TÉCNICO LISBOA

Sparse dynamic Bayesian network estimation using generalized linear models

João Bernardo Almeida Santos

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. Alexandra Sofia Martins de Carvalho
Prof. Susana de Almeida Mendes Vinga Martins

Examination Committee

Chairperson: Prof. Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Prof. Alexandra Sofia Martins de Carvalho
Member of the Committee: Prof. Eunice Isabel Ganhão Carrasquinha Trigueirão

September 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Ao meu pai.

Agradecimentos

Primeiro, agradeço profundamente à minha mãe e ao meu irmão pela odisseia que foi aturar-me durante este ano e meio em dever de recolhimento. Não foram tempos fáceis para ninguém e sem o apoio deles este trabalho teria sido ainda mais impossível. Ao Tomás dedico, como prometido, todas as páginas em branco desta dissertação: foram as que ele ajudou a escrever.

O meu maior obrigado aos meus amigos chegados e colegas do técnico Diogo Oliveira, Pedro Rodrigues, Alexandre Abreu, André Godinho e José Malaquias. Irei para sempre lembrar os jantares, as noites na Cervetoria, as idas a Belém, os *Chefs on Fire* e todas as aventuras em Lisboa. Um obrigado especial à trupe do Interrail que durante os confinamentos me transformaram num *gamer*, permitindo que nos mantivéssemos em contacto. Sem essas noitadas esta dissertação não seria possível.

Uma menção honrosa para os meus *house mates* e amigos – aliás, família – Pedro Cruz, Francisco Mendes e João Cruz, que me acompanharam ao longo destes cinco (seis?) anos. Por mais complicado que o dia tivesse sido, por mais tarde que saísse do IST, era sempre um alívio voltar para casa e descontrair. Lembrar-me-ei sempre das nossas viagens, jantares, noites, almoços no Rui dos Pregos, sustos, filmes e séries, guitarradas, *talkshows* originais e restantes aventuras.

Quero agradecer também ao Prof. Luís Miguel Silveira por me conceder a oportunidade de lecionar um semestre de Algoritmos e Estruturas de Dados durante a realização desta dissertação. Esta experiência contribuiu imenso para melhorar as minhas capacidades de comunicação e aprofundou o meu conhecimento. Além disso, ajudou-me a estabelecer horários: foram os meses mais produtivos a desenvolver este trabalho. Agradeço ainda a todo o restante corpo docente de AED 1º Semestre 2020/2021, Prof. Carlos Bispo, Prof. João Ascenso e Eng. Gonçalo Mestre por terem sido uma excelente equipa, na qual sempre me senti integrado. Peço desculpa a todos os alunos que foram vítima das minhas aulas de laboratório. Há piores.

Agradeço às minhas orientadoras Prof. Alexandra Carvalho e Prof. Susana Vinga por todos os conselhos e toda a ajuda na elaboração desta tese. Este projeto foi parcialmente financiado por uma bolsa de investigação inserida no projeto PREDICT (PTDC/CCI-CIF/29877/2017), pela qual estou muito grato – o meu obrigado à Fundação para a Ciência e Tecnologia (FCT).

Por último, mas não menos importante, agradeço ao meu pai, a quem dedico este trabalho. Obrigado por teres recomendado um livro de "Delphi 6" – uma monstrousidade com mais de 1000 páginas – a um míudo com 10 anos que queria aprender a programar, ensinando-me talvez a maior lição de todas: **desenrasca-te**. Obrigado por me teres mostrado a porta da frente do IST com a sugestão: é para ali que tens de ir. Obrigado... por tudo.

Resumo

As redes Bayesianas Dinâmicas são modelos probabilísticos gráficos usados para a modelação de processos estocásticos. Atendendo às recentes aplicações de *data mining*, estes modelos podem ser treinados com séries temporais multivariadas e a sua interpretação permite encontrar relações interessantes entre as variáveis medidas. Os algoritmos de aprendizagem ótimos têm, contudo, uma elevada complexidade computacional. Esta dificuldade inspira o desenvolvimento de novas técnicas que recorrem a procedimentos heurísticos para reduzir o espaço de procura.

Esta dissertação introduz um método de treino chamado `sDBN`, que possui uma complexidade computacional mais atrativa e pode ser utilizado para dados com alta dimensionalidade. Esta nova técnica recorre a métodos recentes que ligam o treino de redes Bayesianas discretas com modelos lineares generalizados. O método proposto treina redes estacionárias e não estacionárias, recebendo como entrada um atraso de Markov.

Os resultados experimentais mostram que o algoritmo proposto consegue identificar redes válidas e atinge recuperação perfeita de estruturas com dimensão elevada, utilizando dados simulados. Neste contexto, o algoritmo supera os métodos mais avançados disponíveis em termos de qualidade e de tempo de treino. Ao classificar séries temporais reais, o `sDBN` mostra também resultados competitivos. Em dados médicos de pacientes com Espondilite Anquilosante, retirados do `Reuma.pt`, uma base de dados nacional de doentes reumáticos, o algoritmo recupera estruturas inteligíveis e consegue prever a progressão da doença. Estes resultados posicionam o `sDBN` como um método alternativo viável para o treino de redes Bayesianas dinâmicas e sugerem que o algoritmo pode ser treinado com dados de alta dimensionalidade.

Palavras-chave: séries temporais multivariadas, redes Bayesianas dinâmicas, modelos lineares generalizados, aprendizagem automática de estrutura, dados de alta dimensionalidade, *data mining*

Abstract

Dynamic Bayesian Networks (DBNs) are probabilistic graphical models used to predict the evolution of stochastic processes. Following recent trends in data mining, these models can be trained from multivariate time-series data to uncover interesting temporal relationships between measured variables. However, optimal training algorithms are computationally prohibitive which inspires the development of heuristic techniques to prune the search space.

This dissertation introduces `sDBN`, an alternative training algorithm with better computational complexity that can handle high-dimensional data. This method leverages state-of-the-art techniques that bridge discrete Bayesian network training with generalized linear models, extending them to handle temporal data. The proposed method handles both stationary and non-stationary models and is flexible to a specified Markov lag.

Empirical results show that the algorithm achieves meaningful network identification, accomplishing up to perfect F_1 scores in artificial datasets with a considerable number of dimensions. Using simulated data, `sDBN` outperforms state-of-the-art dynamic Bayesian network training algorithms both in terms of structure quality, and training time. Tests in benchmark public datasets show that `sDBN` is also competitive in time-series classification. Using Ankylosing Spondylitis patient data from `Reuma.pt`, a national rheumatological registry, the new method recovers intelligible models and successfully predicts disease progression. These results validate the novel algorithm as an alternative methodology to identify dynamic Bayesian networks from data and suggest that this algorithm enables training using high-dimensional datasets.

Keywords: multivariate time-series, dynamic Bayesian networks, generalized linear models, structure learning, high-dimensional data, data mining

Contents

List of Tables

List of Figures

Acronyms

AS	Ankylosing Spondylitis
ASDAS	Ankylosing Spondylitis Disease Activity Score
AUC	Area Under The ROC Curve
BASDAI	Bath Ankylosing Spondylitis Disease Activity Index
BASFI	Bath Ankylosing Spondylitis Functional Index
BLAS	Basic Linear Algebra Subprogram
BN	Bayesian Network
CPD	Conditional Probability Distribution
CPU	Central Processing Unit
CRP	C-Reactive Protein
CSV	Comma-Separated Values
DAG	Directed Acyclic Graph
DBN	Dynamic Bayesian Network
ESR	Erythrocyte Sedimentation Rate
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GLM	Generalized Linear Model
GPU	Graphics Processing Unit
JSON	JavaScript Object Notation
KKT	Karush-Kuhn-Tucker
LASSO	Least Absolute Shrinkage And Selection Operator
LL	Log-Likelihood
MAP	Maximum *A posteriori*
MDL	Minimum Descriptor Length
MKL	Math Kernel Library
MLE	Maximum Likelihood Estimation
MMHC	Max-Min Hill Climbing
MPI	Message Passing Interface
MTS	Multivariate Time Series

PGM	Probabilistic Graphical Model
ROC	Receiver Operation Characteristic
RV	Random Variable
SPR	Portuguese Society Of Rheumatology
TP	True Positive
TPR	True Positive Rate
VAS	Visual Analogue Scale

Chapter 1

Introduction

1.1 Motivation

Data science has become, over the last few years, ubiquitous in society and all our lives. Data is currently being collected everywhere due to the digitalization of society and the advent of the “information age”. Vast amounts of information call for the development of automatic techniques for its analysis. Using concepts from statistics and computer science, this field of study has exponentially increased in popularity in the last decade. Under this subject, two distinct areas can be identified: machine learning and data mining. Machine learning studies a set of techniques that allow for computers to learn some task and then generalize when faced with previously unseen data, hopefully still achieving satisfactory results. State-of-the-art techniques involve using some form of deep neural networks due to the phenomena of dual descent. Success stories bleed from everywhere, across a vast amount of areas such as games, image classification, natural language processing, translation, and even self-driving cars. On the other hand, data mining aims to extract patterns from data and help in decision making. The focus is not on performing a task, but instead on analyzing the data and getting insights. “Data mining turns a large collection of data into knowledge” [1]. Examples include abnormality detection, business intelligence, prediction of future stock market value, and even the prediction of a treatment outcome based on clinical indicators.

One type of datasets are multivariate time series (MTS), representing how a set of multiple variables evolves over time. MTS datasets frequently arise in several contexts like meteorology, robotics, economics and finance, and electronic health records. In the latter, for example, the patient evolution is tracked over the course of several doctor appointments. Typical medical exams and questionnaires are repeated for each appointment, resulting in the measurement of several features over a time period. The increase in the availability of this type of data, also associated with the digitalization of society, justifies the growing interest in data mining techniques for the analysis of MTS.

Dynamic Bayesian networks (DBNs) are a class of mathematical models that can be used for the analysis of time-series. They are a probabilistic graphical model (PGM) and therefore can be easily schematized using a graph and provide interpretable information on the relationships between measured variables. This is especially important in medical applications, where decisions should follow a well-defined rationale instead of simply being outputs of black-box type models. Additionally, these models are flexible and allow the computation of complex probability queries through inference algorithms and the prediction of the future evolution of the variables based on past observations using a process called "unrolling". Another advantage of these kinds of mathematical models is the possibility of fine-tuning using domain expert knowledge. DBNs can also be trained automatically from data, but exact optimization algorithms are computationally expensive. To counteract this, the training typically includes the use of state-of-the-art heuristic methods to restrict the search space enough so that a computer can tackle it in a reasonable time.

Recent work on discrete Bayesian network (BN) training leverage generalized linear models (GLMs) as an approximate parametrization and employ regularization techniques to train sparse networks. GLMs are non-linear generalizations of regular linear models commonly applied in machine learning and data mining. In these contexts, gradient-based optimization techniques are used to identify these models from data and characterize the underlying process. GLMs are also the heart of some deep neural networks, and the recent advances in this field provide a sophisticated computational framework that can leverage modern hardware to accelerate the training procedure meaningfully. Training BNs approximating them as a collection of GLMs profits from these novel techniques and are mandatory in high-dimensional contexts.

Although plenty of literature is found on identifying static models, extending these techniques to include temporal information is still unexplored. Therefore, this thesis focuses on alternative heuristic methods for training DBNs leveraging GLMs and aiming to recover a sparse structure and allow the possibility of training of these kinds of models using high-dimensional MTS, *i.e.*, time-series with several measured variables per timestep with few observations.

1.2 Objectives and contributions

The main objective of this thesis is the development of a new method for learning sparse discrete dynamic Bayesian networks, introducing the possibility of handling high-dimensional data previously tricky due to the computational complexity of the heuristic methods involved. The algorithm should learn from provided multivariate time-series data with categorical distributed features, outputting a locally optimal network structure, preserving time course causality. The method produces different types of models, namely for the description of stationary and non-stationary stochastic processes.

The main contributions of this thesis are the following:

1. A review of the literature on probabilistic graphical models and generalized linear models.

2. The proposal of a new method for training dynamic Bayesian network based on the reparametrization of the network and state-of-the-art regularization techniques and gradient-based optimization.

3. The implementation of the proposed method in C++, providing a new and flexible software solution for the training of these models allowing for training acceleration using Intel® math kernel library (MKL) and distribution across a computing cluster using the message passing interface (MPI)

protocol. The proposed implementation was made available as open-source software on: `https://github.com/JBSants/sDBN`.

4. A set of validation results, obtained by training the network with artificial and benchmark datasets.

5. The application of the method to real data from the national registry of rheumatological patients. `Reuma.pt`.

1.3 Thesis Outline

Chapter 2 briefly introduces linear regression, maximum likelihood estimation, and generalized linear models, as well as some regularization techniques usually employed. Chapter 3 dives into probabilistic graphical models, introducing Bayesian networks, dynamic Bayesian networks, and Bayesian multinets. It describes some applications of these models and state-of-the-art training techniques. Chapter 4 exposes in detail the developed method and the optimization technique used in the provided implementation. Chapter 5 presents some results obtained in generated artificial datasets, benchmark MTS data, and Ankylosing Spondylitis medical data and discusses them, validating the proposed solution. Chapter 6 concludes the thesis, balancing the achievements, and provides some thoughts on future directions to extend this work.

Chapter 2

Generalized Linear Models

A linear model is a model that assumes that the output variable y is given, in terms of the input variables $\mathbf{X}$, by a linear relationship. In the one-dimensional case, where both the response and input variables are scalars, they are related by the equation $y = \beta_0 + \beta_1 X$. Plotting this relationship yields a straight line.

Given a set of observations and assuming that these observations are "ruled" by a linear model, machine learning aims to use computers to identify such models from the data automatically. Intuitively, the algorithm tries to find the best straight line that fits the observations. The problem is how to characterize mathematically what the best line is. Having this mathematical description of what it means to be the best line, optimization techniques are used to find it.

Several methods may be used, but the first to be discovered was the least-squares approach, mostly credited to Gauss [2], despite being first published by Legendre in 1805 [3]. Using this method, the model parameters β_0 and β_1 are estimated by minimizing the square of the error term between the observed data and the predicted data, with respect to the model parameters, *i.e.*,

$$\underset{\beta_0,\,\beta_1}{\text{minimize}} \quad e^2(\beta_0, \beta_1, D), \tag{2.1}$$

where D is a dataset with M observations. The squared error term is simply given by the squared distance between the observed value and the estimated value for the corresponding output

$$e^2(\beta_0, \beta_1, D) = \sum_{i=1}^{M} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 = \sum_{i=1}^{M} \left(y^{(i)} - \beta_0 - \beta_1 X^{(i)}\right)^2,$$

where $y^{(i)}$ is the i-th observation of the output variable, $X^{(i)}$ the corresponding input variable, and $\hat{y}^{(i)}$ the predicted response variable by the linear model with parameters β_0 and β_1.

This model may be extended to handle multidimensional input variables by considering $\mathbf{X} = (1, x_1 \ldots, x_n)$ and a parameter vector $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_n)$. In this case, assuming the vectors to be column vectors, $y = \boldsymbol{\beta}^\top \mathbf{X}$. The error term is computed, replacing the prediction using this equation.

Another way to pose the problem of finding the best line is to use a probabilistic framework and assign a probability value to a model given a dataset of observed values. This probability value is usually referred to as likelihood and expresses how likely a model would produce the data if sampled from it.

The best model is the model that leads to the highest probability value. This method is called maximum likelihood estimation (MLE), and it is based on early work by Laplace, and Gauss [4], consolidated by Fisher [5].

To transport the original line fitting problem to a probabilistic framework, usually, it is considered that the system has a linear model relationship perturbed by an error term that is drawn from a probability distribution, *i.e.*, in the one-dimensional case,

$$y = \beta_0 + \beta_1 X + e,$$

where e is the error term. Additionally, it is made an assumption on the distribution of this error on the observed dataset due to the measurement process while attaining the observations.

Considering the observation errors to be independent and identically distributed, and that the i-th observation error $e^{(i)} \sim \mathcal{N}(0, \sigma^2)$, the observation $y^{(i)}$ also becomes normally distributed with parameters $\mathcal{N}(\beta_0 + \beta_1 X, \sigma^2)$. The probability that the observation takes a certain value ζ can be computed as

$$P(y^{(i)} = \zeta \mid \beta_0, \beta_1, X^{(i)}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(\frac{\zeta - \beta_0 - \beta_1 X^{(i)}}{\sigma}\right)^2\right\}.$$

The maximum likelihood estimation of the parameters is obtained by solving the optimization problem

$$\underset{\beta_0, \beta_1}{\text{maximize}} \quad \prod_{i=1}^{M} P(y^{(i)} \mid \beta_0, \beta_1, X^{(i)}).$$

Solving this maximum likelihood problem is equivalent to solving the least-squares problem (Equation (2.1)). For increasingly large datasets, the value of likelihood becomes increasingly low. This most likely originates precision problems when using computers. For this reason, and also because of the usually better mathematical properties, the equivalent problem of minimizing the logarithm of the likelihood is solved instead of directly solving the MLE problem. The log-likelihood in the linear case with gaussian errors is

$$\ell(\beta_0, \beta_1, D) = -M \log \sigma - \frac{M}{2}\log(2\pi) - \frac{1}{2\sigma^2}\sum_{i=1}^{M}\left(y^{(i)} - \beta_0 - \beta_1 X^{(i)}\right)^2.$$

Note that only the third term depends on the parameters, thus being the only term that must be maximized, reducing the problem to simple least squares. Maximization of differentiable convex losses is usually done using a gradient descent method. However, the solution to linear regression can be computed in closed form [6] and is given by

$$\boldsymbol{\beta} = (\mathbf{O}^\top \mathbf{O})^{-1}\mathbf{O}^\top Y,$$

where $\mathbf{O}$ is a $M \times n$ matrix constructed by stacking the input variables for all observations and $\mathbf{Y} \in \mathbb{R}^M$ is a vector with every observation of the response variable. Note that this is a solution provided that some conditions are met by the observation matrix, namely that $\mathbf{O}^\top \mathbf{O}$ is invertible.

This reformulation of the linear regression problem as a likelihood problem and the translation of the linear model to a probabilistic framework allows for the generalization of the linear model. It enables the assumption of different distributions of the output variable, and alternative link functions between the distribution parameters and the linear mapping of the input variables with the model parameters.

A GLM [7] is a model with parameters $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_n)$ that relates the response variable y with the features $\mathbf{X} = (1, x_1, \ldots, x_n)$, via a linear mapping $Z = \mathbf{X}^T\boldsymbol{\beta}$, a linking function $g(Z)$ and an assumption on the distribution of the response variables $y \sim \Psi(\theta)$, where Ψ is a distribution of the exponential family with parameter $\boldsymbol{\theta} = g(Z)$.

With $\Psi(\theta) = \mathcal{N}(\boldsymbol{\theta}, \sigma^2)$ and $g(Z) = Z$, the GLM specializes to linear regression. Different choices of distribution and link function lead to different, well-known models. In this chapter, logistic regression is introduced, and an extension of logistic regression to multinomial models. Then, some regularization techniques that can be employed while training these models are introduced.

2.1 Univariate Logistic Regression

Let $\Psi(\boldsymbol{\theta}) = \text{Bernoulli}(\boldsymbol{\theta})$ and the link function be the logistic function

$$g(Z) = \frac{e^Z}{1 + e^Z}.$$

This generalized linear model is called logistic regression [8], and it is used to predict the outcome of a binary response variable from a set of features. This can be achieved due to the S shape of the logistic function, as seen in Figure 2.1. For larger values of the input map Z, the probability becomes almost one, and for smaller values, almost zero. The transition region increases rapidly, and the steepness can be controlled by the parameters of the model when seen by the input variables.

Considering the probability mass function associated with the Bernoulli distribution, the expansion

$$P(y = k \mid \mathbf{X}, \boldsymbol{\beta}) = \theta^k (1-\theta)^{1-k} = \left(\frac{e^{\mathbf{X}^\top\boldsymbol{\beta}}}{1 + e^{\mathbf{X}^\top\boldsymbol{\beta}}}\right)^k \left(1 - \frac{e^{\mathbf{X}^\top\boldsymbol{\beta}}}{1 + e^{\mathbf{X}^\top\boldsymbol{\beta}}}\right)^{1-k},$$

can be deducted, leading to the characteristic probability mass function of logistic regression

$$P(y = k \mid \mathbf{X}, \boldsymbol{\beta}) = \left(\frac{\exp(\mathbf{X}^\top\boldsymbol{\beta})}{1 + \exp(\mathbf{X}^\top\boldsymbol{\beta})}\right)^k \left(\frac{1}{1 + \exp(\mathbf{X}^\top\boldsymbol{\beta})}\right)^{1-k}, \tag{2.2}$$

where $k \in \{0, 1\}$.

The log-likelihood of a given set of model parameters explaining a dataset D with M training pairs $(\mathbf{X}^{(i)}, y^{(i)})$ is given by

$$\ell(\boldsymbol{\beta}|D) = \sum_{i=1}^{M} \left\{ y^{(i)}\, \boldsymbol{\beta}^\top \mathbf{X}^{(i)} - \log\left[1 + \exp(\boldsymbol{\beta}^\top \mathbf{X}^{(i)})\right] \right\}.$$

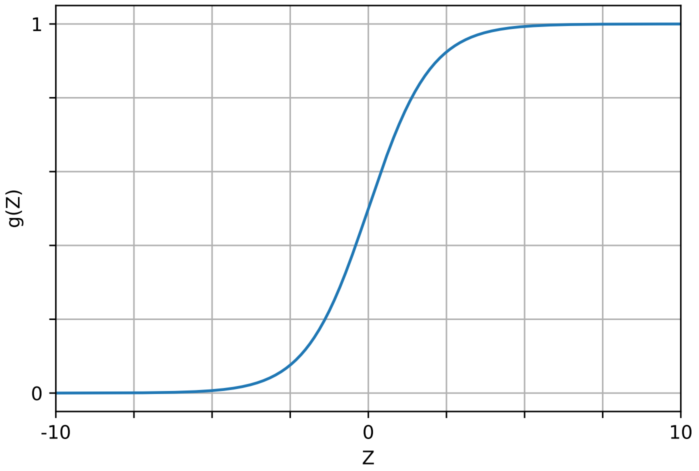


Figure 2.1: A plot of the logistic function. This illustrates the S-shaped curve originated by the logistic function, allowing it to model binary outcomes from a continuous perspective. The curve has two asymptotes and is limited by the value one when $Z \to +\infty$ and by the value zero when $Z \to -\infty$, being suitable to be used directly as a probability value.

2.2 Multinomial Logistic Regression

Multinomial Logistic Regression extends logistic regression to handle the prediction of categorical distributed random variables, allowing the response variables to take r possible categories. Therefore, $\Psi(\boldsymbol{\theta})$ is set to be the categorical distribution, and $\boldsymbol{\theta} = (\theta_1, \dots, \theta_r)$ is simply a vector determining the probability for each possible category. This parameter vector has to satisfy

$$\sum_{i=1}^{r} \theta_i = 1.$$

The link function has to specify every parameter independently. To achieve this, preserving the only dependency on a linear map of the features vector, the features vector must be extended, repeating itself for each category, allowing for a different set of parameters for each possible response. The extended features vector has the form

$$\mathbf{X}' = \begin{bmatrix} \mathbf{X} & \cdots & \mathbf{X} \end{bmatrix}^\top \in \mathbb{R}^{r(n+1)},$$

and the corresponding model parameters vector has the form

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta_1} & \cdots & \boldsymbol{\beta_r} \end{bmatrix} \in \mathbb{R}^{r(n+1)},$$

where $\boldsymbol{\beta_i} \in \mathbb{R}^{n+1}$ are the model parameters associated with category i. Subsequently, the link function exponentiates every linear map obtained for each set of model parameters and normalizes the distribution parameters, forcing them to sum to one, *i.e.*,

$$g(Z) = \left(\frac{\exp(\mathbf{X}^\top \boldsymbol{\beta_1})}{\sum_{i=1}^{r} \exp(\mathbf{X}^\top \boldsymbol{\beta_i})}, \dots, \frac{\exp(\mathbf{X}^\top \boldsymbol{\beta_r})}{\sum_{i=1}^{r} \exp(\mathbf{X}^\top \boldsymbol{\beta_i})} \right).$$

Note that $Z = \mathbf{X'}^\top \boldsymbol{\beta}$, but every internal product $\mathbf{X}^\top \boldsymbol{\beta_i}$ can be obtained from Z by selecting the appropriate components (via another linear map), due to the way $\mathbf{X'}$ and $\boldsymbol{\beta}$ were constructed.

The probability mass function for multinomial logistic regression is obtained directly from the categorical distribution

$$P(y = k \mid \mathbf{X}, \boldsymbol{\beta_1}, \dots, \boldsymbol{\beta_r}) = \theta_k = \frac{\exp(\mathbf{X}^\top \boldsymbol{\beta_k})}{\sum_{i=1}^r \exp(\mathbf{X}^\top \boldsymbol{\beta_i})}, \tag{2.3}$$

where $k \in \{1, \dots, r\}$.

The log-likelihood of a given set of model parameters explaining a dataset D with M training pairs $(\mathbf{X}^{(i)}, y^{(i)})$ is given by

$$\ell(\boldsymbol{\beta}|D) = \sum_{i=1}^{M} \left[\sum_{l=1}^{r} I(y^{(i)} = l) \boldsymbol{\beta_l}^\top \mathbf{X}^{(i)} - \log \left\{ \sum_{l=1}^{r} \exp(\boldsymbol{\beta_l}^\top \mathbf{X}^{(i)}) \right\} \right]. \tag{2.4}$$

Several model parameters $\boldsymbol{\beta}$ can lead to the same probability distribution. In fact, by choosing a new beta $\boldsymbol{\beta'} = \boldsymbol{\beta} + \alpha \mathbb{1}$ for any $\alpha \in \mathbb{R}$ then

$$g_k(\mathbf{X'}^\top \boldsymbol{\beta'}) = \frac{\exp(\mathbf{X}^\top \boldsymbol{\beta_k} + \alpha)}{\sum_{i=1}^r \exp(\mathbf{X}^\top \boldsymbol{\beta_i} + \alpha)} = \frac{\exp(\alpha) \exp(\mathbf{X}^\top \boldsymbol{\beta_k})}{\exp(\alpha) \sum_{i=1}^r \exp(\mathbf{X}^\top \boldsymbol{\beta_i})} = \frac{\exp(\mathbf{X}^\top \boldsymbol{\beta_k})}{\sum_{i=1}^r \exp(\mathbf{X}^\top \boldsymbol{\beta_i})}.$$

Due to this, when identifying these models from data, a model parameter is usually constrained to a given value, effectively eliminating this ambiguity.

2.3 Regularization Techniques

Maximum Likelihood Estimation, although effective, is prone to overfit the model to the observed data. In order to fix this issue, typically, regularization techniques are used, adding a new term to the cost function achieving a smaller model variance at the cost of adding some bias to the prediction.

Ridge Regression

Ridge regression [9] consists of imposing an ℓ_2-norm constraint $||\boldsymbol{\beta}||^2 \leq t$ on the model parameters, where t is an arbitrary bound on the norm. This optimization problem can be solved by solving an equivalent unconstrained optimization problem, using the Lagrangian multipliers method, given by

$$\underset{\beta}{\text{minimize}} \quad -\frac{1}{M} \ell(\boldsymbol{\beta} \mid D) + \lambda ||\boldsymbol{\beta}||^2. \tag{2.5}$$

The parameter λ determines how much regularization is applied, and there is a value of λ that corresponds directly to a value of t. Ridge promotes shrinkage of the model parameters and is typically used to obtain a more stable model with usually better accuracy prediction on the test data [10]. Moreover, in situations of high-dimensional data, where $M < n$, or when the observed data does not fulfill certain conditions, traditional linear regression has an infinite number of solutions. Ridge regression may be applied, yielding a set of parameters that cannot be obtained by solving the unregularized problem.

LASSO Regression

Similar to Ridge regression, the regression using the least absolute shrinkage and selection operator

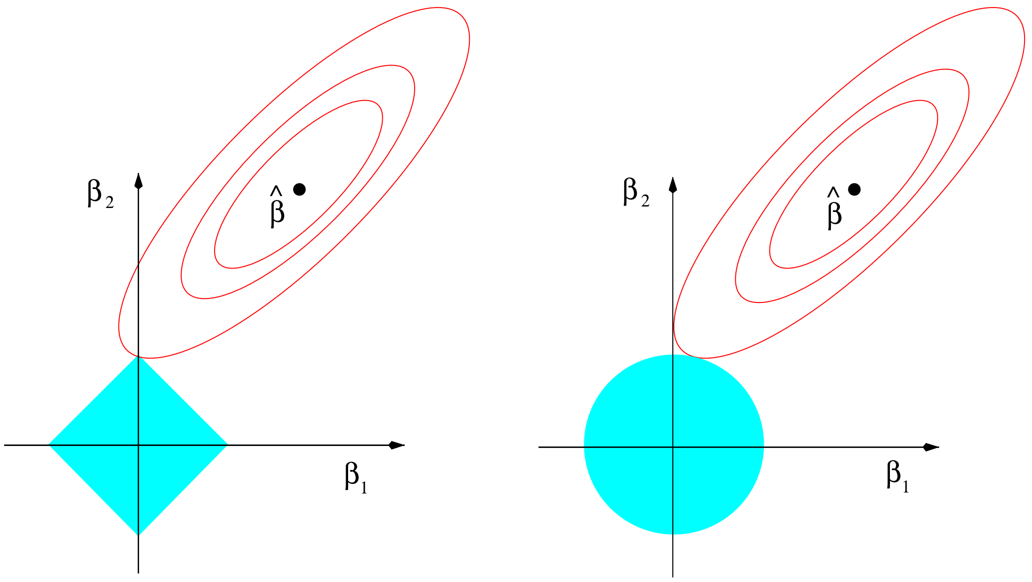


Figure 2.2: Comparison between LASSO (left) and Ridge (right) regression. To illustrate why the LASSO penalty yields a sparse vector, a toy problem was constructed using only two dimensions. The curves drawn in red are the level curves of the negative log-likelihood loss, and the regions in blue are the constraints imposed on the parameters by each regression. The point $\hat{\beta}$ is the unconstrained problem solution, *i.e.*, the MLE estimation of the parameters. Source: [10]

(LASSO) [11] consists of imposing an ℓ_1-norm constraint on the model parameters $||\boldsymbol{\beta}||_1 \leq t$. Like Ridge, the parameter set is typically obtained by solving the equivalent unconstrained optimization problem given by

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad -\frac{1}{M}\ell(\boldsymbol{\beta} \mid D) + \lambda ||\boldsymbol{\beta}||_1. \tag{2.6}$$

LASSO is a special case of regularization methods using ℓ_q-norms because it yields a sparse parameter vector, *i.e.*, some irrelevant or redundant parameters are set exactly to 0. As seen in Figure 2.2 for a toy problem, the level curves of the negative log-likelihood cost are likely to intercept the constraint on a vertice of the square, thus setting one of the parameters to exactly zero. On the other hand, in Ridge regression, the parameters become small, close to zero, but not quite exactly zero. This picture provides an intuition for why this constraint yields not only small coefficients, but also some of them exactly zero.

This property is desirable because it applies feature selection while training the model, remaining a convex optimization problem. Models obtained via LASSO are simpler, more stable, and easier to interpret because the response variable is most likely explained by fewer features than their ridge counterpart. Convexity is essential because it allows the use of computationally efficient optimization algorithms, and the ℓ_1-norm is the norm with smaller q that is still convex.

Group LASSO

Group LASSO [12] is a regularization technique used to obtain sparse solutions where predictors are grouped, and a group is either considered to explain the response variable or discarded, setting all the features in the group to zero. Let the model parameters $\boldsymbol{\beta}$ be split in J groups of coefficients. These groups may have different lengths.

Let $\boldsymbol{\gamma} \in \mathbb{R}^n$ be an arbitrary vector of dimension $n \geq 1$ and $\mathbf{K}$ be a symmetric positive definite matrix of dimension $d \times d$. Consider the norm

$$||\boldsymbol{\gamma}||_{\mathbf{K}} = \sqrt{\boldsymbol{\gamma}^\top \mathbf{K} \boldsymbol{\gamma}}.$$

Sparse group regularization is applied by solving the unconstrained optimization problem given by

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad -\frac{1}{M}\ell(\boldsymbol{\beta} \mid D) + \lambda \sum_{i=1}^{J} ||\boldsymbol{\beta_i}||_{\mathbf{K_i}}, \tag{2.7}$$

where $\mathbf{K_1}, \ldots, \mathbf{K_J}$ are positive definite matrices of the appropriate dimension to be applied with the norm of the group.

Typical values for the $\mathbf{K_i}$ matrices are identity matrices, transforming the regularization term in the sum of simple ℓ_2-norms for each group.

Chapter 3

Probabilistic Graphical Models

3.1 Introduction

Probabilistic Graphical Models [6, 13] are schematic representations of probabilistic relationships between random variables (RVs) that allow better visualization and understanding of the underlying probabilistic model than pure algebraic expressions. In these models, probability distributions are represented via graphs, where each node represents a random variable, and the edges encode relationships between them (*e.g.*, causality). These representations have several advantages – they provide an easy way to visualize the properties of a given probability model, like conditional independence; they offer an alternative representation of uncertain models that is more compact when compared with joint distributions; they define a structure where complex computations can be defined rigorously, allowing inference and learning algorithms.

In Pearl [13], two types of PGMs are defined – Bayesian networks and Markov networks. Bayesian networks rely on directed graphs to describe relationships between random variables. This approach provides an even more intuitive way to reason about the underlying model, considering that arcs can be interpreted as a causation relationship. On the other hand, Markov networks use undirected graphs and are more suited for handling models with relationships that do not have a sense of directionality. Both these models can be reduced to a factorized representation of a joint distribution.

Being probabilistic models, they encode the uncertainty associated with the root process aimed to be described. Therefore, one use of them is to query the likelihood of a specific event to happen, given the evidence observed so far. This process is called inference. Mathematically, this translates to computing $P(\mathbf{x} \mid \mathbf{e})$, where $\mathbf{x}$ is some arbitrary realization of a set of RVs and $\mathbf{e}$ is some observed evidence. For example, consider `Rainy`, `Cloudy`, two boolean RVs, and `Season`, an RV that can take four values $\{\text{Summer}, \text{Autumn}, \text{Winter}, \text{Spring}\}$. One may want to assess $P(\texttt{Rainy} = \text{True} \mid \texttt{Cloudy} = \text{True}, \texttt{Season} = \text{Summer})$, that is, the probability of today being a rainy day given that it is cloudy and that we are in the summer.

However, in order to be used, a model has to be built, in a process called learning. In short, there are two possible paths to model construction – a manual rationally derived, approach and an automatic, data-driven, approach. The first, although time-consuming, may encompass expert knowledge of the underlying system, which can be beneficial to the overall performance of the model. This may be the only approach possible if there is insufficient data, both in terms of quantity and quality. The latter results in an automatically generated model from a given dataset and may harness the computational power of modern computers to find structure in vast chunks of data. Given the amount of data being collected nowadays, this approach, also known as machine learning, is being widely used. Due to the schematic nature of these models, automatically discovered graph topologies could reveal interesting relationships between RVs that might be previously unknown.

In this chapter, two PGMs – Bayesian networks and Dynamic Bayesian networks – are defined, and some automatic learning algorithms for both are described. There is also an overview of the applications of these models. To conclude, Bayesian multinets are also introduced as a way to perform classification tasks using Bayesian networks.

3.2 Bayesian networks

3.2.1 Definition

A BN is a representation of a joint probability distribution over a set of random variables using a directed graph [13]. It can be defined as a triple $B = (\boldsymbol{X}, G, \boldsymbol{\theta})$, where $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$ is a vector of random variables, $G = (\boldsymbol{X}, E)$ is a directed acyclic graph (DAG) which nodes are random variables, and the edges encode dependencies between them and $\boldsymbol{\theta}$ is a set of parameters that quantify the conditional probability distributions (CPDs) of the network.

Let $pa(X_i)$ denote the parent nodes of the random variable X_i as defined in G. The joint probability distribution is then defined by the network B as

$$P_B(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P_B(X_i \mid pa(X_i)). \tag{3.1}$$

Conditional probability distributions

The structure G of the Bayesian network uniquely defines the joint probability over all random variables. However, how each conditional probability distribution is computed remains open. One common modeling choice is to impose that each random variable results from a linear combination of their parents' value (with prior probabilities defined for nodes without parents) summed with error terms, assumed to be white. This results in Gaussian Bayesian networks [14, 15].

Another common way, which this thesis is going to focus, is assuming a categorical distribution and using tables to parameterize the CPDs. This leads to discrete Bayesian networks, defining $\boldsymbol{\theta} = \{\theta_{ijk}\}$, where $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, q_i\}$ and $k \in \{1, \ldots, r_i\}$. The parameters θ_{ijk} are defined by

$$\theta_{ijk} = P_B(X_i = x_{ik} \mid pa(X_i) = \boldsymbol{w_{ij}}), \tag{3.2}$$

and specify the probability of X_i assuming the value x_{ik}, given a parent configuration $\boldsymbol{w_{ij}}$. Accordingly,

q_i is the number of different configurations for the parents of X_i, and r_i is the number of different values that X_i takes in the dataset. Note that, this way, the parameters have to be specified for every possible case, and there is no formula defined to obtain them. This leads to powerful modeling capabilities, given that no structural equations need to be defined, *i.e.*, there is no need for a way of expressing in a formula a random variable X_i in terms of its parents $pa(X_i)$.

Conditional independences

According to Equation (3.1), it can be noted that the DAG implies independence statements between variables: X_i is independent of every non-descendant node, given its parents' configuration, *i.e.*,

$$\forall_{X_i, X_j \in \boldsymbol{X} \,:\, i \neq j} X_j \notin des(X_i) \Rightarrow X_j \perp\!\!\!\perp X_i \mid pa(X_i), \tag{3.3}$$

where $des(X_i)$ is a set of all the descendant nodes of X_i. Also, in [13], a more general criterion to assess conditional independences guaranteed by the graph structure is introduced. It is called d-separation.

Let $X_i \rightleftharpoons \ldots \rightleftharpoons X_j$ be a trail between X_i and X_j in G. Let also E be a subset of observed nodes in the trail. This trail is active if both these conditions are verified:

- Whenever a structure $X_{k-1} \rightarrow X_k \leftarrow X_{k+1}$ (called a v-structure) is present, then X_k or one of its descendants belong to E.

- There is no other node in E that belongs to the trail.

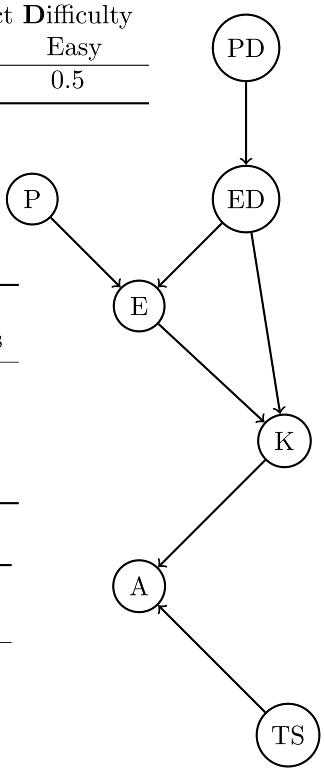
Let X, Y, E be three non-overlapping sets of nodes in G. X and Y are d-separated in G given some evidence E, *i.e.*, d-sep$_G(X; Y \mid E)$, if there is no active trail between any two nodes, one in X and one in Y, given E. Note that d-separation implies conditional independence, although two variables may not be d-separated and still be independent due to a particular configuration of the parameters set θ.

Example

Suppose that an Algorithms and Data Structures student is trying to predict whether he will pass or fail the subject's final exam. By speaking with older students, he learned that the project deeply influences the exam's contents, and he knows he is better prepared if graphs algorithms are the main content. The student may use the Bayesian network of Figure 3.1 to infer the probability of approval (A) given the project's contents (P), difficulty (PD), the time he will spend studying (TS), and his *a priori* knowledge of the subject (K). This model allows him to vary the different parameters, like time studying and can help him decide to use his free time to study for the subject or do some other (productive) work. Similarly, a model that predicts a patient's response to treatment can help the physician prescribe the correct treatment (or the one with the highest probability of success).

The shown example Bayesian network defines the joint probability distribution over its variables

$$P(\mathrm{P}, \mathrm{PD}, \mathrm{E}, \mathrm{ED}, \mathrm{K}, \mathrm{A}, \mathrm{TS}) = P(\mathrm{P})\, P(\mathrm{PD})\, P(\mathrm{ED} \mid \mathrm{PD})\, P(\mathrm{E} \mid \mathrm{P}, \mathrm{ED})\, P(\mathrm{K} \mid \mathrm{E}, \mathrm{ED})\, P(\mathrm{A} \mid \mathrm{K}, \mathrm{TS}). \tag{3.4}$$



Project **D**ifficulty	
Hard	Easy
0.5	0.5

Project	
Graphs	Trees
0.5	0.5

Exam **D**ifficulty		
PD	Hard	Easy
Hard	0.2	0.8
Easy	0.9	0.1

Exam Contents			
P	ED	Graphs	Trees
Graphs	Hard	0.2	0.8
Graphs	Easy	0.4	0.6
Trees	Hard	0.2	0.8
Trees	Easy	0.4	0.6

Knowledge			
E	ED	Abundant	Scarce
Graphs	Hard	0.65	0.35
Graphs	Easy	0.9	0.1
Trees	Hard	0.2	0.8
Trees	Easy	0.4	0.6

Approval			
TS	K	Yes	No
Long	Abundant	0.98	0.02
Long	Scarce	0.75	0.25
Short	Abundant	0.70	0.30
Short	Scarce	0.4	0.6

Time **S**tudying	
Long	Short
0.3	0.7

Figure 3.1: A simple Bayesian network example that allows predicting whether an Algorithms and Data Structures student will be approved in the course. This network defines a joint probability distribution over seven random variables: Project Difficulty (PD), Exam Difficulty (ED), Project Contents (P), Exam Contents (E), Knowledge (K), Approval (A), and Time Studying (TS). The edges in the structure constrain independence between random variables, and the parameters of the network are specified using conditional probability distribution tables, shown here next to each node.

This formula and the conditional probability distributions associated with the Bayesian network allow computing complex queries, using only the conditional probability formula and the law of total probability. For example, a possible query is

$$P(\mathrm{A} \mid \mathrm{P}, \mathrm{PD}, \mathrm{TS}) = \frac{P(\mathrm{A}, \mathrm{P}, \mathrm{PD}, \mathrm{TS})}{P(\mathrm{P}, \mathrm{PD}, \mathrm{TS})},$$

where both the numerator and denominator can be obtained by marginalizing the joint probability in Equation (3.4) appropriately.

3.2.2 Learning

There are two steps in learning a Bayesian network – learning the structure of the G graph and learning the parameters set $\boldsymbol{\theta}$.

Parameters learning

The parameters can be estimated using a MLE approach. The analysis hereby described follows Koller [16] closely. Given a structure graph G and a dataset D with M entries, the likelihood of the parameter set $\boldsymbol{\theta}$ is defined by

$$L(\boldsymbol{\theta} \mid D) = \prod_{m=1}^{M} \prod_{i=1}^{n} P(x_{im} \mid pa(X_i) = \boldsymbol{w_{im}} \mid \boldsymbol{\theta}), \tag{3.5}$$

where m is a dataset entry, x_{im} and $\boldsymbol{w_{im}}$ are the values of the RV X_i and of the configuration of X_i's parents, respectively, in that dataset entry. Equation (3.5) results from the factorized distribution defined by the graph structure, as introduced by Equation (3.1).

Therefore, due to the factorization imposed by the network structure, the likelihood function can be decomposed in terms that only depend on a single RV and its parents. Then, the likelihood is

$$L(\boldsymbol{\theta} \mid D) = \prod_{i=1}^{n} L_i(\boldsymbol{\theta} \mid D) = \prod_{i=1}^{n} L_i(\boldsymbol{\theta}_{X_i|pa(X_i)} \mid D), \tag{3.6}$$

because each local likelihood function L_i is specified by a small subset of $\boldsymbol{\theta}$ concerning X_i and $pa(X_i)$. The global likelihood function is maximized if every local likelihood function is maximized because each local likelihood function is not influenced by any other.

Recalling the table parameterization used for the CPDs, the local likelihood function regarding an RV X_i is given by

$$L_i(\boldsymbol{\theta} \mid D) = \prod_{m=1}^{M} P(x_{im} \mid pa(X_i) = \boldsymbol{w_{im}} \mid \boldsymbol{\theta}_{X_i|pa(X_i)}) = \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} \theta_{ijk}^{N_{ijk}}, \tag{3.7}$$

where N_{ijk} is the number of times a particular configuration of $X_i = x_{ik}$ and $pa(X_i) = \boldsymbol{w_{ij}}$ appears in the dataset. Maximizing this function is the same as maximizing $\log L_i(\boldsymbol{\theta} \mid D)$. Aditionally, the resulting parameters must sum to one, for a given parent configuration, *i.e.*, $\sum_{k=1}^{r_i} \theta_{ijk} = 1$. Therefore, optimization can be achieved using the Lagrangian multiplier method, resulting in maximizing the function

$$f(\theta_{ijk}, \lambda) = \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \theta_{ijk} + \lambda \left(-1 + \sum_{k=1}^{r_i} \theta_{ijk} \right), \tag{3.8}$$

both in order to θ_{ijk} and λ. This concludes in the maximum likelihood estimator for the parameter,

$$\hat{\theta}_{ijk} = \frac{N_{ijk}}{\sum_{k=1}^{r_i} N_{ijk}} = \frac{N_{ijk}}{N_{ij}}, \tag{3.9}$$

where N_{ij} is the number of times a particular configuration of $pa(X_i) = \boldsymbol{w_{ij}}$ appears in the dataset, disregarding the value of the RV X_i itself.

Structure learning

There are three main categories of structure learning algorithms – constraint based, score based and hybrid.

The first methods try to estimate the conditional independence between random variables from data using statistical hypothesis testing. The PC algorithm [17] is considered state-of-the-art in this technique [18]. It starts with a fully connected graph and removes edges between RVs, as independences are established.

The second category, which is more used, relies on a score function that evaluates the fit of a structure to the data used. The strategy for training is to search for the best structure, given a particular score, from the space of all possible structures, therefore solving the optimization problem

$$\begin{aligned} \underset{G}{\text{maximize}} \quad & \phi(G \mid D) \\ \text{subject to} \quad & G \text{ is a DAG}, \end{aligned} \tag{3.10}$$

where ϕ is a scoring function, G is the structure of the network, and D is a dataset with observations of the process to model. Note that the constraint imposes that the problem is non-convex and, unfortunately, this is an NP-hard problem [19], and the search space grows superexponentially ($2^{\mathcal{O}(n^2)}$) with the node count [20]. This is the general case, even if a restriction on the number of possible parents of a node is imposed. Only if the structure is forced to be a tree, *i.e.*, a network allowing only one parent for each node, worst-case polynomial algorithms [21] are possible. Due to the computational complexity of the problem, heuristic methods are usually applied to restrict the search space, and only locally optimal solutions are found.

The hybrid methods combine these methodologies and usually work by restricting the search space based on a measure of independence between nodes. One example is the sparse candidate algorithm [22] that restricts the parents of a particular RV to a set of most relevant nodes using an information theory approach and then uses search to find a locally optimal structure. Also worth noting, the max-min hill climbing (MMHC) algorithm [23] tries to obtain a skeleton of the Bayesian Network, *i.e.*, a partially directed graph that encodes possible associations between random variables, using statistical tests for RVs independence. These partial DAGs may contain undirected edges. Afterward, a greedy hill-climbing step is run, but only using the edges that belong to the skeleton. The authors claim that this method obtains better results in practice than the PC and sparse candidate.

Score-based structure learning

To solve the optimization problem in Equation (3.10), a score and a heuristic search procedure must be defined.

A basic score metric that can be considered is the log-likelihood (LL). Estimating the parameters using their MLE estimation as shown in Equation (3.9), the likelihood formula of Equation (3.5) can be used as a comparison measure. Therefore, the log-likelihood score of a structure G for a given dataset D is defined as

$$\phi_{\mathrm{LL}}(G, D) = \log L(G \mid D) = \sum_{m=1}^{M} \sum_{i=1}^{n} P(x_{im} \mid pa(X_i) = \boldsymbol{w_{im}}) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}}. \tag{3.11}$$

A common problem using the log-likelihood score is overfitting since it usually returns a graph with all the nodes connected that explains the given data exceptionally well but fails to generalized to unseen data. This is unwanted behavior, and therefore a new score that penalizes complex network structures was introduced based on model selection criteria. The minimum descriptor length (MDL) [24] metric is defined by

$$\phi_{\mathrm{MDL}}(G, D) = \phi_{\mathrm{LL}}(G, D) - \frac{1}{2}|B| \log M, \tag{3.12}$$

where $|B|$ is the number of elements in the set $\boldsymbol{\theta}$ and is given by

$$|B| = \sum_{i=1}^{n} (r_i - 1) q_i. \tag{3.13}$$

One common way to conduct the searching procedure is to use greedy hill-climbing (Algorithm 1). Starting with an empty structure (a graph with no edges), each iteration executes the operation leading to the best scoring function increment. Typically, three operations are considered possible: edge addition, edge removal, or edge reversal. Naturally, since the interest is finding an acyclic graph, operations that would result in a cycle are not allowed. The algorithm stops when, for example, no edge addition results in a positive increment. Other stopping criteria are possible.

Algorithm 1 Greedy hill-climbing

```
Input: A dataset $D$ and a scoring function $\phi$
Output: A local optimal structure $G$
loop
    $S^* \leftarrow -\infty$
    for all edge addition, removal and reversals resulting in a DAG do
        $G' \leftarrow$ result of applying operation to $G$

        if $\phi(G') - \phi(G) > S^*$ then
            $S^* \leftarrow \phi(G') - \phi(G)$
            $G^* \leftarrow G'$
        end if
    end for

    if $S^* < 0$ then
        break
    end if

    $G \leftarrow G^*$
end loop
```

This method is a heuristical searching procedure, and therefore the solution obtained is not to the global optimum. There are strategies to try to improve the quality of the solution and try to escape local maxima.

Tabu search [25] applied after hill-climbing attempts to do this by moving to a worse neighbor of the solution and then putting the original solution in a tabu list, therefore not allowing the hill-climbing algorithm to circle back. Then, multiple steps can be performed forbidding the reversal of the operations applied until some stopping criterion is met. Note that the resulting structure is not necessarily a local optimum. The solution is the best structure visited during all the searching procedure. Given that this forces different structures to be considered, the searching procedure may be guided to unexplored regions of the searching space and to obtain possibly better solutions.

Another way to try to escape local maxima is by applying random restarts. After a solution is found, this method consists of performing some random operations on the structure and then do hill-climbing again. This can be repeated several times, keeping the best structure visited.

Simulated Annealing [26] can also be applied to Bayesian network learning [27]. Inspired by physical processes, this method consists of several moves to a neighbor of a given structure. If the move considered improves the score, it is always accepted. Otherwise, it is accepted with probability $p = \exp\left(\frac{\phi(G')-\phi(G)}{T}\right)$, where G is the current structure, G' is the structure after the operation is applied and T is a temperature parameter. As the searching procedure is carried, this temperature decays over time, reducing the probability of an operation that decreases the score to be performed. Eventually, a local optimum is reached, and with $T = 0$ this method converges since the probability of picking a neighbor with a worse score also becomes $p = 0$.

Trying to improve the greedy hill-climbing search efficiency, Chickering [28] proposes a different way to conduct a search by moving on the search space of equivalence classes instead of DAGs. The author defines a complete set of operators to move directly between those classes. Two DAGs are said to be equivalent if the set of probability distributions that can be encoded by one of these structures is the same as the set that can be represented using the other one. An equivalence class is a set of all possible equivalent DAGs for a given structure. If the scoring function used is score equivalent, then structures belonging to the same class are indistinguishable and, therefore, comparing one to the other is a waste of computing resources. This algorithm outputs a partially directed acyclic graph that can be arbitrarily directed to form the local optimal solution.

More recently, ℓ_1-norm based optimization, in the context of neighborhood selection [29], was applied to structure learning in an attempt to obtain sparse Bayesian networks [30]. Regularization terms are useful and may prevent overfitting since a network with fewer edges but with similar (or even better) modeling accuracy should lead to a better generalization. For Gaussian Bayesian networks, the structural equations, *i.e.*, how a random variable is expressed in terms of its parents, are by hypothesis linear equations. Each random variable is given by a linear combination of its parents and some error terms. This implies that in a network with n nodes, for each node, there are at most $n-1$ parameters. Thus, a simple ℓ_1 penalty can be added to the cost function for each parameter, causing some of them to be set exactly to zero. Given that the DAG space is non-convex, in Schmidt *et al.* [30] the authors propose to find a skeleton estimate for the DAG (similar to MMHC) using convex continuous optimization. The graph is then directed to form a DAG and define a joint probability. To apply this method directly to discrete Bayesian networks, a penalization term should be added for every parameter. However, the number of possible parameters for a given node grows exponentially with the number of possible parents ($\mathcal{O}(r^{n-1})$, where r is the number of levels that a node can take). In fact, a given node has, for each possible level, different parameters, as seen in Equation (3.13). To overcome this, the author proposes applying the method only to binary discrete Bayesian networks and approximating the multinomial distribution with logistic regression. This way, only the structure is learned since the multinomial parameters cannot be directly trained. The authors also suggest that this method can be extended to general discrete networks by using multinomial logistic regression and group ℓ_1-norm penalty. This work is later consolidated by Schmidt [31].

Building on this work, Fu et. al [32] propose a coordinate descent algorithm to learn Gaussian Bayesian networks using ℓ_1-norm regularization. Instead of first trying to learn a skeleton and then directing the graph, the authors propose a heuristical searching procedure that simultaneously fits the linear regression and finds the structure. This can be achieved given that the coordinate descent algorithm performs minimization one coordinate at a time, which allows acyclicity to be enforced. The authors claim that this heuristic can save computation time and still obtain satisfactory results, allowing this method to be applied to structure estimation in high-dimensional contexts.

Using the same search procedure, Gu et al. [33] extend this approach to discrete Bayesian networks, approximating the multinomial distribution with multinomial logistic regression using one-hot encoding. The likelihood is then penalized using a group norm penalty, resulting in a sparse structure because this term prunes some edges.

These last two methods are both a part of an R package dedicated to learning Bayesian networks in high dimensional data contexts called `sparsebn` [34]. The authors claim that their software can handle graphs with more than 1000 nodes (on their Github page, it is mentioned 8000 nodes) with superior performance when compared to several other standard software packages.

Also applying continuous optimization procedures to learning Bayesian networks, Zheng *et al.* [35] propose a new mathematical characterization for acyclicity in directed graphs using matrix exponentials. This leads to a smooth and differentiable function that allows characterizing "DAG-ness" and to be used directly in continuous optimization using gradient techniques. Although this problem remains non-convex, the authors claim that this method gives reasonable solutions. Possibly, this allows for new gradient optimization techniques that are being heavily studied due to the rise of deep learning to be applied to Bayesian network learning. However, they only apply this method to Gaussian Bayesian networks, also adding a penalization term. No extension to discrete Bayesian networks was found during the preparation of this thesis.

3.2.3 Applications

Due to their versatility, Bayesian networks are applied across a wide variety of areas, and they can be seen used from analyzing clinical data to decision making in entirely different fields. The bioinformatics use is evident due to the explicability of the models obtained and the possibility to include expert knowledge in the models. This faces particularly well with black box models like neural networks and random forests.

In studies by Loghmanpour et al. [36], Bayesian network based classifiers are trained using pre-implant clinical data and used to predict short and long-term mortality in patients with left ventricular assistant devices. The authors showed that these methods perform very well compared to standard classification metrics used to predict survival in these patients.

Also using clinical data, Seixas *et al.* [37] use BNs to diagnose Alzheimer's disease and other similar illnesses. Those networks have structures developed using expert knowledge, but the parameters were estimated from data. They compared the performance to other classifiers and obtained competitive results. A network with an automatically learned structure was also compared, but the results were very similar to the ones using the manual structure.

Employing structure learning in epidemiological data, [38] uses these networks to analyze the risk of cardiovascular disease and of metabolic syndrome development. The algorithms identify several interesting relationships, including a strong influence of smoking habits and physical activity in these risks.

Regarding other fields, they were used to determine if vessels should be inspected by the maritime authorities in a port [39].

3.3 Dynamic Bayesian Networks

3.3.1 Definition

Dynamic Bayesian Networks are an extension of Bayesian networks used to model temporal processes. Like Bayesian networks, these probabilistic graphical models define a joint probability distribution over a set of random variables. However, these variables are assumed to change over time due to an underlying process and are observed in several discrete time instants.

They are defined by a pair $D = (B_0, B_{\rightarrow})$ [40]. Let $\boldsymbol{X}(t) = (X_1(t), X_2(t), \ldots, X_n(t))$ denote a vector of the random variables at time $t \in [0, T]$, also known as a slice. B_0 is a Bayesian network defined over $\boldsymbol{X}(0)$, called the prior network. $B_{\rightarrow}$ is another Bayesian network defined over $\boldsymbol{X}(0:T) = \boldsymbol{X}(0) \cup \boldsymbol{X}(1) \cup \ldots \cup \boldsymbol{X}(T)$, called the transition network. In the transition network, it is always true that,

$$\forall_{i \in \{1,\ldots,n\}} \forall_{t \in \{0,\ldots,T\}} P(X_i(t) \mid \boldsymbol{X}(0:T)) = P(X_i(t) \mid \boldsymbol{X}(0:t)), \tag{3.14}$$

i.e., a random variable realization must not be dependent on future realizations.

Markov property

The underlying stochastic process is said to be first-order Markovian if future values of the random variables only depend on the present values. Therefore,

$$P(\boldsymbol{X}(t+1) \mid \boldsymbol{X}(0:t+1)) = P(\boldsymbol{X}(t) \mid \boldsymbol{X}(t:t+1)). \tag{3.15}$$

This means that in our transition network, nodes that belong to a timestep $t+1$ can only have parents from that timestep or from nodes belonging to the previous timestep t. This property may be generalized by the so-called higher-order Markovian processes. In these, random variables realizations may depend on several past values. The number of previous timesteps allowed is fixed, and it is called the Markov lag of the process. Let υ be the Markov lag of an arbitrary process, then

$$P(\boldsymbol{X}(t+1) \mid \boldsymbol{X}(0:t+1)) = P(\boldsymbol{X}(t+1) \mid \boldsymbol{X}[t-(\upsilon-1):t+1]). \tag{3.16}$$

Stationarity

A random process is said to be stationary if $P(\boldsymbol{X}(t) \mid \boldsymbol{X}(0:t))$ is the same for all t. This assumption allows extrapolating the probability distribution of future timesteps by "unrolling" the network. This is achieved by repeating the transition network starting at each timestep until a future time instant is reached.

3.3.2 Learning

Similar to Bayesian networks, the network parameters are easily learned once a network structure has been found. Therefore, the hard part is to find the structure that best explains the data. Inference algorithms used for BNs can also be applied to DBNs once a model has been determined.

In general, DBN structure learning is as hard as in BNs, because acyclicity has to be ensured in the intra-slice connections. Inter-slice connections can't generate cycles due to the constraint in Equation (3.14). If intra-slice edges are ignored, then DBNs can be learned in polynomial time [41, 42]. Some work has also been done considering that intra-slice structure does not change between timesteps [43]. This simplifies the learning process since only one intra-slice structure has to be learned, and the remaining inter-slice dependencies can be found using efficient algorithms. Monteiro *et al.* [44] introduce a polynomial-time algorithm for DBNs (`tDBN`) was achieved, restricting the intra-slice connections to a tree (just one parent) and admitting only a fixed number of parents from previous slices. In the work by Sousa *et al.* [45], the intra-slice restriction was relaxed, but an ordering for the variables was obtained using the `tDBN` algorithm. Leão *et al.* [46] improved further on the tree-restricted method, allowing training with static variables, *i.e.*, that don't change over timesteps.

For general dynamic Bayesian networks, score+search strategies can be used [40], as discussed for Bayesian networks, pruning the search space in order to impose the necessary restrictions.

3.4 Bayesian Multinets

3.4.1 Definition

Let $P(X_1, \ldots, X_N)$ be a probability distribution and $H_1, \ldots, ..., H_k$ be a collection of mutually disjoint sets of realization of the random variables in P. A Bayesian multinet is a set of k Bayesian networks, where each network B_i is a comprehensive local network associated with H_i, *i.e.*, a Bayesian network of $P(X_1, \ldots, X_N \mid H_i)$ [47]. This multinet allows the definition of a joint probability distribution

$$P(X_1, \ldots, X_N) = \sum_{i \in \mathcal{B}} P_{B_i}(X_1, \ldots, X_N \mid H_i)\, P(H_i), \tag{3.17}$$

where $\mathcal{B} = \{i, \ldots, k : H_i \subseteq \mathbf{X}\}$, $\mathbf{X} = \{X_1, \ldots, X_N\}$.

These models were introduced in order to allow for asymmetrical independences to be expressed and therefore allow for even more sparse networks that can benefit computational complexity. Since fewer edges may be selected, this could also benefit learning. Asymmetrical independence is an independence constraint that only occurs when some hypothesis is assured.

Example

To illustrate the concept of asymmetrical independences, Geiger *et al.* [47] introduce the example of a guard that has to distinguish between workers, visitors, and spies entering a secured building. When

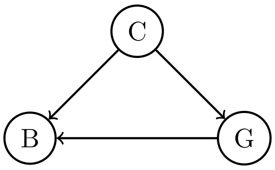


Figure 3.2: The structure of a Bayesian network that can detect whether a person entering a secured building is a spy, visitor or a worker. This network is defined over 3 random variables: the Class (C) may be spy, visitor or worder; Gender (G) may be male or female; Badge (B) may take the values wearing or not wearing. (Source: [47])

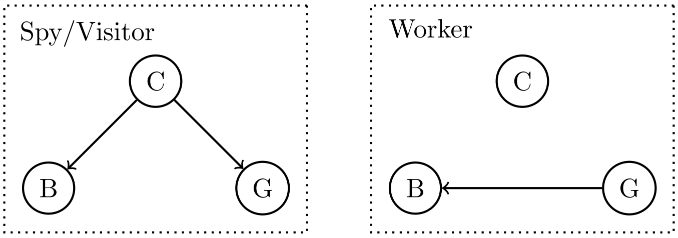


Figure 3.3: A schematic representation of a Bayesian multinet that can detect whether a person entering a secured building is a spy, visitor, or worker. This set of networks is defined over three random variables: the Class (C) may be spy, visitor, or worker; Gender (G) may be male or female; Badge (B) may take the values wearing or not wearing. (Source: [47])

someone enters the building, the guard may notice two particularities – the person's gender and whether the person is wearing a badge. Visitors do not have a badge, so they never wear one. Spies always wear a badge because they do not want to be uncovered. Workers sometimes wear a badge (when they do not forget it at home), and female workers tend to wear it more often than males. Figure 3.2 represents the structure of a Bayesian network that may be used by the guard to compute the probability of a spy entering the building. Note that wearing a badge is independent of gender, given that the person is a spy. Using only this structure, such independence is not guaranteed and must be assured by specific parameter values.

In Bayesian multinets, the structure of the network is allowed to vary based on a value of a specific random variable. Figure 3.3 shows a multinet that can be used by the guard. Note that the independence between gender and badge can now be established via network topology when someone is guaranteed to be a spy or a visitor.

3.4.2 Classification

Bayesian multinets can be used to perform classification of given data [48]. In the classification task, the probability of a class given some evidence has to be estimated. Let C be the class random variable of some realization, and $X_1, \ldots, X_N$ be N random variables that correspond to the feature variables used in the classification. To classify the observation, the probability $P(C \mid X_1, \ldots, X_N)$ must be computed, and the class is the realization of C that has the highest probability.

To achieve this with Bayesian multinets, in a supervised learning context, a Bayesian network must be learned for each class, using a training dataset. The dataset must be split by class, and for each group, automatic techniques for learning Bayesian networks (like some discussed in Section 3.2.2) can be used to learn a network over the features $X_1, \ldots, X_N$. Note that, using this approach, the class C is assumed to be independent of the features in each network. According to Equation (3.17), the multinet defines the joint distribution

$$P(C = c_i, X_1, \ldots, X_N) = P(C = c_i)\, P_{B_i}(X_1, \ldots, X_N),$$

where B_i is the Bayesian network learned for class c_i. This distribution may then be used to compute each class probability. For a dataset with k classes,

$$P(C = c_i \mid X_1, \ldots, X_N) = \frac{P(C = c_i, X_1, \ldots, X_N)}{\sum_{l=1}^{k} P(C = c_l, X_1, \ldots, X_N)}. \tag{3.18}$$

Summing up, classification using a learned Bayesian multinet is done by computing the joint probability assuming each class is true and choosing the class that has the highest probability.

Chapter 4

The sDBN method

Extending the work of Schmidt *et al.* [30] and Gu *et al.* [33] to include temporal information, this thesis proposes a three-step method (Figure 4.1) called `sDBN` to train sparse dynamic Bayesian networks leveraging the approximate problem of finding a graph skeleton using a group ℓ_1-norm penalized multinomial logistic regression loss function. First, edge penalization weights are estimated by fitting an unregularized multinomial logistic loss to the dataset, as suggested by the adaptative lasso method. Then, a sparse skeleton, a set of possible edges for our network, is estimated from our dataset, penalizing every possible edge using the weights computed in the previous step. Finally, a resulting network is obtained by directing the skeleton using greedy hill-climbing, complying with the temporal restrictions.

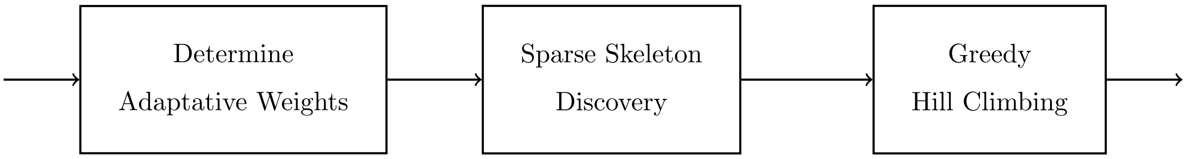


Figure 4.1: Schematic of the pipeline of the proposed method for training dynamic Bayesian networks.

4.1 Network Parametrization

In the first two steps of the training pipeline, a parametrization based on multinomial logistic regression is used to reduce the dimension of the parameter set and enable gradient based optimization with regularization, promoting sparse structures and enabling training in high-dimensional contexts.

Let $B = (\mathbf{X}, G, \boldsymbol{\beta})$ be a discrete Bayesian network, where $\mathbf{X} = (X_1, \ldots, X_n)$ is a vector of discrete random variables, G is any given DAG and $\boldsymbol{\beta}$ is a set of parameters. It is assumed, without loss of generality, that a discrete random variable X_j can take a value in the set $\{0, \ldots, r_j - 1\}$. In order to increase the logistic model flexibility, each random variable X_j is encoded by a set of $r_j - 1$ dummy variables that can take only two possible levels and work in an one-hot enconding configuration, that is, if one of them takes the value 1 the others are necessarily 0; the variable set to 1 determines the level of X_j, being 0 if there are no variables set. For convenience, let $d_i = r_i - 1$. Using this transformation, in total there are now $c = \sum_{j=1}^{n}(r_j - 1) = \sum_{j=1}^{n} d_j$ random variables. In this context, a realization of the vector of random variables of the network can be considered itself a vector $\mathbf{x} = (x_{0,1}, \ldots, x_{0,d_0}, \ldots, x_{n,1}, \ldots, x_{n,d_n}) \in \{0,1\}^c$, where $x_{a,b}$ is the dummy variable associated with the level b of the random variable X_a. A dataset is a collection of N such vectors.

Let X_j denote a random variable that belongs to our network. Let $\boldsymbol{\beta_{jli}} \in \mathbb{R}^{d_i}$ denote the vector of parameters associated with the influence of the value of X_i in the level l of X_j. If the multinomial logistic regression is used to parameterize the probability distribution of this node then

$$P(X_j = l \mid pa(X_j)) = \frac{\exp(\bar{\mathbf{x}}^\top \boldsymbol{\beta_{jl\cdot}})}{\sum_{m=1}^{r_j} \exp(\bar{\mathbf{x}}^\top \boldsymbol{\beta_{jm\cdot}})}, \tag{4.1}$$

where $\bar{\mathbf{x}} = \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix}^\top \in \mathbb{R}^{c+1}$ is an augmented evidence vector and $\boldsymbol{\beta_{jl\cdot}} = \begin{bmatrix} \boldsymbol{\beta_{jl0}}^\top & \boldsymbol{\beta_{jl1}}^\top & \cdots & \boldsymbol{\beta_{jln}}^\top \end{bmatrix}^\top \in \mathbb{R}^{c+1}$ is the parameter vector associated with level l of X_j. Note that this vector is built by stacking the components of $\boldsymbol{\beta_{jli}}$, because those are not scalars due to the one-hot encoding of the dataset. Using vectors enables a different parameter for each level of the parent random variable, increasing the versatility of the multinomial logistic regression approximation to the categorical distribution. $\beta_{jl\,0}$ is a scalar intercept term. To address the identifiability issues of the parameters, as discussed for regular multinomial logistic regression in Chapter 2, it is enforced $\beta_{j\,0\,0} = 0$.

Let $\boldsymbol{\beta_{j\cdot i}} = \begin{bmatrix} \boldsymbol{\beta_{j0i}}^\top & \boldsymbol{\beta_{j1i}}^\top & \cdots & \boldsymbol{\beta_{jd_ji}}^\top \end{bmatrix}^\top \in \mathbb{R}^{r_j d_i}$, be the vector of parameters of the edge $X_i \to X_j$. Analyzing the multinomial approximation, observe that

$$X_i \notin pa(X_j) \Leftrightarrow \boldsymbol{\beta_{j\cdot i}} = \mathbf{0}, \tag{4.2}$$

from the conditional of the probability in Equation (4.1). This equivalence allows determining a sparse graphical structure from the set of parameters if many are set exactly to zero.

Example

Consider the simple Bayesian network present in Figure 4.2. Attending to Equation (4.1) and the equivalence in Equation (4.2), the following equalities hold:

$$\boldsymbol{\beta_{2\,0\,\cdot}} = \begin{bmatrix} 0 & \beta_{2\,0\,1} & 0 & \beta_{2\,0\,3} & 0 \end{bmatrix}^\top,$$

$$P(X_2 = 0 \mid X_1 = x_1, X_3 = x_3) = \frac{\exp(\beta_{2\,0\,1} \cdot x_1 + \beta_{2\,0\,3} \cdot x_3)}{\exp(\beta_{2\,0\,1} \cdot x_1 + \beta_{2\,0\,3} \cdot x_3) + \exp(\beta_{2\,1\,0} + \beta_{2\,1\,1} \cdot x_1 + \beta_{2\,1\,3} \cdot x_3)},$$

$$P(X_2 = 1 \mid X_1 = x_1, X_3 = x_3) = \frac{\exp(\beta_{2\,1\,0} + \beta_{2\,1\,1} \cdot x_1 + \beta_{2\,1\,3} \cdot x_3)}{\exp(\beta_{2\,0\,1} \cdot x_1 + \beta_{2\,0\,3} \cdot x_3) + \exp(\beta_{2\,1\,0} + \beta_{2\,1\,1} \cdot x_1 + \beta_{2\,1\,3} \cdot x_3)}.$$

Likelihood of a given structure

Following Equation (3.5), the likelihood of a given parameter set $\boldsymbol{\beta}$ for a given dataset D can be defined as

$$\mathcal{L}(\boldsymbol{\beta} \mid D) = \prod_{m=1}^{M} \prod_{i=1}^{n} P(X_i = x_i^{(m)} \mid x^{(m)}, \boldsymbol{\beta_{i\cdot\cdot}}), \tag{4.3}$$

where $x^{(m)}$ is the m-th evidence vector from the dataset and $x_i^{(m)}$ denotes the observed value of the

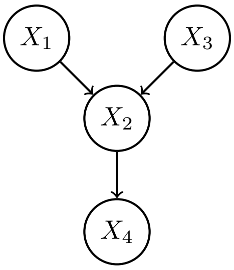


Figure 4.2: A small Bayesian network to illustrate the multinomial logistic regression parametrization. This network is defined over four random variables $X_1, \ldots, X_4$ and contains only three edges. Every random variable is discrete and can take only two levels, 0 or 1.

random variable X_i. Using the parametrization of Equation (4.1) and taking the logarithm, the log-likelihood of the parameter set expands to

$$\ell(\boldsymbol{\beta} \mid D) = \sum_{m=1}^{M} \sum_{i=1}^{n} \left[\sum_{l=0}^{d_i} I\left(x_i^{(m)} = l\right) \cdot \left(\bar{\boldsymbol{x}}^{(m)}\right)^\top \boldsymbol{\beta_{il\cdot}} - \log \left\{ \sum_{l=0}^{d_i} \exp\left(\left(\bar{\boldsymbol{x}}^{(m)}\right)^\top \boldsymbol{\beta_{il\cdot}} \right) \right\} \right]. \tag{4.4}$$

Since the parameter set defines a graphical structure via the equivalence in Equation (4.2), this likelihood can be used in a loss function specifically constructed to provide us with a sparse structure using group LASSO regularization. In this specific case, the groups considered are the $\boldsymbol{\beta_{j\cdot i}}$, *i.e.*, the parameters associated with an edge $X_i \to X_j$. Due to the variable selection properties of group LASSO, many of these groups will be set to exactly 0 after the optimization, eliminating an edge and thus providing a sparse graph that can be used to discover the network's structure. Note that restricting this optimization to the space of DAGs is an NP-hard problem. Therefore a relaxed problem is considered instead, allowing cyclicity and outputting a network skeleton instead of a final structure.

4.2 Parametrization and dataset construction in dynamic Bayesian networks

In the context of dynamic Bayesian networks, the dataset used in training has a value for every random variable in each timestep. In non-stationary networks, the number of timesteps in the transition network is equal to the number of measured timesteps in the dataset. That does not happen in stationary networks. Stationary networks imply that edges are kept the same across timesteps. In practice, this is achieved by training just one timestep and then repeating the edges for every timestep needed for modeling, unrolling the network. The original dataset with measurements across all timesteps must be collapsed down to accommodate the needs for stationary network training without loss of information.

Each timestep has a collection of its own parameters $\boldsymbol{\beta}(t)$. The parameter notation from the Bayesian network approximation can be applied directly to DBNs. Then, $\boldsymbol{\beta_{jl\cdot}}(t) \in \mathbb{R}^{cT+1}$ is the vector of coefficients associated with the level l of the random variable $X_j(t)$. Accordingly, and in DBNs introducing a way to identify inter-temporal edges, $\boldsymbol{\beta_{j\cdot i}}(\tau, t)$ denotes the parameters associated with the edge $X_i(\tau) \to X_j(t)$.

Non-stationary networks

For training of non-stationary networks, the evidence vector can be thought of as a collection of static evidence vectors for each timestep, *i.e.*,

$$\mathbf{x} = \begin{bmatrix}\mathbf{x}(0) & \cdots & \mathbf{x}(T)\end{bmatrix} = \begin{bmatrix}x_{0,1}(0) & \cdots & x_{n,r_n-1}(0) & \cdots & x_{0,1}(T) & \cdots & x_{n,r_n-1}(T)\end{bmatrix} \in \{0,1\}^{cT}.$$

To preserve temporal causality (Equation (3.14)), the parameter vector $\boldsymbol{\beta_{jl\cdot}}(t)$ has to be restricted in such a way that forbids parents from future timesteps. Plus, edges disrespecting the Markov lag should also be disregarded. In order to maintain compatibility with the shorthand notation used for the linear combination in the exponential terms in Equation (4.1) and the dynamic evidence vector shown above, this vector should be defined as belonging to $\mathbb{R}^{cT+1}$, as it also includes an intercept term. To fulfill its imposed constraints, the vector is forced to be zero everywhere except when specifying an edge from an allowed parent (in this case, it is allowed to be freely optimized). Thus,

$$\boldsymbol{\beta_{jl\cdot}}(t) = \begin{bmatrix}\beta_{jl0}(t) & 0 & \cdots & \boldsymbol{\beta_{jl1}}(t-v,t) & \cdots & \boldsymbol{\beta_{jln}}(t-v,t) & \cdots & \boldsymbol{\beta_{jl1}}(t,t) & \cdots & \boldsymbol{\beta_{jln}}(t,t) & \cdots & 0\end{bmatrix}, \tag{4.5}$$

where v is the Markov lag of the network.

Stationary Networks

In the specific case of stationary dynamic Bayesian networks, the transition model is assumed to be always the same. To train these kinds of models, it is only needed to determine the intra-temporal edges of one timestep and edges going into that timestep from every possible past instant according to the specified lag. Random variables from other timesteps are added as nodes to the transition network, allowing the formation of inter-temporal edges, but they only act as a stub, are not trained, and are not considered actual network nodes. A valid dynamic Bayesian network is only obtained when unrolling the transition network (joined with the prior network).

The number of timesteps in the evidence vector for these networks is connected with the Markov lag. Since only one real timestep is used for training and intra-temporal edges are only allowed within the lag restriction, the number of timesteps in the evidence vector is always $v+1$. The dataset has to be collapsed in such a way that results in $v+1$ timesteps, and all the information that may lead to an inclusion of an edge remains present. Let $\mathbf{D}$ be a $M \times cT$ matrix obtained by stacking all evidence vectors of the input dataset, *i.e.*,

$$\mathbf{D} = \begin{bmatrix}\mathbf{x}^{(0)} \\ \vdots \\ \mathbf{x}^{(M)}\end{bmatrix} = \begin{bmatrix}\mathbf{x}^{(0)}(0) & \cdots & \mathbf{x}^{(0)}(T) \\ \vdots & \vdots & \vdots \\ \mathbf{x}^{(M)}(0) & \cdots & \mathbf{x}^{(M)}(T)\end{bmatrix}.$$

The associated collapsed dataset with lag v, $\mathbf{D}^{\dagger}$ is a $M(T-v+1) \times c(v+1)$ matrix constructed as

$$\mathbf{D}^{\dagger} = \begin{bmatrix} \mathbf{x}^{(0)}(0) & \mathbf{x}^{(0)}(1) & \cdots & \mathbf{x}^{(0)}(v) \\ \mathbf{x}^{(0)}(1) & \mathbf{x}^{(0)}(2) & \cdots & \mathbf{x}^{(0)}(v+1) \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{x}^{(0)}(T-v) & \mathbf{x}^{(0)}(T-v+1) & \cdots & \mathbf{x}^{(0)}(T) \\ \mathbf{x}^{(1)}(0) & \mathbf{x}^{(1)}(1) & \cdots & \mathbf{x}^{(1)}(v) \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{x}^{(M)}(T-v) & \mathbf{x}^{(M)}(T-v+1) & \cdots & \mathbf{x}^{(M)}(T) \end{bmatrix}.$$

Therefore, the new evidence vector used in the training of the stationary network can be thought of simply as

$$\mathbf{x}^{\dagger} = \begin{bmatrix} x_{0,1}(0) & \cdots & x_{n,r_n-1}(0) & \cdots & x_{0,1}(v) & \cdots & x_{n,r_n-1}(v) \end{bmatrix} \in \mathbb{R}^{c(v+1)}.$$

Note that the timesteps inside the parenthesis correspond to the collapsed dataset pseudo-timesteps and not the original input dataset.

Every parameter vector is forced to zero, except the ones associated with the last timestep. Those are the only ones that need to be optimized and may take any value except, naturally, the parameter associated with the edge to itself, which is set to zero. Therefore,

$$\boldsymbol{\beta_{jl\cdot}}(t) = \begin{cases} 0 & \text{, if } t \neq v \\ (\boldsymbol{\beta_{jl0}}(t), \ldots, \boldsymbol{\beta_{jln}}(t,t)) & \text{, if } t = v. \end{cases} \tag{4.6}$$

4.3 Loss Function

In this section, a loss function will be constructed using the structure log-likelihood derived in Section 4.1. The final result should be a pair of Bayesian networks skeletons that can be directed, resulting in a full dynamic Bayesian network that attempts to uncover the underlying stochastic process and the relationships between random variables from empirical data.

Foremost, our loss is desired to be decomposable, *i.e.*, it can be separated into a sum of components for each random variable that only depends on it and its parents. This decomposition implies that the training of the prior network can be done separately from the transition network since no node in the prior network can admit a parent from the transition network due to the restriction in Equation (3.14). In fact, if our score is decomposable and the acyclicity constraint on the structure is dropped, the parent set for each node can be estimated independently. A negative log-likelihood score has this property.

Secondly, for this method to be applicable to high-dimensional data and to suit better “real” networks arguably with a small number of edges, a sparse structure is preferred. To achieve this, a group lasso regularization term should be applied to the parameters related to every possible edge of the structure, both intra-temporal and inter-temporal, respecting the temporal causality and Markov lag.

Combining them both, regularizing the log-likelihood score with a group lasso penalty, a loss function can be constructed, summing the cost for each timestep, resulting in

$$f(\boldsymbol{\beta}) = \sum_{t=0}^{T} \left[-\frac{1}{M}\ell(\boldsymbol{\beta}(t)) + \sum_{\tau \in W(t)} \sum_{j=1}^{n} \sum_{i=1}^{n} \lambda_{ij}(\tau, t) \|\boldsymbol{\beta_{j \cdot i}}(\tau, t)\| \right], \tag{4.7}$$

where $\ell(\boldsymbol{\beta}(t))$ is a shorthand notation to represent the log-likelihood of a set of parameters associated to the timestep t according to Equation (4.4), M is the size of the dataset, $\lambda_{ij}(\tau, t)$ are the regularization weights, associated with the edge $X_i(\tau) \to X_j(t)$ and $W(t) = \{\tau \in \mathbb{N}_0 : t - \upsilon \leq \tau \leq t\}$ is a set of previous timesteps within the specified Markov lag υ. Since this loss is decomposable, every node – and therefore every timestep – can be seen as a separate optimization problem. The prior network skeleton is obtained by solving the term associated with timestep 0, and the transition network skeleton can be found optimizing the remaining timesteps.

The regularization weights are obtained using the strategy of the so-called adaptative LASSO [49, 50], and constitutes the first step of the proposed pipeline. First, an unregularized optimization problem

$$\underset{\tilde{\boldsymbol{\beta}}(t)}{\text{minimize}} \quad -\frac{1}{M}\ell(\tilde{\boldsymbol{\beta}}(t)), \tag{4.8}$$

is solved for each timestep to find the set of relevant parameters $\tilde{\boldsymbol{\beta}}(t)$. Then, these parameters are used to compute the adaptative regularization weights using the expression

$$\lambda_{ij}(\tau, t) = \lambda \frac{1}{\|\tilde{\boldsymbol{\beta}}_{\boldsymbol{j \cdot i}}(\tau, t)\|^{\gamma}},$$

where λ and γ are hyper-parameters. λ may act as a tuning parameter and specifies "how much" regularization is applied. Larger lambdas lead to sparser structures, and smaller values result in denser skeletons. If no regularization is applied, the full possible skeleton is typically outputted, possibly resulting in an overfitted model.

4.4 Optimization Method

To find a skeleton to the network, a parameter set $\boldsymbol{\beta}$ needs to be estimated by minimizing the loss function. To achieve this, a coordinate descent algorithm was employed following Gu *et al.* [33] – from which this subsection is largely based – which in turn closely followed Meier *et al.* [51]. This algorithm was chosen because of its simplicity, facilitating the development of new software, and its recognized use in this context, for example, in the `sparsebn` R package.

Since the proposed loss is decomposable, every node may be independently optimized. Therefore, consider the parameters associated with the parents of a node $X_j(t)$. The parcel of the loss associated with this is

$$
\begin{aligned}
f_{j,t}(\boldsymbol{\beta}_{\boldsymbol{j}\cdot\cdot}(t)) &= -\frac{1}{M}\ell_{j,t}(\boldsymbol{\beta}_{\boldsymbol{j}\cdot\cdot}(t)) + \sum_{\tau\in W(t)}\sum_{i=1}^{n}\|\boldsymbol{\beta}_{\boldsymbol{j}\cdot\boldsymbol{i}}(\tau,t)\| \\
&= -\frac{1}{M}\sum_{m=1}^{M}\left[\sum_{l=0}^{d_j} I\left(x_j^{(m)}(t)=l\right)\cdot\left(\mathbf{x}^{(m)}\right)^{\top}\boldsymbol{\beta}_{\boldsymbol{jl}\cdot}(t) - \log\left\{\sum_{l=0}^{d_j}\exp\left(\left(\mathbf{x}^{(m)}\right)^{\top}\boldsymbol{\beta}_{\boldsymbol{jl}\cdot}(t)\right)\right\}\right] \\
&\quad + \sum_{\tau\in W(t)}\sum_{i=1}^{n}\|\boldsymbol{\beta}_{\boldsymbol{j}\cdot\boldsymbol{i}}(\tau,t)\|,
\end{aligned}
\tag{4.9}
$$

where $\ell_{j,t}(\boldsymbol{\beta}_{\boldsymbol{j}\cdot\cdot})$ is the log-likelihood function of the parameter vector $\boldsymbol{\beta}_{\boldsymbol{j}\cdot\cdot}(t)$, which denotes the collection of parameters for all levels of the node $X_j(t)$. Minimizing this loss for all nodes across all timesteps is equivalent to minimizing the function in Equation (4.7). The coordinate descent algorithm method, presented in Algorithm 2, consists of optimizing the cost function along one dimension, keeping all the others constant, and iterating throughout the dimensions until convergence to a solution point is attained. To optimize the function in Equation (4.9) the algorithm circles through every $\boldsymbol{\beta}_{\boldsymbol{j}\cdot\boldsymbol{i}}(\tau,t)$ needed, moving each coordinate following a descent direction and repeating until convergence. Some coordinates are forced to be zero due to the constraints imposed by the DBNs. Since these restrictions are always applied only to a single coordinate, the algorithm can simply set those to zero and skip their optimization, never moving in directions that may cause illegal structures. In fact, it is as if the multinomial parametrization never depended on those coordinates. They are only on the vector due to notation convenience and to keep the parametrization general.

Algorithm 2 Coordinate descent

```
Input: A dataset $D$ and an initial estimate $\boldsymbol{\beta}^{\mathbf{(0)}}_{\boldsymbol{j}\cdot\cdot}(t)$
Output: The minimizer of the cost function $\boldsymbol{\beta}^{*}_{\boldsymbol{j}\cdot\cdot}(t)$
$\boldsymbol{\beta}^{*}_{\boldsymbol{j}\cdot\cdot}(t) \Leftarrow {\boldsymbol{\beta}_{\boldsymbol{j}\cdot\cdot}}^{(0)}(t)$
$k \leftarrow 0$
loop
    for all coordinates $\boldsymbol{\beta}_{\boldsymbol{j}\cdot\boldsymbol{i}}$ of $\boldsymbol{\beta}_{\boldsymbol{j}\cdot\cdot}(t)$ do
        Compute the minimizer of the quadratic approximation around the current estimate ${\bar{\boldsymbol{\beta}}_{\boldsymbol{j}\cdot\boldsymbol{i}}}^{(k)}$

        $\alpha \leftarrow \alpha_0$
        while inequality in Equation (4.13) is not fulfilled do
            $\alpha \leftarrow \eta\alpha$
        end while

        $\boldsymbol{\beta}^{*}_{\boldsymbol{j}\cdot\boldsymbol{i}} = \boldsymbol{\beta}^{*}_{\boldsymbol{j}\cdot\boldsymbol{i}} + \alpha{\mathbf{s_{ji}}}^{(k)}$
    end for
    Update the intercept term

    if stopping criteria is met then
        break
    end if

    $k \leftarrow k + 1$
end loop
```

To overcome the fact that this loss is undifferentiable because of the ℓ_1-norm, a linear second-order approximation of the function on the current estimate is minimized instead. This allows solving for the minimum in close form using the Karush-Kuhn-Tucker (KKT) [52, 53] theorem. The algorithm always moves towards the minimum of this approximation until it coincides with the minimum of the function.

Let $\boldsymbol{\beta_{j\cdot i}}^{(k)}$ represent the value of an arbitrary coordinate $\boldsymbol{\beta_{j\cdot i}}$ on the k-th iteration of the algorithm. The cost function, approximating the log-likelihood keeping only the second-order terms of the Taylor series expansion on the current point $\boldsymbol{\beta_{j\cdot i}}^{(k)}$, for a chosen coordinate $\boldsymbol{\beta_{j\cdot i}}$ is

$$\begin{aligned} Q_{j,t}^{(k)}(\boldsymbol{\beta_{j\cdot i}}) = & -\frac{1}{M}\left\{\left(\boldsymbol{\beta_{j\cdot i}} - \boldsymbol{\beta_{j\cdot i}}^{(k)}\right)^{\top} \boldsymbol{\nabla}\ell_{j,t}(\boldsymbol{\beta_{j\cdot i}}^{(k)}) + \frac{1}{2}\left(\boldsymbol{\beta_{j\cdot i}} - \boldsymbol{\beta_{j\cdot i}}^{(k)}\right)^{\top} \mathbf{H_{ji}}^{(k)}(\boldsymbol{\beta_{j\cdot i}}^{(k)}) \left(\boldsymbol{\beta_{j\cdot i}} - \boldsymbol{\beta_{j\cdot i}}^{(k)}\right)\right\} \\ & + \lambda_{j,i}\|\boldsymbol{\beta_{j\cdot i}}\|. \end{aligned} \tag{4.10}$$

The gradient of the log-likelihood on the current point can be computed as

$$\boldsymbol{\nabla}\ell_{j,t}(\boldsymbol{\beta_{j\cdot i}}^{(k)}) = \sum_{m=1}^{M} \begin{bmatrix} \left\{I\left[x_j^{(m)}(t) = 0\right] - P\left[X_j(t) = x_j^{(m)}(t)\right]\right\}\mathbf{x_i}^{(m)}(t) \\ \vdots \\ \left\{I\left[x_j^{(m)}(t) = d_j\right] - P\left[X_j(t) = x_j^{(m)}(t)\right]\right\}\mathbf{x_i}^{(m)}(t) \end{bmatrix}, \tag{4.11}$$

where the probabilities are computed using the current iteration (k) parameter values. Observe that $\mathbf{x_i}^{(m)}(t) \in \{0,1\}^{d_i}$, and thus $\boldsymbol{\nabla}\ell_{j,t}(\boldsymbol{\beta_{j\cdot i}}^{(k)}) \in \mathbb{R}^{r_j d_i}$. $\mathbf{H_{ji}}^{(k)}(\beta_{j\cdot i}^{(k)})$ is the Hessian matrix computed on the current point.

The calculation of this matrix can be computationally prohibitive, especially in high-dimensional contexts. Moreover, the algorithm update rule (shown further on) will need the inverse matrix of the Hessian, increasing even further the computation complexity of our algorithm. To prevent this, instead of using the true Hessian matrix, a diagonal approximation $\mathbf{H_{ji}}^{(k)}(\boldsymbol{\beta_{j\cdot i}}^{(k)}) = h_{ji}^{(k)}\mathbb{I}_{\boldsymbol{r_j d_i}}$ is used. The scalar $h_{ji}^{(k)} < 0$ can be chosen to be $h_{ji}^{(k)} = -\max\left\{\operatorname{diag}(-\mathbf{H_{ji}}^{(k)}(\boldsymbol{\beta_{j\cdot i}}^{(k)})), b\right\}$, where b is a small positive integer to improve numerical stability when $\max\left\{\operatorname{diag}(\mathbf{H_{ji}}^{(k)}(\boldsymbol{\beta_{j\cdot i}}^{(k)}))\right\} \to 0$. Using this approach, there is no need to compute the full Hessian matrix because only the diagonal is used. Furthermore, the resulting Hessian approximation can be trivially inverted by computing the reciprocal of every term in the diagonal. Also, a benefit for the computational time is the fact that the Hessian approximation needs not be re-computed every iteration and may be determined only for the starting point. It is proven that the algorithm still converges to the solution if the scalars $h_{ji}^{(k)}$ are chosen according to the strategy above [51, 54].

Applying the KKT conditions to minimize the cost in Equation (4.10), the minimizer $\bar{\boldsymbol{\beta}}_{\boldsymbol{j\cdot i}}^{(k)}$ is given by

$$\bar{\boldsymbol{\beta}}_{\boldsymbol{j\cdot i}}^{(k)} = \begin{cases} 0 & \text{if } \|\mathbf{d_{ji}}^{(k)}\| \leq \lambda_{ij} \\ -\frac{1}{h_{ji}^{(k)}}\left(1 - \frac{\lambda_{ij}}{\|\mathbf{d_{ji}}^{(k)}\|}\right)\mathbf{d_{ji}}^{(k)} & \text{otherwise,} \end{cases} \tag{4.12}$$

where $\mathbf{d_{ji}}^{(k)} = \boldsymbol{\nabla}\ell_{j,t}(\boldsymbol{\beta_{j\cdot i}}^{(k)}) - h_{ji}^{(k)}\boldsymbol{\beta_{j\cdot i}}^{(k)}$. The calculation of the minimum of the approximation is referred to in line 5 of Algorithm 2.

If the minimizer is not equal to the last estimate (meaning the algorithm had converged), the Armijo rule is applied to find a suitable step size in order to attain sufficient descent. Let $\mathbf{s_{ji}}^{(k)} = \bar{\boldsymbol{\beta}}_{\boldsymbol{j \cdot i}}{}^{(k)} - \boldsymbol{\beta}_{\boldsymbol{j \cdot i}}{}^{(k)}$ denote the distance of the current estimate to the minimum of the quadratic approximation, and $\boldsymbol{\Delta}^{(k)}$ represent the corresponding change in the value of the quadratic approximation. Then,

$$\boldsymbol{\Delta}^{(k)} = -(\mathbf{s_{ji}}^{(k)})^\top \boldsymbol{\nabla} \ell_{j,t}(\boldsymbol{\beta}_{\boldsymbol{j \cdot i}}{}^{(k)}) + \lambda_{ij} \|\bar{\boldsymbol{\beta}}_{\boldsymbol{j \cdot i}}{}^{(k)}\| - \lambda_{ij} \|\boldsymbol{\beta}_{\boldsymbol{j \cdot i}}{}^{(k)}\|.$$

The Armijo rule specifies that the step $\alpha^{(k)}$ should be the maximum step that verifies

$$f_{j,t}(\boldsymbol{\beta}_{\boldsymbol{j \cdot i}}{}^{(k)} + \alpha^{(k)} \boldsymbol{s_{ji}}^{(k)}) \leq f_{j,t}(\boldsymbol{\beta}_{\boldsymbol{j \cdot i}}{}^{(k)}) + \delta \alpha^{(k)} \boldsymbol{\Delta}^{(k)}, \tag{4.13}$$

for an arbitrary value of $\delta \in]0, 1[$. A backtracking-like algorithm (lines 7-10 in Algorithm 2) is employed to find this value, starting with an initial estimate for the step size α_0 and updating it successively by multiplying with an arbitrary factor $\eta \in]0, 1[$, until it achieves sufficient descent.

The update rule to obtain the next estimate for $\boldsymbol{\beta}_{\boldsymbol{j \cdot i}}$ (line 12 of Algorithm 2) is

$$\boldsymbol{\beta}_{\boldsymbol{j \cdot i}}{}^{(k+1)} = \boldsymbol{\beta}_{\boldsymbol{j \cdot i}}{}^{(k+1)} + \alpha^{(k)} \mathbf{s_{ji}}^{(k)}. \tag{4.14}$$

Since the intercepts $\boldsymbol{\beta}_{\boldsymbol{j \cdot 0}}$ are unpenalized, they can be simply updated by the minimum of the quadratic approximation (line 14 of Algorithm 2), *i.e.*,

$$\beta_{j \cdot 0}^{(k+1)} = \bar{\beta}_{j \cdot 0}^{(k)} = -\frac{d_{j0}^{(k)}}{h_{j0}^{(k)}}.$$

Note that, in spite of this update rule not fulfilling it, the constraint $\beta_{j00} = 0$ is mandatory to address identifiability issues. This means that this coordinate is always set to 0 throughout the execution of the algorithm.

This update rule was derived for an arbitrary coordinate $\boldsymbol{\beta}_{\boldsymbol{j \cdot i}}$, but observe that the cost function (Equation (4.9)) has several of them, including inter-temporal edges. The coordinate descent algorithm (Algorithm 2) finds the minimum by cycling through all coordinates, updating one at a time until it converges.

4.5 Directing the edges and learning parameters

Solutions to the minimization of the function in Equation (4.7) define a skeleton for our prior and transition network through the equivalence in Equation (4.2), but in order for it to define a Bayesian network structure, it has to be a directed acyclic graph. To obtain one, and following the suggestion from the MMHC algorithm, a greedy hill-climbing searching procedure is conducted, starting with an empty structure, but instead of checking every possible edge addition, reversal, and deletion on each iteration, it checks only in the set of allowed edges. In addition, in the transition network, constraining the structures to DBNs and imposing the temporal causality further decreases the allowed actions on each iteration. These constraints reduce the complexity of greedy hill-climbing substantially, especially in high dimensional contexts since the number of allowed edges is $\mathcal{O}(n^2)$. Algorithm 3 summarizes this method to direct the skeleton, being a slightly modified version of traditional hill-climbing (Algorithm 1). The process is similar to direct the prior network, but no temporal causality is imposed because it only concerns the first timestep.

Algorithm 3 DBN restricted greedy hill-climbing

Input: A temporal dataset D and a set of allowed edges E

Output: A restricted local optimal structure for a DBN G

```
loop
  $S^* \leftarrow -\infty$

  for all timestep $t \in [0, T]$ do

    for all timestep $\tau$ such that $t - v \leq \tau \leq t$ do
      for all addition or removal, resulting in a DAG, of edge $X_i(\tau) \rightarrow X_j(t)$ in $E$ do
        $G' \leftarrow$ result of applying operation to $G$

        if $\phi_{LL}(G') - \phi_{LL}(G) > S^*$ then
          $S^* \leftarrow \phi_{LL}(G') - \phi_{LL}(G)$
          $G^* \leftarrow G'$
        end if
      end for
    end for

    for all reversal, resulting in a DAG, of every edge $X_i(t) \rightarrow X_j(t)$ in $E$ do
      $G' \leftarrow$ result of applying operation to $G$

      if $\phi_{LL}(G') - \phi_{LL}(G) > S^*$ then
        $S^* \leftarrow \phi_{LL}(G') - \phi_{LL}(G)$
        $G^* \leftarrow G'$
      end if
    end for

  end for

  if $S^* < 0$ then
    break
  end if

  $G \leftarrow G^*$
end loop
```

Having a directed structure, parameter estimation is trivial (as long as the structure is sparse because the number of parameters increases exponentially in the number of parents) using a maximum likelihood estimation method (see Chapter 3). Note that, since the algorithm enforces temporal causality, no modification is needed for dynamic Bayesian networks. The parameters are estimated for both networks (prior and transition) independently and seeing them as separate Bayesian networks. Learning full conditional probability tables might be more beneficial than other more compressive methods, preserving the flexibility of the discrete Bayesian network model.

4.6 Implementation Details

In the context of this dissertation, a C++ implementation of the proposed method was developed and is available as open-source software on `https://github.com/JBSants/sDBN`. Two binaries can be built from source: `bntrain` which implements the method for non-time-series data (limiting the number of timesteps to one and thus eliminating all parameter restrictions), and it is used to train the prior network; `dbntrain` which is the main binary and implements the complete method and it is used to train the transition network.

Two training modes are provided. In normal mode, the user writes to standard input an arbitrary set of values for λ as targets for training. Alternatively, as a convenience feature, the user may specify the number of networks and a start and ending λ value via command-line argument. A geometric grid of lambdas is automatically generated and used as input to the program. In the alternative mode, networks are trained until a specified number of total edges are obtained, starting with an initial value for λ and specifying a step and a maximum number of trained networks. All modes begin the training with the largest λ value, following the specified path and using the parameters of the last network as a hot-start to the new network. The stopping criterion used was based on the best improvement of all the coordinates. If no coordinate changes more than $\epsilon = 10^{-4}$, then it is determined that the algorithm has converged.

Command-line arguments to the program are read and parsed using the `cxxopts`[1] open source library. The dataset input format is custom built and specifies all the necessary information to the algorithm, such as the number of timesteps and of nodes per timestep and the one-hot encoded dataset. A Python script was developed to convert between longitudinal data comma-separated valuess (CSVs) to this specific format. The output format is a JavaScript object notation (JSON) file with the trained networks, providing for each requested value of the λ parameter the obtained skeleton and the final directed structure along with the likelihood of it generating the training data. Both formats are human-readable, and additional details can be found on the Github page.

The unregularized problem in Equation (4.8) is solved using `liblbfgs`[2], a C implementation of the Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm [55, 56], because of its proven use in the `sklearn`[3] Python library. Matrix multiplication and other operations are handled by the `Eigen`[4] library. This is a widely used linear algebra template C++ library, including well-known gradient-based optimization projects such as Google® Tensorflow. This library allows changing the underlying math engine to Intel® MKL[5] or other basic linear algebra subprogram (BLAS) providers, accelerating an algorithm iteration considerably.

MPI paralellization was developed using the OpenMPI[6] library. MPI is a standard that specifies and facilitates inter-process communication and permits the distribution of the optimization problem through a cluster of computing devices. It also can leverage the multicore capabilities of modern central processing units (CPUs). In the provided implementation only the first two stages of the pipeline were

[1] Available on `https://github.com/jarro2783/cxxopts`.

[2] Available on `http://www.chokkan.org/software/liblbfgs/`.

[3] Available on `https://scikit-learn.org/stable/`.

[4] Available on `https://eigen.tuxfamily.org`.

[5] Available on `https://software.intel.com/mkl`.

[6] Available on `https://www.open-mpi.org`.

paralellized. This stages can be naturally distributed, because each node of the network can be optimized independently.

To find the adaptative weights, the work is distributed across every process *a priori*, splitting the nodes equally. Afterward, the weights are gathered to the rank 0 process, and the flow of execution continues from there. To find the skeleton, a work pool distribution model was implemented. In this model, a process acts as a master coordinating a pool of workers that serve requests from the master. A request is simply the node to be optimized as well as associated hyper-parameters and initial estimates. After a solution is found, the worker sends a response to the master process, storing the results and carrying on with the algorithm. If there is still work to be done, the master gives the worker a new node to optimize. If every node is already complete, the program continues to the next stage of directing the network. This stage is done only in rank 0 as the algorithm is not parallelized.

The implementation also relies on the MD5 hash implementation of the `openssl`[7] library, to facilitate dataset comparison and adaptative weights cache resolution. The program caches the adaptative weights, and if the user supplies the same dataset twice, the adaptative weights are no re-computed because they stay the same.

4.7 Complexity Analysis

An iteration of the implemented coordinate descent optimization algorithm has theoretical $\mathcal{O}(Mnr^2)$ complexity, where M is the size of the dataset, n is the total number of nodes to optimize, and r is the maximum number of allowed levels per node.

Consider that matrix multiplication with dimensions $a \times b$ and $b \times c$ has computational cost $\mathcal{O}(abc)$. Actual multiplication complexity could be lower, but for the purposes of this discussion, the naive scenario is considered. Also, without loss of generality, assume that every node has the same number of allowed levels r. The coordinate descent iteration has three main steps: the update of every coordinate, the update of the intercept, and the checking of the stopping criteria.

The coordinate update itself has three main steps: the closed-form computation of the minimum of the quadratic approximation, the backtracking-like algorithm to determine the step size, and the actual update of the coordinate.

The minimum of the quadratic approximation is determined using the gradient and hessian approximation of the log-likelihood function around the current point. In order to compute the gradient and Hessian, the level probabilities have to be determined (Equation (4.11)) for every observation in the dataset. This has to be done through the multinomial logistic regression equations. In practice, the linear map for every observation is determined via matrix multiplication, with cost $\mathcal{O}(Mnr)$. The resulting matrix has dimensions $M \times r$, and each column has a map with the parameters for each level. To speed up the iterations, instead of re-computing the linear map on every iteration, the linear map is cached and only updated incrementally each time a coordinate is updated, summing the appropriate difference to every line of the matrix, with cost $\mathcal{O}(Mr)$.

[7] Available on `https://www.openssl.org`

Probabilities are then determined via exponentiation of the linear mapping and posterior normalization, but these operations only require linear time on the matrix. Having them, they are subtracted to an indicator variable (all in $\mathcal{O}(Mr)$ time) and then multiplied to the corresponding dataset block (according to Equation (4.11)). This multiplication has complexity $\mathcal{O}(Mr^2)$. This is the more expensive operation, and thus gradient computation has this complexity. To save time, the new cost, assuming a unitary step, is calculated at the same time as the gradient adding no additional complexity.

The hessian approximation only requires the computation of the diagonal, which in turn only depends on probabilities. The diagonal has r^2 entries, and every entry involves the sum of a quantity involving probabilities for every observation. In the proposed implementation, hessian coefficient determination has $\mathcal{O}(Mr^2)$ complexity.

Having the gradient and the hessian, the approximation minimum is computed according to Equation (4.12) in $\mathcal{O}(r^2)$ time, because every vector has size r^2 and only linear time operations are involved.

Determination of the step size is done using several computations of the cost function. Similar to the linear map, instead of determining the regularization term for every iteration, it is cached and updated every time a coordinate changes. This term is not linear, so the *delta* has to be computed with the norm of the old parameter value and the norm of the new. This update has complexity $\mathcal{O}(r^2)$, the size of the coordinate vectors. Then, cost function calculation has $\mathcal{O}(Mr)$ complexity because it only depends on the probabilities for each observation (like the gradient) and the cached regularization term. The backtracking algorithm introduces constant complexity because a limit of iterations (in this implementation 15) is imposed.

Having the step size and the step direction, updating the coordinate is a simple sum having $\mathcal{O}(r^2)$ complexity. These three steps are repeated for each coordinate. Since gradient and hessian computation are the most expensive, this inner loop has $\mathcal{O}(Mnr^2)$ complexity.

Intercept update is done similarly to coordinate update, but since every node only has r intercepts, the complexity is just $\mathcal{O}(Mr)$. The stopping criterion is checked by caching the best improvement along with the algorithm execution and thus only introduces constant complexity. Since the inner coordinate update loop is the most expensive step, the complexity of the complete coordinate descent iteration is dominated by this term.

The skeleton discovery step repeats this optimization for every node of the network. The overall complexity of this step is $\mathcal{O}(kMn^2r^2)$, where k is the limit of iterations of the coordinate descent optimization for a single node. In this implementation, the algorithm is limited to 5000 iterations.

Chapter 5

Results

5.1 Synthetic Datasets

Several experiments were conducted to validate the proposed method. Transition networks were trained using data sampled from known, randomly generated, stationary dynamic Bayesian networks. The resulting transition structures were compared to the true one, to measure the algorithm's ability to recover the underlying process. Prior networks structures were not considered as they are simple Bayesian networks trained using the same method but without temporal restrictions.

Because the underlying structure is known, the quality of the recovered structure is measured by simply accounting for the true positive, false positive, and false negative edges. True positive (TP) edges appear in the estimated network and correspond to correct edges in the original network. False positive (FP) edges show in the resulting network but are not part of the original structure. False negative (FN) (FN) edges take part in the ground truth network but were not identified by the algorithm.

Although direct comparison is possible and seen in the literature, the algorithm may output a network that is different from the original structure but still indistinguishable from an observational standpoint, *i.e.*, both networks have the same score. They cannot be distinguished using only the observed data. In fact, both networks are in the same equivalence class. To mitigate this issue, the edges of the original network are labeled as reversible or unreversible based on an algorithm developed by Chickering [19]. Reversible edges that appear reversed on the estimated network are still counted as true positives. This alternative counting establishes a better framework to compare Bayesian networks and eliminates the possibility of getting higher or lower scores just by shuffling the order of the nodes, causing the hill-climbing procedure to consider first a particular edge direction.

Two metrics were computed to evaluate the obtained network: the precision of the recovered structure is computed as

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{5.1}$$

the recall, also known as true positive rate, is given by

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{5.2}$$

Summing up these two metrics, the F_1 score is computed as the harmonic mean between precision and recall, *i.e.*,

$$F_1 = 2\,\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{5.3}$$

In addition to structure comparison, the resulting networks were also subject to evaluation of the ϕ_{LL} and ϕ_{MDL}, of Equation (3.11) and Equation (3.12), respectively. Both scores were computed using a different, independently sampled dataset from the same DBNs. This dataset constitutes a test dataset as it is never used during the training process and is previously completely "unseen" by the estimated networks. These scores can assess structure quality and whether or not the obtained structure under or overfits the training data.

Structure generation procedure

Two stationary transition networks generators were used. In the first, stationary transition network structures are generated by taking a backbone network containing only identity edges, *i.e.*, edges from the same random variable on the previous timestep, and progressively adding randomly selected edges to form several v-structures between nodes. This procedure takes as input the number of nodes in each timestep, the desired Markov lag, and the number of inter and intra-temporal v-structures. The desired Markov lag simply influences the number of stub timesteps available. In practice, this means that the inter-temporal v-structures have a larger set of possible parent nodes.

After adding all identity edges, the intra-temporal v-structures are fixed as they may originate cycles in the transition network. To avoid this, children nodes are selected from all nodes in the timestep without replacement. Then, the in-degree for each child is sampled from a truncated discrete normal distribution with an adjustable mean. Once the degree is established, parents are selected by choosing from all non-children nodes uniformly.

The procedure is similar to fix the inter-temporal v-structures. Children nodes are fixed (they can be the same as the intra-temporal v-structures children), and then the number of parents is determined. Next, parents are chosen uniformly from the pool of stub nodes in the transition network.

The second generator returns a network with a tree intra-temporal structure augmented with inter-temporal edges. The tree structure is constructed first by choosing a random topological order for the nodes and then choosing a single parent for each node respecting that order. Inter-temporal edges are also randomly picked by choosing a maximum of p parents uniformly from previous timesteps. These structure types are used to fairly compare the proposed method with the `tDBN` algorithm.

Sampling Method

Datasets were simulated using the sampling methods available in the open-source package `tDBN`.[1] This software library provides a set of utility classes for the training and sampling of dynamic Bayesian networks. To generate observations, the desired network structure is loaded along with the possible states for each node. Then, random parameters are generated for each node, according to the underlying structure and possible states. The software then outputs a simulated dataset with the desired number of observations.

[1] Available in `http://josemonteiro.github.io/tDBN/`.

Experimental Setup

Under this experimental framework, four different sets of tests were conducted. First, a total of 16 transition networks with a Markov lag of 1 were generated for 20, 50, 100, and 300 nodes per timestep, four networks for each size with an increasingly larger number of v-structures, and therefore, increasing complexity. Training datasets were sampled with 1000 observations for each network, and test datasets were sampled with 250 observations. All nodes were considered to have three possible discrete states. The training dataset was used as input to the `dbntrain` executable, and a path of 80 stationary transition networks was requested with a Markov lag of 1. The path follows a geometric grid of λ values, chosen to originate a significant variation on the outputted networks scores. The backtracking algorithm parameters were set to $\alpha_0 = 1$, $\eta = 0.5$ and $\delta = 0.1$, typical values found in the literature. Different values were tried, but often resulted in a decrease in the performance of the backtracking algorithm.

Additionally, DBNs with 300 nodes per timestep were generated fusing two distinct networks. First, two DBNs with 150 nodes per timestep were independently generated and sampled. Then, the two structures and datasets were fused into one, shuffling the position of the nodes on the final network. The resulting dataset served as input to the algorithm, requesting a path of 80 networks following a geometric grid of λ values. This was repeated three times with increasing total v-structures. This evaluates the algorithm's ability to separate two distinct networks from observational data.

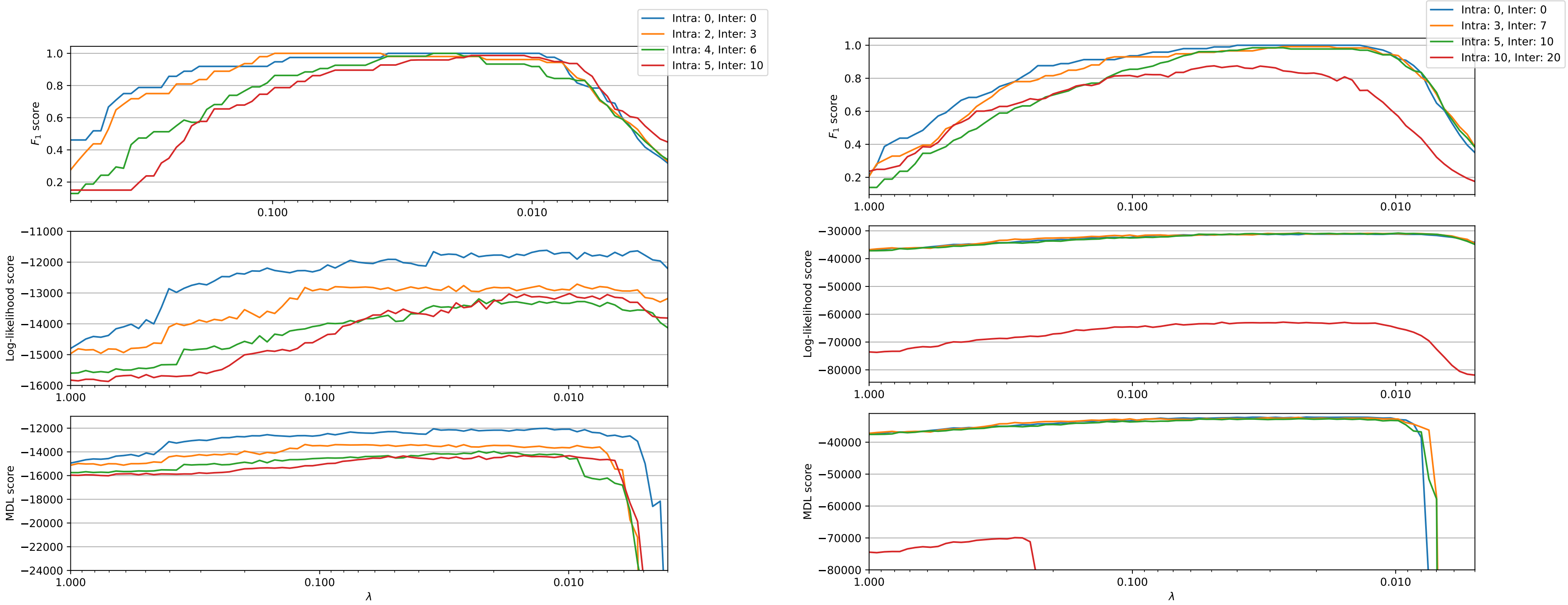
Following that, a DBN with 30 intra-temporal and 60 inter-temporal v-structures was generated. To evaluate the algorithm's performance on different dataset sizes, a linear grid with increasing dataset sizes was trained. A path of 80 networks was trained for each dataset size, also following a geometric grid of λ values. The experiments were repeated three times, using different networks with the same number of v-structures.

Finally, the performance metrics were compared between the proposed algorithm and the `tDBN` algorithm. Three different structure of stationary networks types were sampled and used: tree structures with inter-temporal edges restricted to a maximum of 1, 2, and 3 parents from previous timesteps; inter-temporal only structures with random parents also restricted to a maximum of 1, 2, and 3 parents; random structures with a fixed number of v-structures similar to those used in the other experiments. All structures were generated with Markov lag 1. For each structure type, the algorithms were run five times using different original networks. `sDBN` was used to train a path of 80 networks following a geometric grid of λ values. `tDBN` was run using the log-likelihood score and limited to 2 parents from the previous timesteps, except for runs using structures with three parents from the previous timesteps.

Structure identification results

Regarding the first experiment, Figure 5.1 and Figure 5.2 plot the F_1, ϕ_{LL} and ϕ_{MDL} scores for each obtained network in the geometric grid of λ values. Table 5.1 summarizes the results by reporting all metrics for the best network on each trained path.

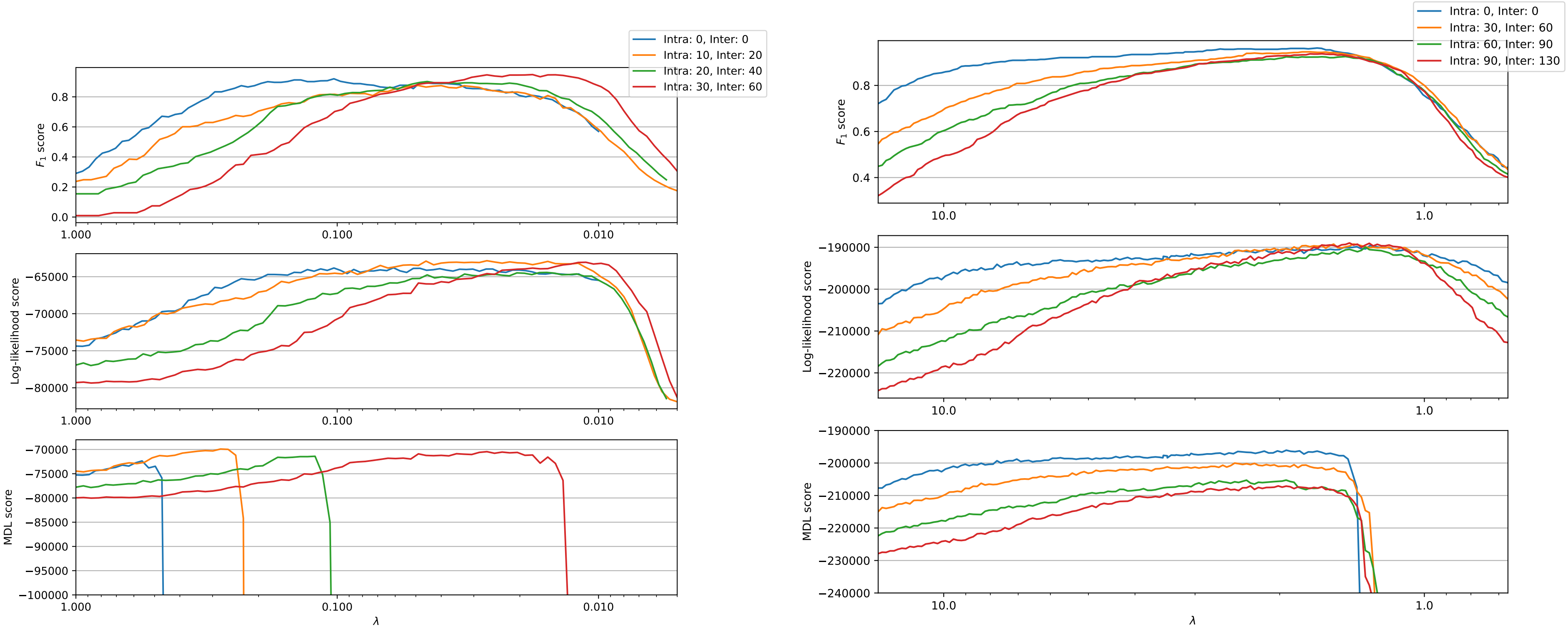
By observing the F_1 evolution over the grid of λ values, it can be established that trained networks follow comparable trends. For higher regularization, the score starts very low. These networks are characterized by high precision values, but the F_1 score is very low due to the low recall. This means



(a)

(b)

Figure 5.1: Evolution of the performance metrics F_1, ϕ_{LL} and ϕ_{MDL} evaluated along a resulting path of 80 networks using a geometric grid of λ values. Generated networks have 20 (a) and 50 (b) nodes per timestep, and each node admits three different levels. The networks are trained from observational data, using datasets with 1000 observations across four timesteps. ϕ_{LL} and ϕ_{MDL} are determined using a different four timestep dataset with 250 observations. Four different configurations of v-structures are compared for every network size, starting with no v-structures (only identity edges) and progressively increasing them. The plot legend associates each configuration to the plotted color.



(a)

(b)

Figure 5.2: Evolution of the performance metrics F_1, ϕ_{LL} and ϕ_{MDL} evaluated along a resulting path of 80 networks using a geometric grid of λ values. Generated networks have 100 (a) and 300 (b) nodes per timestep, and each node admits three different levels. The networks are trained from observational data, using datasets with 1000 observations across four timesteps. ϕ_{LL} and ϕ_{MDL} are determined using a different four timestep dataset with 250 observations. Four different configurations of v-structures are compared for every network size, starting with no v-structures (only identity edges) and progressively increasing them. The plot legend associates each configuration to the plotted color.

that the recovered edges are, in fact, present in the original network, but only a small portion of the original network is recovered.

Table 5.1: Best F_1 score and corresponding λ obtained by direct structure comparison and by evaluating ϕ_{LL} and ϕ_{MDL} on test datasets. Generated networks have $n = 20, 50, 100, 300$ nodes per timestep and each node admits 3 different levels. The networks are trained from observational data, using datasets with 1000 observations across 4 timesteps. ϕ_{LL} and ϕ_{MDL} are determined using a different 4 timestep dataset with 250 observations. Structure comparison reports the precision, recall, and F_1 scores for the single best network – according to the F_1 metric – of the full path. Log-likelihood and MDL report the associated ϕ and F_1 scores for the single best network – according to the corresponding ϕ_{LL} or ϕ_{MDL} score – of the full path. Each row refers to a different original structure with the specified number of intra-temporal and inter-temporal v-structures.

V-structures		Structure Comparison				Log-likelihood			MDL		
Intra	Inter	Pre	Rec	F_1	λ	ϕ_{LL}	F_1	λ	ϕ_{MDL}	F_1	λ
n = 20											
0	0	1.00	1.00	1.00	0.0359	-11616.0	0.98	0.0122	-12013.2	0.98	0.0122
2	3	1.00	1.00	1.00	0.0981	-12713.3	0.85	0.0093	-13378.3	1.00	0.1146
4	6	1.00	1.00	1.00	0.0240	-13191.3	0.98	0.0230	-13946.0	0.98	0.0230
5	10	0.97	1.00	0.99	0.0172	-13018.9	0.94	0.0099	-14303.2	0.99	0.0174
n = 50											
0	0	1.00	1.00	1.00	0.0400	-31126.4	0.95	0.0105	-32123.2	1.00	0.0219
3	7	1.00	0.98	0.99	0.0267	-30756.3	0.99	0.0234	-32226.0	0.99	0.0234
5	10	1.00	0.97	0.98	0.0350	-30814.8	0.92	0.0098	-32637.9	0.98	0.0234
10	20	0.99	1.00	0.99	0.0128	-31198.8	0.99	0.0128	-34110.5	0.98	0.0267
n = 100											
0	0	0.87	0.98	0.92	0.1030	-63790.0	0.86	0.0609	-72362.3	0.58	0.5583
10	20	0.84	0.92	0.88	0.0489	-62841.3	0.84	0.0267	-69903.2	0.64	0.2796
20	40	0.92	0.88	0.90	0.0453	-64423.2	0.84	0.0158	-71399.1	0.80	0.1215
30	60	0.96	0.93	0.95	0.0179	-63049.7	0.91	0.0112	-70448.8	0.95	0.0267
n = 300											
0	0	0.99	0.94	0.96	1.6714	-189563.2	0.91	1.2972	-196058.3	0.96	1.9376
30	60	0.98	0.91	0.95	1.7303	-189229.1	0.93	1.4044	-200027.3	0.92	2.4815
60	90	0.95	0.90	0.93	1.4587	-189926.4	0.91	1.3266	-205256.4	0.92	1.9390
90	130	0.98	0.90	0.94	1.6346	-188964.8	0.93	1.4313	-207060.1	0.91	2.3001

Progressing on the grid, the score attains a maximum on the optimal λ value and then begins to drop. This drop is explained by drastically lower precision values. This means that the algorithm starts to select edges that are not present in the original network, evidencing insufficient regularization. Recall values may still not achieve the maximum, and missing edges can still exist after the optimal λ is crossed.

The results suggest that the `sDBN` algorithm can correctly recover transition structures with high accuracy. The best F_1 score tends to decrease as network complexity increases. The majority of tested structures scored a F_1 score greater than 0.90. The notable exception is the network with 100 nodes per timestep, 10 intra-temporal and 20 inter-temporal v-structures that scored a F_1 of 0.88. This is caused by a lowering of the precision score. A possible explanation is a particularly difficult structure type for the algorithm to recover.

The log-likelihood score on the test dataset follows the same tendency as the F_1 score. For higher regularization, ϕ_{LL} starts low, attains a maximum near the best F_1 score, and then drops. This plot shape is more clearly visible in networks with a larger number of nodes per timestep and with a higher number of v-structures and therefore edges. Log-likelihood is expected to drop with the decrease of precision because a higher number of edges not present in the underlying structure usually causes the model to overfit on the training dataset.

The λ value with maximum log-likelihood does not coincide with the one for maximum F_1, and almost always scored strictly lower relative to the best retrieved F_1. Nevertheless, the maximum log-likelihood still serves as a valid approximation to the optimal λ, with the majority of networks scoring a F_1 higher than 0.90 using this criterion. On the tested structures, 12 out of 16 runs present a likelihood estimated network with F_1 within 5% of the best. This suggests that cross-validation may be used to estimate the optimal λ.

MDL scores are consistent with log-likelihood for networks with 20, 50, and 300 nodes per timestep. This score drops abruptly when the rise of log-likelihood stagnates. This is expected as it results from an increase in the number of network parameters not followed by a substantial increase in likelihood. Networks with 100 nodes per timestep and a small number of v-structures do not follow this pattern, as the MDL score cuts off very distant from the maximum likelihood λ. Maximum MDL λ values can also serve as good approximations to the optimal λ. However, these scores tend to favor networks with fewer edges.

Structure splitting results

Figure 5.3 plots F_1 score and the number of edges incorrectly identified between the two separate networks for each obtained network in the geometric grid of λ values. Even in this challenging dataset, the algorithm still performs adequately, recovering networks with F_1 score above 0.90 for all considered configurations of v-structures. These results are consistent with previous experiments: an initial rise of the score is followed by a decline after a maximum is reached on the optimal λ.

Using more regularization, there are no identified edges between the two networks. However, this is not the case for lower values of λ. At these lower levels of regularization, the algorithm incorrectly associates nodes belonging to different networks. Table 5.2 reports the results for the best network according to F_1 score. As expected, the misidentified edges increase with network complexity. Albeit, these inter-network edges are reduced and only represent a small fraction of all identified edges. Overall, the `sDBN` algorithm has a good performance splitting the networks.

Table 5.2: Best F_1 score and associated number of edges identified between two disjoint structures trained from the same dataset. For each configuration of v-structures, two separate DBNs with 150 nodes per timestep were generated and sampled, and the datasets were then fused to train a single network with 300 nodes per timestep. The percentages in parenthesis are the proportion of misclassified edges between the two structures on the total number of predicted edges.

Intra	Inter	F1	Inter-network edges
20	30	0.936	2 (0,6 %)
60	90	0.939	6 (1,3 %)
90	150	0.904	8 (1,5 %)

Results quality vs. Dataset size

Figure 5.4 plots the best F_1 score achieved for different number of observations. As observed, the algorithm has consistently great performance even when using a smaller dataset. The resulting score is

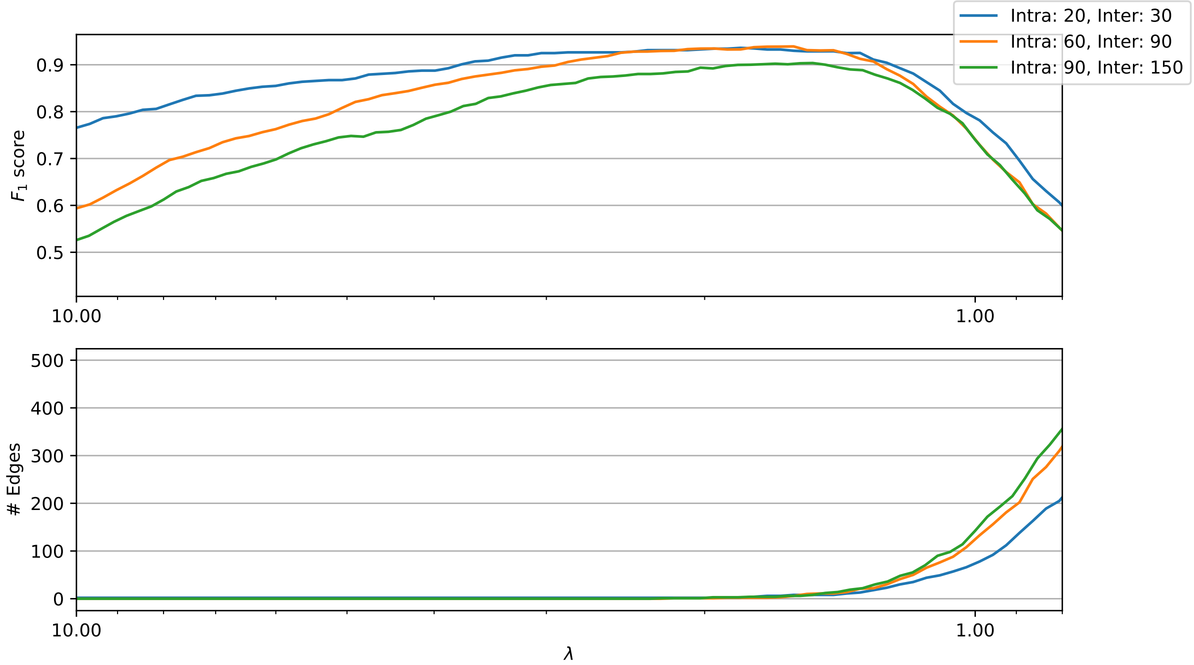


Figure 5.3: Evolution of the F_1 metric and the number of incorrectly identified edges between two disjoint structures evaluated along a resulting path of 80 networks using a geometric grid of λ values. For each configuration of v-structures, two separate DBNs with 150 nodes per timestep were generated and sampled, and the datasets were then fused to train a single network with 300 nodes per timestep. Each node admits 3 different levels. The networks are trained from observational data, using datasets with 1000 observations across 4 timesteps.

fairly consistent between the three runs, and all networks trained with a dataset with 300 or more observations scored a F_1 higher than 0.90. Oscillations on the best F_1 are explained by the non-deterministic nature of dataset sampling, which in these runs led to a worse score on the largest dataset, even though it is expected that a larger amount of data leads to a better result. Although the F_1 scores are consistently good, cross-validation estimations may become worse on smaller datasets. Unless domain knowledge can be applied in real data situations, this may lead to low-quality networks. Further work is needed to test the algorithm in these conditions meaningfully.

`sDBN` vs. `tDBN` comparison

Table 5.3 compares precision, recall, F_1 and ellapsed time between the `sDBN` and the `tDBN` algorithms. All experiments were run on an Intel® Xeon® E3-1220 v3 @ 3.10GHz quad-core CPU running Ubuntu Server[2]. The `sDBN` algorithm was run with Intel® MKL acceleration enabled. The `tDBN` algorithm was run using OpenJDK[3] version 11.0.11.

The results show that `sDBN` beats `tDBN + LL` on structure recovery. These experiments statistically reject the hypothesis[4] that both algorithms perfom equally well in terms of the F_1 metric. Therefore, it is fair to extrapolate that, on the tested structure types, `sDBN` performs better than `tDBN`. Only for $n = 50$, $p = 3$ the latter algorithm scored, on average, a better F_1, but the proposed method still achieved better

[2] Available on `https://ubuntu.com/download/server`.
[3] Available on `https://openjdk.java.net`.
[4] Wilcoxon test [57] yields a p-value of 1×10^{-12}.

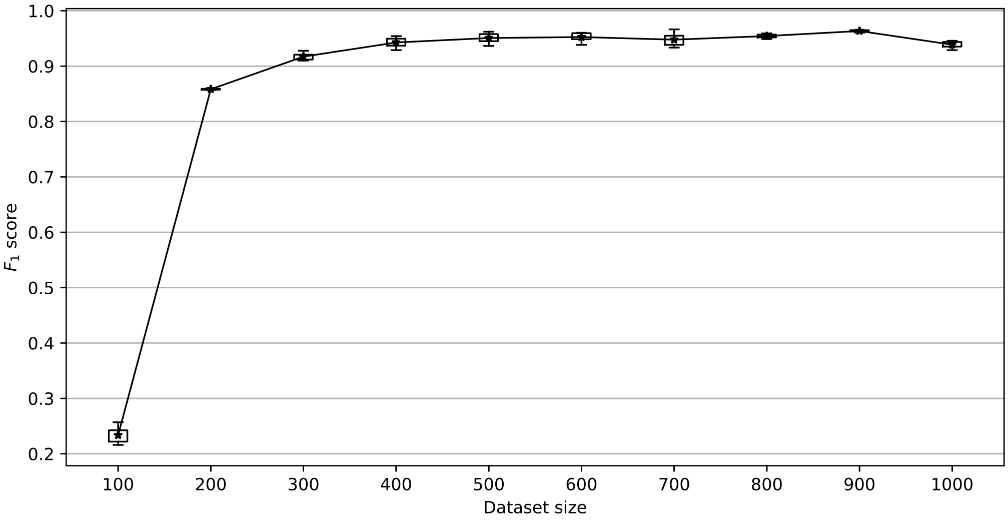


Figure 5.4: F_1 score of the best network obtained for several dataset sizes. For each dataset size, the original network was sampled, and the resulting dataset was used as input to the `dbntrain` executable. A path of 80 stationary networks with a Markov lag of 1 was requested, following a geometric grid of lambda values. This was repeated three times for distinct randomly generated networks with 300 nodes per timestep and with 30 and 60 intra-temporal and inter-temporal v-structures, respectively. The reported scores are centered on the mean scores over the three runs and present the box plot with the median and quantiles for each dataset size.

precision.

The `tDBN` algorithm tends to have excellent recall, meaning that the algorithm selects the entire original structure but adds additional false positive edges. In the complete tree setting, this suggests that `tDBN` cannot discern the true parents of the nodes correctly but still outputs the correct tree structure. It is expected that `tDBN` improves by using the MDL score on these structure types, so further work is needed to compare it with `sDBN`. In inter-temporal only networks, the precision is even lower because there are no tree intra-temporal relationships. On variable v-structures networks, the `tDBN` algorithm is expected to perform poorly as the intra-temporal underlying structure violates the algorithm's assumptions. The fact that the proposed algorithm performs well in all these contexts shows its increased versatility.

Consistently `sDBN` outperforms `tDBN` considerably in terms of running time. Even for $n = 50$, `sDBN` outputs the entire path of 80 networks in under 30 minutes. On the other hand, `tDBN` takes more than 4 hours to train a single network. This is consistent with the theoretical quadratic complexity of the novel pipeline that is fairly better when compared to `tDBN` which has a $\mathcal{O}(n^{p+3})$ complexity. The exponential complexity in p explains the severely increased running time for $n = 20$, $p = 3$. `sDBN` has weaker assumptions, better theoretical complexity, lower practical running time and can achieve good structure identification. Unfortunately, it has the additional overhead of optimal λ estimation, if domain knowledge is not available.

Table 5.3: Precision, recall, F_1 score and elapsed time t for various types of networks comparing the `sDBN` and `tDBN` algorithm. Performance metrics are expressed in percentages. The table reports average values along with the standard deviation in parenthesis over five runs for each type of network. On every run, a dataset with 1000 observations over four timesteps was sampled and used as input to both algorithms. Variable p is the maximum number of parents from previous timesteps allowed; v is a tuple characterizing the random structure generation with the number of intra-temporal and inter-temporal v-structures, respectively; n is the number of nodes per timestep. Values in bold are the best obtained for each metric per row. In `sDBN` the performance metrics are the ones obtained for the best network in the trained path (t is the elapsed time for the entire path).

	sDBN				tDBN + LL			
Type	Pre	Rec	F_1	t / s	Pre	Rec	F_1	t / s
Complete Tree + Inter-temporal (n = 20)								
$p = 1$	**97.5** (3.2)	93.3 (3.1)	**95.3** (1.8)	**147.5** (14.0)	66.1 (0.0)	**100** (0.0)	79.6 (0.0)	264.6 (4.2)
$p = 2$	**96.0** (3.2)	89.6 (6.3)	**92.6** (4.2)	**138.6** (14.1)	82.7 (4.3)	**100** (0.0)	90.5 (2.6)	264.7 (5.4)
$p = 3$	**93.6** (4.2)	87.6 (4.7)	**90.5** (3.9)	**113.3** (15.5)	73.4 (6.5)	**100** (0.0)	84.5 (4.4)	4338.8 (78.4)
Complete Tree + Inter-temporal (n = 50)								
$p = 1$	**95.8** (2.2)	96.2 (3.0)	**96.0** (1.6)	**952.7** (64.4)	66.4 (0.0)	**100** (0.0)	79.8 (0.0)	16474.4 (161.2)
$p = 2$	**91.6** (2.6)	91.7 (2.4)	**91.6** (2.1)	**862.0** (56.6)	81.3 (3.3)	**100** (0.0)	89.7 (2.0)	16502.0 (58.0)
$p = 3$	**89.7** (2.5)	82.9 (2.6)	86.1 (1.7)	**775.7** (51.3)	88.1 (3.2)	**90.8** (1.6)	**89.3** (1.3)	16517.8 (53.3)
Inter-temporal only (n = 20)								
$p = 1$	**100** (0.0)	**100** (0.0)	**100** (0.0)	**168.4** (15.1)	33.9 (0.0)	**100** (0.0)	50.6 (0.0)	254.8 (4.7)
$p = 2$	**99.4** (1.2)	**100** (0.0)	**99.7** (0.6)	**148.4** (17.7)	49.8 (1.7)	**100** (0.0)	66.5 (1.5)	261.0 (4.3)
$p = 3$	**98.4** (2.1)	99.5 (1.1)	**98.9** (1.3)	**132.2** (10.1)	45.3 (2.2)	**100** (0.0)	62.3 (2.1)	4273.6 (61.7)
Inter-temporal only (n = 50)								
$p = 1$	**100** (0.0)	**100** (0.0)	**100** (0.0)	**1293.5** (80.2)	33.6 (0.0)	**100** (0.0)	50.3 (0.0)	16286.2 (51.0)
$p = 2$	**96.4** (5.3)	98.7 (1.4)	**97.5** (3.4)	**1043.5** (178.9)	51.5 (2.5)	**100** (0.0)	68.0 (2.2)	16332.9 (72.9)
$p = 3$	**98.0** (1.0)	**97.6** (1.4)	**97.8** (0.9)	**854.1** (96.3)	53.8 (3.0)	86.4 (1.0)	66.3 (2.3)	16616.2 (425.6)
Variable v-structures (n = 20)								
$v = (2, 3)$	**97.2** (3.9)	**100** (0.0)	**98.6** (2.1)	**172.8** (21.6)	41.7 (1.4)	98.5 (1.9)	58.6 (1.2)	265.3 (3.3)
$v = (4, 6)$	**99.4** (1.2)	**98.8** (1.5)	**99.1** (0.8)	**150.4** (15.0)	50.8 (0.0)	92.6 (1.4)	65.6 (0.4)	261.5 (8.3)
$v = (5, 10)$	**98.9** (1.3)	**96.7** (1.0)	**97.8** (1.0)	**151.4** (20.0)	58.0 (1.3)	94.6 (3.4)	71.9 (1.1)	261.6 (2.9)

5.2 Multinet classification on benchmark Datasets

To test the performance of this learning procedure on real data, a classification task was performed on a collection of public multivariate time-series from the UCI Machine Learning Repository [58] and the UCR Time Series Classification Archive [59]. These datasets were already pre-processed and discretized by Samuel Arcadinho [34] using the SAX algorithm [60], ready to be used by discrete DBN training algorithms. Each random variable is assumed to take four possible levels. No missing values exist. Table 5.4 presents a summary of the considered datasets.

Each dataset was shuffled and then split into train and test datasets using stratified cross-validation with five folds. This ensures that the datasets keep the original class balance. A cross-validation approach was used due to the small size of the dataset, as this can better evaluate algorithms performance than on a single test dataset.

For each dataset, a DBN multinet was trained and then used for prediction on a test dataset. The DBNs were trained considering a Markov lag of 1 and a stationary process, which effectively helps to increase the number of observations on the training dataset. In the directing process, nodes were restricted to have a maximum of 3 parents. The observations were stratified by class and, for each sub-dataset, a DBN was trained using the proposed algorithm. The predicted class is predicted computing the joint query for each network and considering the class associated with the highest probability (see Equation (3.18)).

To select the best λ value, a cross-validation procedure with 10 folds was used in the training data. Using each fold as a test dataset, a path of 80 networks was trained, and then the ϕ_{LL} score was computed for each network. The λ selected is the one that has the best average score across all folds. Every class has a different value.

Serving as a comparison, DBN multinets were trained using the `tDBN` algorithm, a state-of-the-art DBN structure discovery algorithm. These networks were trained using the MDL score, considering a stationary process, and limiting the number of parents from previous timesteps to three.

Table 5.4: Summary of the MTS benchmark datasets used for classification to assess the performance of the proposed method. The reported number of nodes is per timestep.

Dataset	Nodes	Timesteps	Observations	Classes
ArabicDigits	13	4	8800	10
CharacterTrajectories	3	100	2858	20
ECG	2	39	200	2
JapaneseVowels	12	7	640	9
Libras	2	45	360	15
NetFlow	4	50	1337	2
UWave	3	100	4478	8
Wafer	6	100	1194	2

Results

Table 5.5 reports the average performance metrics on the test dataset, comparing `sDBN` and `tDBN` with MDL. Figure 5.5 shows the receiver operation characteristic (ROC) plot curve for all benchmark datasets with binary classes. Using sDBN, ECG, NetFlow and Wafer scored an average area under the ROC curve (AUC) metric [61], resulting from the integration of each single ROC curve and posterior averaging, of 0.836, 0.852 and 0.854, respectively. On the other hand, tDBN scored 0.812, 0.870 and 0.887, respectively. Figure 5.6 and Figure 5.7 report the sum of the confusion matrices for both methods.

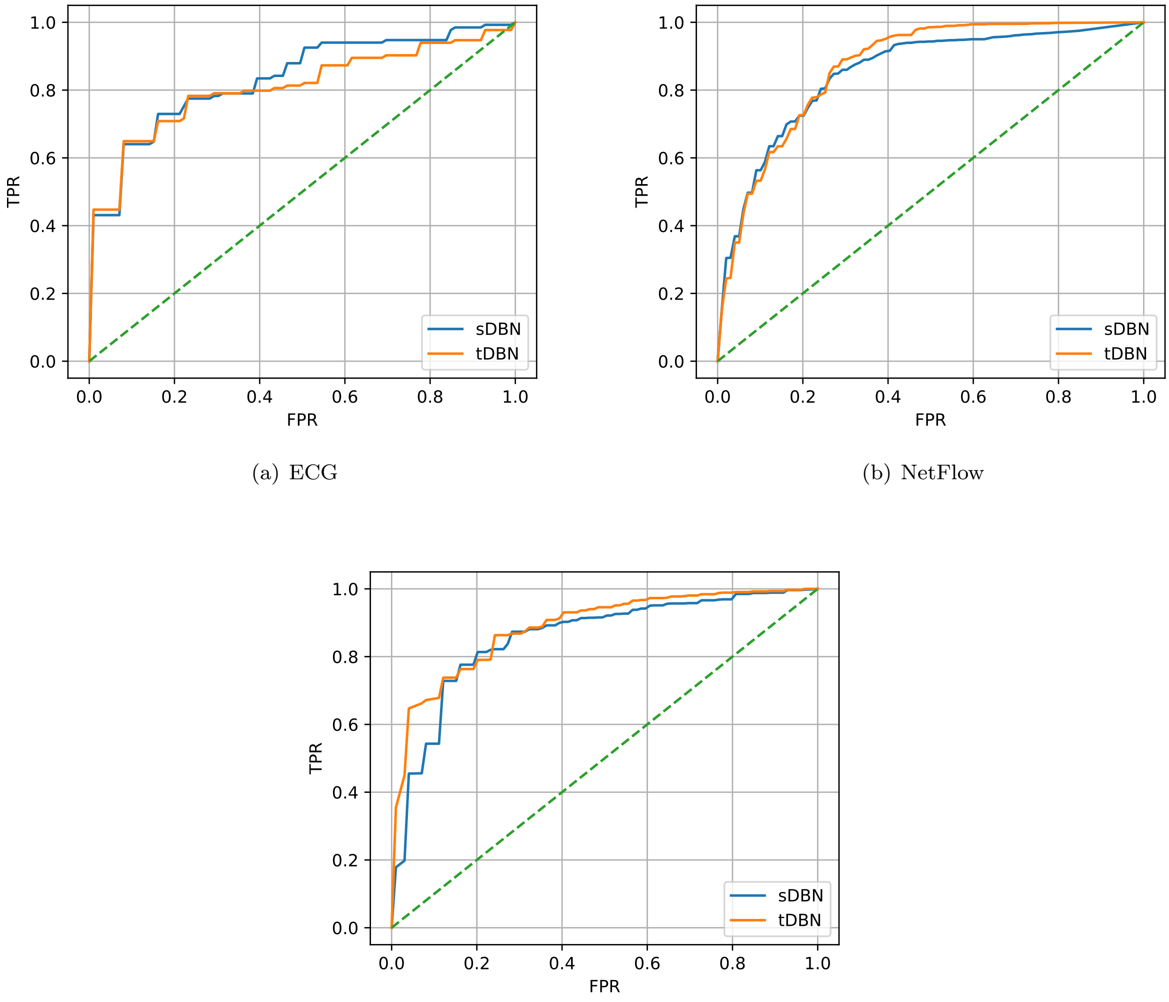
On 5 out of 8 datasets, tDBN shows better performance than sDBN. Still, on all datasets, the proposed method performs within 3% of the accuracy of the state-of-the-art method. Accordingly, ROC plots show that both algorithms have very similar performances on the binary MTS datasets. AUC scores are also similar, but with advantage for tDBN + MDL, scoring higher in 2 out of 3 datasets.

Poor sDBN performance can be associated with difficulty to select the optimal λ value. The benchmark datasets do not have a large number of observations, which can result in an unsatisfactory cross-validation estimate. This is consistent with results in synthetic datasets, as the best network in the path scored in the test dataset often did not coincide with the optimal one. Also, in these experiments, each network that belongs to the multinet classifier was trained using the same geometric grid. However, different classes datasets may have different optimal λ values, and therefore the request grid may not be suitable for every class, which can contribute to reduced classification performance.

Overall results show very competitive accuracies. Although tDBN wins most cases, both in terms of accuracy and F_1 score, the winning margins are not large. According to the Wilcoxon statistical test, the available results are not sufficient to reject the null hypothesis that the two classifiers have different mean accuracy (p=0.898). The results validate the use of sDBN algorithm in real data as it still can output better classifiers when faced with a state-of-the-art DBN training algorithm.

Table 5.5: Average accuracy, precision, recall, and F_1 score on test MTS benchmark datasets for classifiers trained with the sDBN and tDBN algorithms, along with standard deviation inside parenthesis. For each dataset, a five-fold stratified cross-validation evaluation methodology was used. Precision, recall, and F_1 on multiclass datasets were computed using a macro-averaging perspective. The best classifiers for each metric are highlighted in bold. All values are expressed in percentages.

Dataset	sDBN				tDBN with MDL			
	Acc	Pre	Rec	F_1	Acc	Pre	Rec	F_1
ArabicDigits	**78.1** (0.5)	**78.4** (0.6)	**78.1** (0.5)	**78.2** (0.6)	75.5 (0.6)	75.9 (0.7)	75.5 (0.6)	75.6 (0.6)
CharacterTrajectories	85.8 (1.7)	86.9 (1.3)	85.1 (1.7)	84.2 (2.0)	**86.3** (0.7)	**87.2** (0.8)	**85.2** (0.6)	**85.1** (0.7)
ECG	**76.5** (4.4)	**74.2** (4.8)	**76.1** (5.1)	**74.7** (4.9)	75.0 (7.6)	72.6 (8.4)	73.1 (9.1)	72.3 (8.7)
JapaneseVowels	86.7 (3.1)	89.0 (2.2)	86.3 (3.5)	86.7 (3.3)	**89.1** (1.3)	**89.6** (1.4)	**88.5** (1.7)	**88.8** (1.6)
Libras	**63.9** (6.7)	**65.1** (7.0)	**63.7** (6.5)	**62.2** (6.5)	62.2 (5.7)	63.2 (7.2)	62.2 (5.7)	60.1 (6.5)
NetFlow	81.9 (1.0)	74.4 (1.4)	78.8 (3.0)	75.9 (1.7)	**84.4** (2.6)	**77.2** (3.4)	**80.1** (4.7)	**78.4** (3.8)
UWave	92.3 (1.0)	92.5 (1.0)	92.3 (1.0)	92.3 (1.0)	**92.7** (0.4)	**92.8** (0.4)	**92.7** (0.4)	**92.7** (0.4)
Wafer	89.8 (0.8)	72.4 (6.4)	**61.4** (6.5)	63.5 (7.0)	**90.5** (1.7)	**77.0** (9.2)	**61.4** (6.4)	**64.5** (7.5)

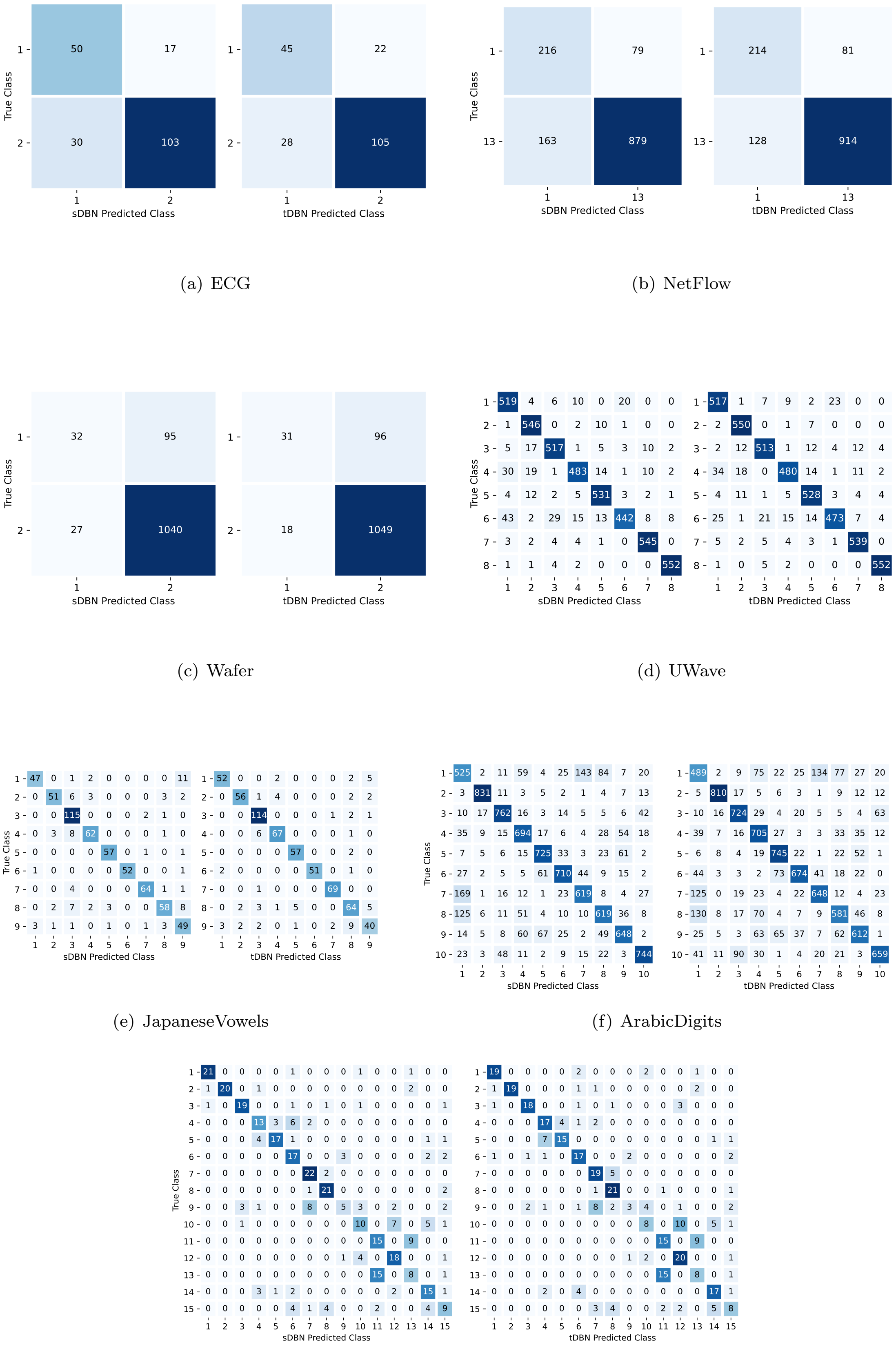


(a) ECG

(b) NetFlow

(c) Wafer

Figure 5.5: Average ROC plots comparing classifiers generated using `sDBN` and `tDBN` for the ECG, NetFlow and Wafer test datasets. The curves were averaged by fixing a grid of FPR values and computing the corresponding TPR using linear interpolation.



	sDBN Predicted Class	
True Class	1	2
1	50	17
2	30	103

	tDBN Predicted Class	
True Class	1	2
1	45	22
2	28	105

(a) ECG

	sDBN Predicted Class	
True Class	1	13
1	216	79
13	163	879

	tDBN Predicted Class	
True Class	1	13
1	214	81
13	128	914

(b) NetFlow

	sDBN Predicted Class	
True Class	1	2
1	32	95
2	27	1040

	tDBN Predicted Class	
True Class	1	2
1	31	96
2	18	1049

(c) Wafer

	sDBN Predicted Class							
True Class	1	2	3	4	5	6	7	8
1	519	4	6	10	0	20	0	0
2	1	546	0	2	10	1	0	0
3	5	17	517	1	5	3	10	2
4	30	19	1	483	14	1	10	2
5	4	12	2	5	531	3	2	1
6	43	2	29	15	13	442	8	8
7	3	2	4	4	1	0	545	0
8	1	1	4	2	0	0	0	552

	tDBN Predicted Class							
True Class	1	2	3	4	5	6	7	8
1	517	1	7	9	2	23	0	0
2	2	550	0	1	7	0	0	0
3	2	12	513	1	12	4	12	4
4	34	18	0	480	14	1	11	2
5	4	11	1	5	528	3	4	4
6	25	1	21	15	14	473	7	4
7	5	2	5	4	3	1	539	0
8	1	0	5	2	0	0	0	552

(d) UWave

	sDBN Predicted Class								
True Class	1	2	3	4	5	6	7	8	9
1	47	0	1	2	0	0	0	0	11
2	0	51	6	3	0	0	0	3	2
3	0	0	115	0	0	0	2	1	0
4	0	3	8	62	0	0	0	1	0
5	0	0	0	0	57	0	1	0	1
6	1	0	0	0	0	52	0	0	1
7	0	0	4	0	0	0	64	1	1
8	0	2	7	2	3	0	0	58	8
9	3	1	1	0	1	0	1	3	49

	tDBN Predicted Class								
True Class	1	2	3	4	5	6	7	8	9
1	52	0	0	2	0	0	0	2	5
2	0	56	1	4	0	0	0	2	2
3	0	0	114	0	0	0	1	2	1
4	0	0	6	67	0	0	0	1	0
5	0	0	0	0	57	0	0	2	0
6	2	0	0	0	0	51	0	1	0
7	0	0	1	0	0	0	69	0	0
8	0	2	3	1	5	0	0	64	5
9	3	2	2	0	1	0	2	9	40

(e) JapaneseVowels

	sDBN Predicted Class									
True Class	1	2	3	4	5	6	7	8	9	10
1	525	2	11	59	4	25	143	84	7	20
2	3	831	11	3	5	2	1	4	7	13
3	10	17	762	16	3	14	5	5	6	42
4	35	9	15	694	17	6	4	28	54	18
5	7	5	6	15	725	33	3	23	61	2
6	27	2	5	5	61	710	44	9	15	2
7	169	1	16	12	1	23	619	8	4	27
8	125	6	11	51	4	10	10	619	36	8
9	14	5	8	60	67	25	2	49	648	2
10	23	3	48	11	2	9	15	22	3	744

	tDBN Predicted Class									
True Class	1	2	3	4	5	6	7	8	9	10
1	489	2	9	75	22	25	134	77	27	20
2	5	810	17	5	6	3	1	9	12	12
3	10	16	724	29	4	20	5	5	4	63
4	39	7	16	705	27	3	3	33	35	12
5	6	8	4	19	745	22	1	22	52	1
6	44	3	3	2	73	674	41	18	22	0
7	125	0	19	23	4	22	648	12	4	23
8	130	8	17	70	4	7	9	581	46	8
9	25	5	3	63	65	37	7	62	612	1
10	41	11	90	30	1	4	20	21	3	659

(f) ArabicDigits

	sDBN Predicted Class														
True Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	21	0	0	0	0	1	0	0	0	1	0	0	1	0	0
2	1	20	0	1	0	0	0	0	0	0	0	0	2	0	0
3	1	0	19	0	0	1	0	1	0	1	0	0	0	0	1
4	0	0	0	13	3	6	2	0	0	0	0	0	0	0	0
5	0	0	0	4	17	1	0	0	0	0	0	0	0	1	1
6	0	0	0	0	0	17	0	0	3	0	0	0	0	2	2
7	0	0	0	0	0	0	22	2	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	21	0	0	0	0	0	0	2
9	0	0	3	1	0	0	8	0	5	3	0	2	0	0	2
10	0	0	1	0	0	0	0	0	0	10	0	7	0	5	1
11	0	0	0	0	0	0	0	0	0	0	15	0	9	0	0
12	0	0	0	0	0	0	0	0	1	4	0	18	0	0	1
13	0	0	0	0	0	0	0	0	0	0	15	0	8	0	1
14	0	0	0	3	1	2	0	0	0	0	0	2	0	15	1
15	0	0	0	0	0	4	1	4	0	0	2	0	0	4	9

	tDBN Predicted Class														
True Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	19	0	0	0	0	2	0	0	0	2	0	0	1	0	0
2	1	19	0	0	0	1	1	0	0	0	0	0	2	0	0
3	1	0	18	0	0	1	0	1	0	0	0	3	0	0	0
4	0	0	0	17	4	1	2	0	0	0	0	0	0	0	0
5	0	0	0	7	15	0	0	0	0	0	0	0	0	1	1
6	1	0	1	1	0	17	0	0	2	0	0	0	0	0	2
7	0	0	0	0	0	0	19	5	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	21	0	0	1	0	0	0	1
9	0	0	2	1	0	1	8	2	3	4	0	1	0	0	2
10	0	0	0	0	0	0	0	0	0	8	0	10	0	5	1
11	0	0	0	0	0	0	0	0	0	0	15	0	9	0	0
12	0	0	0	0	0	0	0	0	1	2	0	20	0	0	1
13	0	0	0	0	0	0	0	0	0	0	15	0	8	0	1
14	0	0	0	2	0	4	0	0	0	0	0	0	0	17	1
15	0	0	0	0	0	0	3	4	0	0	2	2	0	5	8

(g) Libras

Figure 5.6: Confusion matrices for ECG, NetFlow, Wafer, UWave, JapaneseVowels, ArabicDigits, and Libras. Squares with darker blue colors signal a larger number of observations classified as the position implies.

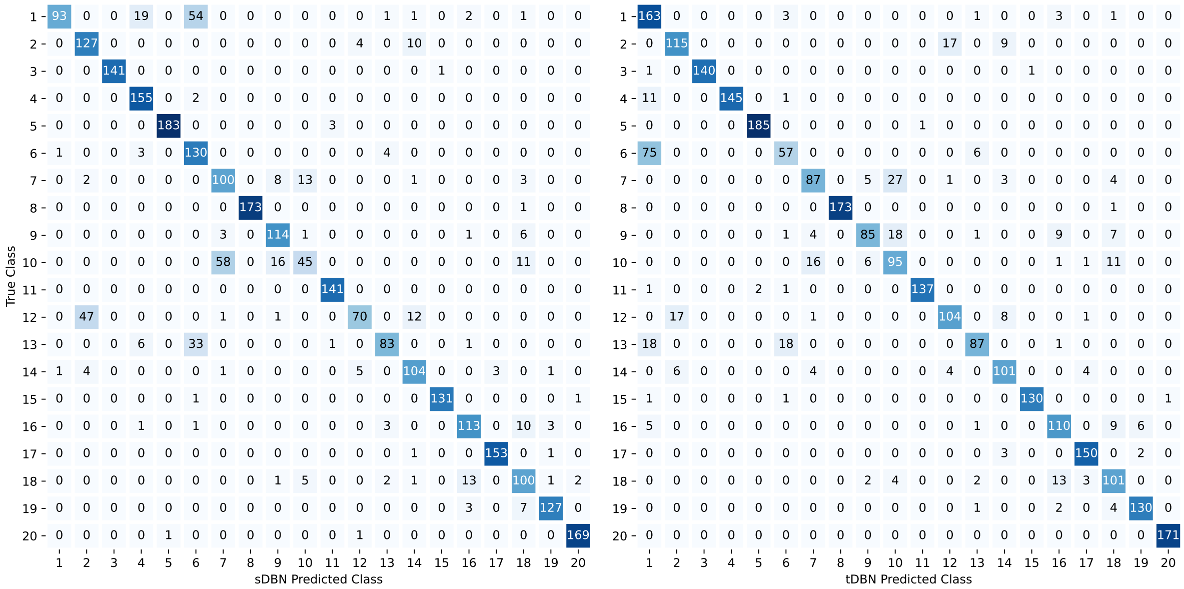


Figure 5.7: Confusion matrix for CharacterTrajectories.

5.3 Rheumatological health records dataset

Ankylosing spondylitis (AS) is a condition considered to be a rheumatological disease [62]. This disease is characterized by inflammation in skeleton joints and can lead to total fusion of the axial skeleton and consequent loss of spinal mobility. AS affects predominantly young adult males and has a social and economical impact as it usually leads to disability and loss of employment [63]. It has no cure and is typically treated with non-steroidal anti-inflammatory drugs [62]. For patients that no longer respond to typical therapy, treatment with biological agents is recommended [64].

Since it is a chronic disease with deep sociological impact, there is interest in modeling the disease using a mathematical framework [65] that can lead to early diagnosis, treatment outcome prediction, personalized treatment, better disease progression understanding, and hopefully reduce the impact of the disease in the patient's life.

`Reuma.pt` or the Rheumatic Diseases Portuguese Register, is a database of rheumatological patients health records in Portugal created by the portuguese society of rheumatology (SPR) operating since 2008 [66]. The registry includes patients of several diseases in the rheumatological scope, including rheumatoid arthritis, psoriatic arthritis, and ankylosing spondylitis.

To evaluate the `sDBN` performance on real data, a subset of records from the `Reuma.pt` AS dataset was cleaned and pre-processed. Table 5.6 summarizes the kept features and their associated discretization scheme. Data discretization was largely based on the proposed categories by Martins [67]. Painful and swelled articulations were discretized as a binary feature.

Data pre-processing

Provided data had to be cleaned as it is subject to filling errors and missing values. It is also continuous and has to be discretized.

Typical faults on the data include the use of non-numeric characters, for example, indicating that

Table 5.6: Summary of the features considered composed of measured clinical indicators from the Reuma.pt dataset for Ankylosing Spondylitis patients and used discretization. BASDAI and BASFI questionnaires are 6 and 10 individual features, respectively.

Feature name	Description	Discretization
BASDAI	The bath ankylosing spondylitis disease activity index (BASDAI) [68] is a metric to assess disease activity based on a patient answered questionnaire.	**0**: $[0,4[$; **1**: $[4,7[$; **2**: $[7,10]$
BASDAI Q1-Q6	Questionnaire filled for evaluating the BASDAI index.	**0**: $[0,4[$; **1**: $[4,7[$; **2**: $[7,10]$
BASFI	The bath ankylosing spondylitis functional index (BASFI) [69] is a metric to assess the level of a patient disability based on a self answered questionnaire.	**0**: $[0,3[$; **1**: $[3,7[$; **2**: $[7,10]$
BASFI Q1-Q10	Questionnaire filled for evaluating the BASFI index.	**0**: $[0,3[$; **1**: $[3,7[$; **2**: $[7,10]$
PCR	C-reactive protein (CRP) / mg/L	**0**: $[0,5[$; **1**: $[5,10[$; **2**: $[10,100]$; **3**: $[100,\infty[$
VS	Erythrocyte sedimentation rate (ESR) / mm/hour	**0**: $[0,6[$; **1**: $[6,17[$; **2**: $[17,\infty[$
TemCorticActivo	Whether the patient is medicated with corticosteroids	**0**: No; **1**: Yes
ArtTumefactas	Number of swelled articulations	**0**: $[0,1[$; **1**: $[1,\infty[$
ArtDolorosas	Number of painful articulations	**0**: $[0,1[$; **1**: $[1,\infty[$
EVAMedico	visual analogue scale (VAS) from the physician prespective	**0**: $[0,1[$; **1**: $[1,3[$; **2**: $[3,10]$
EVADoente	VAS from the patient prespective	**0**: $[0,2[$; **1**: $[2,5[$; **2**: $[5,10]$
ASDAS	The Ankylosing Spondylitis Disease Activity Score (ASDAS) [70] is a composite index, computed using the C-reactive protein value, applied to evaluate disease activity. See Equation (5.4).	**0**: $[0,1.3[$; **1**: $[1.3,2.1[$; **2**: $[2.1,3.5[$; **3**: $[3.5,\infty[$
ASDASvs	The ASDAS-ESR score is a composite index, computed using the ESR value, applied to evaluate disease activity.	**0**: $[0,1.3[$; **1**: $[1.3,2.1[$; **2**: $[2.1,3.5[$; **3**: $[3.5,\infty[$
NEW_BioActivo	Biological agent currently used for treatment.	Adalimumab, Certolizumab, Etanercept, Golimumab, Infliximab, Secucinumab, None

there is no result. It is also common that the observed value is specified to be in an interval. In those cases, a random value was assigned picked with uniform probability in the specified interval. CRP values were also sometimes specified directly as a negative outcome instead of a measured number. In those cases, a random number was picked in the appropriate discretization interval. Random numbers are a way to fill the data leading to the same final result because, after discretization, every possible value gets in the same category.

After basic input cleaning, missing values were filled using simple filling methods. For each patient, values from previous appointments were first forward carried to later appointments. After that, a backfill method was used to carry back observed values to previous appointments. Foward fill and backfill were only applied on data from a single patient. After this filling procedure, there were still observations containing missing values.

Subjects were also filtered by the number of appointments. On the examined data, there were patients with more than 80 identical medical evaluations. This was clearly an input error and should not be considered as part of the dataset. Additionally, patients with a low number of observations should be considered as outliers as the disease progression may not be normal or simply dropped treatment. Due to this, patients with less than four and more than 80 observations were disregarded.

Additionally, patients were filtered and stratified by applied treatment. Appointments in which the biological agent was changed were dropped, as well as subsequent medical evaluations. Treatment plans were the patient started with no treatment using biological agents, but then one was applied are kept.

Afterwards, all observations containing missing values were dropped. Figure 5.8 shows the final distribution of patients per treament. Groups treated with *Abatacept*, *Certolizumab*, *Rituximab*, *Secucinumab*, *Tocilizumab* and *Ustecinumab* were dropped due to the low level of observations. After discretization, the result was a discrete dataset with 1679 subjects and 27469 observations.

Experimental Setup

Two different sets of results were obtained. For every group of treatment, stationary DBNs with Markov lag of 1 and 2 were trained using the rheumatological dataset. The dataset used has a variable number of appointments per patient. Therefore, stationary DBNs are better for this data as only transitions are used to train the networks. Non-stationary network training without missing values, imperative in the proposed algorithm, forces a homogeneous number of appointments per subject.

The best lambda value according to each score was obtained using 10 fold cross-validation. Each cross-validation run used a path of 80 networks with a geometric grid of λ values with arbitrarily chosen limits to obtain a reasonable amount of edges. The log-likelihood and MDL scores were tested on the left-out dataset for each run. The λ with the best average score was selected. The resulting networks were inspected visually and qualitatively analyzed.

For a quantitative study, a classification task was designed aiming to predict the treatment outcome. To this end, stationary DBNs were trained on a transition dataset between two consecutive appointments. The best λ value was chosen similarly to the other experiments. For comparison purposes, networks were trained with the `tDBN` algorithm with different scores.

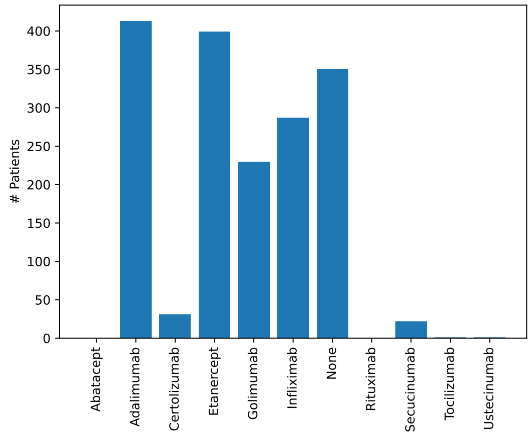


Figure 5.8: Number of patients per applied treatment on the final, pre-processed, `Reuma.pt` dataset.

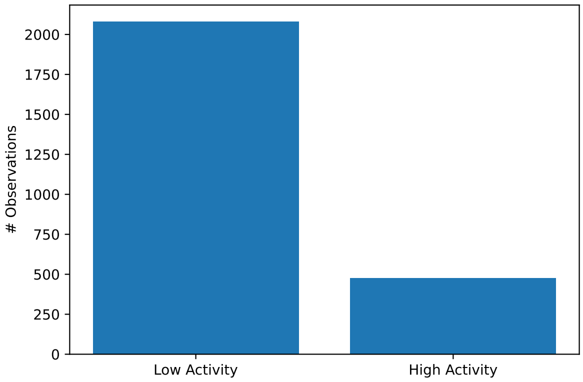


Figure 5.9: Number of observations per class on the final `Reuma.pt` dataset used for classification.

A class variable was designed discretizing the ASDAS feature differently. Instead of admitting four categories, only two were considered – low or high disease activity – being 3.5 the threshold between the two. This is the threshold to very high disease activity defined by Machado *et al.* [71]. Additionally, patients with more than 10 appointments were disregarded. Figure 5.9 shows the distribution of observations for each considered class. This class variable resulted in a very unbalanced dataset, with the low activity class representing over 81% of the observations. This is expected, as patients under treatment usually do not develop high disease activity. Nevertheless, predicting it is a relevant classification problem. Observations with low disease activity were undersampled at random to tackle the class imbalance. The final dataset used for the classification task had 956 observations.

To predict treatment outcome, the resulting network was queried for the probability of the class variable in the latter timestep using evidence only for the first timestep. The predicted class is the node level with the highest *a posteriori* probability, given the evidence. The maximum *a posteriori* (MAP) query was made using the Variable Elimination algorithm [72] implementation in the `pgmpy`[5] package [73].

Results

Figure 5.10 shows the best MDL network with lag 1 using the full `Reuma.pt` dataset. Figure 5.11 shows the best networks for patients treated with Adalimumab and Etanercept and Figure 5.12 for those treated with Golimumab and Infliximab. Full trained path, best log-likelihood networks and networks with lag 2 can be interactively explored on `http://jbsants.com/viewer/reuma`.

All networks are consistent, and there are no significant changes on the edges identified for each group. All trained networks have a high prevalence of identity edges, *i.e.*, edges originating in the same node but on the previous timestep. This is expected as the value of a particular clinical variable should influence its condition in the next medical consultation. There are very few non-identity inter-temporal edges, appearing only in Figure 5.10. The Golimumab group yields the sparsest network due to having

[5] Available in `https://github.com/pgmpy/pgmpy`.

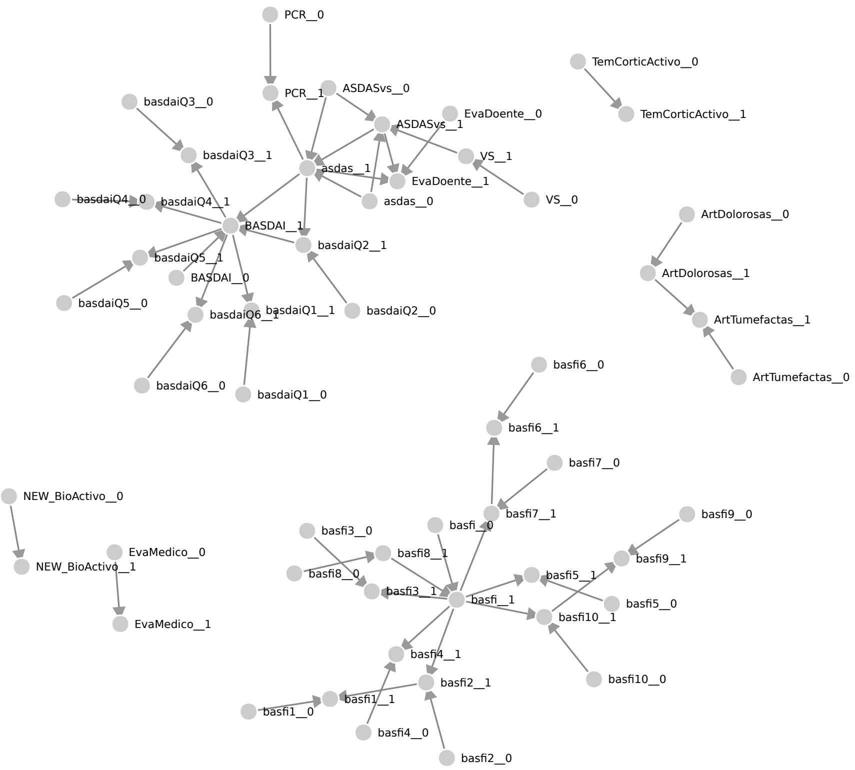


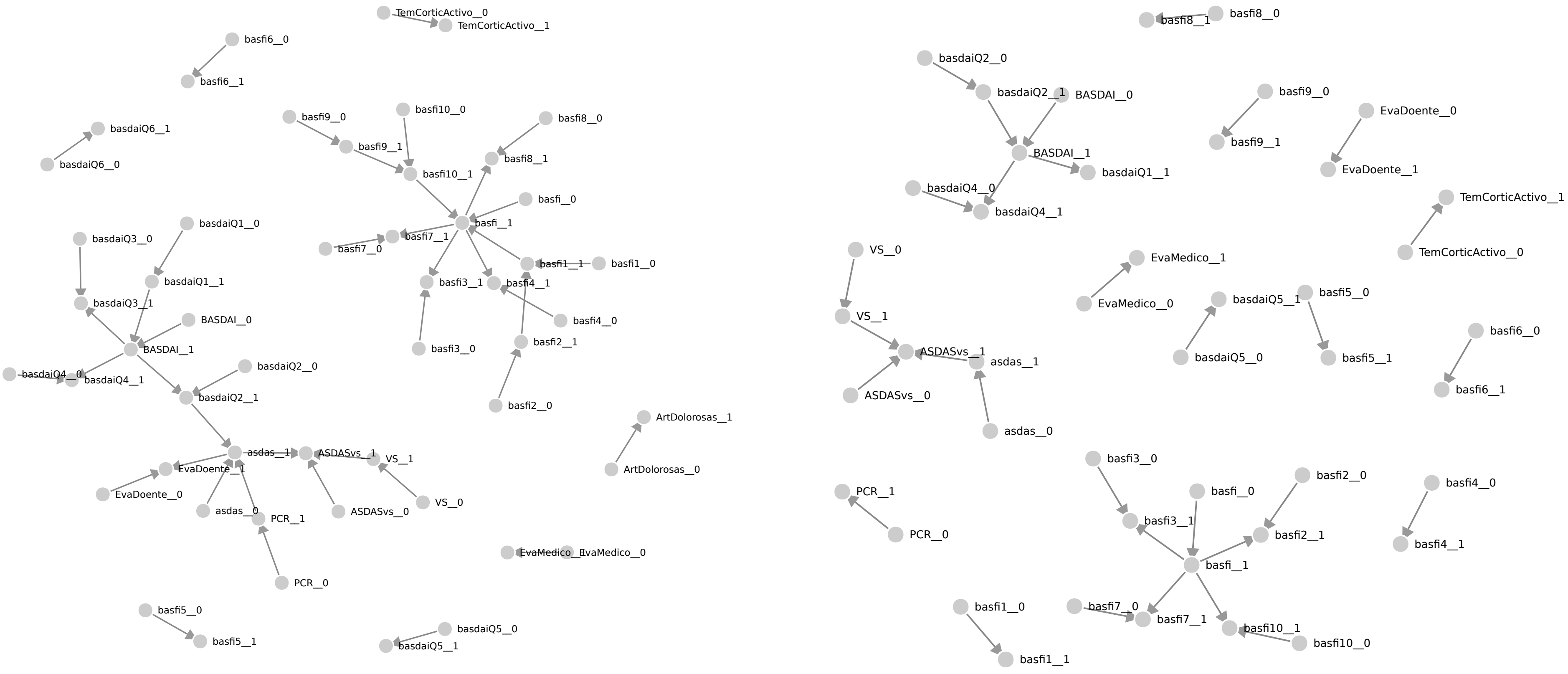
Figure 5.10: Transition network trained with data from the Reuma.pt dataset considering only patients that were treated by a single or none biological agent. The dataset used was obtained joining all considered treatment groups. The networks were trained considering a Markov lag of 1. To determine a single network, a cross-validation step was conducted with 10 folds. The shown network is the one with the best mean MDL score on the folds test dataset. Nodes are labelled as `variable name__timestep`.

fewer patients and, subsequently, observations lead to more conservative networks according to the MDL criteria. Observational evidence is not sufficient to support the increase in complexity of the model.

All networks identify consistent clusters of nodes around the BASDAI and BASFI indices. These nodes correspond to the individual questions on each questionnaire. This proves the algorithm's ability to find good relationships between variables as the indices are determined using the individual questions, leading to an evident relationship. The proposed method was able to identify this relationship without any prior knowledge of it. This also shows that resulting networks have intra-temporal v-structures, which are not allowed by the `tDBN` algorithm and is a clear advantage of the proposed algorithm. Clusters surrounding BASDAI and BASFI are not possible to be recovered by algorithms restricted to tree structures. In this dataset, this relationship is known *a priori*. However, when that is not the case, `tDBN` can fail to fully characterize the underlying model due to the substantial restrictions.

The same happens for the ASDAS, and ASDAS-ESR indices as the corresponding clinical indicators applied in the calculation (CRP and ESR, respectively) are tied to both indices. In four out of the five shown networks, ASDAS is associated with the VAS level from the patient perspective and Q2 from the BASDAI questionnaire. The formula used for calculating ASDAS is

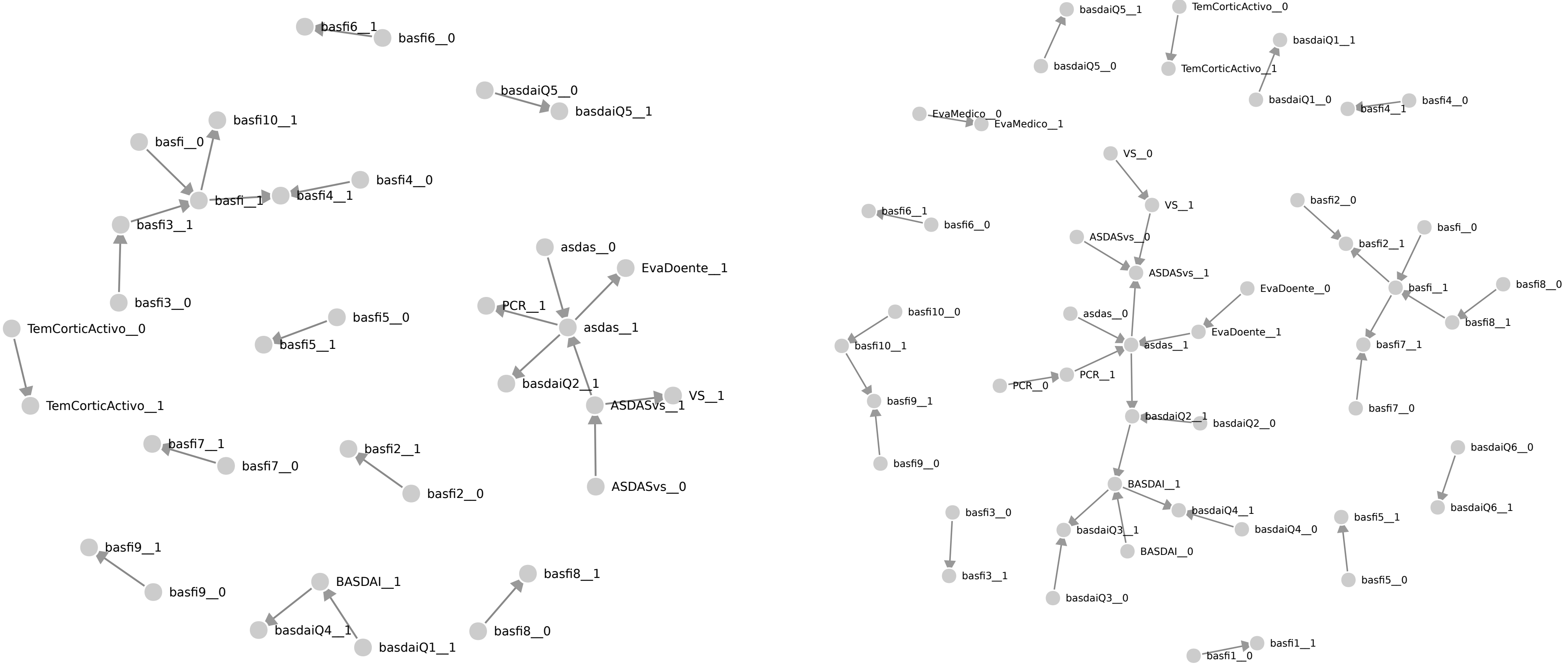
$$\begin{aligned}\text{ASDAS} = {} & 0.12 \cdot \text{BASDAI\_Q2} + 0.06 \cdot \text{BASDAI\_Q6} + 0.11 \cdot \text{EVADoente} + 0.07 \cdot \text{BASDAI\_Q3} \\ & + 0.58 \log\left(\text{PCR} + 1\right),\end{aligned} \tag{5.4}$$



(a) Adalimumab

(b) Etanercept

Figure 5.11: Transition networks obtained from the `Reuma.pt` dataset considering only the patients that were treated by a single biological agent and splitting them into different datasets. This figure shows the networks for Adalimumab and Etanercept. The networks were trained considering a Markov lag of 1. To determine a single network, a cross-validation step was conducted with 10 folds. The shown networks are the ones with the best mean MDL score on the folds test dataset. Nodes are labelled as `variable name__timestep`.



(a) Golimumab

(b) Infliximab

Figure 5.12: Transition networks obtained from the `Reuma.pt` dataset considering only the patients that were treated by a single biological agent. This figure shows the networks for Infliximab and Golimumab. The networks were trained considering a Markov lag of 1. To determine a single network, a cross-validation step was conducted with 10 folds. The shown networks are the ones with the best mean MDL score on the folds test dataset. Nodes are labelled as `variable name__timestep`.

using the `Reuma.pt` feature terminology. In fact, these two variables also have a considerable influence on ASDAS, corroborating that the algorithm outputs a good structure. Edges between ASDAS and ASDAS-ESR can be explained by similarities in the used formula for the latter, which also account for VAS and BASDAI Q2, Q3, and Q6. These results are not consistent with Martins [67] as, in most networks, a relationship is found between ASDAS and VAS from the doctor's perspective instead. Since ASDAS is calculated with the patient perspective, the shown results are more reasonable. Nevertheless, the trained network with the least amount of regularization for *Golimumab* also found an edge between ASDAS and the physician VAS.

No network included a relationship with corticosteroids use. An identity edge is present from the previous timestep because, most times, the value is the same as in the previous timestep. Martins [67] also found that corticosteroids use is very disconnected from the considered features. This suggests that the use of these drugs does not influence disease progression. In fact, the use of steroids can provide pain relief, but it has not been proved to be effective with AS [62].

Regarding the classification task, Table 5.7 shows performance metrics for the trained classifiers. Figure 5.13 compares the mean ROC curves of the classifiers. All classification algorithms accurately predicted the disease activity in the medical consultation, with average accuracies greater than 80%. Given that a balanced binary dataset was used, this shows that these classifiers can be helpful in aiding clinical decisions. The used imputation and balancing methods are very naive, and overall accuracy might improve using different approaches, suggesting future work to assess the impact of such pre-processing on the classifiers' performance.

Table 5.7: Average classifier performance on the `Reuma.pt` test dataset over a 10 fold cross-validation. Values inside the parenthesis are the standard deviation over all runs. The best classifier for each metric is emphasized in bold. Both algorithms were trained assuming a stationary process with Markov lag 1. The sDBN algorithm was trained cross-validating the training dataset with 10 folds to find the optimal lambda value according to the ϕ_{LL} or the ϕ_{MDL} score. Every node was limited to having a maximum of three parents in the directing step. The tDBN algorithm was run limiting the number of parents from the previous timestep to just 1.

Algorithm	Accuracy	Precision	Recall	F_1 score	AUC
sDBN + LL	0.875 (0.048)	**0.890** (0.046)	0.869 (0.083)	0.877 (0.050)	0.913 (0.039)
sDBN + MDL	0.838 (0.151)	0.871 (0.159)	0.873 (0.092)	0.856 (0.105)	0.846 (0.125)
tDBN + LL	0.873 (0.050)	0.886 (0.051)	0.873 (0.083)	0.876 (0.050)	0.920 (0.036)
tDBN + MDL	**0.882** (0.065)	0.887 (0.054)	**0.888** (0.093)	**0.885** (0.064)	**0.923** (0.037)

The ROC curves show that the `sDBN` classifier chosen with ϕ_{LL} is very competitive with both tDBN classifiers. Using the MDL score to choose the best network is not the best approach, leading to a clearly worse classifier. `tDBN` with MDL score, on the other hand, is distinctively the best classifier. Note that MDL is computed using the training data in this case, not on the test dataset. The proposed algorithm wins in the precision score and is within 0,7% of the best classifier in terms of accuracy. Additionally, it performs better than `tDBN` with the LL score, despite having a lower AUC score. These results, once again, validate the performance of `sDBN` and affirms it as an alternative learning algorithm.

In this particular classification task, `tDBN` with MDL is the best classifier. Not only has greater

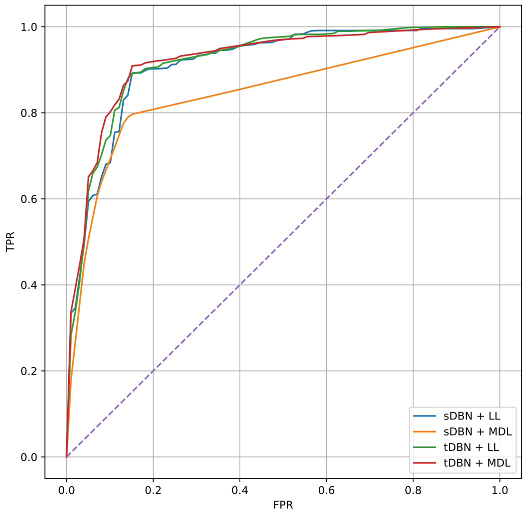


Figure 5.13: Mean ROC curve for all `Reuma.pt` classifiers evaluated on the test dataset over a 10 fold cross-validation for all the considered classifiers. The mean curve was computed fixing a linear grid of values for FPR and then averaging the TPR values for every point in the grid determined using linear interpolation.

accuracy, but it also has no additional overhead of discovering the optimal λ. Also, it was trained admitting only one parent from the previous timestep, the fastest configuration possible. Therefore, `sDBN` does not prove itself useful in this context, as the dimensionality is still tractable to be trained by other methods with better results. The lower performance of `sDBN` can be linked to difficulty in choosing the optimal λ. Additional studies have to be done to determine if increasing the number of folds on the cross-validation step to find the optimal network can lead to a better result. The number of networks in the requested path can also be increased. Moreover, as the author of `tDBN` suggests, tree-like classifiers have been shown to perform very well [48]. Structures with added complexity may still model the underlying process correctly. However, they can lead to a final classifier with a higher variance that overfits the training data. These results in medical datasets are consistent with the results found with multinet classification on MTS benchmark datasets shown in Section 5.2.

Chapter 6

Conclusions

6.1 Achievements

This dissertation introduces a new pipeline to train dynamic Bayesian networks based on recently developed regularization-based approaches to Bayesian network training. The algorithm successfully applies temporal causation restrictions to these methods, allowing high-dimensional DBN training using time-series data. In this work, an implementation of the new methodology was developed and validated in synthetic and real data.

The proposed method achieves great structure identification in low and high-dimensional contexts. This thesis reports excellent results in DBNs with up to 300 nodes per timestep in artificial datasets, dimensions simply unattainable by state-of-the-art techniques like `tDBN` in a reasonable time. Even in low dimensions, `sDBN` excels in structure discovery with considerably lower training times. The new algorithm also identified suitable transition networks on rheumatological data that uncover known relationships between dataset features. These structures are full of intra-temporal v-structures that optimal tree algorithms can't recover.

Classification results, on the other hand, do not favor the use of these techniques on datasets with low dimensions. Even though a competitive performance was achieved, the added overhead of cross-validation to find the optimal regularization level is a clear disadvantage of this method. Also, tree classifiers achieve good performances as more complex structures are prone to overfit and offer better accuracy variance across datasets.

6.2 Future Work

The validation results of this dissertation are by no means extensive. This thesis focused on training stationary networks with Markov lag 1, so additional work can be done to test the algorithm for networks with different configurations. Also, non-stationary network training is possible, but it is not tested. Finding public high-dimensional temporal datasets proved to be a challenge. Therefore, further studies to assess the performance of the algorithm in these kinds of datasets are needed. As suggested, the pre-processing used in the Ankylosing Spondylitis may not be the most suitable for the classification task. Therefore, more detailed studies to treatment outcome prediction are a possibility.

The developed C++ implementation has a huge improvement potential. The skeleton directing step can also be parallelized to improve performance. Additional work is needed to enhance single-core performance, like CPU cache hit rate analysis. Since the algorithm heavily relies on matrix multiplication, graphics processing unit (GPU) acceleration can be leveraged to reduce computation time.

The proposed algorithm conducts a full optimization procedure on the multinomial logistic parametrization, but only the support of the parameter vector is considered when directing the network. It should be possible to improve the method by stopping the optimization if the active set did not change in a certain number of iterations. Another possibility is to change to stopping criteria, accepting less precise solutions.

Following recent advances in characterizing graph acyclicity using differentiable functions [35], new studies can be conducted joining these restrictions with the reparametrization of DBNs to introduce new network training methods using gradient descent. These methods can provide different local minimum escaping procedures than greedy methods and can serve as alternative optimization procedures that may benefit from the latest research in gradient optimization techniques.

Bibliography

[1] J. Han, M. Kamber, and J. Pei. 1 - Introduction. In J. Han, M. Kamber, and J. Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 1–38. Morgan Kaufmann, Boston, Jan. 2012. ISBN 978-0-12-381479-1. doi: 10.1016/B978-0-12-381479-1.00001-0. URL https://www.sciencedirect.com/science/article/pii/B9780123814791000010.

[2] S. M. Stigler. Gauss and the Invention of Least Squares. *The Annals of Statistics*, 9(3):465–474, 1981. ISSN 0090-5364. URL https://www.jstor.org/stable/2240811. Publisher: Institute of Mathematical Statistics.

[3] A. M. Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes.* F. Didot, 1805.

[4] S. M. Stigler. The Epic Story of Maximum Likelihood. *Statistical Science*, 22(4), Nov. 2007. ISSN 0883-4237. doi: 10.1214/07-STS249. URL http://arxiv.org/abs/0804.2996. arXiv: 0804.2996.

[5] R. A. Fisher and E. J. Russell. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222(594-604):309–368, Jan. 1922. doi: 10.1098/rsta.1922.0009. URL https://royalsocietypublishing.org/doi/10.1098/rsta.1922.0009. Publisher: Royal Society.

[6] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, apr 2011. ISBN 9780387310732. URL https://www.xarg.org/ref/a/0387310738/.

[7] J. A. Nelder and R. W. M. Wedderburn. Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384, 1972. ISSN 0035-9238. doi: 10.2307/2344614. URL https://www.jstor.org/stable/2344614. Publisher: [Royal Statistical Society, Wiley].

[8] J. Berkson. Application of the Logistic Function to Bio-Assay. *Journal of the American Statistical Association*, 39(227):357–365, 1944. ISSN 0162-1459. doi: 10.2307/2280041. URL https://www.jstor.org/stable/2280041. Publisher: [American Statistical Association, Taylor & Francis, Ltd.].

[9] A. E. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67, 1970. ISSN 0040-1706. doi: 10.2307/1267351. URL https://www.jstor.org/stable/1267351. Publisher: [Taylor & Francis, Ltd., American Statistical Association, American Society for Quality].

[10] T. Hastie, R. Tibshirani, and J. Friedman. Linear Methods for Regression. In T. Hastie, R. Tibshirani, and J. Friedman, editors, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics, pages 43–99. Springer, New York, NY, 2009. ISBN 978-0-387-84858-7. doi: 10.1007/978-0-387-84858-7_3. URL https://doi.org/10.1007/978-0-387-84858-7_3.

[11] R. Tibshirani. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 0035-9246. URL https://www.jstor.org/stable/2346178. Publisher: [Royal Statistical Society, Wiley].

[12] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, Feb. 2006. ISSN 1369-7412, 1467-9868. doi: 10.1111/j.1467-9868.2005.00532.x. URL http://doi.wiley.com/10.1111/j.1467-9868.2005.00532.x.

[13] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0934613737.

[14] R. D. Shachter and C. R. Kenley. Gaussian Influence Diagrams. *Management Science*, 35(5):527–550, May 1989. ISSN 0025-1909. doi: 10.1287/mnsc.35.5.527. URL https://pubsonline.informs.org/doi/abs/10.1287/mnsc.35.5.527. Publisher: INFORMS.

[15] D. Geiger and D. Heckerman. Learning Gaussian Networks. *arXiv:1302.6808 [cs, stat]*, Feb. 2013. URL http://arxiv.org/abs/1302.6808. arXiv: 1302.6808.

[16] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning.* The MIT Press, 2009. ISBN 0262013193.

[17] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search.* MIT press, 2nd edition, 2000.

[18] M. Scanagatta, A. Salmerón, and F. Stella. A survey on Bayesian network structure learning from data. *Progress in Artificial Intelligence*, 8(4):425–439, Dec. 2019. ISSN 2192-6360. doi: 10.1007/s13748-019-00194-y. URL https://doi.org/10.1007/s13748-019-00194-y.

[19] M. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks: Search methods and experimental results. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, January 1995. URL https://www.microsoft.com/en-us/research/publication/learning-bayesian-networks-search-methods-and-experimental-results/.

[20] R. W. Robinson. Counting labeled acyclic digraphs. *New Directions in the Theory of Graphs*, pages 239–273, 1973.

[21] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, May 1968. ISSN 1557-9654. doi: 10.1109/TIT.1968.1054142. Conference Name: IEEE Transactions on Information Theory.

[22] N. Friedman, I. Nachman, and D. Pe'er. Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. *arXiv:1301.6696 [cs, stat]*, Jan. 2013. URL http://arxiv.org/abs/1301.6696. arXiv: 1301.6696.

[23] 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, Oct. 2006. ISSN 1573-0565. doi: 10.1007/s10994-006-6889-7. URL https://doi.org/10.1007/s10994-006-6889-7.

[24] G. Schwarz. Estimating the Dimension of a Model. *Annals of Statistics*, 6(2):461–464, Mar. 1978. ISSN 0090-5364, 2168-8966. doi: 10.1214/aos/1176344136. URL https://projecteuclid.org/euclid.aos/1176344136. Publisher: Institute of Mathematical Statistics.

[25] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, Jan. 1986. ISSN 0305-0548. doi: 10.1016/0305-0548(86)90048-1. URL http://www.sciencedirect.com/science/article/pii/0305054886900481.

[26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220 (4598):671–680, May 1983. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.220.4598.671. URL https://science.sciencemag.org/content/220/4598/671. Publisher: American Association for the Advancement of Science Section: Articles.

[27] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, Sept. 1995. ISSN 1573-0565. doi: 10.1007/BF00994016. URL https://doi.org/10.1007/BF00994016.

[28] D. M. Chickering. Learning Equivalence Classes of Bayesian-Network Structures. *Machine Learning Research*, page 54, 2002.

[29] N. Meinshausen and P. Bühlmann. High-dimensional graphs and variable selection with the Lasso. *Annals of Statistics*, 34(3):1436–1462, June 2006. ISSN 0090-5364, 2168-8966. doi: 10.1214/009053606000000281. URL https://projecteuclid.org/euclid.aos/1152540754. Publisher: Institute of Mathematical Statistics.

[30] M. Schmidt, A. Niculescu-Mizil, and K. Murphy. Learning graphical model structure using L1-regularization paths. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI'07, pages 1278–1283, Vancouver, British Columbia, Canada, July 2007. AAAI Press. ISBN 978-1-57735-323-2.

[31] M. Schmidt. *Graphical Model Structure Learning with ℓ_1-Regularization.* PhD thesis, University of British Columbia, 2010.

[32] F. Fu and Q. Zhou. Learning Sparse Causal Gaussian Networks With Experimental Intervention: Regularization and Coordinate Descent. *Journal of the American Statistical Association*, 108(501): 288–300, Mar. 2013. ISSN 0162-1459. doi: 10.1080/01621459.2012.754359. URL https://amstat.tandfonline.com/doi/full/10.1080/01621459.2012.754359. Publisher: Taylor & Francis.

[33] J. Gu, F. Fu, and Q. Zhou. Penalized estimation of directed acyclic graphs from discrete data. *Statistics and Computing*, 29(1):161–176, Jan. 2019. ISSN 1573-1375. doi: 10.1007/s11222-018-9801-y. URL `https://doi.org/10.1007/s11222-018-9801-y`.

[34] B. Aragam, J. Gu, and Q. Zhou. Learning Large-Scale Bayesian Networks with the sparsebn Package. *Journal of Statistical Software*, 91(11), 2019. ISSN 1548-7660. doi: 10.18637/jss.v091.i11. URL `http://arxiv.org/abs/1703.04025`. arXiv: 1703.04025.

[35] X. Zheng, B. Aragam, P. K. Ravikumar, and E. P. Xing. DAGs with NO TEARS: Continuous Optimization for Structure Learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9472–9483. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/8157-dags-with-no-tears-continuous-optimization-for-structure-learning.pdf`.

[36] N. A. Loghmanpour, M. K. Kanwar, M. J. Druzdzel, R. L. Benza, S. Murali, and J. F. Antaki. A New Bayesian Network-Based Risk Stratification Model for Prediction of Short-Term and Long-Term LVAD Mortality. *ASAIO Journal*, 61(3):313–323, June 2015. ISSN 1058-2916. doi: 10.1097/MAT.0000000000000209. URL `https://journals.lww.com/asaiojournal/fulltext/2015/05000/A_New_Bayesian_Network_Based_Risk_Stratification.13.aspx`.

[37] F. L. Seixas, B. Zadrozny, J. Laks, A. Conci, and D. C. Muchaluat Saade. A Bayesian network decision model for supporting the diagnosis of dementia, Alzheimer's disease and mild cognitive impairment. *Computers in Biology and Medicine*, 51:140–158, Aug. 2014. ISSN 0010-4825. doi: 10.1016/j.compbiomed.2014.04.010. URL `http://www.sciencedirect.com/science/article/pii/S0010482514000961`.

[38] P. Fuster-Parra, P. Tauler, M. Bennasar-Veny, A. Ligęza, A. A. López-González, and A. Aguiló. Bayesian network modeling: A case study of an epidemiologic system analysis of cardiovascular risk. *Computer Methods and Programs in Biomedicine*, 126:128–142, Apr. 2016. ISSN 0169-2607. doi: 10.1016/j.cmpb.2015.12.010. URL `http://www.sciencedirect.com/science/article/pii/S0169260715301140`.

[39] Z. Yang, Z. Yang, and J. Yin. Realising advanced risk-based port state control inspection using data-driven Bayesian networks. *Transportation Research Part A: Policy and Practice*, 110:38–56, Apr. 2018. ISSN 0965-8564. doi: 10.1016/j.tra.2018.01.033. URL `http://www.sciencedirect.com/science/article/pii/S0965856417309047`.

[40] N. Friedman, K. Murphy, and S. Russell. Learning the structure of dynamic probabilistic networks. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, UAI'98, pages 139–147, San Francisco, CA, USA, July 1998. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-555-8.

[41] N. Dojer. Learning Bayesian Networks Does Not Have to Be NP-Hard. In R. Královič and P. Urzyczyn, editors, *Mathematical Foundations of Computer Science 2006*, Lecture Notes in Com-

puter Science, pages 305–314, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-37793-1. doi: 10.1007/11821069_27.

[42] N. X. Vinh, M. Chetty, R. Coppel, and P. P. Wangikar. Polynomial Time Algorithm for Learning Globally Optimal Dynamic Bayesian Network. In B.-L. Lu, L. Zhang, and J. Kwok, editors, *Neural Information Processing*, pages 719–729, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-24965-5.

[43] K. P. Murphy. *Dynamic Bayesian networks: representation, inference and learning.* PhD thesis, University of California, 2002.

[44] J. L. Monteiro, S. Vinga, and A. M. Carvalho. Polynomial-time algorithm for learning optimal tree-augmented dynamic Bayesian networks. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI'15)*, pages 622–631, 2015.

[45] M. Sousa and A. M. Carvalho. Polynomial-Time Algorithm for Learning Optimal BFS-Consistent Dynamic Bayesian Networks. *Entropy*, 20(4):274, Apr. 2018. doi: 10.3390/e20040274. URL https://www.mdpi.com/1099-4300/20/4/274. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[46] T. Leão, S. C. Madeira, M. Gromicho, M. de Carvalho, and A. M. Carvalho. Learning dynamic Bayesian networks from time-dependent and time-independent data: Unraveling disease progression in Amyotrophic Lateral Sclerosis. *Journal of Biomedical Informatics*, 117:103730, May 2021. ISSN 1532-0464. doi: 10.1016/j.jbi.2021.103730. URL https://www.sciencedirect.com/science/article/pii/S1532046421000599.

[47] D. Geiger and D. Heckerman. Knowledge representation and inference in similarity networks and Bayesian multinets. *Artificial Intelligence*, 82(1):45–74, Apr. 1996. ISSN 0004-3702. doi: 10.1016/0004-3702(95)00014-3. URL https://www.sciencedirect.com/science/article/pii/0004370295000143.

[48] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29 (2):131–163, Nov. 1997. ISSN 1573-0565. doi: 10.1023/A:1007465528199. URL https://doi.org/10.1023/A:1007465528199.

[49] H. Wang and C. Leng. A note on adaptive group lasso. *Computational Statistics & Data Analysis*, 52(12):5277–5286, Aug. 2008. ISSN 0167-9473. doi: 10.1016/j.csda.2008.05.006. URL https://www.sciencedirect.com/science/article/pii/S0167947308002582.

[50] H. Zou. The Adaptive Lasso and Its Oracle Properties. *Journal of the American Statistical Association*, 101(476):1418–1429, Dec. 2006. ISSN 0162-1459. doi: 10.1198/016214506000000735. URL https://doi.org/10.1198/016214506000000735. Publisher: Taylor & Francis _eprint: https://doi.org/10.1198/016214506000000735.

[51] L. Meier, S. V. D. Geer, P. Bühlmann, and E. T. H. Zürich. The group Lasso for logistic regression. *Journal of the Royal Statistical Society, Series B*, 2008.

[52] W. Karush. Minima of functions of several variables with inequalities as side constraints. Master's thesis, Depatment of Mathematics, University of Chicago, Chicago, IL, 1939.

[53] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *2nd Berkeley Symposium*, pages 481 – 492. Berkeley: University of California Press, 1951.

[54] P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1):387–423, Mar. 2009. ISSN 1436-4646. doi: 10.1007/s10107-007-0170-0. URL `https://doi.org/10.1007/s10107-007-0170-0`.

[55] J. Nocedal. Updating Quasi-Newton Matrices with Limited Storage. *Mathematics of Computation*, 35(151):773–782, 1980. ISSN 0025-5718. doi: 10.2307/2006193. URL `https://www.jstor.org/stable/2006193`. Publisher: American Mathematical Society.

[56] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, Aug. 1989. ISSN 1436-4646. doi: 10.1007/BF01589116. URL `https://doi.org/10.1007/BF01589116`.

[57] F. Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945. ISSN 0099-4987. doi: 10.2307/3001968. URL `https://www.jstor.org/stable/3001968`. Publisher: [International Biometric Society, Wiley].

[58] D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[59] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The ucr time series classification archive, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.

[60] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, Oct. 2007. ISSN 1573-756X. doi: 10.1007/s10618-007-0064-z. URL `https://doi.org/10.1007/s10618-007-0064-z`.

[61] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, July 1997. ISSN 0031-3203. doi: 10.1016/S0031-3203(96)00142-2. URL `https://www.sciencedirect.com/science/article/pii/S0031320396001422`.

[62] F. Mahmood and P. Helliwell. Ankylosing Spondylitis: A review. *European Medical Journal*, pages 134–139, Nov. 2017.

[63] E. L. Healey, K. L. Haywood, K. P. Jordan, A. Garratt, and J. C. Packham. Impact of ankylosing spondylitis on work in patients across the UK. *Scandinavian Journal of Rheumatology*, 40(1):34–40, Jan. 2011. ISSN 1502-7732. doi: 10.3109/03009742.2010.487838.

[64] A. R. Cravo, V. Tavares, and J. C. d. Silva. Terapêutica anti-tnf alfa na espondilite anquilosante. *Acta Médica Portuguesa*, 19:141–150, 2006.

[65] A. Deodhar, M. Rozycki, C. Garges, O. Shukla, T. Arndt, T. Grabowsky, and Y. Park. Use of machine learning techniques in the development and refinement of a predictive model for early diagnosis of ankylosing spondylitis. *Clinical Rheumatology*, 39(4):975–982, Apr. 2020. ISSN 1434-9949. doi: 10.1007/s10067-019-04553-x. URL `https://doi.org/10.1007/s10067-019-04553-x`.

[66] H. Canhão, A. Faustino, F. Martins, and J. E. Fonseca. Reuma.pt — the rheumatic diseases portuguese register. *Acta Reumatológica Portuguesa*, 36:45–56, 2011.

[67] A. R. L. B. Martins. Dynamic Bayesian networks for clinical decision support on ankylosing spondylitis patients under biological therapies. Master's thesis, Insituto Superior Técnico, Lisboa, 2021.

[68] S. Garrett, T. Jenkinson, L. G. Kennedy, H. Whitelock, P. Gaisford, and A. Calin. A new approach to defining disease status in ankylosing spondylitis: the Bath Ankylosing Spondylitis Disease Activity Index. *The Journal of Rheumatology*, 21(12):2286–2291, Dec. 1994. ISSN 0315-162X.

[69] A. Calin, S. Garrett, H. Whitelock, L. G. Kennedy, J. O'Hea, P. Mallorie, and T. Jenkinson. A new approach to defining functional ability in ankylosing spondylitis: the development of the Bath Ankylosing Spondylitis Functional Index. *The Journal of Rheumatology*, 21(12):2281–2285, Dec. 1994. ISSN 0315-162X.

[70] C. Lukas, R. Landewé, J. Sieper, M. Dougados, J. Davis, J. Braun, S. v. d. Linden, D. v. d. Heijde, and f. t. A. o. S. i. Society. Development of an ASAS-endorsed disease activity score (ASDAS) in patients with ankylosing spondylitis. *Annals of the Rheumatic Diseases*, 68(1):18–24, Jan. 2009. ISSN 0003-4967, 1468-2060. doi: 10.1136/ard.2008.094870. URL `https://ard.bmj.com/content/68/1/18`. Publisher: BMJ Publishing Group Ltd Section: Clinical and epidemiological research.

[71] P. Machado, R. Landewé, E. Lie, T. K. Kvien, J. Braun, D. Baker, D. v. d. Heijde, and f. t. A. o. S. i. Society. Ankylosing Spondylitis Disease Activity Score (ASDAS): defining cut-off values for disease activity states and improvement scores. *Annals of the Rheumatic Diseases*, 70(1):47–53, Jan. 2011. ISSN 0003-4967, 1468-2060. doi: 10.1136/ard.2010.138594. URL `https://ard.bmj.com/content/70/1/47`. Publisher: BMJ Publishing Group Ltd Section: Clinical and epidemiological research.

[72] N. L. Zhang and D. Poole. A simple approach to Bayesian network computations, 1994.

[73] A. Ankan and A. Panda. pgmpy: Probabilistic graphical models using python. In *Proceedings of the 14th Python in Science Conference (SCIPY 2015)*. Citeseer, 2015.