive dissimilarity index for measuring time series proximity. *Advances in Data Analysis and Classification*, 1(1):5–21, 2007.

R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. STL: A seasonal-trend decomposition. *Journal of Official Statistics*, 6(1):3–73, 1990.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

E. Cornelius, O. Akman, and D. Hrozencik. COVID-19 mortality prediction using machine learning-integrated random forest algorithm under varying patient frailty. *Mathematics*, 9 (17):2043, 2021.

P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial intelligence*, 60(1):141–153, 1993.

J. Dai, J. Ren, W. Du, V. Shikhin, and J. Ma. An improved evolutionary approach-based hybrid algorithm for Bayesian network structure learning in dynamic constrained search space. *Neural Computing and Applications*, 32(5):1413–1434, 2020.

D. Dash and M. J. Druzdzel. A hybrid anytime algorithm for the construction of causal models from sparse data. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 142–149, 1999.

E. Davoudi and B. Vaferi. Applying artificial neural networks for systematic estimation of degree of fouling in heat exchangers. *Chemical Engineering Research and Design*, 130: 138–153, 2018.

L. M. De Campos, J. M. Fernández-Luna, and J. M. Puerta. An iterated local search algorithm for learning Bayesian networks with restarts based on conditional independence tests. *International Journal of Intelligent Systems*, 18(2):221–235, 2003.

J.-B. Denis and M. Scutari. *rbmn: Handling linear Gaussian Bayesian networks*, 2021. URL https://CRAN.R-project.org/package=rbmn. R package version 0.9-4.

S. Dhamodharavadhani, R. Rathipriya, and J. M. Chatterjee. COVID-19 mortality rate prediction for India using statistical neural network models. *Frontiers in Public Health*, 8: 441, 2020.

A. L. Diaby, S. J. Miklavcic, and J. Addai-Mensah. Optimization of scheduled cleaning of fouled heat exchanger network under ageing using genetic algorithm. *Chemical Engineering Research and Design*, 113:223–240, 2016.

E. Diaz-Bejarano, F. Coletti, and S. Macchietto. Modeling and prediction of shell-side fouling in shell-and-tube heat exchangers. *Heat Transfer Engineering*, 40(11):845–861, 2019.

F. J. Diez. Parameter adjustment in Bayes networks. The generalized noisy OR–gate. In *Uncertainty in Artificial Intelligence*, pages 99–105. Elsevier, 1993.

M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2021. URL https://CRAN.R-project.org/package=data.table. R package version 1.14.2.

T. Du, S. Zhang, and Z. Wang. Efficient learning Bayesian networks using PSO. In *International Conference on Computational and Information Science*, pages 151–156. Springer, 2005.

T. Duan. Auto regressive dynamic Bayesian network and its application in stock market inference. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 419–428. Springer, 2016.

G. Ducamp, C. Gonzales, and P.-H. Wuillemin. aGrUM/pyAgrum: A toolbox to build models and algorithms for probabilistic graphical models in Python. In *10th International Conference on Probabilistic Graphical Models*, pages 609–612, 2020.

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: 10.18637/jss.v040.i08.

D. Edwards. *Introduction to Graphical Modelling*. Springer Science & Business Media, 2012.

L. Ehwerhemuepha, S. Danioko, S. Verma, R. Marano, W. Feaster, S. Taraman, T. Moreno, J. Zheng, E. Yaghmaei, and A. Chang. A super learner ensemble of 14 statistical learning models for predicting COVID-19 severity among patients with cardiovascular conditions. *Intelligence-Based Medicine*, 5:100030, 2021.

G. Fan, C. Tu, F. Zhou, Z. Liu, Y. Wang, B. Song, X. Gu, Y. Wang, Y. Wei, H. Li, et al. Comparison of severity scores for COVID-19 patients with pneumonia: A retrospective study. *European Respiratory Journal*, 56(3), 2020.

G. Feng, L. Zhang, J. Yang, and W. Lu. Long-term prediction of time series using fuzzy cognitive maps. *Engineering Applications of Artificial Intelligence*, 102:104274, 2021.

N. E. Fenton, S. McLachlan, P. Lucas, K. Dube, G. A. Hitman, M. Osman, E. Kyrimi, and M. Neil. A Bayesian network model for personalised COVID-19 risk assessment and contact tracing. *MedRxiv*, pages 2020–07, 2021.

R. Fernandes. *dbnlearn: Dynamic Bayesian network structure learning, parameter learning and forecasting*, 2020. URL https://CRAN.R-project.org/package=dbnlearn. R package version 0.1.0.

N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In *Learning in Graphical Models*, pages 421–459. Springer, 1998.

J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37, 2014.

A. Gatt and E. Krahmer. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61:65–170, 2018.

D. Geiger and D. Heckerman. Learning Gaussian networks. In *Uncertainty in Artificial Intelligence Proceedings 1994*, pages 235–243. Elsevier, 1994.

D. Geiger, T. Verma, and J. Pearl. Identifying independence in Bayesian networks. *Networks*, 20(5):507–534, 1990.

S. Gheisari and M. R. Meybodi. BNC-PSO: Structure learning of Bayesian networks by particle swarm optimization. *Information Sciences*, 348:272–289, 2016.

T. Giorgino. Computing and visualizing dynamic time warping alignments in R: The dtw package. *Journal of Statistical Software*, 31:1–24, 2009.

M. Grzegorczyk and D. Husmeier. Non-homogeneous dynamic Bayesian networks for continuous data. *Machine Learning*, 83:355–419, 2011.

A. Haque, M. Rao, and M. A. J. Qamar. Inter-factor determinants of return reversal effect with dynamic Bayesian network analysis: Empirical evidence from Pakistan. *The Journal of Asian Finance, Economics and Business*, 9(3):203–215, 2022.

S. Højsgaard. Graphical independence networks with the gRain package for R. *Journal of Statistical Software*, 46(10):1–26, 2012. doi: 10.18637/jss.v046.i10.

M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga. A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys*, 51(6):1–36, 2019.

R. J. Hyndman, Y. Khandakar, et al. *Automatic Time Series for Forecasting: The Forecast Package for R.* Number 6/07. Monash University, Department of Econometrics and Business Statistics, 2007.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

E. J. Keogh and M. J. Pazzani. Learning the structure of augmented Bayesian classifiers. *International Journal on Artificial Intelligence Tools*, 11(04):587–601, 2002.

N. Kianfar, M. S. Mesgari, A. Mollalo, and M. Kaveh. Spatio-temporal modeling of COVID-19 prevalence and mortality using artificial neural network algorithms. *Spatial and Spatio-temporal Epidemiology*, 40:100471, 2022.

B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

W. Kirchgässner, O. Wallscheid, and J. Böcker. Estimating electric motor temperatures with deep residual machine learning. *IEEE Transactions on Power Electronics*, 36(7):7480–7488, 2021.

N. K. Kitson, A. C. Constantinou, Z. Guo, Y. Liu, and K. Chobtham. A survey of Bayesian network structure learning. *Artificial Intelligence Review*, pages 1–94, 2023.

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

K. B. Korb and A. E. Nicholson. *Bayesian Artificial Intelligence*. CRC press, 2010.

B. Kosko. Fuzzy cognitive maps. *International Journal of Man-Machine Studies*, 24(1):65–75, 1986.

S. Kotz, N. Balakrishnan, and N. L. Johnson. *Continuous Multivariate Distributions, Models and Applications*. John Wiley & Sons, 2004.

S. Lalot, O. Palsson, G. Jonsson, and B. Desmet. Comparison of neural networks and Kalman filters performances for fouling detection in a heat exchanger. *International Journal of Heat Exchangers*, 8(1):151, 2007.

H. Langseth, T. D. Nielsen, R. Rumı, and A. Salmerón. Mixtures of truncated basis functions. *International Journal of Approximate Reasoning*, 53(2):212–227, 2012.

P. Larranaga, M. Poza, Y. Yurramendi, R. H. Murga, and C. M. H. Kuijpers. Structure learning of Bayesian networks by genetic algorithms: A performance analysis of control parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(9):912–926, 1996.

P. Larrañaga, H. Karshenas, C. Bielza, and R. Santana. A review on evolutionary algorithms in Bayesian network learning and inference tasks. *Information Sciences*, 233:109–125, 2013.

D. R. Larsen and P. L. Speckman. Multivariate regression trees for analysis of abundance data. *Biometrics*, 60(2):543–549, 2004.

S. L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, pages 31–57, 1989.

U. Lerner, E. Segal, and D. Koller. Exact inference in networks with discrete children of continuous parents. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 319–328, 2001.

C. Li, S. Mahadevan, Y. Ling, S. Choze, and L. Wang. Dynamic Bayesian network for aircraft wing health monitoring digital twin. *American Institute of Aeronautics and Astronautics Journal*, 55(3):930–941, 2017.

Z. Li, P. Li, A. Krishnan, and J. Liu. Large-scale dynamic gene regulatory network inference combining differential equation models with local dynamic Bayesian network analysis. *Bioinformatics*, 27(19):2686–2691, 2011.

H. Liang, U. Ganeshbabu, and T. Thorne. A dynamic Bayesian network approach for analysing topic-sentiment evolution. *IEEE Access*, 8:54164–54174, 2020.

X. Liu and X. Liu. Structure learning of Bayesian networks by continuous particle swarm optimization algorithms. *Journal of Statistical Computation and Simulation*, 88(8):1528–1556, 2018.

X. Liu, J. Lu, Z. Cheng, and X. Ma. A dynamic Bayesian network-based real-time crash prediction model for urban elevated expressway. *Journal of Advanced Transportation*, 2021: 1–12, 2021.

Y. Ma, L. Wang, J. Zhang, Y. Xiang, and Y. Liu. Bridge remaining strength prediction integrated with Bayesian network and in situ load testing. *Journal of Bridge Engineering*, 19(10), 2014.

D. Margaritis. *Learning Bayesian Network Model Structure from Data*. PhD thesis, Carnegie-Mellon University, School of Computer Science, Pittsburgh, PA, 2003.

K. Masmoudi and A. Masmoudi. A new class of continuous Bayesian networks. *International Journal of Approximate Reasoning*, 109:125–138, 2019.

J. A. Mauricio. Análisis de series temporales. *Universidad Complutence de Madrid*, 2007.

D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. *e1071: Misc functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2022. URL https://CRAN.R-project.org/package=e1071. R package version 1.7-12.

B. Mihaljević, C. Bielza, and P. Larrañaga. bnclassify: Learning Bayesian network classifiers. *The R Journal*, 10(2):455–468, 2018.

A. Momeni-Boroujeni, R. Mendoza, I. J. Stopard, B. Lambert, and A. Zuretti. A dynamic Bayesian model for identifying high-mortality risk in hospitalized COVID-19 patients. *Infectious Disease Reports*, 13(1):239–250, 2021.

P. Montero, J. A. Vilar, et al. TSclust: An R package for time series clustering. *Journal of Statistical Software*, 62(1):1–43, 2014.

S. Moral, R. Rumí, and A. Salmerón. Mixtures of truncated exponentials in hybrid Bayesian networks. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 156–167. Springer, 2001.

S. Moritz and T. Bartz-Beielstein. imputeTS: Time series missing value imputation in R. *The R Journal*, 9(1):207–218, 2017.

S. Moritz, A. Sarda, T. Bartz-Beielstein, M. Zaefferer, and J. Stork. Comparison of different methods for univariate time series imputation in R. *arXiv preprint arXiv:1510.03924*, 2015.

K. Mullen, D. Ardia, D. Gil, D. Windover, and J. Cline. DEoptim: An R package for global optimization by differential evolution. *Journal of Statistical Software*, 40(6):1–26, 2011.

K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division, 2002.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

M. Naili, M. Bourahla, M. Naili, and A. Tari. Stability-based dynamic Bayesian network method for dynamic data mining. *Engineering Applications of Artificial Intelligence*, 77: 283–310, 2019.

O. Orang, P. C. d. L. e Silva, and F. G. Guimarães. Time series forecasting using fuzzy cognitive maps: A survey. *arXiv preprint arXiv:2201.02297*, 2022.

R. Pamfil, N. Sriwattanaworachai, S. Desai, P. Pilgerstorfer, K. Georgatzis, P. Beaumont, and B. Aragam. Dynotears: Structure learning from time-series data. In *International Conference on Artificial Intelligence and Statistics*, pages 1595–1605. PMLR, 2020.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.

D. Peña, G. C. Tiao, and R. S. Tsay. *A Course in Time Series Analysis*. John Wiley & Sons, 2011.

I. Pérez-Bernabé, A. D. Maldonado, T. D. Nielsen, and A. Salmerón. Hybrid Bayesian networks using mixtures of truncated basis functions. *R Journal*, 12(2):321–341, 2020.

V. C. Pezoulas, K. D. Kourou, C. Papaloukas, V. Triantafyllia, V. Lampropoulou, E. Siouti, M. Papadaki, M. Salagianni, E. Koukaki, N. Rovina, et al. A multimodal approach for the

risk prediction of intensive care and mortality in patients with COVID-19. *Diagnostics*, 12 (1):56, 2021.

G. Pinter, I. Felde, A. Mosavi, P. Ghamisi, and R. Gloaguen. COVID-19 pandemic prediction for Hungary: A hybrid machine learning approach. *Mathematics*, 8(6):890, 2020.

T. Pogiatzis, E. M. Ishiyama, W. R. Paterson, V. S. Vassiliadis, and D. I. Wilson. Identifying optimal cleaning cycles for heat exchangers subject to fouling and ageing. *Applied Energy*, 89(1):60–66, 2012.

M. Pourhomayoun and M. Shakibi. Predicting mortality risk in patients with COVID-19 using machine learning to help medical decision-making. *Smart Health*, 20:100178, 2021.

D. Quesada. *dbnR: Dynamic Bayesian network learning and inference*, 2021. URL https://CRAN.R-project.org/package=dbnR. R package version 0.7.5.

D. Quesada, C. Bielza, and P. Larrañaga. Structure learning of high-order dynamic Bayesian networks via particle swarm optimization with order invariant encoding. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 158–171. Springer, 2021a.

D. Quesada, G. Valverde, P. Larrañaga, and C. Bielza. Long-term forecasting of multivariate time series in industrial furnaces with dynamic Gaussian Bayesian networks. *Engineering Applications of Artificial Intelligence*, 103:104301, 2021b.

D. Quesada, C. Bielza, P. Fontán, and P. Larrañaga. Piecewise forecasting of nonlinear time series with model tree dynamic Bayesian networks. *International Journal of Intelligent Systems*, 37(11):9108–9137, 2022.

D. Quesada, P. Larrañaga, and C. Bielza. Classifying the evolution of COVID-19 severity on patients with combined dynamic Bayesian networks and neural networks. *arXiv preprint arXiv:2303.05972*, 2023.

J. R. Quinlan. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, volume 92, pages 343–348. World Scientific, 1992.

V. Radhakrishnan, M. Ramasamy, H. Zabiri, V. Do Thanh, N. Tahir, H. Mukhtar, M. Hamdi, and N. Ramli. Heat exchanger fouling model and preventive maintenance scheduling tool. *Applied Thermal Engineering*, 27(17-18):2791–2802, 2007.

J. W. Robinson, A. J. Hartemink, and Z. Ghahramani. Learning non-stationary dynamic Bayesian networks. *Journal of Machine Learning Research*, 11(12), 2010.

R. W. Robinson. Counting unlabeled acyclic digraphs. In *Combinatorial Mathematics V*, pages 28–43. Springer, 1977.

B. J. Ross and E. Zuviria. Evolving dynamic Bayesian networks with multi-objective genetic algorithms. *Applied Intelligence*, 26:13–23, 2007.

F. L. Santamaria and S. Macchietto. Integration of optimal cleaning scheduling and control of heat exchanger networks under fouling: MPCC solution. *Computers & Chemical Engineering*, 126:128–146, 2019.

F. P. Santos and C. D. Maciel. A PSO approach for learning transition structures of higher-order dynamic Bayesian networks. In *5th ISSNIP-IEEE Biosignals and Biorobotics Conference (2014): Biosignals and Robotics for Better and Safer Living*, pages 1–6. IEEE, 2014.

M. Scanagatta, A. Salmerón, and F. Stella. A survey on Bayesian network structure learning from data. *Progress in Artificial Intelligence*, 8(4):425–439, 2019.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

M. Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010. doi: 10.18637/jss.v035.i03.

M. Scutari, P. Howell, D. J. Balding, and I. Mackay. Multiple quantitative trait analysis using Bayesian networks. *Genetics*, 198(1):129–137, 2014.

M. Scutari, C. E. Graafland, and J. M. Gutiérrez. Who learns better Bayesian network structures: Accuracy and speed of structure learning algorithms. *International Journal of Approximate Reasoning*, 115:235–253, 2019.

R. D. Shachter and C. R. Kenley. Gaussian influence diagrams. *Management Science*, 35(5): 527–550, 1989.

H. Sharma and S. Kumar. A survey on decision tree algorithms of classification in data mining. *International Journal of Science and Research*, 5(4):2094–2097, 2016.

P. P. Shenoy and J. C. West. Inference in hybrid Bayesian networks using mixtures of polynomials. *International Journal of Approximate Reasoning*, 52(5):641–657, 2011.

P. Singh and Y.-P. Huang. A high-order neutrosophic-neuro-gradient descent algorithm-based expert system for time series forecasting. *International Journal of Fuzzy Systems*, 21(7): 2245–2257, 2019.

P. Spirtes, C. N. Glymour, R. Scheines, and D. Heckerman. *Causation, Prediction, and Search*. MIT press, 2000.

L. E. Sucar. *Probabilistic Graphical Models: Principles and Applications*. Springer Publishing Company, 2015.

B. Sun and Y. Zhou. Bayesian network structure learning with improved genetic algorithm. *International Journal of Intelligent Systems*, 37(9):6023–6047, 2022.

S. Sundar, M. C. Rajagopal, H. Zhao, G. Kuntumalla, Y. Meng, H. C. Chang, C. Shao, P. Ferreira, N. Miljkovic, S. Sinha, and S. Salapaka. Fouling modeling and prediction approach for heat exchangers using deep learning. *International Journal of Heat and Mass Transfer*, 159:120112, 2020.

T. Talvitie, R. Eggeling, and M. Koivisto. Finding optimal Bayesian networks with local structure. In *International Conference on Probabilistic Graphical Models*, pages 451–462. PMLR, 2018.

E. Taskesen. *bnlearn - Library for Bayesian Network Learning and Inference*, 2020. URL https://erdogant.github.io/bnlearn.

F. Tezza, G. Lorenzoni, D. Azzolina, S. Barbar, L. A. C. Leone, and D. Gregori. Predicting in-hospital mortality of patients with COVID-19 using machine learning techniques. *Journal of Personalized Medicine*, 11(5):343, 2021.

A. Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 17(1): 168–192, 2021.

L. Torgo. *Data Mining with R. Learning with Case Studies*. Chapman and Hall/CRC, 2010.

G. Trabelsi. *New Structure Learning Algorithms and Evaluation Methods for Large Dynamic Bayesian Networks*. PhD thesis, Université de Nantes, 2013.

G. Trabelsi, P. Leray, M. B. Ayed, and A. M. Alimi. Dynamic MMHC: A local search algorithm for dynamic Bayesian network structure learning. In *International Symposium on Intelligent Data Analysis*, pages 392–403. Springer, 2013.

N. Trifonova, M. Karnauskas, and C. Kelble. Predicting ecosystem components in the Gulf of Mexico and their responses to climate variability with a dynamic Bayesian network model. *PLOS ONE*, 14(1), 2019.

M. Tsagris. Bayesian network learning with the PC algorithm: An improved and correct variation. *Applied Artificial Intelligence*, 33(2):101–123, 2019.

I. Tsamardinos, C. F. Aliferis, and A. Statnikov. Time and sample efficient discovery of Markov blankets and direct causal relations. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 673–678, 2003.

I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

A. Tucker and X. Liu. Learning dynamic Bayesian networks from multivariate time series with changing dependencies. In *Advances in Intelligent Data Analysis V: 5th International Symposium on Intelligent Data Analysis, Proceedings 5*, pages 100–110. Springer, 2003.

A. Tucker, X. Liu, and A. Ogden-Swift. Evolutionary learning of dynamic probabilistic models with large time lags. *International Journal of Intelligent Systems*, 16(5):621–645, 2001.

P. K. Vaddi, M. C. Pietrykowski, D. Kar, X. Diao, Y. Zhao, T. Mabry, I. Ray, and C. Smidts. Dynamic Bayesian networks based abnormal event classifier for nuclear power plants in case of cyber security threats. *Progress in Nuclear Energy*, 128:103479, 2020.

A. Vaid, S. Somani, A. J. Russak, J. K. De Freitas, F. F. Chaudhry, I. Paranjpe, K. W. Johnson, S. J. Lee, R. Miotto, F. Richter, et al. Machine learning to predict mortality and critical events in a cohort of patients with COVID-19 in New York City: Model development and validation. *Journal of Medical Internet Research*, 22(11), 2020.

R. J. van Berlo, E. P. van Someren, and M. J. Reinders. Studying the conditions for learning dynamic Bayesian networks to discover genetic regulatory networks. *Simulation*, 79(12): 689–702, 2003.

G. Van Houdt, C. Mosquera, and G. Nápoles. A review on the long short-term memory model. *Artificial Intelligence Review*, 53(8):5929–5955, 2020.

N. Vanli and S. Kozat. A comprehensive approach to universal nonlinear regression based on trees. *IEEE Transactions on Signal Processing*, 62(20):5471–5486, 2014.

A. Vepa, A. Saleem, K. Rakhshan, A. Daneshkhah, T. Sedighi, S. Shohaimi, A. Omar, N. Salari, O. Chatrabgoun, D. Dharmaraj, et al. Using machine learning algorithms to develop a clinical decision-making tool for COVID-19 inpatients. *International Journal of Environmental Research and Public Health*, 18(12):6228, 2021.

M. Villegas, A. Gonzalez-Aguirre, A. Gutiérrez-Fandiño, J. Armengol-Estapé, C. P. Carrino, D. P. Fernández, F. Soares, P. Serrano, M. Pedrera, N. García, et al. Predicting the evolution of COVID-19 mortality risk: A recurrent neural network approach. *MedRxiv*, pages 2020–12, 2021.

N. X. Vinh, M. Chetty, R. Coppel, and P. P. Wangikar. GlobalMIT: Learning globally optimal dynamic Bayesian network with the mutual information test criterion. *Bioinformatics*, 27 (19):2765–2766, 2011.

H. Wang and B. Raj. On the origin of deep learning. *arXiv preprint arXiv:1702.07800*, 2017.

Y. Wang, Z. Yuan, Y. Liang, Y. Xie, X. Chen, and X. Li. A review of experimental measurement and prediction models of crude oil fouling rate in crude refinery preheat trains. *Asia-Pacific Journal of Chemical Engineering*, 10(4):607–625, 2015.

G. Wu, P. Yang, Y. Xie, H. C. Woodruff, X. Rao, J. Guiot, A.-N. Frix, R. Louis, M. Moutschen, J. Li, et al. Development of a clinical decision support system for severity risk prediction and triage of COVID-19 patients at hospital admission: An international multicentre study. *European Respiratory Journal*, 56(2), 2020.

H. Wu and X. Liu. Dynamic Bayesian networks modeling for inferring genetic regulatory networks by search strategy: Comparison between greedy hill climbing and MCMC methods. *International Journal of Computer and Information Engineering*, 2(8):2585–2595, 2008.

Y. Wu, J. McCall, and D. Corne. Two novel ant colony optimization approaches for Bayesian network structure learning. In *IEEE Congress on Evolutionary Computation*, pages 1–7. IEEE, 2010.

H. Xing-Chen, Q. Zheng, T. Lei, and L.-P. Shao. Research on structure learning of dynamic Bayesian networks by particle swarm optimization. In *2007 IEEE Symposium on Artificial Life*, pages 85–91, 2007.

D. Xiong, L. Zhang, G. L. Watson, P. Sundin, T. Bufford, J. A. Zoller, J. Shamshoian, M. A. Suchard, and C. M. Ramirez. Pseudo-likelihood based logistic regression for estimating COVID-19 infection and case fatality rates by gender, race, and age in California. *Epidemics*, 33:100418, 2020.

A. S. Yadaw, Y. Li, S. Bose, R. Iyengar, S. Bunyavanich, and G. Pandey. Clinical predictors of COVID-19 mortality. *MedRxiv*, 2020.

J. York. Use of the Gibbs sampler in expert systems. *Artificial Intelligence*, 56(1):115–130, 1992.

Y. Yu, M. Travaglio, R. Popovic, N. S. Leal, and L. M. Martins. Alzheimer's and Parkinson's diseases predict different COVID-19 outcomes: A UK Biobank study. *Geriatrics*, 6(1):10, 2021.

C. Yuan and M. J. Druzdzel. Importance sampling algorithms for Bayesian networks: Principles and performance. *Mathematical and Computer Modelling*, 43(9-10):1189–1207, 2006.

J. Zhu, W. Zhang, and X. Li. Fatigue damage assessment of orthotropic steel deck using dynamic Bayesian networks. *International Journal of Fatigue*, 118:44–53, 2019.