DANIELE CODECASA

CONTINUOUS TIME BAYESIAN NETWORK CLASSIFIERS

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer science.

Dipartimento di Informatica, Sistemistica e Comunicazione

Università degli Studi di Milano-Bicocca

Dottorato di Ricerca in Informatica – Ciclo XXVI

Continuous Time Bayesian Network Classifiers

Daniele Codecasa

Advisor: **Fabio Stella**

Tutor: **Gabriella Pasi**

To my parents,
who always supported me.

ABSTRACT

Streaming data are relevant to finance, computer science, and engineering, while they are becoming increasingly important to medicine and biology. Continuous time Bayesian networks are designed for analyzing efficiently multivariate streaming data, exploiting the conditional independencies in continuous time homogeneous Markov processes. Continuous time Bayesian network classifiers are a specialization of continuous time Bayesian networks designed for multivariate streaming data classification when time duration of events matters and the class occurs in the future.

Continuous time Bayesian network classifiers are presented and analyzed. Structural learning is introduced for this class of models when complete data are available. A conditional log-likelihood scoring is derived to improve the marginal log-likelihood structural learning on continuous time Bayesian network classifiers. The expectation maximization algorithm is developed to address the unsupervised learning of continuous time Bayesian network classifiers when the class is unknown.

Performances of continuous time Bayesian network classifiers in the case of classification and clustering are analyzed with the help of a rich set of numerical experiments on synthetic and real data sets. Continuous time Bayesian network classifiers learned by maximizing marginal log-likelihood and conditional log-likelihood are compared with continuous time naive Bayes and dynamic Bayesian networks. Results show that the conditional log-likelihood scoring combined with Bayesian parameter estimation outperforms marginal log-likelihood scoring and dynamic Bayesian networks in the case of supervised classification. Conditional log-likelihood scoring becomes even more effective when the amount of available data is limited. Continuous time Bayesian network classifiers outperform dynamic Bayesian networks even on data sets generated from discrete time models. Clustering results show that in the case of unsupervised learning the marginal log-likelihood score is the most effective way to learn continuous time Bayesian network classifiers. Continuous time models again outperform dynamic Bayesian networks even when applied on discrete time data sets.

A Java software toolkit implementing the main theoretical achievements of the thesis has been designed and developed under the name of the CTBNCToolkit. It provides a free stand-alone toolkit for multivariate trajectory classification and an open source library, which can be extend in accordance with the GPL v.2.0 license. The CTBNCToolkit allows classification and clustering of multivariate trajectories using continuous time Bayesian network classifiers. Structural learning, maximizing marginal log-likelihood and conditional log-likelihood scores, is provided.

ACKNOWLEDGMENTS

This dissertation is the main result of a long journey. I would like to gratefully acknowledge all the people that walked with me along this path.

A special thank goes to Professor Fabio Stella who trusted in me and provided priceless suggestions and inspiration.

I wish to thank Professor Finn Jensen and the Aalborg University Machine Intelligence group. Even if the research work done in Aalborg is not strictly related to this thesis, the hours spent sharing ideas with Professor Jensen were a source of inspiration and priceless knowledge.

I want to thank Project Automation S.p.A. for funding my Ph.D. and in particular Paolo Confalonieri who believed in the contributions that my research could give to the Project Automation's mission. I also thank Alberto Feroldi and Yeser Amer, who shared ideas and experiences with me during the time spent in Project Automation.

I am grateful to Professor Julia Weekes, who first improved my English and then, worried about the Italian-English, helped me to proofread this dissertation.

I am grateful to Luca Manzoni for many influential conversations and to Anisa Rula and Enzo Acerbi for their valuable encouragement. All of them are precious colleagues and friends. I also want to thank Alessandro La Torraca, Marco Rossetti, Margherita Bodini and Davide Magatti because they made the time spent in the MAD laboratory fun.

I thank all my friends: the ones who share my love for research, but in other areas not as exciting as mine the ones that

asked me “why are you studying again?”, and the ones that did not ask any questions, but were just there.

Finally, special thanks go to my parents who patiently supported me in the never ending journey of the learning process. First, teaching me during childhood and then later asking me what I was talking about when I started to fall in love with research. In both situations their support was fundamental.

I apologize to all the people who are not in this list, but who deserve to be thanked. I am trying to include all of you in the following tag cloud. If you cannot find yourself, blame the stop words removing process.

ACRONYMS

ACTNB	Augmented Continuous Time Naive Bayes
BN	Bayesian Network
CIM	Conditional Intensity Matrix
CLL	Conditional Log-likelihood
CPT	Conditional Probability Tables
CTBN	Continuous Time Bayesian Network
CTBNC	Continuous Time Bayesian Network Classifier
CTNB	Continuous Time Naive Bayes
CT-NOR	Continuous Time Noisy-OR
CV	Cross-validation
DBN	Dynamic Bayesian Network
DBNC	Dynamic Bayesian Network Classifier
DTW	Dynamic Time Warping
EM	Expectation Maximization
EP	Expectation Propagation
fm	Fowlkes–Mallows index
HMM	Hidden Markov Model
j	Jaccard's coefficient
LCM	Latent Classification Model

LL	Log-likelihood
MLE	Maximum Likelihood Estimation
MLL	Marginal Log-likelihood
NB	Naive Bayes
NN	Neural Network
NNC	Nearest Neighbor Classifier
OE-DTW	Open-End Dynamic Time Warping
PCIM	Piecewise-constant Conditional Intensity Model
r	Rand index
SVM	Support Vector Machine
TAN	Tree-Augmented Naive Bayes
UTC	Urban Traffic Control

LIST OF NOTATIONS

Variables have their names capitalized (i.e. X).

A state of a variable is written in lower case letters (i.e. x).

Vectors are represented by boldface letters (i.e. $\mathbf{X}$, $\mathbf{x}$).

Sets are represented by calligraphic letters (i.e. $\mathcal{D}$).

$\mathcal{B}$	Bayesian Network.
$\mathcal{C}$	Continuous Time Bayesian Network Classifier.
$\mathcal{D}$	Data set of trajectories.
$\mathbf{FamScore}(X, Pa_{\mathcal{G}}(X) : \mathcal{D})$	Local score function for CTBN and CTBNC structural learning.
L	Likelihood function.
$M[x, x' \mid pa(X)]$	Number of times X transitions from state x to state x' when the state of its parents (i.e. $Pa(X)$) is set to $pa(X)$.
$M[x \mid pa(X)]$	Number of transitions from state x of variable X when the state of its parents (i.e. $Pa(X)$) is set to $pa(X)$.
$M[y]$	Number of trajectories in the data set labeled with class $Y = y$.
$\bar{M}$	Expected value of sufficient statistics M.
M^i	M sufficient statistics computed on the i-th trajectory.
$\mathbf{MLLscore}\,(\mathcal{G} : \mathcal{D})$	Marginal Log-likelihood score function for CTBN and CTBNC structural learning.

$\aleph$	Continuous Time Bayesian Network.
$P_{\aleph}(t)$	Probability distribution of the variables in $\aleph$ at time t.
$P_{\aleph}(s,t)$	Joint probability distribution of the variables in $\aleph$ over two time points.
P_X^0	Initial probability distribution over X.
$Pa(X)$	Parent set of variable X.
$pa(X)$	Instantiation of the variable X's parent set (i.e. $Pa(X)$).
$q_{xx'}^{pa(X)}$	Rate of arriving to state x' from state x for a specific instantiation $pa(X)$ of $Pa(X)$.
$q_x^{pa(X)}$	Rate of leaving state x for a specific instantiation $pa(X)$ of $Pa(X)$.
$\mathbf{Q}_X^{Pa(X)}$	Conditional Intensity Matrices of X, given its parents (i.e. $Pa(X)$).
score $(\mathcal{G} : \mathcal{D})$	Bayesian score function for CTBN and CTBNC structural learning.
$T[x \mid pa(X)]$	Amount of time spent in state x by variable X when the state of its parents (i.e. $Pa(X)$) is set to $pa(X)$.
$\bar{T}$	Expected value of sufficient statistics T.
T^i	T sufficient statistics computed on the i-th trajectory.
$\mathcal{V}al(X)$	State space of X.
$\mathbf{X} = \{X_1 \ldots X_n\}$	Set of variables (class excluded).
$X^t = x$	Point evidence at time t.
$X^{[t1,t2)} = x^{[t1,t2)}$	Continuous evidence over the time interval $[t1, t2)$.

$(\mathbf{x}^1, \ldots \mathbf{x}^J)$	J-evidence-stream of $\mathbf{X}$ variables over the J-time-stream (i.e. $[0, t_1); [t_1, t_2); ...; [t_{J-1}, T)$).
Y	Class variable.
α	Imaginary counts for the M sufficient statistics.
$\Gamma(\ldots)$	Gamma function.
$\theta_{xx'}^{pa(X)}$	Probability of transitioning from state x to state x', when it is known that the transition occurs at a given instant in time.
θ_y	Probability of class Y of being in state y.
τ	Imaginary counts for the T sufficient statistics.
π_n	Instantiation of variable X_n's parent set without the $Y = y$ class (i.e. $\pi_n = \{pa(X_n)/y\}$).

CONTENTS

1 INTRODUCTION

1.1 MOTIVATIONS

The number of sources generating streaming data has rapidly increased over the last few years. Streaming data are relevant in finance, computer science, and engineering while they are becoming increasingly important in medicine and biology (Barber and Cemgil, 2010). High frequency trading is an example where streaming data is relevant to finance (Dacorogna, 2001). Computer science offers many examples of streaming data, system error logs, web search query logs, network intrusion detection and social networks, to mention just a few (Simma and Jordan, 2010). In image processing, acoustic and vision applications streaming data are used to solve engineering problems (Yilmaz et al., 2006). An emerging paradigm in medicine is that of patient monitoring based on sensor data and that of continuous time diagnosis, including the study of computational firing pattern of neurons (Truccolo et al., 2005). Finally, the increasing amount of time course data generated in biology allows to discover gene regulatory networks to model the evolution of infections and to learn and analyze metabolic networks (Voit, 2012).

Data streaming problems may be approached with algorithms and models that can represent dynamics, sequences and time. Among these, Dynamic Bayesian Networks (DBNs) (Dean and Kanazawa, 1989) and Hidden Markov Models (HMMs) (Rabiner, 1989) have received great attention for modeling temporal dependencies. However, DBNs are concerned with discrete time and thus suffer from several limitations due to the fact that it is not clear how timestamps should be discretized. In the case where a too slow sampling rate is used the data will be poorly represented, while a too fast sampling rate rapidly makes learning and inference prohibitive. Furthermore, it has been pointed out (Gunawardana et al., 2011) that when allowing long term dependencies, it is necessary to condition on multiple steps into the past. Thus the choice of a too fast sampling rate will increase the number of such steps that need to be conditioned on. Continuous Time Bayesian Networks (CTBNs) (Nodelman et al., 2005), Continuous Time Noisy-OR (CT-NOR) (Simma et al., 2008), Poisson cascades (Simma and Jordan, 2010) and Poisson networks (Rajaram et al., 2005) together with the Piecewise-constant Conditional Intensity Model (PCIM) (Gunawardana et al., 2011) are interesting models to represent and analyze continuous time processes. CT-NOR and Poisson cascades are devoted to model event streams while they require the modeler to specify a parametric form for temporal dependencies. This aspect significantly impacts performance, and the problem of model selection in CT-NOR and Poisson cascades has not been addressed yet. This limitation is overcome by PCIMs which perform structure learning to model how events in the past affect future events of interest. Continuous Time Bayesian Networks are continuous time homogeneous Markov models which allow to represent joint trajectories of discrete finite variables, rather than models of event streams in continuous time.

Temporal classification is one of the most interesting problems concerning stream data analysis. In this dissertation attention is focused on temporal classification in the case where data stream measurements are available over a period of time in history, while the class is expected to occur in the future. This kind of problem can be addressed by discrete and continuous time models. Discrete time models such as dynamic Latent Classification Models (Zhong et al., 2012) and Dynamic Bayesian Networks (Dean and Kanazawa, 1989), both also able to make classification in the case where the class changes over time. Continuous time models, such as Continuous Time Bayesian Network Classifiers (CTBNCs) (Stella and Amer, 2012), overcame the problem of timestamps discretization. For this reason CTBNCs deserve particular attention.

CTBNCs parameter learning and inference algorithm were introduced by Stella and Amer (2012). Nevertheless, many open issue have to be addressed. How is it possible to learn the structure of CTBNCs? Is the Bayesian score introduced by Nodelman et al. (2002b) for CTBN the best approach to learn CTBNCs? How is it possible to address unsupervised learning (i.e. clustering) using CTBNCs? How does this model perform over real world data sets?

This dissertation answers these open issues providing the additional contribution of the CTBNCToolkit, an open source Java toolkit for CTBNCs, which makes freely available a toolkit for temporal classification. It can be used for scientific purposes, such as model comparison and temporal classification of interesting scientific problems, but it can be used as well as a prototype to address real world problems.

1.2 CONTRIBUTIONS

The class of CTBNCs is introduced and analyzed. The structural learning of CTBNCs is addressed by applying the algorithm introduced by Nodelman et al. (2002b) for CTBN. This approach, based on Bayesian scoring maximization, is improved introducing a Conditional Log-likelihood scoring function. The Conditional Log-likelihood scoring is derived and used to develop a structural learning algorithm for CTBNC.

Unsupervised learning (i.e. clustering) is addressed by using the Expectation Maximization (EM) algorithm to learn CTBNCs when the class is unknown.

Classification and clustering are studied on a rich set of synthetic data sets and on real world data sets as well. Comparison between CTBNCs, Continuous Time Naive Bayes (CTNB) and the state of the art is provided by using Dynamic Bayesian Network Classifiers (DBNCs).

Two important real world problems are analyzed by using CTBNCs: gesture recognition for post-stroke rehabilitation movements and traffic profile classification.

The main contributions of the dissertation are the following:

- definition of new models from the class of CTBNCs;
- derivation of a Conditional Log-likelihood scoring function for CTBNCs structural learning;
- development of soft and hard assignment EM algorithms for CTBNCs clustering;
- classification and clustering performance analysis between CTBNCs learned by maximizing the Marginal Log-likelihood and Conditional Log-likelihood scoring functions, CTNB, and DBNCs;
- performance analysis of CTBNCs on the post-stroke rehabilitation problem and the traffic profile classification problem;

- development of CTBNCToolkit, an open source Java toolkit for CTBNCs, which can be used as a stand-alone application and as an open source library as well.

1.3 OVERVIEW

The dissertation is organized as follows.

Chapter 2 provides an overview of the classification state of the art, with particular interest for temporal classification.

Preliminary notions about CTBNs and CTBNCs are provided in Chapter 3.

The contributions of the dissertation are presented and discussed in the following chapters. In Chapter 4, the structural learning problem for CTBNCs is addressed. A Conditional Log-likelihood scoring function is derived. A rich set of experiments on synthetic data sets are provided comparing CTBNCs learned by maximizing the Marginal Log-likelihood scoring function, the Conditional Log-likelihood scoring function, CTNB and DBNCs.

Chapter 5 introduces the clustering problem by using the EM algorithm to learn CTBNCs when the class is unknown. Tests are made to compare the clustering performances, over a rich set of synthetic data sets, by using external measures. Again CTBNCs learned by maximizing the Marginal Log-likelihood and Conditional Log-likelihood scores are compared to CTNB and DBNCs.

In Chapter 6 two interesting real world problems are addressed. Gesture recognition for post-stroke rehabilitation and traffic profile classification are introduced. Tests are made in the case of supervised and unsupervised learning, by using the CTBNCs introduced and comparing them with DBNCs.

Conclusions are made in Chapter 7, which outlines the work done in this dissertation and proposes the future works.

Additionally, two appendixes are provided. Appendix A explains in detail the CTBNCToolkit, analyzing the stand-alone usage, which allows to replicate the experiments reported in this dissertation and analyzing the code, which can be extended in accordance with the CTBNCToolkit license.

Finally, Appendix B provides all the results which, for matter of clarity and brevity, were summarized with graphics.

2

STATE OF THE ART

"Classification is a basic task in data analysis and pattern recognition that requires the construction of a classifier*, that is, a function that assigns a* class *label to instances described by a set of* attributes*"* (Friedman et al., 1997).

Classification often is the first step in dealing with complex decision problems and is a challenging task. Starting from a training set, i.e. a set of instances used to learn a model, a classifier must be learned. The learned model is then used to address the classification task on new instances (i.e. different instances never seen before). The learning process can be *supervised* and *unsupervised*. In supervised classification each instance of the training set has a class associated to the attributes, so the classifier is learned starting from labeled examples. In unsupervised learning, usually referred to as *clustering*, the training set instances are not associated with a class or label, and the classifier must learn to distinguish the classes by exploiting the values of the attributes.

In both supervised and unsupervised classification it is possible to speak about static classification (Section 2.1) and dynamic classification (Section 2.2). Static classification is the problem of associating a label to an instance, which is described with attributes values that do not change in time. Dynamic classification regards the problem of classifying a process that evolves over time.

The last distinction that must be made is between *complete observability* and *partial observability*. A problem is completely observable if all the attributes which describe its state are specified. Instead, a partially observable problem is characterized by instances where some attributes may be not specified. In this dissertation learning and classification are addressed only on completely observable problems.

2.1 STATIC CLASSIFICATION

Static classification is one of the first tasks that was studied in data mining and pattern recognition. For this reason many classification models have been developed over the years.

Decision trees (Quinlan, 1986; Dietterich, 2000) and random forests (i.e. a collection of decision trees) (Breiman, 2001) are algorithms that allow classification through successive steps in a tree. The path in the tree brings to a leaf which is associated to the classification label. Neural Networks (NNs) (Bishop, 1995; Ripley, 2007) are a well studied classification framework introduced with the purpose of simulating the human brain. A neural network can be interpreted as a black box that through the learning process (i.e. back propagation algorithm) is capable of learning complex functions, mapping the attributes to the class variable. Support Vector Machine (SVM) (Vapnik and Kotz, 1982; Burges, 1998; Joachims, 1998) is a model capable of separating the training data according to their classes. Each training instance is seen as a vector in the attribute state space. Classification is addressed as the problem of finding the hyperplanes which maximize the margin that separates the training data according to the classes. *Kernel functions* are often used to map the data in a new state space which allows for better classification.

Bayesian Networks (BNs) (Pearl, 1988) are a framework that allows general inference over static domains. BNs efficiently represent a probability distribution over the problem state space. The idea behind the BNs is to exploit the *conditional independencies* relationships between the problem variables. Using BNs it is possible to address many types of inference problems: one of these is classification (Friedman et al., 1997).

Naive Bayes (NB) (Duda et al., 1973; Langley et al., 1992) is the simplest classification model that originates from the BN framework. BNs allow to exploit all the relationships between the attributes, while NBs force the conditional independence assumption between attributes given the class. This assumption is rarely verified and forces the relationships between all the attributes and the class, even if some of these relationships do not hold. Nevertheless, NB is one of the most used classification models due to its simplicity. It usually offers good classification performances compared to the computational effort needed to learn a BN. Structural learning for BNs is a NP-hard problem, while NBs require only parameter learning.

Friedman et al. (1997) analyzed the class of BN classifiers and proposed the Tree-Augmented Naive Bayes (TAN) model. TAN is an extension of the NB where the conditional independence assumption of the attributes is relaxed thanks to a tree structure learned between the attributes. Friedman et al. (1997) showed that by limiting the attribute dependencies to a tree structure, it is possible to learn these relationships in polynomial time. For this reason TANs offer a solution to improve NBs performances without requiring too much computational effort. In the same work Friedman et al. (1997) introduced the concept of Conditional Log-likelihood. To learn a Bayesian model usually scores which use Log-likelihood are maximized. Log-likelihood is a function that measures how well the learned model represents the training set. In the case of a classification problem, maximizing the Log-likelihood function is the same as learning the best representation of the studied problem. This representation may not correspond to the structure which allows the best classification performances. Friedman et al. (1997) showed that maximizing Conditional Log-likelihood corresponds learning the model that maximizes the classification performances (see Section 4.3.2).

However, it is not possible to learn in a close form the BN parameters that maximize Conditional Log-likelihood. For this reason other approaches have been proposed. Greiner and Zhou (2002) described a gradient-descent algorithm to optimize an empirical Conditional Log-likelihood scoring function. Grossman and Domingos (2004) proposed an algorithm for BN learning where structural learning is performed by maximizing the Conditional Log-likelihood, while parameter learning relies on Log-likelihood maximization.

2.2 DYNAMIC CLASSIFICATION

Dynamic or temporal classification is the task of classifying a process that evolves over time. An instance which describes the process behavior through attributes which change over time is called *trajectory*.

Trajectories can be specified in discrete or continuous time. Discrete time trajectories describe the evolution of the process using time steps. At each time step the attribute values represent the process state in that particular instant. On the contrary, continuous time trajectories are representations of the world during the whole time interval. In this case each attribute can change state asynchronously, and it is possible to distinguish two types of evidence: *continuous evidence* where an attribute is specified for a complete interval and *point evidence* where an attribute is specified only for a particular instant.

Since the system evolves over the time, it is possible to deal with the classification problem in two ways:

- classify the process in order to find a hidden class or a class that will occur in the future (i.e. one single class that explains the entire trajectory);
- find the classes associated with each single instant of time during which the trajectory evolves (i.e. the class can change over time).

Some models are able to address both classification types, while others are designed to deal with the first type.

2.2.1 *Dynamic Time Warping*

The trajectory attributes can take value on a discrete or continuous state space. One of the most successful approaches in the specialized literature to deal with discrete time and continuous state space trajectories is the Dynamic Time Warping (DTW) (Keogh and Pazzani, 2000; Ratanamahatana and Keogh, 2004).

DTW uses dynamic programming to efficiently compare two trajectories. DTW is in some way similar to the text edit distance. The first step to calculate DTW distance is to generate a $N \times M$ matrix, where N and M are the dimensions of the two compared trajectories (i.e. $\mathbf{X}, \mathbf{Y}$). Each cell (i, j) contains the distance between the i^{th} state of the first trajectory and the j^{th} state of the second trajectory (i.e. $d(x_i, y_j)$). Once the matrix is generated, the goal is to find the *warping path* ($\mathbf{W} = w_1, \dots, w_K$), a continuous path that defines the mapping between the two trajectories.

Since the mapping must be complete, the path starts from one corner of the matrix to arrive to the opposite one. The warping path is the path in the matrix that minimizes the warping cost (i.e. $d_{DTW}(\mathbf{X},\mathbf{Y}) = \min\left\{\sqrt{\sum^{K} w_k}\right\} = \min\left\{\sqrt{\sum^{K} d(x_i, y_j)_k}\right\}$). This path can be efficiently calculated using dynamic programming (Keogh and Ratanamahatana, 2005).

Once defined how to efficiently calculate a distance between trajectories, the classification problem can be addressed with a Nearest Neighbor Classifier (NNC) (Cover and Hart, 1967; Dasarathy, 1991). NNCs are classifiers that use the k closest training set instances to classify a new instance according to the most common label.

To improve the computational effort of the classification process some solutions have been proposed. Regarding the DTW calculation, studies have focused on how far the warping path may stray from the main diagonal of the distance matrix (Keogh and Ratanamahatana, 2005). Following this line an approximation often used consists of defining a *warping window* that bounds the portion of the distance matrix in which the warping path is allowed to go through (Sakoe and Chiba, 1978; Itakura, 1975). NNC algorithm requires a linear computation time respect to the dimension of the training set. To improve its efficiency data reduction techniques are used (Dasarathy, 1991; Pękalska et al., 2006; Xi et al., 2006). The idea is to reduce the number of training set instances, maintaining its representativity and removing only instances in which information is duplicated.

Dynamic Time Warping was developed to compare two complete trajectories. In many real world problems it is necessary compare a complete trajectory (i.e. new instance) to find the best partial matching with the training set instances (i.e. reference instances). Tormene et al. (2009) introduced Open-End

Dynamic Time Warping (OE-DTW) to address this task. The idea behind OE-DTW is to calculate the DTW distances between the trajectory to classify and the reference trajectory truncated in all its points (i.e. $d_{OE}(\mathbf{X}, \mathbf{Y}) = \min_{j=1,\dots M}\left\{d_{DTW}(\mathbf{X}, \mathbf{Y}^{(j)})\right\}$). This operation can be done efficiently thanks again to dynamic programming.

Theoretically both DTW and OE-DTW can be used over discrete state space, since a distance is defined between the states. In the case where the states do not have a clear ordering the *trivial metric* must be used (Hazewinkel, 1993):

$$d_T(x_i, y_j) = \begin{cases} 0 & \text{if } x_i = y_j \\ 1 & \text{if } x_i \neq y_j \end{cases} .$$

In this case the DTW distances greatly lose their effectiveness.

Since this work is mainly focused on the problems of classification and clustering of continuous time multivariate trajectories, DTW distances are not used in performance comparison. An exception is made for the post-stroke rehabilitation problem that provides continuous state space trajectories (see Section 6.1).

2.2.2 *Dynamic Bayesian Network Classifiers*

Dynamic Bayesian Networks (DBNs) are an extension of Bayesian Networks introduced to model systems which evolve over discrete time (Dean and Kanazawa, 1989; Ghahramani, 1998; Friedman et al., 1998; Murphy, 2002). DBNs represent a Markov process over time using the factorization introduced with the BNs.

A DBN consists of a sequence of BNs, one for each time instant. Each of these BNs is a local model called *time slice*. Since the local probability distribution does not change over time, we can define a DBN by a *local probability distribution* that models the intra slice dependencies of the variables, a *prior distribution* that defines the knowledge about the initial state of the process, and a *transition distribution* that models how the variables evolve over time. The transition distribution must satisfy the *Markov property*: the knowledge of the past does not influence the future, if the present state is known (i.e. $\forall s, k, t \in \mathbb{N} : 0 \leqslant s \leqslant t, P\left(X(t+k) \mid X(s), \ldots, X(t)\right) = P\left(X(t+k) \mid X(t)\right)$. Since both the local and the transition distributions do not change over time and because of the Markov property, a DBN can be defined by two adjacent time slices. The first models the prior distribution, the second models the local distribution, while the transition distribution is modeled by the dependencies between the variables of the two time slices.

DBNs allow general inference such as *filtering*, *prediction*, *smoothing*, and the *most probable explanation*. Filtering consists of calculating the probability distribution of a present variable, given the evidence from the past. Prediction consists of calculating the probability distribution of a future variable, given the evidence from the past. Smoothing consists of calculating the probability distribution of a past variable, given all the time evidence (past and future). The most probable explanation is the sequence of variable states that better explain the evidence over time. The most probable explanation for a class variable corresponds to the classification problem over a variable that changes in time. DBNs can be used to infer a static class that does not change over time, forcing to zero the probability of the class changing state.

Hidden Markov Models (HMMs) (Rabiner and Juang, 1986; Rabiner, 1989) and Kalman filters (Welch and Bishop, 1995) are a simplification of DBNs. An HMM is a DBN where the state is hidden, and only some variables can be used to infer the state of the system (i.e. observations). These observation variables are independent over time, given the hidden state of the system. A Kalman filter is an HMM where exactly one variable is connected to the adjacent time slices.

The Latent Classification Model (LCM) is a Naive Bayes extension that relaxes the conditional independence assumption using latent variables (Zhang et al., 2004; Langseth and Nielsen, 2005). Dynamic latent classification models (Zhong et al., 2010, 2012) are generative models that extend the LCMs to address the classification problem over dynamic systems. The idea is to use a LCM as a local model in each time slice. Nevertheless, dynamic latent classification models are a specialization of DBNs.

Since DBN is the general framework and is one of the most relevant models that the state of the art proposes to address the classification problem of multivariate trajectories, Dynamic Bayesian Network Classifiers (DBNCs) are used as a competitor for the Continuous Time Bayesian Network Classifier framework discussed in this dissertation.

2.2.3 *Continuous Time Bayesian Network Classifier*

Continuous Time Bayesian Networks (CTBNs) are a framework introduced by Nodelman et al. (2002a) to make inference on continuous time trajectories. The idea behind CTBNs (Nodelman et al., 2002a; Nodelman, 2007) is to exploit the conditional independency relationships between the variables that define a homogeneous continuous Markov process. Finding the structure behind a continuous Markov process allows to represent it very efficiently, similarly to how BNs compactly represent a static probability distribution.

CTBNs represent a process that evolves over continuous time. The *memoryless assumption* is made to model the time evolution using the *exponential distribution*. Nevertheless, it has been shown that it is possible to use other probability distributions to model the process evolution with the CTBN framework. Gopalratnam et al. (2005) propose an extension of CTBN that uses Erlang-Coxian approximation to model arbitrary transition time distribution while maintaining the learning process tractable.

CTBNs structural learning has shown to be very efficient because the acyclicity constrain of BN and DBN does not hold (Nodelman et al., 2002b). While learning BNs and DBNs is an NP-hard problem, learning the structure of CTBNs is a polynomial problem with respect to the number of variables and the size of the data set. In addition, the parent set of each variable can be learned independently, allowing the parallelization of the structural learning.

Contrary to DBNs, inference in CTBNs does not require enrolling the time slices. Nevertheless, inference in CTBNs is still an NP-hard problem (Nodelman et al., 2002a). For this reason approximate algorithms were introduced: *Expectation Propagation* (Nodelman et al., 2005; Saria et al., 2007), *Importance Sampling* (Fan and Shelton, 2008) and, *Gibbs sampling* (El-Hay et al., 2008).

Continuous Time Bayesian Network Classifiers (CTBNCs) were introduced by Stella and Amer (2012) as a specialization of the CTBNs, designed to address the problem of multivariate continuous time trajectories classification in polynomial time. The idea behind CTBNCs is to add a static class variable which represents an unknown class that will occur in the future or a static explanation of the trajectory.

The structure of a CTBNC can be learned efficiently as the structure of a CTBN. Structural learning is polynomial with respect to the number of variables and the size of the data set, but the number of variable parameters is exponential with respect to the maximum number of parents which must be fixed in advance. For this reason Stella and Amer (2012) introduced the Continuous Time Naive Bayes (CTNB) model. CTNB is a Naive Bayes model designed to classify in continuous time.

2.3 MULTIVARIATE TRAJECTORIES CLUSTERING

Clustering is the classification problem when the training set instances are not labeled (Hartigan, 1975). For this reason clustering relies on finding a natural division between the instances.

Clustering algorithms are often based on some measure of distance. The most known example of distance based clustering is the k*-means algorithm* (Pelleg et al., 2000). k-means is an iterative algorithm. It requires the definition of the number of clusters (i.e. k) before running the algorithm. The initialization consists of randomly choosing k points in the state space. These points are the centroids of the initial clusters. Then the algorithm requires the iteration of the following two steps until the convergence:

- assigns to each instance of the training set the closest centroid (i.e. cluster);
- calculates the mean between the points in each cluster and sets the means as new centroids.

This iteration process will arrive to a convergence when the centroids are the same after two consecutive iterations.

Alternatives distance based clustering are the *hierarchical clustering* algorithms (Witten and Frank, 2005). Hierarchical clustering is a category of clustering algorithms that proposes a hierarchy of clusters. There are two types of hierarchical clustering: *divisive* and *agglomerative*. The first category starts with one single cluster and divides it into many clusters in a topdown approach. The second category starts with many single instance clusters and aggregates them step by step in a bottom-up strategy.

Oates et al. (1999) suggested using DTW distance to realize an agglomerative hierarchical clustering between trajectories. The idea is to start with single trajectory clusters and iteratively merge the couple of clusters that have the minimum average inter-cluster distance. To avoid having to one single cluster a stopping criteria must be defined. Oates et al. (1999) propose to not merge clusters for which the mean inter-cluster distance is significantly different from the mean intra-cluster distance verified with a t-test.

As pointed out in Section 2.2.1, it is not really effective to use Dynamic Time Warping on discrete state space trajectories. For this reason, instead of focusing on distance based clustering that required DTW distance, the focus is directed on probabilistic graphical models and *Expectation Maximization (EM)* algorithm (Koller and Friedman, 2009). EM is an iterative algorithm used to optimize the *likelihood function* in probabilistic graphical models learning when there are missing data. In the case of the clustering on completely observable data, the only missing value is the class.

EM is based on two steps: *expectation* and *maximization*. The idea is to start with a random instantiation of the model parameters or a random filling of the missing values (i.e. the class). Here let's consider starting with a random instantiation of the model parameters. Using the current model we can estimate the probability distribution over the class values. Using this distribution, the first step (i.e. expectation step) consists of calculating the expected sufficient statistics. In the next step (i.e. maximization step), the model parameters are calculated by maximizing the likelihood with respect to the calculated sufficient statistics. EM iteration steps are repeated until reaching a stopping criteria. Common stopping criteria are related to small changes of likelihood values or parameter values between consecutive iterations.

It is possible to distinguish two types of EM clustering: *hard-assignment* and *soft-assignment*. The difference relies on the expectation step. In the case of soft-assignment EM the expected sufficient statistics are calculated using the probability distribution of the class. In the case of hard-assignment EM the sufficient statistics are calculated supposing to fill the class missing values with the most probable class.

3

PRELIMINARY NOTIONS

This Chapter presents the basic notions necessary to understand the rest of the dissertation.

First the Continuous Time Bayesian Network (CTBN) framework is introduced as a viable alternative in order to overcome Dynamic Bayesian Networks (DBNs) limitations (Section 3.1). Inference, parameter learning and structural learning are progressively addressed. Finally, Continuous Time Bayesian Network Classifiers (CTBNCs) are introduced to efficiently address the temporal classification problem of a static class variable (i.e. data change over time, while the class is static).

3.1 CONTINUOUS TIME BAYESIAN NETWORKS

Dynamic Bayesian Networks (DBNs) model dynamic systems without representing time explicitly. They discretize time to represent a dynamic system through several time slices. Nodelman et al. (2002a) pointed out that *"since DBNs slice time into fixed increments, one must always propagate the joint distribution over the variables at the same rate"* . Therefore, if the system consists of processes which evolve at different time granularities and/or the obtained observations are irregularly spaced in time, the inference process may become computationally intractable.

Continuous Time Bayesian Networks (CTBNs) overcome the limitations of DBNs by explicitly representing temporal dynamics and thus allow us to recover the probability distribution over time when specific events occur. CTBNs have been used to discover intrusion in computers (Xu and Shelton, 2008), to analyze the reliability of dynamic systems (Boudali and Dugan, 2006), for learning social networks dynamics (Fan and Shelton, 2009) and to model cardiogenic heart failure (Gatti et al., 2011). CTBNs are based on *homogeneous continuous Markov processes* where transition intensities do not depend on time.

Let X be a random variable whose state changes continuously over time and takes value on domain $Val(X) = \{x_1, ..., x_m\}$ then the continuous Markov process $X(t)$ can be represented with the following *intensity matrix*:

$$\mathbf{Q}_X = \begin{bmatrix} -q_{x_1} & q_{x_1x_2} & \cdots & q_{x_1x_m} \\ q_{x_2x_1} & -q_{x_2} & \cdots & q_{x_2x_m} \\ \vdots & \vdots & \ddots & \vdots \\ q_{x_mx_1} & q_{x_mx_2} & \cdots & -q_{x_m} \end{bmatrix},$$

where $q_{x_i} = \sum_{i \neq j} q_{x_ix_j}$. The $\mathbf{Q}_X$ matrix allows the description of the transient behavior of $X(t)$. If $X(0) = x_i$ is assumed, then the variable X stays in state x_i for an amount of time which is a random variable distributed according to the exponential distribution with parameter value equal to q_{x_i}. Therefore, the probability density function f together with the corresponding distribution function F for the process $X(t)$ to remain in state x_i are defined as follows:

$$f(t) = q_{x_i} \exp(-q_{x_i} t), t \geqslant 0$$

$$F(t) = 1 - \exp(-q_{x_i} t), t \geqslant 0.$$

Thus, the expected time of leaving the state x_i is equal to $\frac{1}{q_{x_i}}$, while $\frac{q_{x_ix_j}}{q_{x_i}}$ is the probability of transitioning from state x_i to state x_j, when it is known that the transition occurs at a given instant in time.

A Continuous Time Bayesian Network (CTBN) is a probabilistic graphical model whose nodes are associated with random variables and whose state evolves continuously over time. Evolution of each variable depends on the state of its parents in the graph associated with the CTBN model. A CTBN consists of two main components: *i)* an initial probability distribution and *ii)* the dynamics which rule the evolution over time of the probability distribution associated with the CTBN.

Definition 3.1.1. (Continuous Time Bayesian Network (CTBN)). (Nodelman et al., 2002a). Let $\mathbf{X}$ be a set of random variables $X_1, X_2, ..., X_N$. Each X_n has a finite domain of values $Val(X_n) = \{x_1, x_2, ..., x_I\}$. A Continuous Time Bayesian Network (CTBN) $\aleph$ over $\mathbf{X}$ consists of two components: the first is an initial distribution $P^0_{\mathbf{X}}$, specified as a Bayesian network $\mathcal{B}$ over $\mathbf{X}$. The second is a continuous transition model, specified as:

- a directed (possibly cyclic) graph $\mathcal{G}$ whose nodes are $X_1, X_2, ..., X_N$; $Pa(X_n)$ denotes the parents of X_n in $\mathcal{G}$.
- a conditional intensity matrix, $\mathbf{Q}^{Pa(X_n)}_{X_n}$, for each variable $X_n \in \mathbf{X}$.

Given the random variable X_n, the *Conditional Intensity Matrix (CIM)* $\mathbf{Q}^{Pa(X_n)}_{X_n}$ consists of a set of intensity matrices, one intensity matrix

$$\mathbf{Q}^{pa(X_n)}_{X_n} = \begin{bmatrix} -q^{pa(X_n)}_{x_1} & q^{pa(X_n)}_{x_1 x_2} & \cdots & q^{pa(X_n)}_{x_1 x_I} \\ q^{pa(X_n)}_{x_2 x_1} & -q^{pa(X_n)}_{x_2} & \cdots & q^{pa(X_n)}_{x_2 x_I} \\ \vdots & \vdots & \ddots & \vdots \\ q^{pa(X_n)}_{x_I x_1} & q^{pa(X_n)}_{x_I x_2} & \cdots & -q^{pa(X_n)}_{x_I} \end{bmatrix},$$

for each instantiation $pa(X_n)$ of the parents $Pa(X_n)$ of node X_n, where $q^{pa(X_n)}_{x_i} = \sum\limits_{x_j \neq x_i} q^{pa(X_n)}_{x_i x_j}$ is the rate of leaving state x_i for a specific instantiation $pa(X_n)$ of $Pa(X_n)$, while $q_{x_i x_j}^{pa(X_n)}$ is the rate of arriving to state x_j from state x_i for a specific instantiation $pa(X_n)$ of $Pa(X_n)$. Matrix $\mathbf{Q}_{X_n}^{pa(X_n)}$ can equivalently be summarized by using two types of parameters, $q_{x_i}^{pa(X_n)}$ which is associated with each state x_i of the variable X_n when its parents are set to $pa(X_n)$, and $\theta_{x_i x_j}^{pa(X_n)} = \frac{q_{x_i x_j}^{pa(X_n)}}{q_{x_i}^{pa(X_n)}}$ which represents the probability of transitioning from state x_i to state x_j, when it is known that the transition occurs at a given instant in time.

Example 3.1.1. Figure 1 shows a part of the drug network introduced in Nodelman et al. (2002a). It contains a cycle, indicating that whether a person is hungry (*H*) depends on how full his/her stomach (*S*) is, which depends on whether or not he/she is eating (*E*), which in turn depends on whether he/she is hungry. Assume that E and H are binary variables with states *no* and *yes* while the variable S can be in one of the following states; *full*, *average* or *empty*. Then, the variable E is fully specified by the [2×2] CIM matrices $\mathbf{Q}_E^n$, and $\mathbf{Q}_E^y$, the variable S is fully specified by the [3×3] CIM matrices $\mathbf{Q}_S^n$ and $\mathbf{Q}_S^y$, while the the variable H is fully specified by the [2×2] CIM matrices $\mathbf{Q}_H^f$,

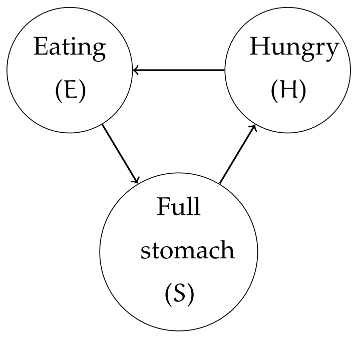


Figure 1.: A part of the drug network.

$\mathbf{Q}_H^a$ and, $\mathbf{Q}_H^e$. For matters of brevity only $\mathbf{Q}_S^y$ with two equivalent parametric representations is shown:

$$\begin{aligned}\mathbf{Q}_S^y &= \begin{bmatrix} -q_f^y & q_{f,a}^y & q_{f,e}^y \\ q_{a,f}^y & -q_a^y & q_{a,e}^y \\ q_{e,f}^y & q_{e,a}^y & -q_e^y \end{bmatrix} \\ &= \begin{bmatrix} -0.03 & 0.02 & 0.01 \\ 5.99 & -6.00 & 0.01 \\ 1.00 & 5.00 & -6.00 \end{bmatrix} \end{aligned} \tag{1}$$

$$\begin{aligned}\mathbf{Q}_S^y &= \begin{bmatrix} q_f^y & 0 & 0 \\ 0 & q_a^y & 0 \\ 0 & 0 & q_e^y \end{bmatrix} \left(\begin{bmatrix} 0 & \theta_{f,a}^y & \theta_{f,e}^y \\ \theta_{a,f}^y & 0 & \theta_{a,e}^y \\ \theta_{e,f}^y & \theta_{e,a}^y & 0 \end{bmatrix} - \mathbf{I} \right) \\ &= \begin{bmatrix} 0.03 & 0 & 0 \\ 0 & 6.00 & 0 \\ 0 & 0 & 6.00 \end{bmatrix} \left(\begin{bmatrix} 0 & \frac{0.02}{0.03} & \frac{0.01}{0.03} \\ \frac{5.99}{6.00} & 0 & \frac{0.01}{6.00} \\ \frac{1.00}{6.00} & \frac{5.00}{6.00} & 0 \end{bmatrix} - \mathbf{I} \right) \end{aligned} \tag{2}$$

where $\mathbf{I}$ is the identity matrix.

If the hours are the units of time, then a person who has an empty stomach (*S=empty*) and is eating (*E=yes*) is expected to stop having an empty stomach in 10 minutes ($\frac{1.00}{6.00}$ hour). The stomach will then transition from state $empty$ (*S=empty*) to state $average$ (*S=average*) with probability $\frac{5.00}{6.00}$ and to state $full$ (*S=full*) with probability $\frac{1.00}{6.00}$. Equation 1 is a compact representation of the CIM while Equation 2 is useful because it explicitly represents the transition probability value from state x to state x', i.e. $\theta_{xx'}^{pa(X)}$.

CTBNs allow two types of evidence, namely *point evidence* and *continuous evidence*, while Hidden Markov Models (HMMs) and DBNs allow only point evidence. *Point evidence* at time *t* for a subset of variables $X_1, X_2, ..., X_k \in \mathbf{X}$ is the knowledge of the states $x_1, x_2, ..., x_k$ at time *t* for the variables $X_1, X_2, ..., X_k$. Point evidence will be referred to as $X_1^t = x_1, X_2^t = x_2, ..., X_k^t = x_k$ or in compact notation as $\mathbf{Z}^t = \mathbf{z}$, where $\mathbf{Z} = (X_1, X_2, ..., X_k)$ while $\mathbf{z} = (x_1, x_2, ..., x_k)$. *Continuous evidence* is the knowledge of the states $x_1, x_2, ..., x_k$ of a set of variables $X_1, X_2, ..., X_k \in \mathbf{X}$ throughout an entire interval of time $[t_1, t_2)$ (that it is taken to be a half-closed interval). It is worthwhile to notice that the states $x_1, x_2, ..., x_k$ of the set of variables $X_1, X_2, ..., X_k$ do not change over the considered time interval $[t_1, t_2)$. Continuous evidence over time interval $[t_1, t_2)$ for the set of variables $X_1, X_2, ..., X_k$ will be referred to as $X_1^{[t_1,t_2)} = x_1, X_2^{[t_1,t_2)} = x_2, ..., X_k^{[t_1,t_2)} = x_k$. Analogously to point evidence a compact notation is introduced where continuous evidence over the time interval $[t_1, t_2)$ for the set of variables $X_1, X_2, ..., X_k$ will be written as $\mathbf{Z}^{[t_1,t_2)} = \mathbf{z}^{[t_1,t_2)}$, where $\mathbf{Z}^{[t_1,t_2)} = (X_1^{[t_1,t_2)}, X_2^{[t_1,t_2)}, ..., X_k^{[t_1,t_2)})$ while $\mathbf{z}^{[t_1,t_2)} = (x_1^{[t_1,t_2)}, x_2^{[t_1,t_2)}, ..., x_k^{[t_1,t_2)})$.

3.1.1 *Inference*

A CTBN exploits conditional independence relationships between variables to obtain a factored representation of a homogeneous continuous Markov process. Given the CIMs associated with the variables of the CTBN, the *amalgamation* operation (Nodelman et al., 2002a) over the CIMs allows to recover the joint intensity matrix of a homogeneous continuous Markov process as follows:

$$\mathbf{Q}_{\aleph} = \prod_{X_i \in \mathbf{X}} \mathbf{Q}_{X_i|pa(X_i)}.$$

Amalgamation takes two CIMs and produces a single bigger CIM. Given the sets of variables $\mathcal{S}1, \mathcal{S}2, \mathcal{C}1, \mathcal{C}2 \in \mathbf{X}$ and two CIMs $\mathbf{Q}_{\mathcal{S}1|\mathcal{C}1}$ and $\mathbf{Q}_{\mathcal{S}2|\mathcal{C}2}$ the amalgamated CIM $\mathbf{Q}_{\mathcal{S}|\mathcal{C}} = \mathbf{Q}_{\mathcal{S}1|\mathcal{C}1} \cdot \mathbf{Q}_{\mathcal{S}2|\mathcal{C}2}$

contains the intensities for the variables in $\mathcal{S} = \mathcal{S}1 \cup \mathcal{S}2$ conditioned on those in $\mathcal{C} = (\mathcal{C}1 \cup \mathcal{C}2) - \mathcal{S}$.

CTBNs can be used to answer to all the questions which can be answered by homogeneous Markov processes. Indeed, it is always possible to recover the joint intensity matrix $\mathbf{Q}_{\aleph}$ from the variables' intensity matrices (i.e. $\mathbf{Q}_{X_i|pa(X_i)}, X_i \in \mathbf{X}$). Given the joint intensity matrix $\mathbf{Q}_{\aleph}$ it is possible to compute the distribution over the values of the variables as follows:

$$P_{\aleph}(t) = P^0_{\aleph} \exp(\mathbf{Q}_{\aleph} t)$$

where $P^0_{\aleph}$ is the initial distribution over the variables, compactly represented as a Bayesian network in the CTBN framework. Similarly the joint distribution over two time points can be computed as follows:

$$P_{\aleph}(s,t) = P_{\aleph}(s) \exp \mathbf{Q}_{\aleph}(t-s)$$

with $t > s$. The above formula shows that CTBNs and homogeneous Markov processes inherit the memoryless property from the exponential distribution.

Inference in CTBNs can be performed by exact and approximate algorithms. *Full amalgamation* (Nodelman et al., 2002a) is an exact algorithm that involves generating an exponentially-large matrix representing the transition model over the entire state space. Exact inference in CTBNs is known to be NP-hard, and thus different approximate algorithms have been proposed. Nodelman et al. (2005) introduced the *Expectation Propagation (EP)* algorithm which allows both point and interval evidence. It exploits message passing in a cluster graph, where the clusters contain distributions over trajectories of the variables through a time duration. Saria et al. (2007) presented a new EP-based algorithm, which uses a flexible cluster graph architecture that fully exploits the natural time-granularity at which different sub-processes evolve. It also dynamically chooses the appropriate level of granularity to use in each cluster at each point in time. Alternatives are offered by sampling based inference algorithms. The *Importance Sampling* algorithm (Fan and Shelton, 2008) computes the expectation of any function of a trajectory, conditioned on any evidence set constraining the values of subsets of the variables over subsets of the timeline. El-Hay et al. (2008) developed a *Gibbs sampling* procedure for CTBNs which iteratively samples a trajectory for one of the components given the remaining ones. This approach naturally exploits the structure of the CTBN to optimize the computational cost of each step. This procedure is the first that can provide asymptotically unbiased approximations in such processes.

3.1.2 *Parameter learning*

Given a data set $\mathcal{D}$ and a fixed structure of a CTBN, parameter learning is based on Bayesian estimation. Parameter learning accounts for the *imaginary counts* of the hyperparameters $\alpha_x^{pa(X)}$, $\alpha_{xx'}^{pa(X)}$ and, $\tau_x^{pa(X)}$. The parameters $q_x^{pa(X)}$ and $\theta_{xx'}^{pa(X_n)}$ can be estimated as follows:

- $q_x^{pa(X)} = \frac{\alpha_x^{pa(X)}+M[x|pa(X)]}{\tau_x^{pa(X)}+T[x|pa(X)]}$;
- $\theta_{xx'}^{pa(X)} = \frac{\alpha_{xx'}^{pa(X)}+M[x,x'|pa(X)]}{\alpha_x^{pa(X)}+M[x|pa(X)]}$.

where $M[x, x' \mid pa(X)]$, $M[x \mid pa(X)]$ and $T[x \mid pa(X)]$ are the *sufficient statistics* computed over $\mathcal{D}$:

- $M[x, x' \mid pa(X)]$: number of times X transitions from state x to state x' when the state of its parents $Pa(X)$ is set to $pa(X)$;

- $M[x \mid pa(X)] = \sum_{x' \neq x} M[x, x' \mid pa(X)]$: number of transitions from state x of variable X when the state of its parents $Pa(X)$ is set to $pa(X)$;
- $T[x \mid pa(X)]$: amount of time spent in state x by variable X when the state of its parents $Pa(X)$ is set to $pa(X)$.

3.1.3 *Structural learning*

The problem of learning the structure of a CTBN from a data set $\mathcal{D}$ has been addressed in (Nodelman et al., 2002b) as the problem of finding the structure $\mathcal{G}$ which maximizes the following Bayesian score:

$$\textbf{score}\,(\mathcal{G} : \mathcal{D}) = \ln P(\mathcal{D}|\mathcal{G}) + \ln P(\mathcal{G}). \tag{3}$$

This is an optimization problem over possible CTBN structures whose search space is significantly simpler than that of Bayesian Networks (BNs) or DBNs. While it is known that learning the optimal structure of a BN is NP-hard, the same does not hold true in the context of CTBN learning where all edges are across time and thus represent the effect of the current value of one variable on the next value of the other variables. Therefore, no acyclicity constraints arise, and it is possible to optimize the parent set for each variable of the CTBN independently. This can be easily done for each variable exploring its parent space with a local search to find the best score value. Fixing the maximum number of parents local search is polynomial with respect to the number of variables and the size of the data set (Nodelman et al., 2002b).

Efficiency of the optimization algorithm relies on the fact that the prior satisfies structure and parameter modularity. Structure prior $P(\mathcal{G})$ satisfies *structure modularity* if $P(\mathcal{G}) = \prod_i P(Pa(X_i) = Pa_{\mathcal{G}}(X_i))$ while parameter prior satisfies *parame-*

ter modularity if given any pair of structures $\mathcal{G}$ and $\mathcal{G}'$ such that $Pa_{\mathcal{G}}(X) = Pa_{\mathcal{G}'}(X)$ the following equality holds $P(\mathbf{q}_X, \theta_X \mid \mathcal{G}) = P(\mathbf{q}_X, \theta_X \mid \mathcal{G}')$. Nodelman et al. (2002b) show that the marginal likelihood $P(\mathcal{D} \mid \mathcal{G})$ of the data $\mathcal{D}$ given the CTBN's structure $\mathcal{G}$ decomposes to

$$P(\mathcal{D} \mid \mathcal{G}) = \left(\int_{\mathbf{q}_{\mathcal{G}}} L(\mathbf{q}_{\mathcal{G}} : \mathcal{D}) P(\mathbf{q}_{\mathcal{G}}) d\mathbf{q}_{\mathcal{G}} \right) \cdot \left(\int_{\theta_{\mathcal{G}}} L(\theta_{\mathcal{G}} : \mathcal{D}) P(\theta_{\mathcal{G}}) d\theta_{\mathcal{G}} \right)$$

where L stands for the likelihood function.

The first term of $P(\mathcal{D} \mid \mathcal{G})$ is equal to

$$\prod_{X} \prod_{pa(X)} \prod_{x} \frac{\Gamma(\alpha_x^{pa(X)} + M[x \mid pa(X)] + 1)(\tau_x^{pa(X)})^{\alpha_x^{pa(X)}+1}}{\Gamma(\alpha_x^{pa(X)} + 1)(\tau_x^{pa(X)} + T[x \mid pa(X)])^{\alpha_x^{pa(X)}+M[x|pa(X)]+1}}$$

while the second term is equal to

$$\prod_{X} \prod_{pa(X)} \prod_{x} \frac{\Gamma(\alpha_x^{pa(X)})}{\Gamma(\alpha_x^{pa(X)} + M[x \mid pa(X)])} \cdot \\ \cdot \prod_{x' \neq x} \frac{\Gamma(\alpha_{xx'}^{pa(X)} + M[x, x' \mid pa(X)])}{\Gamma(\alpha_{xx'}^{pa(X)})}$$

where Γ is the gamma function.

Thanks to the previous decomposition and the structure modularity assumption, the Bayesian score (Equation 3) decomposes in a sum of local scores (i.e. family scores). Each local score (i.e. **FamScore**$(X, Pa_{\mathcal{G}}(X) : \mathcal{D})$) measures the quality of the parent set (i.e. $Pa(X)$) of a variable (i.e. X) given the data (i.e. $\mathcal{D}$). This allows to separately optimize the parent set of each variable by maximizing the variable family score (Nodelman et al., 2002b).

3.2 CTBNC PREVIOUS WORKS AND BASIC NOTIONS

Continuous Time Bayesian Network Classifiers (CTBNCs) (Stella and Amer, 2012) are a specialization of CTBNs. They allow polynomial time classification, while for CTBNs general inference is

NP-hard. Classifiers from this class explicitly represent the evolution in continuous time of the set of random variables X_n, $n = 1, 2, ..., N$ which are assumed to depend on the class node Y. A CTBNC is defined as follows.

Definition 3.2.1. (Continuous Time Bayesian Network Classifier (CTBNC))[1]. A Continuous Time Bayesian Network Classifier is a pair $\mathcal{C} = \{\aleph, P(Y)\}$ where $\aleph$ is a CTBN model with attribute nodes $X_1, X_2, ..., X_N$, Y is the class node with marginal probability $P(Y)$ on states $Val(Y) = \{y_1, y_2, ..., y_K\}$, $\mathcal{G}$ is the graph of the CTBNC, such that the following conditions hold:

- $Pa(Y) = \emptyset$, the class variable Y is associated with a root node;
- Y is fully specified by $P(Y)$ and does not depend on time.

An instance of a CTBNC consisting of six attributes $X_1, X_2, ..., X_6$ and the class Y is depicted in Figure 2. The class variable Y is associated with the only root node. It is worthwhile to note that the model contains a cycle involving nodes X_3 and X_4, which is allowed in CTBNs, while not in BNs.

Parameter learning of CTBNCs corresponds with the parametric learning of CTBNs (Section 3.1.2). The only difference is related to the necessity of learning the probability distribution over the class node. Because the class node is a static node, this can be done easily as follows:

$$\theta_y = \frac{\alpha_y + M[y]}{\sum_{y'} \alpha_{y'} + M[y']}$$

where $M[y]$ is the number of trajectories in the training set with class Y sets to y and, α_y are the imaginary counts related to the class variable.

1 This definition differs from the one proposed by Stella and Amer (2012). In fact, the definition proposed here does not require the CTBNC graph to be connected and thus allows structural learning algorithms to automatically perform feature selection.

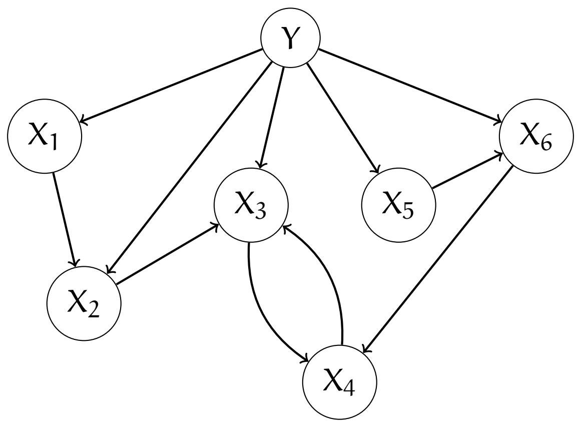


Figure 2.: Continuous Time Bayesian Network Classifier; six attribute nodes $X_1, \dots, X_6$ and the class node Y.

Given a data set $\mathcal{D}$ with no missing data, a CTBNC can be learned by maximizing the score function **score** $(\mathcal{G} : \mathcal{D})$ (3) subjected to the constraints listed in Definition 3.2.1. Exact learning requires to set in advance the maximum number of parents L for the nodes $X_1, X_2, ..., X_N$ (Nodelman et al., 2005). In the case where L is not small a considerable computational effort is required to find the graph structure $\mathcal{G}^*$ which maximizes the score function **score** $(\mathcal{G} : \mathcal{D})$ (3). Therefore, to find an approximate solution to the considered optimization problem it is necessary to resort to hill-climbing optimization procedures. Continuous Time Naive Bayes (CTNB) was introduced to limit the computational effort to find the optimal graph structure $\mathcal{G}*$.

Definition 3.2.2. (Continuous Time Naive Bayes (CTNB) (Stella and Amer, 2012)). A Continuous Time Naive Bayes is a continuous time Bayesian network classifier $\mathcal{C} = \{\aleph, P(Y)\}$ such that $Pa(X_n) = \{Y\}$, $n = 1, 2, ..., N$.

3.2.1 *Classification*

As pointed out before CTBNCs are a specialization of CTBNs. A CTBNC can be written using CTBN framework representing the class probability distribution with a prior distribution P^0_Y, and a CIM that does not allow transitions. The motivation to study CTBNCs is given by inference. Inference for CTBNs is an NP-hard problem. Nodelman et al. (2002a) developed the only known exact inference algorithm. It has an exponential complexity in time and space (see Section 3.1.1). Stella and Amer (2012) showed that with CTBNC it is possible to compute exact inference, in terms of classification over completely observable trajectories, in polynomial time.

According to Stella and Amer (2012) a CTBNC $\mathcal{C} = \{\aleph, P(Y)\}$ classifies a stream of continuous time evidence $\mathbf{z} = (x_1, x_2, ..., x_N)$ for the attributes $\mathbf{Z} = (X_1, X_2, ..., X_N)$ over J contiguous time intervals, i.e. a stream of continuous time evidence $\mathbf{Z}^{[t_1,t_2)} = \mathbf{z}^{[t_1,t_2)}$, $\mathbf{Z}^{[t_2,t_3)} = \mathbf{z}^{[t_2,t_3)}$, $\ldots$, $\mathbf{Z}^{[t_{J-1},t_J)} = \mathbf{z}^{[t_{J-1},t_J)}$, by selecting the value y^* for the class Y which maximizes the posterior probability

$$P(Y|\mathbf{z}^{[t_1,t_2)}, \mathbf{z}^{[t_2,t_3)}, ..., \mathbf{z}^{[t_{J-1},t_J)}),$$

which is proportional to

$$P(Y) \prod_{j=1}^{J} q^{pa(X_{m_j})}_{x^j_{m_j} x^{j+1}_{m_j}} \prod_{n=1}^{N} \exp\left(-q^{pa(X_n)}_{x^j_n} \delta_j\right), \tag{4}$$

where:

- $q^{pa(X_n)}_{x^j_n}$ is the parameter associated with state x^j_n, in which the variable X_n was during the j^{th} time interval, given the state of its parents $pa(X_n)$ during the j^{th} time intervals;
- $q^{pa(X_m)}_{x^j_m x^{j+1}_m}$ is the parameter associated with the transition from state x^j_m, in which the variable X_m was during the

j^{th} time interval, to state x_m^{j+1}, in which the variable X_m will be during the $(j+1)^{th}$ time interval, given the state of its parents $pa(X_m)$ during the j^{th} and the $(j+1)^{th}$ time intervals,

while $\delta_j = t_j - t_{j-1}$ is the length of the j^{th} time interval of the stream $\mathbf{z}^{[t_1,t_2)}, \mathbf{z}^{[t_2,t_3)}, \ldots, \mathbf{z}^{[t_{J-1},t_J)}$ of continuous time evidence.

The inference algorithm for CTBNCs (Algorithm 1) is described in (Stella and Amer, 2012). Stella and Amer (2012) used the Bayesian estimation to learn parameters for CTNBs (see Section 3.1.2). In Section 4 the Naive Bayes limitation is overcome by taking into account the structural learning problem for generic CTBNC. A new scoring function that uses Conditional log-likelihood is proposed for CTBNCs.

Example 3.2.1. Figure 3 depicts the structure of a CTBNC to diagnose eating disorders from the eating process (Figure 1). An example of the eating process is shown in Figure 4.

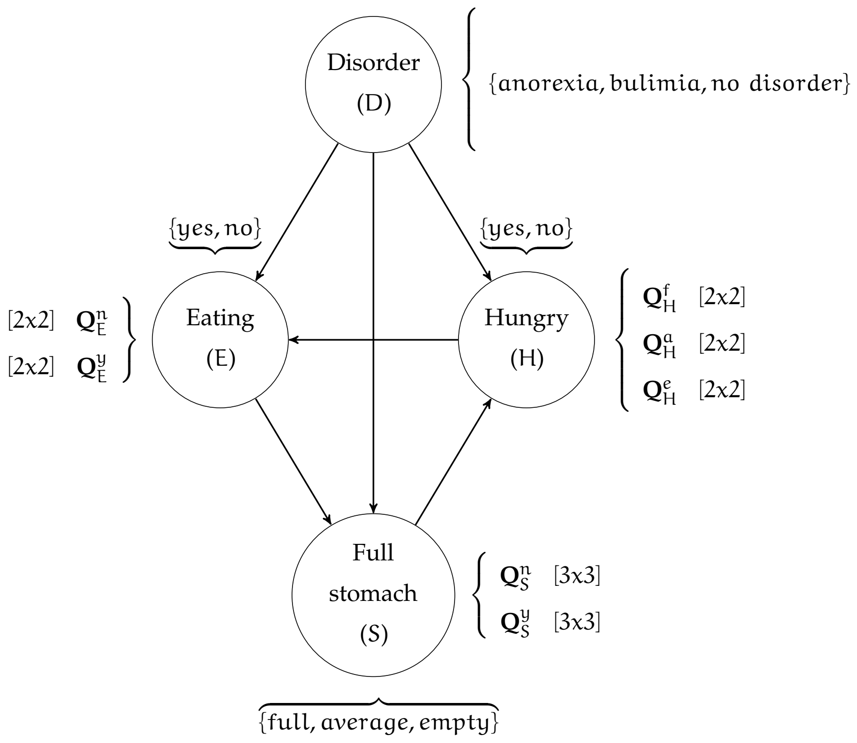


Figure 3.: CTBNC to diagnose eating disorders observing the eating process.

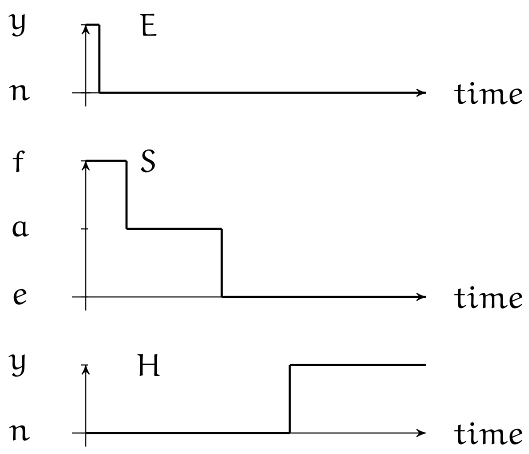


Figure 4.: Example of eating process.

Algorithm 1 Inference algorithm for CTBNCs.

Require: a CTBNC $\mathcal{C} = \{\aleph, P(Y)\}$ consisting of N attribute nodes and a class node Y such that $Val(Y) = \{y_1, y_2, ..., y_K\}$, a *fully observed evidence stream* $\left(\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^J\right)$.

Ensure: the maximum a posteriori classification y^* for the *fully observed evidence-stream* $\left(\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^J\right)$.

```
1:  for $k = 1$ to $K$ do
2:      $\log p(y_k) \leftarrow \log P(y_k)$
3:  end for
4:  for $k = 1$ to $K$ do
5:      for $j = 1$ to $J$ do
6:          for $n = 1$ to $N$ do
7:              $\log p(y_k) := \log p(y_k) - q_{x_n^j}^{pa(X_n)}(t_j - t_{j-1})$
8:              if $x_n^j \neq x_n^{j+1}$ then
9:                  $\log p(y_k) := \log p(y_k) + \log\left(q_{x_n^j x_n^{j+1}}^{pa(X_n)}\right)$
10:             end if
11:         end for
12:     end for
13: end for
14: $y^* \leftarrow \arg\max_{y \in Val(Y)} \log p(y)$.
15: return $y^*$
```

4

CTBNC STRUCTURAL LEARNING

In this Chapter the problem of structural learning for Continuous Time Bayesian Network Classifiers (CTBNCs) is addressed to overcome the Naive Bayes (NB) limitation of Stella and Amer (2012). A parallel between Static Bayesian Classifiers structural learning and Continuous Time Classifiers structural learning is made exploiting Friedman et al. (1997) contribute.

A Conditional Log-likelihood scoring function which outperform Marginal Log-likelihood score especially on limited amount of data is introduced.

A synthetic tests campaign is done by comparing the performances of different CTBNC models with Dynamic Bayesian Networks (DBNs).

Results on real data sets are shown in Chapter 6.

4.1 WHY LEARN THE STRUCTURE?

Stella and Amer (2012) introduced the Continuous Time Bayesian Network Classifiers (CTBNCs) framework, how to make inference on CTBNCs and defined Continuous Time Naive Bayes (CTNB) (Section A.1.2.2). In their work, using CTNBs, the problem of structural learning is not addressed.

CTNBs, as all the Naive Bayes (NB) models, assume the conditional independence of the features given the class. In the specialized literature this is a well known assumption for static Bayesian models (Friedman et al., 1997). In many cases conditional independence is not a realistic assumption and introduces errors which limit the classification performances. The main error sources for NB models are:

- when two or more features are not independent (see Example 4.1.1);
- when there is not a direct relation between a feature and the class node (see Example 4.1.1).

Example 4.1.1. Figure 5 shows an example of CTBNC where the naive Bayes assumption is a strong limitation. The network below is a toy classification model to discover the presence of traffic congestions on a target street. The features are: a sensor on the street where the traffic congestion must be discovered, a sensor on an adjacent street, the traffic light color at the end of the interested street, the traffic light color in the perpendicular direction, the weather condition and an unrelated node informing about the Pope's presence in Rome (Figure 5).

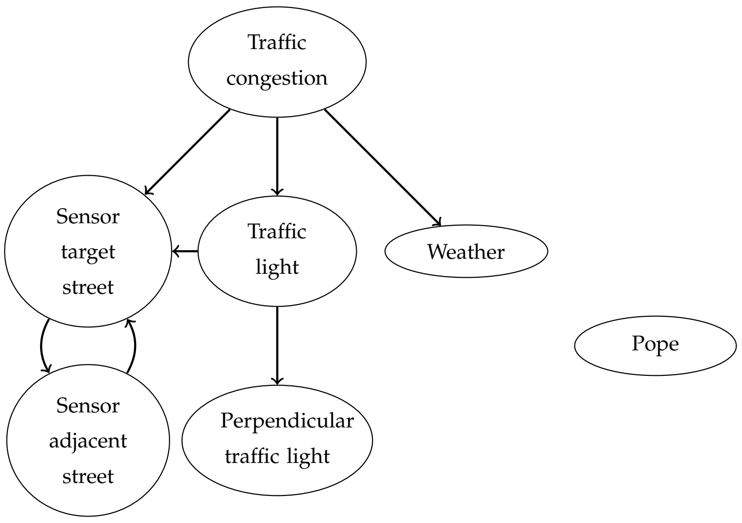


Figure 5.: CTBNC toy example for traffic congestion classification.

If conditional independence is forced, the following errors are introduced:

- the relation between the state of the sensor and the traffic light color is lost (i.e. a red traffic light can be a motivation for a queue on the sensor);
- the relations between the sensors in adjacent streets are lost;
- the independence between the traffic light in the perpendicular street and the classification problem, given the traffic light on the target street is lost (i.e. the colors of the perpendicular street are obviously related: the color in one direction informs about the color in the conflictual direction);
- the introduction of a fake dependency (i.e. noise) between the Pope's presence in Rome and the classification problem.

As shown in Example 4.1.1 the conditional independence assumption can introduce many errors that often bring to wrong classifications. To overcome the Naive Bayes limitations, in this section the problem of structural learning for CTBNCs is studied.

4.2 MAX-k CLASSIFIERS

The Tree-Augmented Naive Bayes (TAN) model was introduced by Friedman et al. (1997) with the aim to achieve a good tradeoff between the model's expressiveness and the model's complexity. In particular, the TAN was designed to model relationships between attributes and thus to overcome the strong assumption of conditional independence between attributes on which the Naive Bayes classifiers relies.

In the case where Bayesian Network (BN) classifiers are considered it must be taken into account the acyclicity constraint which limits the structure of the graph of the corresponding

Bayesian Network. Thus, the TAN model guarantees that structural learning time is still polynomial, while increasing the possibility to improve the classification accuracy compared to the one archived by Naive Bayes classifier.

However, when CTBNCs are considered, the acyclicity constraint is removed. Nevertheless, it is valuable to relax the conditional independence assumption of the CTNB bounding the structure dimension. Indeed, fixing the maximum number of parents, structural learning in Continuous Time Bayesian Networks (CTBNs) is polynomial with respect to the number of variables and the size of the data set, but the number of parameters for a node is exponential with respect to the node parents. Too many parents leads to a heavy computational effort in the structural learning process. Increasing the maximum number of parents also implies working with more data for learning the node parameters conditioned on all the possible parent instantiations. In contrast with static Bayesian classifiers, in CTBNCs there are no reasons to constrain the graph associated with the classifier to have a tree structure. The above consideration suggests the need to define the following instances of the class of CTBNCs: the max-k Augmented Continuous Time Naive Bayes (ACTNB) and the max-k CTBNC.

Definition 4.2.1. (Max-k Continuous Time Bayesian Network Classifier (Codecasa and Stella, 2013)). A max-k Continuous Time Bayesian Network Classifier is a couple $\mathcal{M} = \{\mathcal{C}, k\}$, where $\mathcal{C}$ is a CTBNC $\mathcal{C} = \{\aleph, P(Y)\}$ such that the number of parents $|Pa(X_n)|$ for each attribute node X_n is bounded by a positive integer k. Formally, the following condition holds; $|Pa(X_n)| \leqslant k$, $n = 1, 2, ..., N$, $k > 0$.

Definition 4.2.2. (Max-k Augmented Continuous Time Naive Bayes (Codecasa and Stella, 2013)). A max-k Augmented Continuous Time Naive Bayes is a max-k CTBNC such that the class node Y belongs to the parents set of each attribute node X_n, $n = 1, 2, ..., N$. Formally, the following condition holds; $Y \in Pa(X_n)$, $n = 1, 2, ..., N$.

The definition of max-k CTBNC allows to conclude that a CTNB is a max-1 Augmented Continuous Time Naive Bayes. Max-k ACTNBs are particular cases of max-k CTBNCs. The definition of max-k ACTNB constrains the class variable Y to be a parent for each attribute node X_n, $n = 1, 2, ..., N$. The rationale behind this constraint is to compensate the fact that some direct dependencies between the attribute nodes are discarded due to the upper bound in the number of parents.

4.3 STRUCTURAL LEARNING

4.3.1 *Bayesian scoring*

Learning a CTBNC from data consists of learning a CTBN model where a specific node, i.e. the class node Y, does not depend on time (see Section 3.1.3). The constraint that the class node Y must have no parents makes the learning problem straightforward. Indeed, in such a particular case the learning algorithm runs, for each attribute node X_n, $n = 1, 2, ..., N$, a local search procedure to find its optimal set of parents, i.e. the set of parents which maximizes a given score function. Furthermore, for each attribute node X_n, $n = 1, 2, ..., N$, no more than L parents are selected.

The structural learning algorithm proposed by Nodelman et al. (2002b) uses the score function $\mathbf{score}(\mathcal{G} : \mathcal{D})$ (3). The choice of the scoring function to be optimized is fundamental for structural learning of CTBNCs. Nodelman et al. (2002b) proposed a Bayesian scoring function to learn CTBN models (Equation 3). The same learning algorithm can be adapted to learn a CTBNC by introducing the constraint that the class node Y does not depend on time.

4.3.2 *Conditional log-likelihood scoring*

4.3.2.1 *Why conditional log-likelihood?*

Scores based on the Log-likelihood are not the only scoring functions which can be used to learn the structure of a CTBNC. Following what was presented and discussed by Friedman et al. (1997), the log-likelihood function consists of two components:

$$\begin{aligned} LL(\mathcal{M} \mid \mathcal{D}) &= \textstyle\sum_{i=1}^{|\mathcal{D}|} \log P_{\aleph}(y_i \mid \mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i}) \\ &\quad + \log P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i}). \end{aligned} \quad (5)$$

The first component, i.e. $\log P_{\aleph}(y_i \mid \mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i})$, measures the classification capability of the model. The second component, i.e. $\log P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i})$, measures how well the model represents the join distribution of the attributes.

Friedman et al. (1997) remarked that in the case where the number of attribute nodes X_n, $n = 1, 2, ..., N$ is large, the contribution to the likelihood function (Equation 5) of the second component overwhelms the contribution of the first component. However, the contribution of the second component is not directly related to the classification performance achieved by the learned classifier.

Therefore, to improve the classification performance Friedman et al. (1997) suggested to use the *conditional log-likelihood* in the scoring functions. In such a case the maximization of the conditional log-likelihood results in maximizing the classification performance of the model without paying specific attention to the discovery of the existing dependencies between the attribute nodes X_n, $n = 1, 2, ..., N$.

4.3.2.2 *Conditional log-likelihood scoring*

CTBNCs, as static Bayesian classifiers, can be learned by using log-likelihood and conditional log-likelihood as well.

In the case where CTBNCs are considered, the conditional log-likelihood can be written as follows (Codecasa and Stella, 2013):

$$\begin{aligned}
\mathrm{CLL}(\mathcal{M} \mid \mathcal{D}) &= \sum_{i=1}^{|\mathcal{D}|} \log P_{\aleph}(y_i \mid \mathbf{x}_i^1, \dots, \mathbf{x}_i^{J_i}) \\
&= \sum_{i=1}^{|\mathcal{D}|} \log \left(\frac{P_{\aleph}(\mathbf{x}_i^1, \dots, \mathbf{x}_i^{J_i} \mid y_i) P_{\aleph}(y_i)}{P_{\aleph}(\mathbf{x}_i^1, \dots, \mathbf{x}_i^{J_i})} \right) \\
&= \sum_{i=1}^{|\mathcal{D}|} \log \left(P_{\aleph}(y_i) \right) \\
&\quad + \sum_{i=1}^{|\mathcal{D}|} \log \left(P_{\aleph}(\mathbf{x}_i^1, \dots, \mathbf{x}_i^{J_i} \mid y_i) \right) - \\
&\quad - \sum_{i=1}^{|\mathcal{D}|} \log \left(\sum_{y'} P_{\aleph}(y') P_{\aleph}(\mathbf{x}_i^1, \dots, \mathbf{x}_i^{J_i} \mid y') \right) .
\end{aligned} \tag{6}$$

It is clear from (6) that the conditional log-likelihood function consists of the following three terms: the *class probability term* (7), the *posterior probability term* (8), and finally the *denominator term* (9). These three terms can be estimated by using the available data set $\mathcal{D}$ as described in the following.

The *class probability term* is estimated as follows:

$$\sum_{i=1}^{|\mathcal{D}|} \log \left(P_{\aleph}(y_i) \right) = \sum_{y} M[y] \log(\theta_y) \tag{7}$$

where θ_y represents the parameter associated with the probability of class y (see Section 3.1).

From Equation (4) it is possible to write the following:

$$
\begin{aligned}
&P_{\aleph}(\mathbf{x}^1, ..., \mathbf{x}^J \mid y) = \\
&= \prod_{j=1}^{J} q_{x^j_{m_j} x^{j+1}_{m_j}}^{pa(X_{m_j})} \prod_{n=1}^{N} \exp\left(-q_{x^j_n}^{pa(X_n)} \delta_j\right) \\
&= \prod_{j=1}^{J} q_{x^j_{m_j}}^{pa(X_{m_j})} \theta_{x^j_{m_j} x^{j+1}_{m_j}}^{pa(X_{m_j})} \prod_{n=1}^{N} \exp\left(-q_{x^j_n}^{pa(X_n)} \delta_j\right)
\end{aligned}
$$

Therefore, the *posterior probability term* can be estimated as follows:

$$
\begin{aligned}
&\textstyle\sum_{i=1}^{|\mathcal{D}|} \log\left(P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i} \mid y_i)\right) = \\
&= \sum_{n=1}^{N} \sum_{x_n, pa(X_n)} M[x_n \mid pa(X_n)] \log\left(q_{x_n}^{pa(X_n)}\right) - \\
&\quad - q_{x_n}^{pa(X_n)} T[x_n \mid pa(X_n)] + \\
&\quad + \sum_{x'_n \neq x_n} M[x_n x'_n \mid pa(X_n)] \log(\theta_{x_n x'_n}^{pa(X_n)}).
\end{aligned} \tag{8}
$$

The *denominator term* is similar to the first two components. Unfortunately, because of the sum the *denominator term* cannot be further decomposed, while sufficient statistics allow us to write:

$$
\begin{aligned}
&\textstyle\sum_{i=1}^{|\mathcal{D}|} \log\left(\sum_{y'} P_{\aleph}(y') P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i} \mid y')\right) = \\
&= \log\left(\sum_{y'} \theta_{y'} \prod_{n=1}^{N} \prod_{x_n, pa'(X_n)} (q_{x_n}^{pa'(X_n)})^{M[x_n \mid pa'(X_n)]}\right. \\
&\quad \exp(-q_{x_n}^{pa'(X_n)} T[x_n \mid pa'(X_n)]) \\
&\quad \left.\prod_{x'_n \neq x_n} (\theta_{x_n x'_n}^{pa'(X_n)})^{M[x_n x'_n \mid pa'(X_n)]}\right)
\end{aligned} \tag{9}
$$

where $pa(X_n) = \{\pi_n \cup y\}$, $pa'(X_n) = \{\pi_n \cup y'\}$, while π_n is the instantiation of the non-class parents of the attribute node X_n.

Unfortunately, no closed form solution exists to compute the optimal value of the model parameters, i.e. those parameter values which maximize the conditional log-likelihood (6). Therefore, the approach introduced and discussed by Grossman and Domingos (2004) for static classifiers is followed. The scoring function is computed by using the conditional log-likelihood, while parameters values are obtained by using the Bayesian approach as described by Nodelman et al. (2002b).

4.4 SYNTHETICS TEST

Considering the Bayesian score (Equation 3), the prior distribution of the structure (i.e. $\ln P(\mathcal{G})$) becomes less relevant with the increasing of the data set dimension. In the case where the data set dimension tends to infinity (i.e. $|\mathcal{D}| \to \infty$), the Bayesian score is equivalent to the Marginal Log-likelihood (MLL) score (i.e. $\mathbf{MLLscore}(\mathcal{G} : \mathcal{D}) = \ln P(\mathcal{D}|\mathcal{G})$). To fairly compare the classification performance achieved when using the Conditional Log-likelihood (CLL) score (Equation 6), which does not use a graph's structure penalization term, the Marginal Log-likelihood score is used instead of the Bayesian score.

Numerical experiments are devoted to comparing the performance of the following types of continuous time Bayesian Network classifiers; CTNB, $k = 2$ ACTNB, $k = 2$ CTBNC, $k = 3$ CTBNC, and $k = 4$ CTBNC. Each classifier is associated with a suffix which informs about the particular scoring function used for its structural learning. In particular, the suffix MLL will be used when the Marginal Log-likelihood scoring function is optimized, while the CLL suffix is used when the Conditional Log-likelihood scoring function is optimized for structural learning. It is worthwhile to notice that when the structural learning is performed by optimizing the conditional log-likelihood scoring function, the parametric learning task is based on Bayesian estimation.

In this section the tests performed by exploiting synthetic data sets are described. Real data sets are addressed in Chapter 6.

4.4.1 *Tests campaign*

Synthetic data sets, used to compare classification performances, learning time, and inference time of different classification models, are generated by sampling structures of increasing complexity. The rationale for this choice is that simple classifiers (i.e. Naive Bayes models) tend to work well for simple problems, while are not capable of modeling complex structures. On the other hand, classifiers that can learn complex structures are subject to overfitting.

4.4.1.1 *Competitors' models*

Continuous Time Classifiers' performances achieved by using MLL and CLL scoring functions are compared with the performances achieved by Dynamic Bayesian Network Classifiers (DBNC) (Section 2.2.2).

Regarding Dynamic Bayesian Network Classifiers (DBNCs), experiments are made by using Naive Bayes models and models where the structure over adjacent time slices is learned. MATLAB Bayesian Nets toolbox (Murphy et al., 2001) is used for the experiments.

Two Naive Bayes models are tested. In both cases each variable depends on its state at the previous time slice, but the first model (DBN-NB1) allows the Naive Bayes relations intra slice (i.e. Figure 6a), while the second model (DBN-NB2) allows the Naive Bayes relations through consecutive time slices (i.e. Figure 6a).

Two models are used also in the case where the dependencies

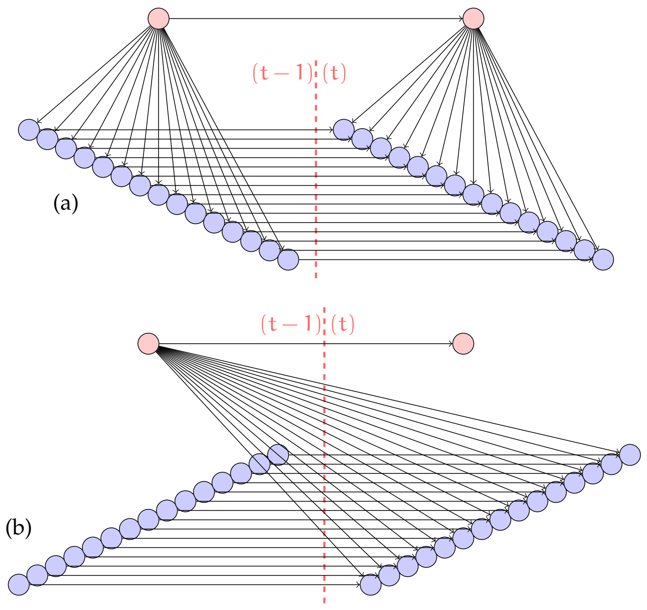


Figure 6.: DBN-NB1 (a) and DBN-NB2 (b) tested models.

over time are learned. The first one (DBNC1) does not allow any intra slice dependencies. The second (DBNC2) allows only Naive Bayes intra slice dependencies, similarly to the DBN-NB1 model. In both cases the maximum number of parents from the previous time slice is set to 2.

It is valuable to recall that Dynamic Bayesian Networks (DBNs) use more parameters compared to the corresponding CTBNs. This is because the DBNs capability to model intra and extra time slice parental relations. This is because to the discrete time models need to capture all the information lost in the time between two consecutive time slices. This is well explained by the concept of Granger causality (Granger, 1969). Granger (1969) shows how causality happens only in time: the appearance of instantaneous causality "arises due to slowness in recording information or because a sufficiently wide class of possible causal variables has not been used".

4.4.1.2 *CTBN synthetic data sets*

Part of the data sets are generated by sampling from CTBNC models of different complexity. While the structure of the models is given, the values of the parameters associated with each of its nodes (q parameters) are obtained by sampling from a given interval. Each pair (*structure*, *parameters assignment*), is used to generate a learning data set.

Data sets consist of $1,000$ trajectories with average length ranging from 300 (CTNBs) to 1,400 (max-4 CTBNCs). Analyzed model structures are CTNB (Figure 7(a)), max-2 ACTNB (Figure 7(b,c)), max-2 CTBNC (Figure 7(d-f)), max-3 CTBNC (Figure 7(g-i)), and max-4 CTBNC (Figure 7(j-l)).

Table 1 summarizes the structures used to generate the synthetic data sets. The table shows the class cardinality and the sampling interval for each node. Performance comparison is based on a 10 fold cross validation on *full data sets* (100%) and *reduced data sets*, i.e. when the number and the length of trajectories are reduced to: 80%, 60%, 40%, and 20%.

Figure 7 shows the structures of the CTBNC models used for generating the synthetic data sets. The number associated with each node represents the cardinality of the corresponding attribute. The color of each node is associated with the interval used to sample the values of the q parameters (see Table 1).

DBNs classifiers are used to compare CTBNCs with the state of the art in multivariate trajectory classification. For this reason a sampling ratio is used to discretize the continuous time trajectories in order to apply DBNCs. Because of the computational effort to deal with DBNs, it was necessary to force a sampling ratio that generates no more then 50 time slices per trajectory. In the same way the generated data sets were too big to allow the structural learning of DBNs. For this reason just DBN-NB1

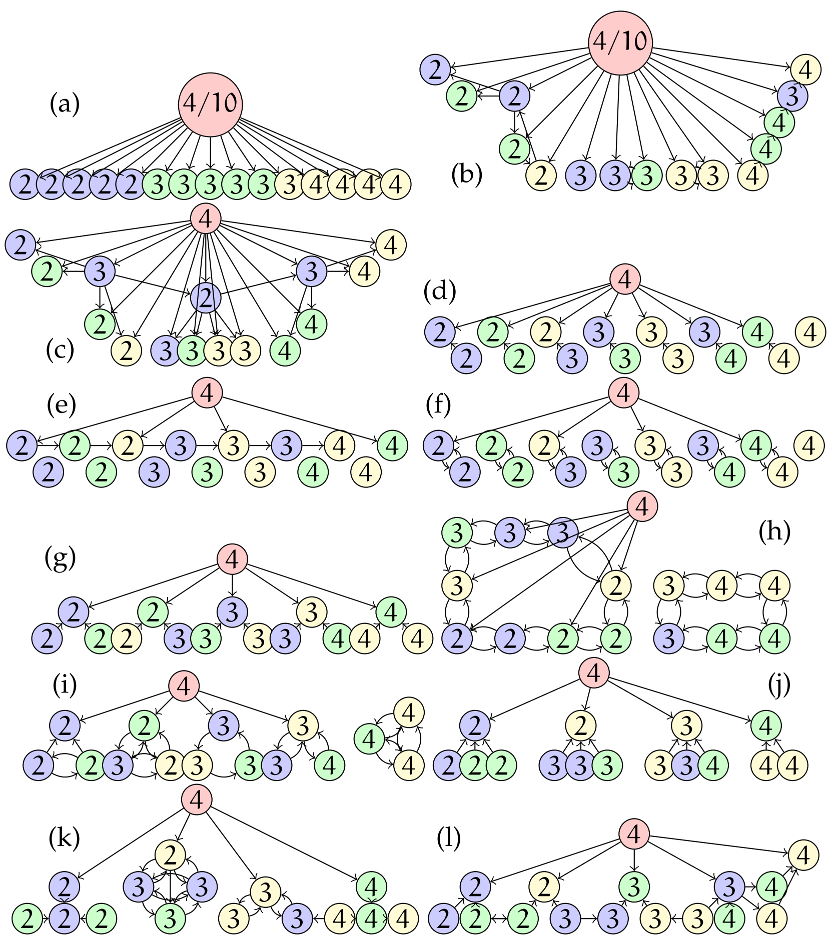


Figure 7.: CTNB (a), max-2 ACTNB (b,c), max-2 CTBNC (d-f), max-3 CTBNC (g-i) and, max-4 CTBNC (j-l) tested structures. Numbers associated with nodes represent the cardinality of the corresponding variables. Red nodes are associated with classes, while the colors of the other nodes represent the intervals used to sample the q parameters (see Table 1).

and DBN-NB2 models are used for comparison of the continuous time approaches over the data sets generated from CTBNCs.

4.4.1.3 *DBN synthetic data sets*

Continuous Time Bayesian Network Classifiers (CTBNCs) are compared with Dynamic Bayesian Network Classifiers (DBNCs). To have a fair comparison a portion of the synthetic data sets are generated starting from Dynamic Bayesian Networks (DBNs).

Test name	Figure	# classes red	qs range blue	qs range green	qs range yellow	Test name	Figure	# classes red	qs range blue	qs range green	qs range yellow
CTNB1	7a	4	[1, 2]	[2, 4]	[4, 8]	k2ACTNB1	7b	4	[1, 2]	[2, 4]	[4, 8]
CTNB2	7a	4	[1, 2]	[4, 8]	[8, 16]	k2ACTNB2	7b	4	[10, 20]	[20, 40]	[40, 80]
CTNB3	7a	4	[10, 20]	[20, 40]	[40, 80]	k2ACTNB3	7b	10	[10, 20]	[20, 40]	[40, 80]
CTNB4	7a	10	[1, 2]	[2, 4]	[4, 8]	k2ACTNB4	7c	4	[1, 2]	[2, 4]	[4, 8]
CTNB5	7a	10	[10, 20]	[20, 40]	[40, 80]	k2ACTNB5	7c	4	[10, 20]	[20, 40]	[40, 80]
k2CTBNC1	7d	4	[1, 2]	[2, 4]	[4, 8]	k3CTBNC1	7g	4	[1, 2]	[2, 4]	[4, 8]
k2CTBNC2	7d	4	[10, 20]	[20, 40]	[40, 80]	k3CTBNC2	7g	4	[10, 20]	[20, 40]	[40, 80]
k2CTBNC3	7e	4	[1, 2]	[2, 4]	[4, 8]	k3CTBNC3	7h	4	[1, 2]	[2, 4]	[4, 8]
k2CTBNC4	7e	4	[10, 20]	[20, 40]	[40, 80]	k3CTBNC4	7h	4	[10, 20]	[20, 40]	[40, 80]
k2CTBNC5	7f	4	[1, 2]	[2, 4]	[4, 8]	k3CTBNC5	7i	4	[1, 2]	[2, 4]	[4, 8]
k2CTBNC6	7f	4	[10, 20]	[20, 40]	[40, 80]	k3CTBNC6	7i	4	[10, 20]	[20, 40]	[40, 80]

Test name	Figure	# classes red	qs range blue	qs range green	qs range yellow
k4CTBNC1	7j	4	[1, 2]	[2, 4]	[4, 8]
k4CTBNC2	7j	4	[10, 20]	[20, 40]	[40, 80]
k4CTBNC3	7k	4	[1, 2]	[2, 4]	[4, 8]
k4CTBNC4	7k	4	[10, 20]	[20, 40]	[40, 80]
k4CTBNC5	7l	4	[1, 2]	[2, 4]	[4, 8]
k4CTBNC6	7l	4	[10, 20]	[20, 40]	[40, 80]

Table 1.: Summary of the CTBNC tested structures. *Test name* represents the name of the structure. *Figure* specifies the figure number of the structure. *# classes* represents the number of classes. The remaining columns show the sampling interval for the value of the q parameter for each node, depending on its color as depicted in the figures.

Trajectory generation follows the same guideline explained in Section 4.4.1.2 for continuous time trajectory generation. Models of growing complexity are used to generate discrete time trajectories by using DBNCs with a maximum of 2, 3, and 4 parents per node. Parameters of DBNC nodes are sampled from the Dirichelet distribution, except for first time slice and class nodes whose parameter values are sampled from the uniform distribution. Because CTBNC makes inference about a static class that will occur in the future, DBNC data sets have been generated by models where the class value does not change over time.

Figure 8 shows two DBNs models used to generate the data sets. The other three models have a similar structure, but each node has a greater number of parental relations from the previous time slice.

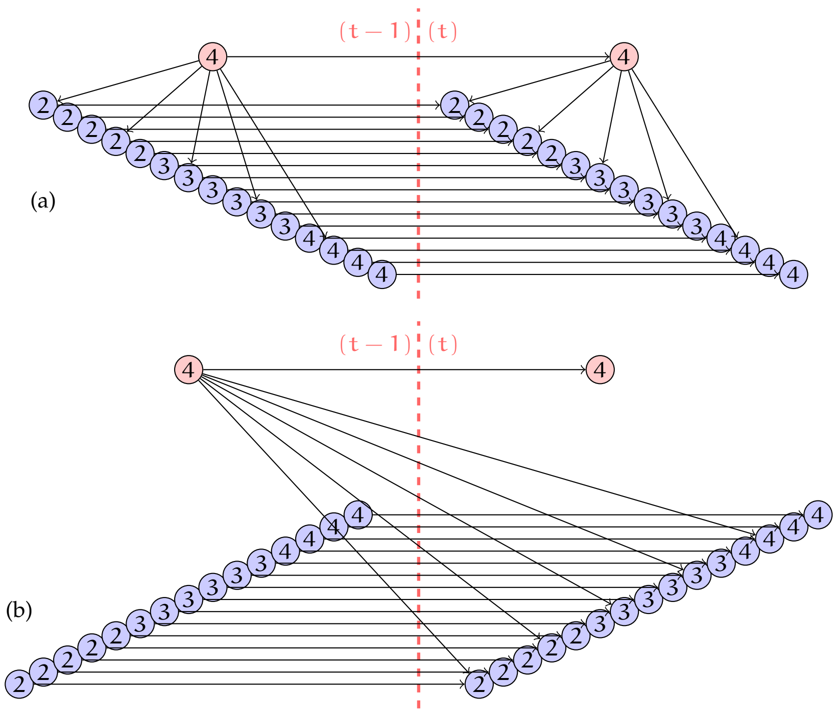


Figure 8.: Example of DBN models used to generate discrete time data sets. Red nodes are associated with classes. Numbers represent the cardinality of each node. Figure (a) depicts the structure of DBNTest1 model. DBNTest2 has the same class relations, but blue nodes in the first time slice are connected with 2 nodes of the next time slice. Figure (b) depicts the structure of DBNTest3. DBNTest4 and DBNTest5 have the same class relations, but blue nodes in the first time slice are connected with 2 (DBNTest4) and 3 (DBNTest5) nodes of the next time slice.

All five data sets consists of 100 trajectories of 20 time slices each. Because DBNC data sets are simpler than data sets generated by using CTBNCs (20 time slices instead of 50), tests show the performance comparisons between all CTBNC models, the DBNC models tested in the CTBNC data sets (i.e. DBN-NB1 and DBN-NB2) and the DBNC models where the intra-slice dependencies are fixed, while the dependencies over time are learned (i.e. DBNC1 and DBNC2) (see Section 4.4.1.1).

4.4.2 *Results*

4.4.2.1 *CTBN synthetic data sets*

Accuracy values on *full data sets* (100% data sets) are summarized in Table 2. The Table shows that DBNCs are strongly outperformed by all the continuous time models, CTNB included.

Test	CTNB	$k=2$ ACTNB (MLL)	$k=2$ ACTNB (CLL)	$k=2$ CTBNC (MLL)	$k=2$ CTBNC (CLL)	$k=3$ CTBNC (MLL)	$k=3$ CTBNC (CLL)	$k=4$ CTBNC (MLL)	$k=4$ CTBNC (CLL)	DBN-NB1	DBN-NB2
CTNB	**0.95**	**0.95**	0.93	0.93	0.92	0.93	0.82	0.93	0.64	0.80	0.81
K2ACTNB	0.78	0.89	**0.92**	0.76	**0.92**	0.76	0.85	0.76	0.72	0.62	0.63
K2CTBNC	0.68	0.84	**0.86**	**0.85**	**0.86**	**0.85**	0.76	**0.85**	0.60	0.48	0.50
K3CTBNC	0.49	0.65	0.63	0.66	0.63	**0.79**	0.75	**0.79**	0.64	0.32	0.33
K4CTBNC	0.64	0.74	0.79	0.69	0.79	0.76	**0.94**	0.79	0.90	0.40	0.40

Table 2.: Classifier's average accuracy value with respect to different categories of the data set generating model, 10 fold cross validation over 100% data sets. Bold characters are associated with the best model with 90% confidence.

Figure 9 depicts the best average performances between DBNCs, CTNB and, CTBNCs when Marginal Log-likelihood and Conditional Log-likelihood scoring functions are used. The fig-

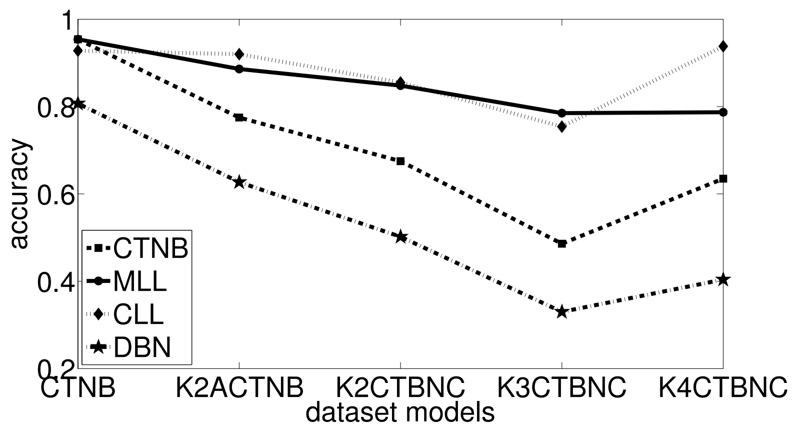


Figure 9.: Comparison of the best average performances between CTBNCs learned by using MLL and CLL scores, CTNB and DBN models. X-axis is associated with the model used to generate the data sets. The continuous line is associated with MLL, the thin dotted line with CLL, the big dotted line with CTNB and the fragmented line with DBN.

ure shows that DBNCs are strongly outperformed by the continuous time models over all the tested levels of structure complexity.

CTNB performs well on simple structures and obviously loses its effectiveness gradually when the maximum number of parents of the data set models is increased. On the max-4 CTBNCs data sets the CTNBs performances are still poor, but not as much as expected compared to the other models. Probably this is due to the necessity to have more data to learn complex structures such as max-4 models. In these cases a simple model (i.e. CTNB) brings suboptimal results, but does not require a significant increase of the amount of data. This is confirmed by the tests on the reduced data sets, shown in the following part of the section (see also Figure 11).

Figure 9 also shows the performances of continuous time models learned with both the scoring functions. Performance are comparable except that for the max-4 data sets. The reason of this is the ability of Conditional Log-likelihood scoring function to perceive weak dependencies, even when this is due to the shortage of data because of the increasing of the structure complexity. Indeed, the increase of the number of node parents leads to the necessity of learning the node parameters over a greater amount of data. This is confirmed by Figure 10.

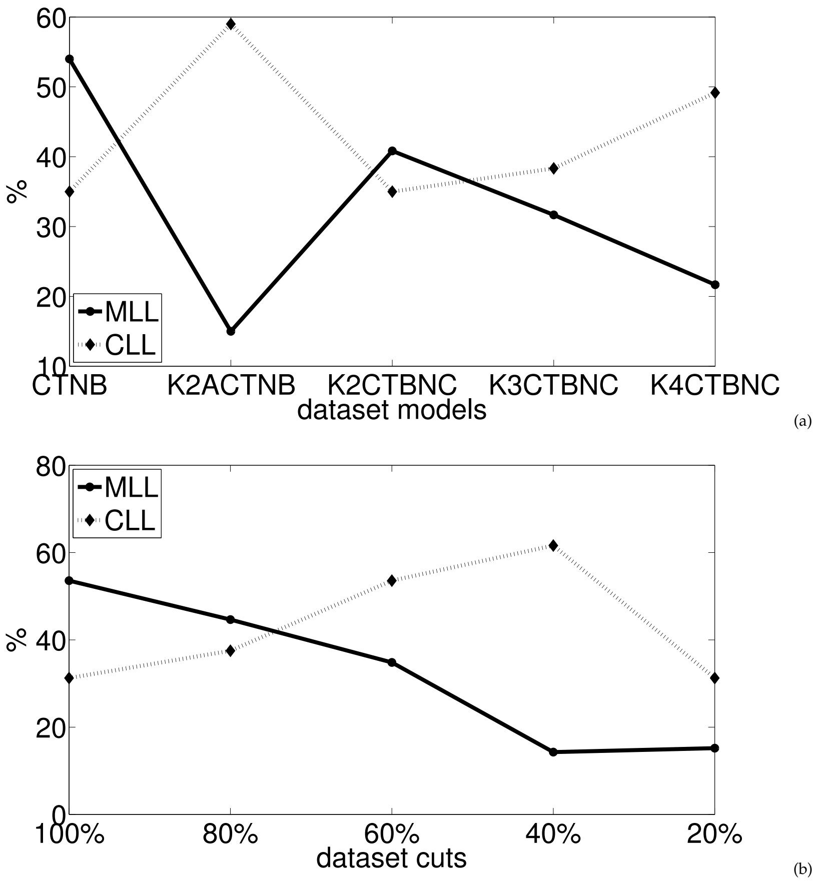


Figure 10.: Percentage of tests when the MLL (CLL) scoring is better than the CLL (MLL) scoring with 90% confidence and while considering all the reduced data sets. X-axis in Figure (a) is associated with the models used to generate the data sets. X-axis in Figure (b) is associated with the data sets' percentage reduction. The continuous line is associated with the Marginal Log-likelihood scoring function, while the thin dotted line is associated with the Conditional Log-likelihood scoring function.

Figure 10 depicts the comparison between Marginal Log-likelihood and Conditional Log-likelihood scoring functions when the reduced data sets are also considered. In detail, Figure 10 shows the percentage of tests in which the choice of one scoring function instead of the other one gives better performances. The comparison is made fixing the structure complexity of the models (i.e. $k = 2$ ACTNB, $k = 2$ CTBNC, ...) and using 90% of confidence[1]. When a great amount of data is provided Marginal Log-likelihood seems to perform better than Conditional Log-likelihood, even if the best models learned with the two scores generate the same average performances (Figure 9). This is probably due to the tendency of Conditional Log-likelihood to overfit (see Figure 12). Instead, Conditional Log-likelihood scoring strongly outperforms Marginal Log-likelihood scoring when the amount of data is limited (Figure 10). This is due, as outlined before, to the effectiveness of the Conditional Log-likelihood scoring function to discover weak dependencies between variables and thus to its tendency to add the class variable as a parent of all nodes which are useful for the classification task. The capability of the Conditional Log-likelihood scoring function to discover the dependencies between the class node and the features can be observed in Figure 10a where Conditional Log-likelihood strongly outperforms Marginal Log-likelihood on the ACTNB data set. On the other hand, when the amount of data is too scarce, Conditional Log-likelihood scoring tends to overfit by learning classifiers which are too complex for the available amount of data. In these cases, where also Marginal Log-likelihood scoring achieves poor accuracy, simple models like CTNBs are the best option (see Figure 11).

1 The values do not sum to 100% because the case when both scoring functions are comparable must be taken into account.

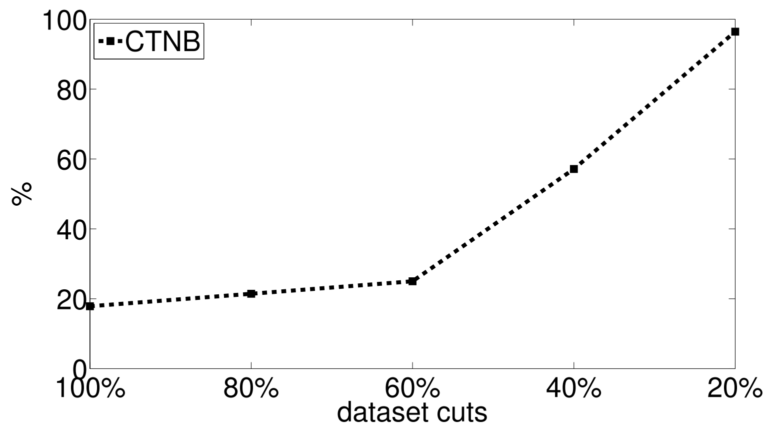


Figure 11.: Percentage of tests in which the CTNB is better or comparable to the best Continuous Time model. The x-axis is associated with the data sets' percentage reduction.

The tendency of the Conditional Log-likelihood scoring function to overfit can be seen in Figure 12. The Figure depicts the structures learned by the different Continuous Time models. The Conditional Log-likelihood scoring function seems to facilitate the presence of arcs (see Figure 12e-g). The structure of the Conditional Log-likelihood scoring function tends to favor an on-off behavior, either many parents or no parents are added to each attribute node. On the contrary, learning by optimizing the Marginal Log-likelihood scoring function tends to miss many arcs with specific reference to data sets where the causal relation is weak (see Figure 12d).

This behavior of the two scoring functions allows to see in the $k = 2$ACTNB classifier learned by optimizing the Conditional Log-likelihood scoring as a good trade-off between model complexity and effectiveness. Of course, the performance of a model strongly depends on the structure of the models that generate the data set. Nevertheless, $k = 2$ ACTNB allows to limit the maximum number of parents; at the same time it allows, thanks to the Conditional Log-likelihood scoring function, to exploit only those relationships between attribute nodes which

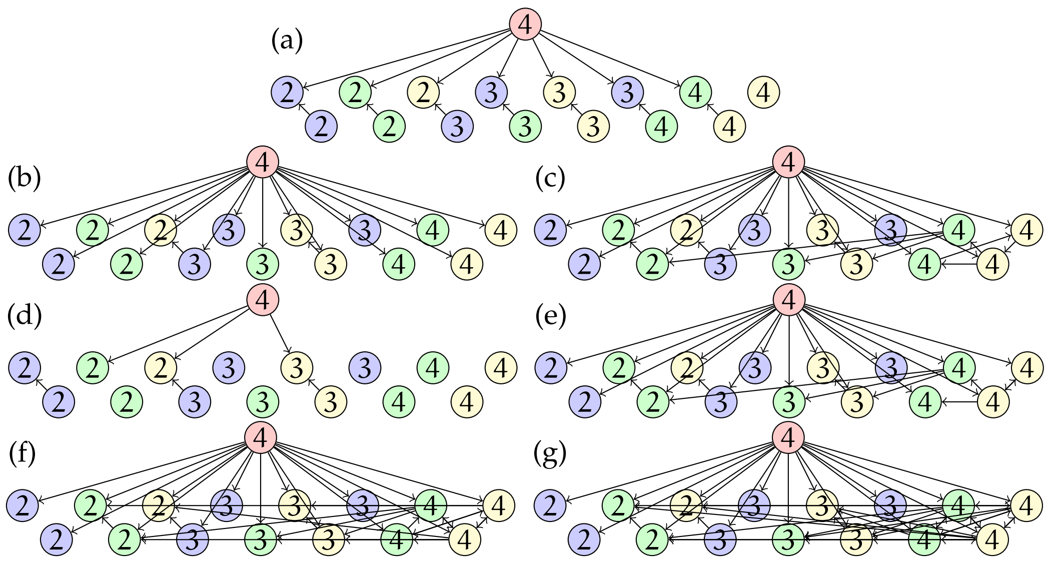


Figure 12.: k2CTBNC1 tests: (a) real structure, (b) structure learned by $k = 2$ACTNB-MLL, (c) structure learned by $k = 2$ACTNB-CLL, (d) structure learned by $k = 2$CTBNC-MLL, $k = 3$CTBNC-MLL, and $k = 4$CTBNC-MLL (e) structure learned by $k = 2$CTBNC-CLL, (f) structure learned by $k = 3$CTBNC-CLL, (g) structure learned by $k = 4$CTBNC-CLL. The learned structures are the same across all folds.

improve the classification performance. This is in contrast to what happens when learning a $k = 2$ ACTNB classifier by optimizing the MLL score function, which tends to miss weak dependency.

Inference times on continuous time models are comparable, while DBNCs inference strongly depends on the time discretization rate. Tests are made by using a rate that reduces the number of data rows. This was mandatory to make the tests feasible using DBNCs. For this reason it is impossible to exactly compare the time performances between continuous time classifiers and DBNCs. Nevertheless, it is clear that dealing with discrete time models requires more computational efforts than working in continuous time. This is because discrete models need to unroll many time slices, while continuous time models implicitly represent time (Nodelman, 2007).

Learning and inference times are summarized in Figure 13. Numerical experiments have been performed on Intel(R) Xeon(R) CPU X5670 2.93GHz, 15Gb RAM. Inference time is almost the same for all the continuous time classifiers (Figure 13b), while learning time varies across classifiers because of the different values of parent bounds (Figure 13a). There is not a

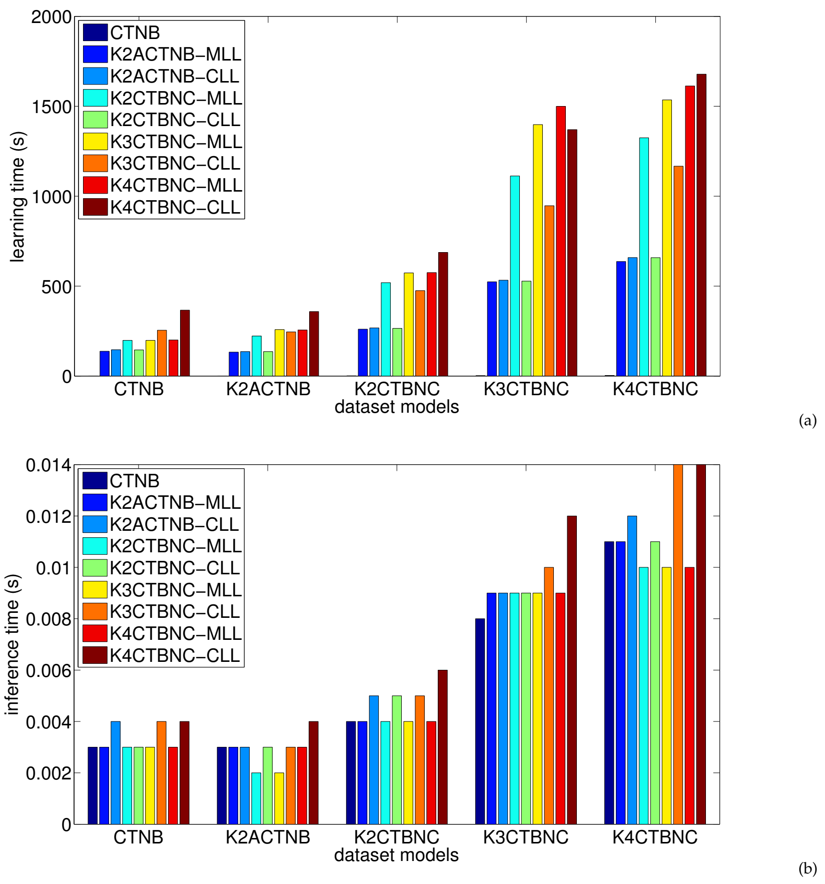


Figure 13.: Average learning (a) and inference (b) time for each model; in order: CTNB, k = 2ACTNB-MLL, k = 2ACTNB-CLL, k = 2CTBNC-MLL, k = 2CTBNC-CLL, k = 3CTBNC-MLL, k = 3CTBNC-CLL, k = 4CTBNC-MLL, k = 4CTBNC-CLL. X-axis is associated with the models used to generate the data sets.

clear relation between learning time required by MLL and CLL.

Theoretically CLL should be more expensive to compute and should require little additional time than MLL, but probably due to the hill climbing algorithm and the structures induced by CLL, it happens that often learning with CLL is faster than learning with MLL. Since CTNB does not require structural learning, but only parameter learning, it is the fastest model to learn.

It is worthwhile to mention that the $k = 2$ACTNB classifier, learned by optimizing the Conditional Log-likelihood score function, also implements a good trade-off between classification accuracy and learning time.

4.4.2.2 *DBN synthetic data sets*

Accuracy values on data sets generated by using DBN modes are reported in Table 3.

Test	CTNB	$k = 2$ ACTNB (MLL)	$k = 2$ ACTNB (CLL)	$k = 2$ CTBNC (MLL)	$k = 2$ CTBNC (CLL)	$k = 3$ CTBNC (MLL)	$k = 3$ **CTBNC !** (CLL)	$k = 4$ CTBNC (MLL)	$k = 4$ CTBNC (CLL)
DBNTest1	**1.00**	**1.00**	**1.00**	**0.99**	0.37	**0.99**	0.35	**0.99**	0.35
DBNTest2	0.86	0.91	**0.98**	0.85	0.26	0.85	0.26	0.85	0.26
DBNTest3	**1.00**	**1.00**	**1.00**	**0.98**	0.67	**0.98**	0.67	**0.98**	0.67
DBNTest4	0.82	0.82	**0.96**	0.81	0.35	0.81	0.35	0.81	0.35
DBNTest5	0.64	0.64	**0.73**	0.64	0.2	0.64	0.2	0.64	0.2

Test	DBN-NB1	DBN-NB2	DBNC1	DBNC2
DBNTest1	**1.00**	**1.00**	**1.00**	**1.00**
DBNTest2	0.90	0.91	**0.95**	0.91
DBNTest3	**0.98**	**0.98**	**0.99**	0.95
DBNTest4	0.80	0.83	0.87	0.74
DBNTest5	0.63	**0.68**	0.62	0.64

Table 3.: Classifiers' accuracy values on the discrete time data sets. Tests made with 10 fold cross validation. Bold digits are associated with the best model with 90% confidence.

Results show that complex CTBNCs do not perform well while CTNB, $k = 2$ ACTNB-MLL and $k = 2$ ACTNB-CLL achieve good performances. This confirms what emerged from the analysis on continuous time synthetic data sets (Section 4.4.2.1) where a too small amount of data made the simple models the most effective ones.

Particularly surprising is the behavior of CTBNCs when learned with the CLL scoring function. In this case CLL seems not able to learn the dependencies between the nodes, in contradiction with what happened on the continuous time synthetic data sets. A reason could be the discretization process. Continuous time trajectories hide two dependency factors: the time ratio of the transitions and the transition probabilities between states. In case of discrete trajectories the time dependencies are less evident and this, added to the short length of the trajectories, seems to have a strong impact on CLL performances.

It is worthwhile to note that almost all the models that perform well on these data sets have naive Bayes relations forced into their structure. Differently from the continuous time data sets, DBNCs perform well. Nevertheless, $k = 2$ ACTNB-CLL once more is the most effective model. Indeed, $k = 2$ ACTNB-CLL performs statistically better than all the DBNCs on the DBNTest4 data set, and on all the other data sets it achieves performance values never inferior to those achieved by the best DBNC. It is also important to recall that CTBNCs are computationally more efficient than DBNCs and have less degrees of freedom because of the smaller number of parameters.

4.4.3 *Synthetic tests summary*

In this Chapter a synthetic tests campaign was addressed. Here follows a short summary of the main results:

- CTBNCs are an efficient and effective framework to classify multivariate trajectories;
- CTBNCs structural learning using Conditional Log-likelihood scoring function allows to also catch the weak dependencies;
- Conditional Log-likelihood scoring is clearly more effective than Marginal Log-likelihood scoring when the amount of data is limited;
- CTBNCs strongly outperform DBNCs on continuous time data sets;
- CTBNCs models also work well on discrete time data sets generated from DBNCs;
- $k = 2$ ACTNB-CLL is the best compromise between model complexity and effectiveness;
- $k = 2$ ACTNB-CLL is comparable and in two cases even better than the best DBNCs on discrete time data sets.

5 CLUSTERING USING CTBNC

Continuous Time Bayesian Network Classifiers (CTBNCs) allow to efficiently address the classification task over continuous time multivariate trajectories. The supervised classification, i.e. the classification of trajectories when the model is learned from labeled data, is not the only classification task that real world problems need to address. Another interesting problem is represented by *clustering*: the problem of learning natural groups from unlabeled data.

In this chapter the clustering problem is addressed using CTBNCs. In Section 5.1 the unsupervised learning process is introduced. The test campaign over synthetic data sets is shown in Section 5.2. Performances are compared with the ones achieved by Dynamic Bayesian Network Classifiers (DBNCs).

Results on real data sets are shown in Chapter 6.

5.1 CTBNC FOR CLUSTERING

5.1.1 *EM parametric learning*

The differences between supervised classification and clustering using CTBNCs is only in the parametric learning. Given the structure, parameters can be learned from sufficient statistics in the same way as for supervised classification. The problem is to estimate the sufficient statistics in which the unobserved class is involved. From Definition 3.2.1, the class is a root node. Therefore, the computation of the sufficient statistics differs from the supervised classification only for the class variable and for the variables which have the class in their parent set.

For parametric learning the well known Expectation Maximization (EM) algorithm can be used (Koller and Friedman, 2009). EM is an iterative algorithm used to optimize the *likelihood* in probabilistic graphical models learned on missing data. The EM algorithm also fits in the case of clustering over complete observable data. In this case only the class is not observed.

EM consists of two steps: *expectation* and *maximization* which are repeated iteratively until reaching a stopping criteria. The expectation step is the step where expected sufficient statistics are calculated. The maximization step is the step where the parameters are calculated from the expected sufficient statistics by maximizing the likelihood.

5.1.1.1 *Expectation*

The main part of the learning process on missing data relies on the expectation step of the EM algorithm. In the case of clustering on observable data the expectation step needs to calculate the sufficient statistics as in a classical learning process, with the exception for the nodes that have the class in the parent set and the class itself.

There are two types of sufficient statistics: the *occurrence counts* (i.e. M) and the *time counts* (i.e. T)[1]. Because only the class node (a static node) is unobserved, all the time spent by the variables in their states is known. Both time counts and occurrence counts of attributes with the class in their parent set

1 Time count sufficient statistics refers to the time spent in a particular state by a variable given the state of its parents.

must be weighted for the class probability. The class prior distribution can be estimated from the class expected occurrences.

Let's suppose that the EM iterative algorithm starts with a random instantiation of the model parameters[2]. Using the current model parameters we can estimate the probability distribution of the class for each trajectory. Let's take the i^{th} trajectory of the data set $\mathcal{D}$ as example. $P(Y \mid \mathbf{z}^{i,1}, \dots, \mathbf{z}^{i,J})$ is the probability distribution of $i \in \{1, \dots, \mid \mathcal{D} \mid\}$, where $\mathbf{z}^{i,j} = (x_1^{i,j}, \dots, x_N^{i,j})$ is the state of each variable of trajectory i at the j^{th} time interval (see notation in Section 3.2.1).

The trajectory contributions to the sufficient statistics are as follows. Contribution to the class count sufficient statistics for class $Y = y$ is:

$$\bar{M}[y] = \bar{M}[y] + P(y \mid \mathbf{z}^{i,1}, \dots, \mathbf{z}^{i,J}).$$

Contribution to the occurrence count sufficient statistics of attribute X_n such that $Y \in Pa(X_n)$ and $Y = y$ is:

$$\begin{aligned}\bar{M}[x_n, x_n' \mid pa(X_n)] = \bar{M}[x_n, x_n' \mid pa(X_n)] \\ + M^i[x_n, x_n' \mid pa(X_n)/y] \cdot P(y \mid \mathbf{z}^{i,1}, \dots, \mathbf{z}^{i,J}),\end{aligned}$$

where $M^i[x_n, x_n' \mid pa(X_n)/y]$ represents the number of times X_n transitions from state x_n to state x_n' in the i^{th} trajectory when X_n's parents without the class node (i.e. $Pa(X_n)/Y$) are set to $pa(X_n)/y$. Contribution to the time count sufficient statistics of attribute X_n such that $Y \in Pa(X_n)$ and $Y = y$ is:

$$\begin{aligned}\bar{T}[x_n \mid pa(X_n)] =& \bar{T}[x_n \mid pa(X_n)] \\ &+ T^i[x_n \mid pa(X_n)/y] \cdot P(y \mid \mathbf{z}^{i,1}, \dots, \mathbf{z}^{i,J}),\end{aligned}$$

where $T^i[x_n \mid pa(X_n)/y]$ is the amount of time along the i^{th} trajectory in which attribute X_n is in state x_n when its parents without the class node (i.e. $Pa(X_n)/Y$) are set to $pa(X_n)/y$.

2 The algorithm could also start with a random instantiation of the class value for each trajectory. Since it is an iterative algorithm that stops in local optimum, different starting points can achieve different results.

Expected sufficient statistics (i.e. $\bar{M}$ and $\bar{T}$) can be calculated by summing the contributions of the sufficient statistics, occurrences (i.e. M^i) and times (i.e. T^i) for each trajectory of the training set (i.e. $i \in \{1, \dots, |\ \mathcal{D}\ |\}$).

5.1.1.2 *Maximization*

Once calculated the expected sufficient statistics, the maximization step is the same as for complete observable trajectories. According to the Maximum Likelihood Estimation (MLE) the parameters are calculated as follows:

- $q_x^{pa(X)} = \frac{\bar{M}[x|pa(X)]}{\bar{T}[x|pa(X)]};$
- $\theta_{xx'}^{pa(X)} = \frac{\bar{M}[x,x'|pa(X)]}{\bar{M}[x|pa(X)]};$
- $\theta_y = \frac{\bar{M}[y]}{\sum_{y'} \bar{M}[y'|pa(X)]};$

where $\bar{M}[x \mid pa(X)] = \sum_{x'} \bar{M}[x, x' \mid pa(X)]$.

Of course also in the case of clustering the *imaginary counts* of the hyperparameters $\alpha_x^{pa(X)}$, $\alpha_{xx'}^{pa(X)}$ and, $\tau_x^{pa(X)}$ can be used (see Section A.1.2.2). Using Bayesian estimation the parameters are calculated as follows:

- $q_x^{pa(X)} = \frac{\alpha_x^{pa(X)} + \bar{M}[x|pa(X)]}{\tau_x^{pa(X)} + \bar{T}[x|pa(X)]};$
- $\theta_{xx'}^{pa(X)} = \frac{\alpha_{xx'}^{pa(X)} + \bar{M}[x,x'|pa(X)]}{\alpha_x^{pa(X)} + \bar{M}[x|pa(X)]};$
- $\theta_y = \frac{\alpha_y + \bar{M}[y]}{\sum_{y'} \alpha_{y'} + \bar{M}[y']}.$

5.1.1.3 *Stopping criteria*

EM algorithm is an optimization algorithm that maximizes the likelihood with iterative steps. It stops when it converges to a local optimum. In practice, EM goes closer to the local optimum iteration after iteration, but never reaches it. For this reason it is necessary to define a stopping criterion (Koller and Friedman, 2009).

The idea is to stop the algorithm when it reaches a point close enough to the local optimum. To do this consecutive iterations are compared, and when the likelihood values or the parameter values are very similar the algorithm is terminated. The two comparison methods bring different results and performances, but it is not possible to define which is better. It differs from data set to data set.

5.1.2 *Hard-assignment EM*

In Section 5.1.1 the soft-assignment EM algorithm was shown. It is called soft-assignment EM because the class probability distribution is used to bias the sufficient statistics. With soft-assignment EM each trajectory contributes to the sufficient statistics for each class.

Hard-assignment EM labels each trajectory with the most probable class. Then, the new labeled trajectories are used to calculate the sufficient statistics.

The trajectory contributions to the sufficient statistics in the case of hard-assignment EM are as follows. Contribution to the class count sufficient statistics for class $Y = y$ is:

$$\bar{M}[y] = \bar{M}[y] + \begin{cases} 1 & \text{if } y = \arg\max_{y'} P(y' \mid \mathbf{z}^{i,1}, \ldots, \mathbf{z}^{i,J}) \\ 0 & \text{otherwise} \end{cases} .$$

Contribution to the occurrence count sufficient statistics of attribute X_n such that $Y \in Pa(X_n)$ and $Y = y$ is:

$$\begin{aligned} \bar{M}[x_n, x_n' \mid pa(X_n)] &= \bar{M}[x_n, x_n' \mid pa(X_n)] \\ &+ \begin{cases} M^i[x_n, x_n' \mid pa(X_n)/y] & \text{if } y = \arg\max_{y'} P(y' \mid \mathbf{z}^{i,1}, \ldots, \mathbf{z}^{i,J}) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

where $M^i[x_n, x'_n \mid pa(X_n)/y]$ represents the number of times X_n transitions from state x_n to state x'_n in the i^{th} trajectory when X_n's parents without the class node (i.e. $Pa(X_n)/Y$) are set to $pa(X_n)/y$. Contribution to the time count sufficient statistics of attribute X_n such that $Y \in Pa(X_n)$ and $Y = y$ is:

$$\begin{aligned}\bar{T}[x_n \mid pa(X_n)] &= \bar{T}[x_n \mid pa(X_n)] \\ &+ \begin{cases} T^i[x_n \mid pa(X_n)/y] & \text{if } y = \arg\max_{y'} P(y' \mid \mathbf{z}^{i,1}, \dots, \mathbf{z}^{i,J}) \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

where $T^i[x_n \mid pa(X_n)/y]$ is the amount of time along the i^{th} trajectory in which attribute X_n is in state x_n when when its parents without the class node (i.e. $Pa(X_n)/Y$) are set to $pa(X_n)/y$.

Once calculated the sufficient statistics (i.e. expectation step) the hard-assignment EM relies on the same maximization step as the soft-assignment EM.

5.1.3 *Structural learning*

Learning the structure of CTBNCs over unlabeled data requires the same algorithms as in the case of supervised learning. Starting from an initial structure an optimization algorithm can be used to locally maximize the scoring function, as done in the case of supervised learning (Chapter 4). In this work the Naive Bayes structure is used as the starting point.

Scoring functions, i.e. Marginal Log-likelihood and Conditional Log-likelihood scores, are based on the training set sufficient statistics, for this reason the sufficient statistics of the last iteration of the EM algorithm are used.

Learning the structure in case of clustering is a computationally demanding procedure that requires to iterate the EM parameter learning and the structural learning algorithm since the reaching of a termination point. The termination point is found when changing the structure does not improve the score with respect to the current model parameters and sufficient statistics. The structural learning process can be addressed independently for each variable (Algorithm 2).

Algorithm 2 CTBNCs unsupervised structural learning algorithm.

```
Require: a data set $\mathcal{D}$ of unlabeled trajectories without missing
    data.
Ensure: a learned CTBNC model $\mathcal{C}$.
 1: $\mathcal{C}_{max} := \text{NaiveBayesModel}(\mathcal{D})$
 2: for $X \in \mathcal{C}_{max}$ do
 3:     $[\mathcal{C}_{max}, \bar{M}, \bar{T}] := \text{EM}(\mathcal{C}_{max}, X, \mathcal{D})$
 4:     $s := \text{Score}(\bar{M}, \bar{T}, \mathcal{C}_{max}, X, \mathcal{D})$
 5:     repeat
 6:         $\mathcal{C} := \mathcal{C}_{max}$
 7:         $\mathcal{M} := \text{Neighbors}(\mathcal{C}, X)$
 8:         for $\mathcal{C}_n \in \mathcal{M}$ do
 9:             $[\mathcal{C}_n, \bar{M}_n, \bar{T}_n] := \text{EM}(\mathcal{C}_n, X, \mathcal{D})$
10:             $s_n := \text{Score}(\bar{M}_n, \bar{T}_n, \mathcal{C}_n, X, \mathcal{D})$
11:             if $s < s_n$ then
12:                 $s := s_n$
13:                 $\mathcal{C}_{max} := \mathcal{C}_n$
14:             end if
15:         end for
16:     until $\mathcal{C} \neq \mathcal{C}_{max}$
17: end for
18: return $\mathcal{C}$
```

5.2 SYNTHETIC TESTS

5.2.1 *Test campaign*

In this section the synthetic test campaign is presented and described, while the real world data set experiments are addressed in Chapter 6.

The clustering synthetic campaign is parallel to the supervised classification synthetic campaign discussed in Section 4.4. Numerical experiments are devoted to comparing the performance of the previously tested CTBNCs, learned by maximizing both the Marginal Log-likelihood and the Conditional Log-likelihood scoring functions, with Dynamic Bayesian Networks (DBNs). CTBNCs and DBNs are both learned using the EM algorithm.

DBNs are implemented using the MATLAB Bayesian Nets toolbox (Murphy et al., 2001). This library does not allow to learn the structure over partially observable trajectories. For this reason DBN-NB1 and DBN-NB2 are the DBNCs tested (see Figure 6).

The synthetic clustering tests are realized over CTBNCs data sets and DBNs data sets as well. CTBNCs data sets were described in Table 1. These data sets were generated by using the models depicted in Figure 7 (Section 4.4.1.2). DBNs data sets were described in Section 4.4.1.3.

In many clustering problems the number of clusters is unknown. Discovering the optimal number of clusters is a difficult problem. Because the cluster number discovery is not in the purpose of this dissertation, it is assumed that the correct number of clusters is known.

Section 5.2.3.1 shows the performances achieved over the continuous time data sets, while Section 5.2.3.2 shows the performances achieved over the discrete time data sets.

5.2.2 *Performance measures*

Clustering evaluation can be performed by using *external measures*, *internal measures* or, *relative measures* (Halkidi et al., 2001; Gan et al., 2007; Xu and Wunsch, 2008).

Internal and external measures are based on statistical tests. Relative measures are not based on statistical tests, and for this reason they are more efficient (Halkidi et al., 2001). The idea of relative approaches is to compare the clustering performance on the basis of a predefined criterion. Relative measures can be used, for example, to test the number of clusters (i.e. Calinski-Harabasz index (Caliński and Harabasz, 1974) and, Davies-Bouldin index (Davies and Bouldin, 1979)).

Internal measures evaluate the similarity of the clusters using distance functions (i.e. the CoPhenetic Correlation Coefficient (Sokal and Rohlf, 1962)). The advantage of internal approaches is the possibility to obtain performance measures also when the label is unknown in the evaluation data sets. On the contrary, if it is difficult to calculate a good quality distance, internal measures lose their effectiveness. This is the case of continuous time multivariate trajectories (see Section 2.2.1).

External measures evaluate the clusters using the label information. In this case the label is ignored during the clustering process, but it is mandatory to calculate the performances. The advantages of external approaches is that they do not require a distance function. On the contrary, the class label must be known, and in many real problems this is difficult, expensive or even impossible.

The labels are available in the synthetic and real world data sets used in this dissertation. This allows to use external measures, which are the most effective way to calculate performances considering the difficulties of finding good quality distances on multivariate continuous time trajectories.

The external measures used are *Rand index (R)* (Rand, 1971), *Jaccard's coefficient (J)* (Halkidi et al., 2001) and, *Fowlkes–Mallows index (FM)* (Fowlkes and Mallows, 1983). These measures can be calculated as follows. Let's consider a clustering partitioning $\mathbf{C} = \{C_1, \dots, C_h\}$ and the real partitioning $\mathbf{P} = \{P_1, \dots, P_s\}$. Given a couple of clustered instances in the data set it is possible to distinguish the following four cases (Halkidi et al., 2002):

- SS: both points belong to the same cluster and the same partition;
- SD: both points belong to the same cluster, but a different partitions;
- DS: the points belong to a different clusters, but the same partition;
- DD: the points belong to different clusters and partitions.

Considering a data set of $\mid \mathcal{D} \mid$ instances and all the possible M couples of instances in the data set (i.e. $M = \frac{|\mathcal{D}|(|\mathcal{D}|-1)}{2}$), let's call #SS, #SD, #DS and, #DD the number of couples in each of the four possible configurations (i.e.$M = \#SS + \#SD + \#DS + \#DD$). The number of occurrences in each of the four cases represents useful information for evaluating the clustering quality. Starting from these values the following external measures are calculated (Halkidi et al., 2002):

- Rand index: $R = \frac{\#SS + \#DD}{M}$
- Jaccard's coefficient: $J = \frac{\#SS}{\#SS + \#SD + \#DS}$

- Fowlkes–Mallows index: $FM = \sqrt{\frac{\#SS}{\#SS+\#SD}\frac{\#SS}{\#SS+\#DS}}$

These three measures take value in $[0,1]$. The more similar the clusters are to the original partitioning, the higher values the three indexes have.

In addition to these measures, also *precision*, *recall* and e *f-measure* are calculated between each cluster-partition couple.

5.2.3 *Results*

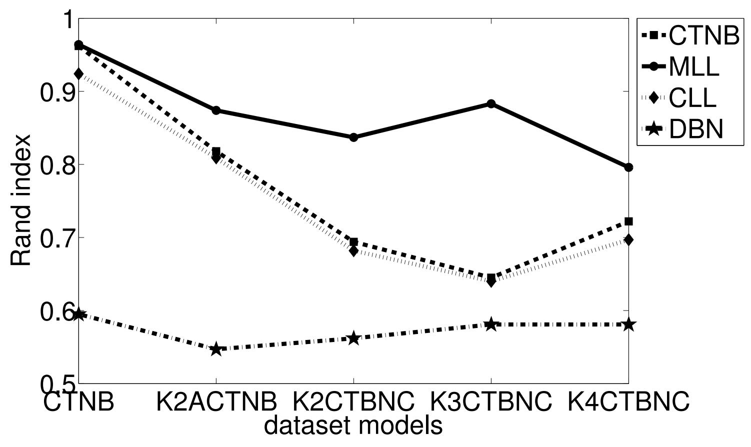
5.2.3.1 *CTBN synthetic data sets*

Figure 14 depicts the best average performances in terms of Rand index, Jaccard's coefficient, and Fowlkes–Mallows index. For each measure and for each class of models (i.e. MLL, CLL, CTNB, and DBN) the best average performances are reported.

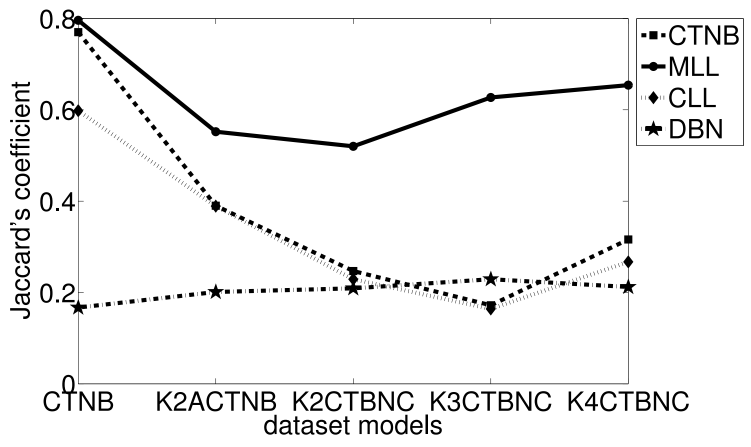
The picture clearly shows the effectiveness of Continuous Time Bayesian Network Classifiers learned by maximizing the Marginal Log-likelihood scoring function. In the clustering case, the Conditional Log-likelihood score does not perform well. Indeed, CTBNCs learned with Conditional Log-likelihood perform comparably or worse than Continuous Time Naive Bayes (CTNB).

DBNs are less effective than the continuous time model. DBNs results are clearly worse than the one achieved by CTBNC learned by maximizing the Marginal Log-likelihood score.

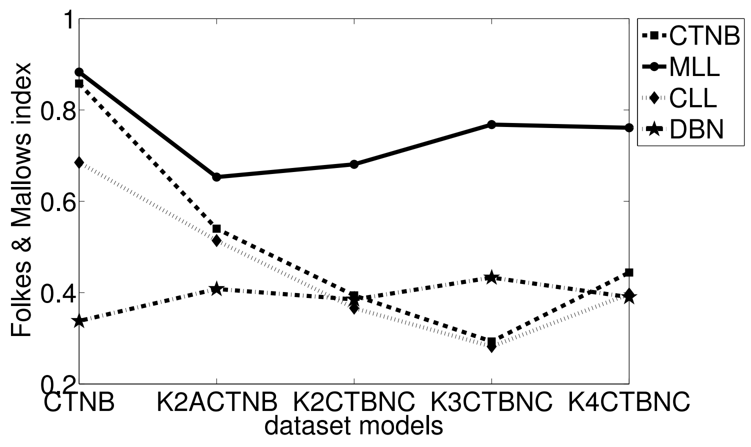
Figure 15 depicts the average learning time for the CTBNCs. As for supervised learning over continuous time trajectories, it was necessary to discretize the trajectories to apply DBNs. For each trajectory a sampling rate has been chosen that allows to generate 50 data rows. The discretization process makes the learning time of CTBNCs and DBNs not directly comparable. Nev-



(a)



(b)



(c)

Figure 14.: Best average performances, i.e. Rand index (a), Jaccard's coefficient (b), and Fowlkes–Mallows index (c), between MLL, CLL, CTNB, and DBN models. The x-axis is associated with the CTBNCs used to generate the data sets.

ertheless, the results clearly showed that CTBNCs are more efficient than DBNs.

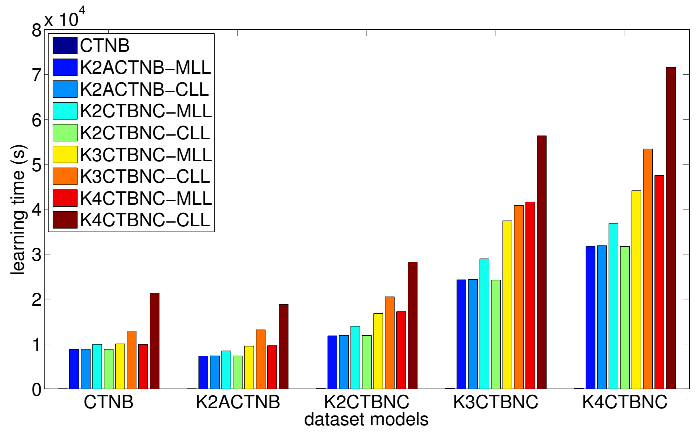


Figure 15.: Average learning time for each CTBNC model. The x-axis is associated with the models used to generate the data sets.

Among CTBNCs, CTNB is of course the most efficient approach because it does not require the structural learning process. The main differences in terms of learning time between CTBNCs emerge when the maximum number of parents is increased. CTBNCs learned by maximizing Marginal Log-likelihood seem to be learned faster than the corresponding models learned by maximizing the Conditional Log-likelihood. This is in contradiction with what observed in the case of supervised learning where the time difference between the two learning functions was not clear and seems to be in favor of the Conditional Log-likelihood scoring function.

As in the case of supervised learning, $k = 2$ Augmented Continuous Time Naive Bayes (ACTNB) offers a good compromise between model complexity and effectiveness. The complete results in Section B.2.2 show the capability of $k = 2$ ACTNB to provide good performance, even on data sets generated by more complex models.

5.2.3.2 *DBN synthetic data sets*

The results over CTBNCs synthetic data sets are confirmed by the results on the DBNs synthetic data sets. Figure 16 depicts the best performances in term of Rand index, Jaccard's coefficient, and Fowlkes–Mallows index. For each measure and for each class of models (i.e. MLL, CLL, CTNB, and DBN) the best performances are reported.

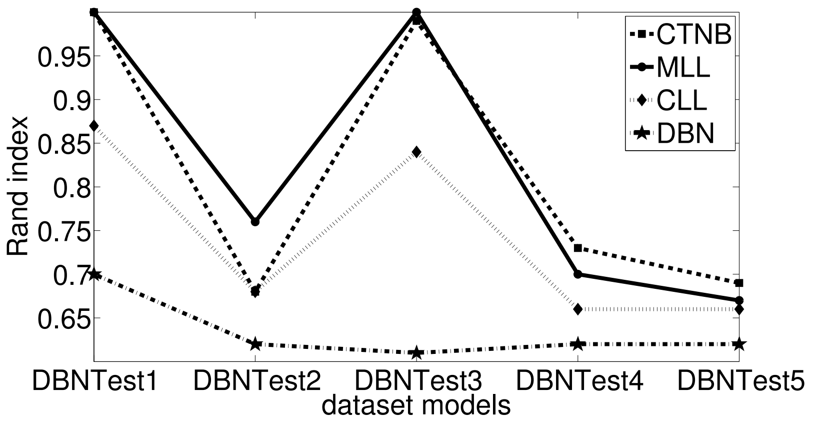
DBNCs synthetic data sets are composed of discrete time trajectories. Nevertheless, discrete time models (i.e. DBNs) do not perform well as continuous time models. Even in this case, CTBNCs learned with the Marginal Log-likelihood score outperform all the other models. CTBNCs learned by maximizing Conditional Log-likelihood perform better than DBNCs, but their performances are inferior to the ones achieved by CTNB.

In the case of discrete time representation $k = 2$ ACTNB shows its capability to provide good performances even more. In the case of clustering over the DBNCs data sets, $k = 2$ ACTNB-MLL is the model that provides the best performances.

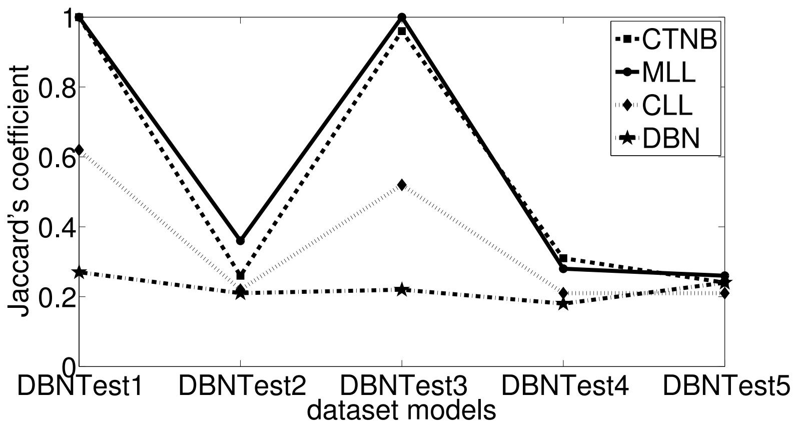
5.2.3.3 *Synthetic test summary*

In this Section a synthetic test campaign for the problem of multivariate trajectories clustering was addressed. Here is a short summary of the main results:

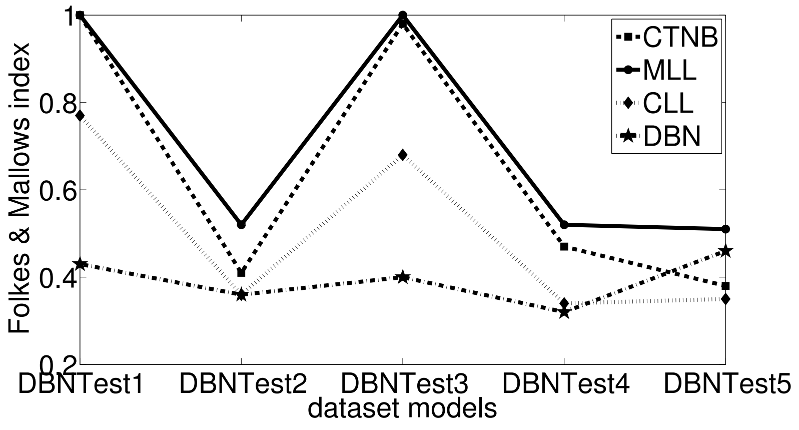
- CTBNCs are an efficient and effective framework to cluster multivariate trajectories;
- Marginal Log-likelihood score performs better than Conditional Log-likelihood score in the case of clustering;
- CTBNCs learned by maximizing the Marginal Log-likelihood scoring function greatly outperform DBNCs, both on continuous and discrete time data sets;
- $k = 2$ ACTNB-Marginal Log-likelihood (MLL) is the best compromise between model complexity and effectiveness.



(a)



(b)



(c)

Figure 16.: Best performances, i.e. Rand index (a), Jaccard's coefficient (b), and Fowlkes–Mallows index (c), between MLL, CLL, CTNB, and DBN models. The x-axis is associated with the DBNCs used to generate the data sets.

6 REAL WORLD PROBLEM APPLICATIONS

This chapter presents two real world problems addressed efficiently and effectively with Continuous Time Bayesian Network Classifiers (CTBNCs).

Post-Stroke rehabilitation (Section 6.1), and classification of traffic profiles (Section 6.2) are described. Both classification and clustering is addressed over the real world data sets.

6.1 POST-STROKE REHABILITATION

6.1.1 *Case study*

Rehabilitation after hospital treatment is an important challenge for health systems all over the world. Tormene et al. (2009) pointed out how this is also true for post-stroke rehabilitation. In terms of costs and effectiveness it is very important to start the physical therapy as soon as possible (Paolucci et al., 2000; Kwakkel et al., 2004; Forster and Young, 2002). This is often hard to realize because a supervisor should be present during the rehabilitation exercises. Another problem is the difficulty for many patients to reach a rehabilitation structure after a stroke, mainly because many of them live alone.

Tormene et al. (2009) proposed an automatic movement recognition system to face these difficulties. The idea is to provide the patient with a system that can recognize his/her movements and inform him/her about the correctness of the rehabilitation exercise. The supervision of the rehabilitation therapies can be guaranteed remotely. The system must be able to recognize the correctness of the movements in real time before the exercises are completed and the full trajectories are recorded.

Tormene et al. (2009) focused on the upper limb post-stroke rehabilitation. Nevertheless, this method can be applied for all rehabilitation therapies. The authors provide a data set of 7 rehabilitation exercises (Table 4). For each exercise 120 multivariate trajectories are provided, thanks to 29 sensors analyzed with a frequency of 30 Hz (Tormene and Giorgino, 2008). Each movement is addressed separately as a classification problem with 2, 4, and 6 classes (see Table 5).

Movement id	Description
1	Abduction-adduction of the upper limb on a frontal plane
2	Abduction-adduction of the upper limb on a sagittal plane
3	External rotation of the forearm
4	Flexion-extension of the elbow
5	Pronation-supination of the forearm
6	Functional activity: eating
7	Functional activity: combing

Table 4.: Description of the 7 rehabilitation exercises (Tormene et al., 2009).

Tormene et al. (2009) addressed the post-stroke rehabilitation problem with 1-Nearest Neighbor Classifier (NNC) using both Dynamic Time Warping (DTW) (Keogh and Ratanamahatana, 2005) and Open-End Dynamic Time Warping (OE-DTW) distances (see Section 2.2.1). OE-DTW is a variation of DTW that allows to effectively deal with incomplete trajectories. The idea is to match the input trajectory with the reference trajectories, considering all the possible points in which the references can

Class index	Correctness	Speed	Description
1	Correct	Slow	Reference
2	Correct	Average	Reference
3	Correct	Fast	Reference
4	Incorrect	Average	Movement too small
5	Incorrect	Average	Typical compensatory action (first)
6	Incorrect	Average	Typical compensatory action (second)

Table 5.: Description of the 6 movement classes (Tormene et al., 2009). The 2 class problem identifies only the correctness of the movement while the 4 class problem identifies correct movements and the different types of incorrect movements.

be truncated. The OE-DTW is the minimum of all the distances calculated for all the possible truncations. The results showed the ability of both distances to deal with the rehabilitation problem. Tests were also made with half length trajectories. In this case DTW was not able to perform efficiently, while OE-DTW was not strongly affected by length reduction.

Dynamic Time Warping distances are an efficient way of calculating the distance between trajectories in a continuous state space. When the state space is discrete and without any states ordering, the DTW distances lose their effectiveness because they are based on the trivial metric (Hazewinkel, 1993). To understand how well CTBNCs perform after discretization compared to DTW approaches on real world problems with continuous state space, tests are made on discretized post-stroke rehabilitation data sets.

6.1.2 *Supervised classification experiments*

Tormene et al. (2009) used leave-one-out cross-validation to estimate the performances. Because of the computational effort needed to test all the models, in particular Dynamic Bayesian Networks (DBNs), the experiments here were made using 10 fold cross-validation.

Table 6 shows accuracy, precision and recall values, averaged over movements, achieved by CTBNCs and Dynamic Bayesian Network Classifiers (DBNCs) for 2 and 6 class classification problems (Codecasa and Stella, 2013). CTBNCs performances are comparable with DTW performances obtained by Tormene et al. (2009)[1], even after the great simplification due to the original state space discretization.

# classes	Measure	CTNB	$k=2$ ACTNB (MLL)	$k=2$ ACTNB (CLL)	$k=2$ CTBNC (MLL)	$k=2$ CTBNC (CLL)	$k=3$ CTBNC (MLL)	$k=3$ CTBNC (CLL)	$k=4$ CTBNC (MLL)	$k=4$ CTBNC (CLL)	DBN-NB1	DBN-NB2
2 classes	Accuracy	0.98	0.97	**0.99**	0.87	0.85	0.87	0.92	0.87	0.95	0.97	0.97
	Precision	0.97	0.97	**0.99**	0.86	0.85	0.86	0.93	0.86	0.95	0.97	0.97
	Recall	0.98	0.98	**0.99**	0.88	0.84	0.88	0.92	0.88	0.96	0.97	0.97
6 classes	Accuracy	**0.91**	**0.91**	**0.89**	0.81	**0.88**	0.81	**0.88**	0.81	**0.88**	0.87	0.87
	Precision	**0.92**	**0.91**	**0.89**	0.84	**0.89**	0.84	**0.89**	0.84	**0.90**	0.88	0.87
	Recall	**0.90**	**0.90**	**0.88**	0.81	**0.88**	0.82	**0.88**	0.82	**0.89**	0.87	0.87

Table 6.: Average accuracy, precision and, recall for the post-stroke rehabilitation data set (10 fold CV). Bold characters indicate the best models with 90% confidence.

Accuracy values achieved by almost all CTBNCs with the Conditional Log-likelihood (CLL) scoring function are better than accuracy values achieved by the corresponding models using the Marginal Log-likelihood (MLL) scoring function.

For the 6 class classification problem, in the case where no information about variable dependency is available (i.e. the links between the class and the other variables), Conditional Log-

1 DTW and OE-DTW obtained 0.99 accuracy values over the 2 class data set, while DTW obtained 0.88 and OE-DTW obtained 0.87 accuracy values over the 6 class data set (Tormene et al., 2009).

likelihood always outperforms Marginal Log-likelihood. Performance achieved by CTBNCs learned with Conditional Log-likelihood are robust with respect to the choice of the imaginary count values, while the same does not apply to Marginal Log-likelihood.

$k = 2$ ACTNB, learned with Conditional Log-likelihood scoring, implements the optimal trade-off between the continuous time models in terms of time and accuracy. Indeed, for both 2 and 6 class classification problems, the $k = 2$ ACTNB model when learned with Conditional Log-likelihood, achieves the highest accuracy value and is the fastest to learn because of the small number of parents.

All the approaches (i.e. CTBNCs, DTW, and DBNs) seem to provide good results when the best model is selected. It is likely that learning different models for each movement brings great simplification that allows to reach good performances with all the approaches. Nevertheless, DBNs are much more computationally demanding than CTBNCs.

Further analyses were made with CTBNCs when the percentage of the data set it reduced. As in Section 4.4, the reduction of the data set dimension is made both in terms of trajectory length and number. This differs from what Tormene et al. (2009) made. In their work only the length of the test set trajectories was cut. Tormene et al. (2009) were interested in understanding the performances of DTW approaches during the movement execution, while in this thesis the interest is focused on understanding the behavior of CTBNCs' learning process when the amount of data decreases.

Figure 17 shows how the Conditional Log-likelihood scoring function outperforms the Marginal Log-likelihood scoring function. The effectiveness of Conditional Log-likelihood grows when the data amount is reduced. In this case Conditional Log-likelihood is able to also perceive weak dependencies between the variables and the class node. When the data amount is greatly reduced, the Conditional Log-likelihood scoring function tends to generate too complex structures with respect to the amount of available data. In these cases, simple models such as Continuous Time Naive Bayes (CTNB) are the most effective ones.

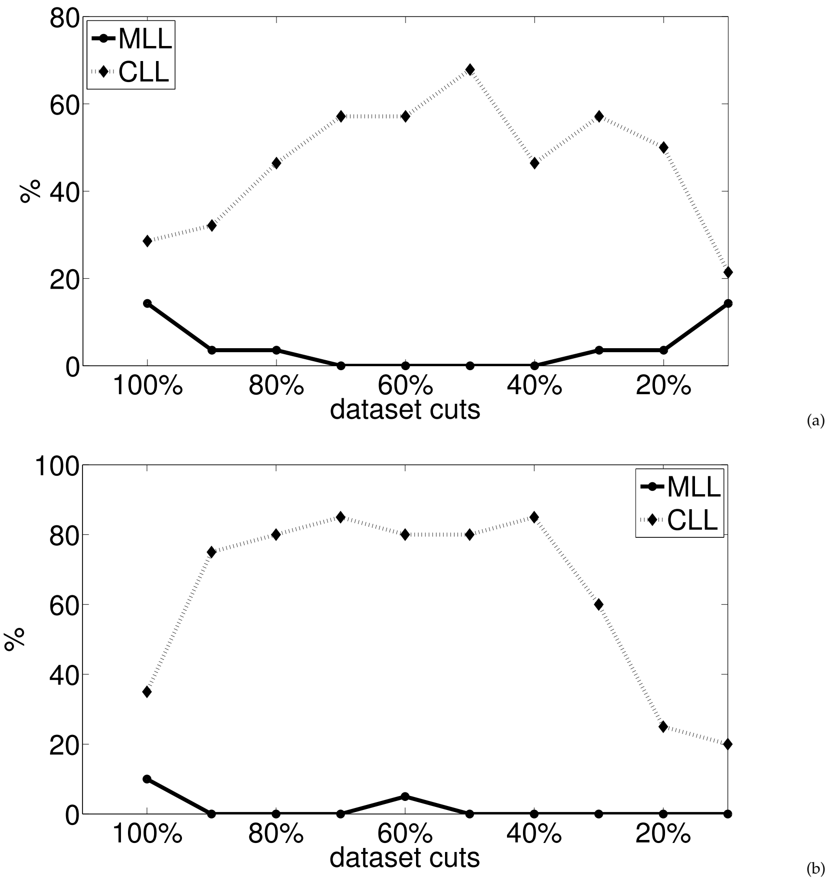


Figure 17.: Percentage of numerical experiments where MLL (CLL) is better than CLL (MLL) with 90% confidence. Analysis is performed on post-stroke reduced data sets. The x-axes represent data set percentage reduction. Figure **(a)** refers to 2 class problem, while Figure **(b)** refers to 6 class problem.

Results on the post-stroke rehabilitation data set confirmed the consideration evidenced in the synthetic experiments (Section 4.4). Additionally, CTBNC is an effective and efficient framework to address post-stroke rehabilitation gesture recognition problems even after the simplification due to state space discretization. Performances are comparable to the one obtained by the DTW approaches. Nevertheless, CTBNCs offer an efficient alternative in order to avoid the computational effort of comparing many trajectories during the 1-NNC classification procedure.

6.1.3 *Clustering experiments*

In order to test the clustering algorithms on real world problems the post-stroke rehabilitation data sets are used.

Table 7 summarizes the average performances (see Section 5.2.2) of the different tested models. The results do not evidence a more effective approach. Different measures seem to give advantage to different models, since they evaluate different aspects of the clustering. The Rand index seems to give advantage to continuous time approaches, CTNB in particular. Jaccard's coefficient does not clearly shows a more efficient ap-

Test	Measure	CTNB	$k=2$ ACTNB (MLL)	$k=2$ ACTNB (CLL)	$k=2$ CTBNC (MLL)	$k=2$ CTBNC (CLL)	$k=3$ CTBNC (MLL)	$k=3$ CTBNC (CLL)	$k=4$ CTBNC (MLL)	$k=4$ CTBNC (CLL)	DBN-NB1	DBN-NB2
2 classes	R	.596	.575	.543	.563	.560	.526	.509	.567	.516	.522	.527
	J	.451	.406	.374	.399	.401	.363	.352	.405	.346	.448	.453
	FM	.595	.575	.543	.566	.569	.533	.521	.574	.513	.636	.640
6 classes	R	.750	.733	.723	.738	.724	.754	.724	.739	.730	.426	.526
	J	.170	.164	.161	.154	.137	.170	.127	.158	.113	.167	.167
	FM	.291	.285	.282	.267	.242	.287	.226	.274	.204	.357	.334

Table 7.: Average clustering performances for 2 and 6 class post-stroke rehabilitation problems. R stands for the Rand index, J stands for Jaccard's coefficient and FM for the Fowlkes–Mallows index.

proach, while the Fowlkes–Mallows index seems to give advantage to discrete time approaches. The reason for this can probably be found in the post-stroke rehabilitation problem. Indeed, the rehabilitation movements are quite similar and do not define well separated clusters of trajectories. A better clustering real world example is provided in the next section.

6.2 TRAFFIC PROFILE CLASSIFICATION

6.2.1 *Case study*

Today traffic congestion is one of the main concerns to cope with in all the big cities of the world. Wasting time driving a car is source of stress and has big impacts on economic activities. Arnott and Small (1994) estimate a cost of 48 billion dollars due to traffic congestion in 39 metropolitan areas in the United States with a population of one million or more. This does not include the cost due to unpredictability of traffic delays, the cost of extra fuel and air pollution.

The environmental issue is not less important. Traffic is one of the main causes of environmental and acoustic pollution which leads to high social costs. The relationships between pollution and allergies, asthma, tumors and, immune system diseases are clear (Gualtieri et al., 2005; Mantecca et al., 2007). Analyzing these data it easy to see that traffic can be considered an important social problem that must be faced to reduce its daily impact.

Urban Traffic Control (UTC) is one of the major challenges for traffic engineers. It promises to be one of the most effective ways to cope with traffic congestion in metropolitan areas. This is particularly true considering the evolution of new technologies, such as sensors to monitor the streets (i.e. loops, cameras, ...) and the increase of computational capabilities of modern computers. Nevertheless, because traffic is a chaotic system, and traffic profile changes continuously during the day, the traffic light control is still an open problem.

Traffic light control and coordination is a well studied problem in the specialized literature (Papageorgiou et al., 2003). The following commercial solutions are very important: TRANSYT (Robertson, 1969), an off-line optimization model; SCOOT (Robertson and Bretherton, 1991), SCATS (Sims and Dobinson, 1980) and UTOPIA[2] which offer traffic-responsive strategies.

The improvement of artificial intelligent models has brought new intelligent transportation system approaches (Stella et al., 2006). The literature shows many solutions to the traffic light control problem which use artificial intelligence models. Yu and Recker (2006) propose a discrete-time, stationary, Markov decision process in order to solve the problem of traffic control. Haijema and van der Wal (2008) proposed a Markov decision process decomposition approach to control isolated intersections. A reinforcement learning approach is used by Thorpe and Anderson (1996) and Wiering (2000) to define a UTC system with learning capability. Also approaches using expert systems (Felici et al., 2006; Hirankitti and Krohkaew, 2007), fuzzy logic (Favilla et al., 1993; Angulo et al., 2011), neural networks (Spall and Chin, 1997), auto-organization systems (Gershenson, 2004; Lämmer and Helbing, 2008) and evolutionary algorithms (Park and Messer, 1998) were proposed.

Only few of these approaches can be applied in the real world. This is because of the need for strong assumptions and approximations to address the computational effort of managing real networks and complex systems. Classification of the traffic profile (i.e. the state of the traffic) is a way to simplify

2 http://www.swarco.net/

the traffic light optimization. The idea is classify or cluster the traffic condition off-line in order to find the best plan for each particular condition. Then, in real time choose the plan which as the most similar precalculated traffic profile and eventually propose some modifications to adapt the traffic light plans to the actual traffic condition. This approach is not new in the literature: Angulo et al. (2011) use fuzzy clustering of origin-destination matrix as an intermediate step to their UTC approach.

Urban Traffic Control is out of the scope of this thesis, but classification and clustering of traffic light profiles using real time square waves generated by loop sensors positioned under the concrete of the roads (see Figure 18) represents an innovative, efficient and effective way to deal with this problem.

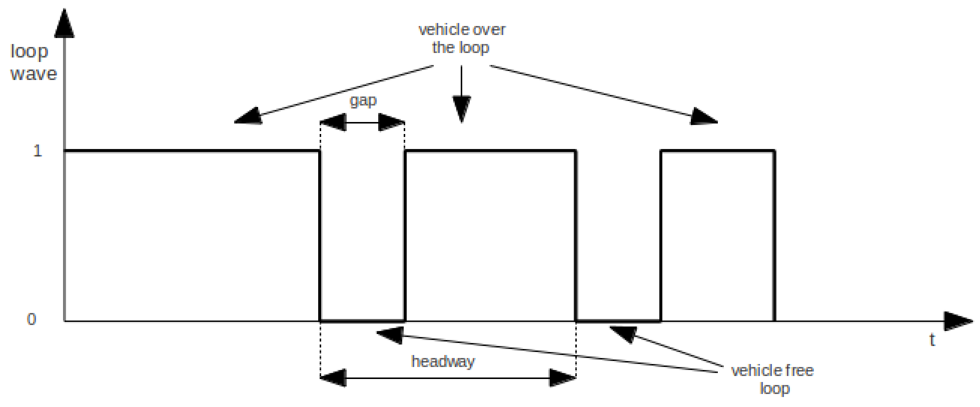


Figure 18.: Example of loop square wave. One indicates a vehicle passing over the loop, zero indicates a loop free of vehicles.

Four data sets were generated using the TSIS-CORSIM simulator (Daigle et al., 1997; Owen et al., 2000). The data sets are generated using 100 seconds and 300 seconds as length of trajectories. Data are sampled with a frequency of 10Hz. Sensors (i.e. loops) are positioned at the beginning and at the end of each link (i.e. a road that links two crossroads). Two data sets from a toy network example (Figure 19a) and two data sets from a portion of Monza's road network (Figure 19b) are generated.

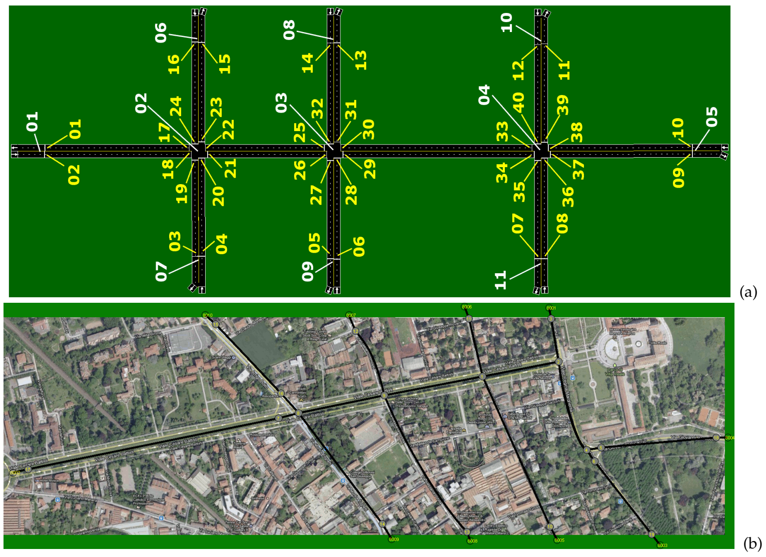


Figure 19.: Toy road network (a) and Monza's road network (b) used to generate the two data sets.

The choice of this portion of Monza's road network is particularly valuable because it is one of the most crucial and most congested areas in Monza. This is confirmed by the partnership of Monza in the CIVITAS ARCHIMEDES European project[3] with the purpose of introducing "innovative, integrated and ambitious strategies for clean, energy-efficient, sustainable urban transport and thereby have a significant impact on policies concerning energy, transport, and environmental sustainability".

Performance of CTBNCs in traffic profile classification (Section 6.2.2) and clustering (Section 6.2.3) are analyzed. Since the real Monza road network is not equipped with 2 sensors for each link, tests are also made using the actual six sensors present on the network. The impact of the sensors in the classification task can suggest if and where to add or to remove sensors.

3 http://www.civitas.eu/archimedes

6.2.2 *Supervised classification experiments*

Table 8 shows the classification performances over the traffic profiling data sets. CTBNC is an effective framework to deal with traffic profile classification.

Road network	Trajectories length	CTNB	k = 2 ACTNB (MLL)	k = 2 ACTNB (CLL)	k = 2 CTBNC (MLL)	k = 2 CTBNC (CLL)	k = 3 CTBNC (MLL)	k = 3 CTBNC (CLL)	k = 4 CTBNC (MLL)	k = 4 CTBNC (CLL)	DBN-NB1	DBN-NB2
Toy network	100 seconds	**0.91**	**0.92**	**0.91**	**0.92**	0.86	**0.93**	0.86	**0.93**	0.86	0.86	0.83
	300 seconds	**0.95**	**0.95**	**0.95**	**0.95**	0.93	**0.95**	0.93	**0.95**	0.93	0.89	0.88
Monza's network	100 seconds	**0.75**	0.73	**0.75**	0.73	0.72	0.73	0.72	0.72	0.72	0.67	0.67
	300 seconds	**0.82**	0.79	**0.82**	0.79	0.80	0.79	0.80	0.78	0.80	0.75	0.75
Monza's network real loops	100 seconds	**0.62**	**0.62**	**0.62**	**0.62**	**0.62**	**0.62**	**0.62**	**0.62**	**0.62**	0.54	0.53
	300 seconds	**0.72**	**0.70**	**0.72**	**0.70**	**0.72**	**0.70**	**0.72**	**0.70**	**0.72**	0.63	0.61

Table 8.: Accuracy for the traffic profiling problem (10 fold CV). The bolt characters indicate the best models with 90% of confidence.

In traffic state of the art the loop square waves are aggregated over time generating three main measures: vehicle counts, vehicle average speed and density. These measures are used to develop intelligent transportation systems. The idea to directly use the square wave generated by loops is new. The results show that with Continuous Time Bayesian Network Classifiers this approach is feasible and effective.

Simple models such as CTNB seem good enough for classification purpose. Probably the reason is the complexity of the traffic dynamics that does not show strong dependency relations between sensors in an urban network. These dependencies would be more evident in freeways, where the flow is more regular, and it would be possible see groups of vehicle moving on the highways.

300 second trajectories allow for better classification, probably due to the smaller influence of traffic light cycles. Regarding

Monza's road network the strong sensor reduction (i.e. from 32 sensors to 6 sensors) has only a limited impact on the classification performances. Accuracy is reduced by 10% when reducing 81% of the sensors. This highlights the fact that 2 sensors for each link generates duplicated information, while the position of the real world sensors is well studied. Nevertheless, some new sensors can offer more information for the traffic profile classification.

In terms of continuous versus discrete time models, CTBNCs perform statistically significantly better than DBNs also in this problem. In the case of the real world road network CTBNCs show an increase of accuracy between 7% and 9%.

6.2.3 *Clustering experiments*

Traffic profile clustering is a solution to simplify the traffic signal optimization problem. Using clustering it is possible to identify, without any supervision, clusters of similar traffic profiles. A different traffic plan can be used to manage each traffic profile in the same cluster. In this section the clustering performances of CTBNCs and DBNs are compared over the traffic profile classification data sets.

Figure 20 depicts the best performances in terms of the Rand index, Jaccard's coefficient, and the Fowlkes–Mallows index (see Section 5.2.2). For each measure and for each class of models (i.e. MLL, CLL, CTNB, and DBN) the best performances are shown[4].

The results, summarized in Table 25, show the effectiveness of continuous time classifiers. CTBNCs strongly outperform DBNs. CTBNCs learned by maximizing Marginal Log-likelihood

4 For computational reasons a random subset of trajectories is used in the Monza road network tests.

are the best approaches to deal with the clustering problem. Nevertheless, CTNB and CTBNCs learned by maximizing Conditional Log-likelihood perform well. Both these approaches are more effective than DBNs.

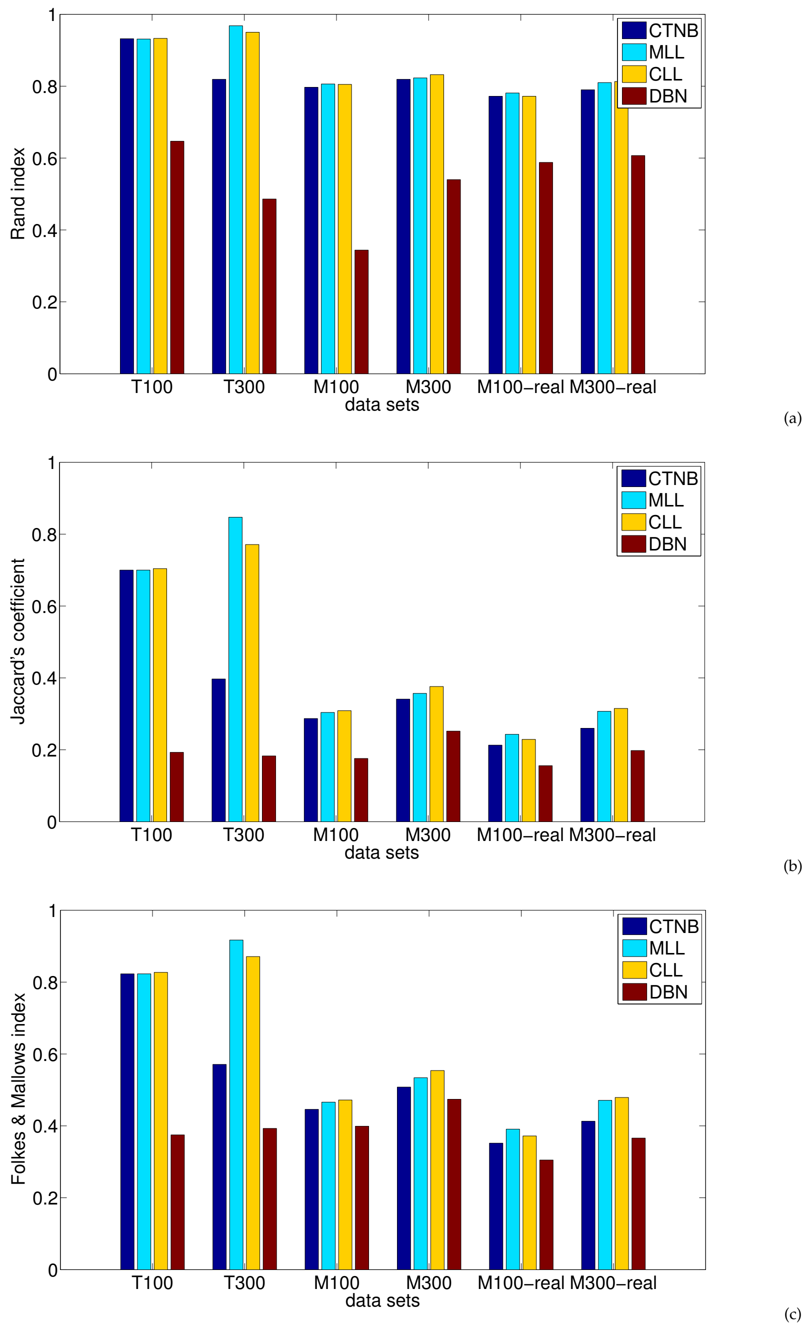


Figure 20.: The Rand index (a), Jaccard's coefficient (b), and the Fowlkes–Mallows index (c), between MLL, CLL, CTNB, and DBN models. The x-axis is associated with the used data sets[4]. "T100" and "T300" indicate the toy network data sets. "M100", "M300", "M100-real", and "M300-real" indicate the Monza road network data sets with all the sensors and with only the real sensors.

7 CONCLUSIONS

7.1 WORK OVERVIEW

Real world problems which generate streaming data have rapidly increased over the last years. This has made data mining of temporal data to be an increasingly important problem to cope with. Recently, many approaches have been proposed in the specialized literature. Continuous Time Bayesian Networks (CTBNs) is one of these approaches introduced to efficiently model temporal data (Nodelman et al., 2002a). CTBNs represent multivariate trajectories using the exponential distribution to model variable evolution over continuous time. This allows to avoid the time discretization which can introduce unnecessary complexity (Nodelman et al., 2002a).

Classification is one of the most important problems in the mining of streaming data. In this dissertation temporal classification of multivariate trajectories was analyzed. Continuous Time Bayesian Network Classifiers (CTBNCs), a specialization of CTBNs, which allows temporal classification when the class is static were studied. Before this work, CTBNCs were introduced and tested only using the naive Bayes approach (Stella and Amer, 2012). In this thesis the structural learning of CTBNCs was addressed. First, the Marginal Log-likelihood scoring, function introduced by Nodelman et al. (2002b) for CTBNs, was used to learn CTBNCs. Then, a Conditional Log-likelihood scoring function was derived and applied to learn CTBNCs improving their classification performances (Chapter 4).

Unsupervised learning, i.e. the clustering problem, was addressed for the first time using CTBNCs. The Expectation Maximization (EM) algorithm with soft and hard assignment was developed and tested (Chapter 5).

A rich test campaign using synthetic and real world data sets was performed to compare the classification and the clustering performances of the proposed models. CTBNCs learned by maximizing Marginal Log-likelihood, and Conditional Log-likelihood scores were compared with the existing approaches in the state of the art: Continuous Time Naive Bayes (CTNB) and Dynamic Bayesian Network Classifiers (DBNCs). Regarding the supervised classification problem, CTBNCs learned by maximizing the Conditional Log-likelihood score are the most effective approaches. This is particularly true when the amount of data is reduced. When the clustering problem is addressed, CTBNCs learned by maximizing the Marginal Log-likelihood score are the most effective approaches. In both cases, i.e. supervised classification and clustering, CTBNCs outperformed DBNCs even on discrete time problems. This is also true on synthetic data sets generated from Dynamic Bayesian Networks (DBNs). The experiments showed that max-2 Augmented Continuous Time Naive Bayes (ACTNB) is the best compromise between model complexity and effectiveness.

Two interesting real world problems were addressed using the proposed models: gesture recognition for post-stroke rehabilitation and traffic profile classification (Chapter 6). CTBNCs were found to be an effective approach to dealing with both the problems. CTBNCs performed better than DBNCs, the approach of the state of the art used for comparison.

The last contribution of the thesis was the CTBNCToolkit: an open source toolkit for CTBNCs. The toolkit is described in Appendix A. Using it, it is possible to replicate the numerical experiments described in this dissertation.

7.2 FUTURE WORKS AND PERSPECTIVES

CTBNCs were recently introduced in the literature. After the contributions of this thesis (Section 1.2) there are still open issues that must be taken into account.

CTBNCs efficiently address the classification problem of multivariate trajectories when the class is static and the trajectories are completely observable. Relaxing these two assumptions is the first natural future step. It would be interesting to try to understand if there is an efficient way to deal with missing value trajectories under particular conditions. The same question holds when the class changes over time. Is it possible to preserve the efficient inference of the CTBNCs under some assumption even when the class changes over time?

Another natural extension is to model the time evolution using different distributions. Some work has been done in this direction for CTBNs (Gopalratnam et al., 2005; Nodelman et al., 2012). This is particularly interesting in the case of real world applications where the memoryless property of the exponential distribution can be a limitation. In this sense further experiments, especially addressed to real world clustering problems, would be interesting.

Enlarging the horizon, two other paths seem promising: to try to apply the inference algorithm of CTBNCs on Piecewise-constant Conditional Intensity Models (PCIMs) (Gunawardana et al., 2011; Weiss and Page, 2013) and to try to add decision and utility nodes on CTBNs in order to model the decision process over continuous time.

A

CTBNCTOOLKIT

In this dissertation Continuous Time Bayesian Network Classifiers (CTBNCs) have been studied. An open source Java toolkit has been developed to provide a CTBNCs implementation. The toolkit can be used as a stand-alone application (Section A.1), it can be used as an external library or it can be extended to provide new features (Section A.4).

Section A.1 provides a guide about the CTBNCToolkit stand-alone usage; download information is provided in Section A.1.1. Section A.2 explains how to read the results of the classification inference, while Section A.3 provides a tutorial set of examples to use the package as a stand-alone application. Finally, the code is presented and analyzed in Section A.4.

A.1 CTBNCTOOLKIT HOW TO

A.1.1 *Download*

CTBNCToolkit can be downloaded as a stand-alone application. It requires *opencsv-2.3* library[1] to read the csv files and *commons-math3-3.0* library[2] for the Gamma function calculation.

1 http://opencsv.sourceforge.net/

2 http://commons.apache.org/proper/commons-math/download_math.cgi

To download and use the compiled CTBNCToolkit follow these steps:

- download CTBNCToolkit `.jar` file from http://dcodecasa.wordpress.com/ctbnc/ctbnctoolkit/ website[3];
- download opencsv library from http://sourceforge.net/projects/opencsv/ web site (tests were made with version 2.3);
- download commons math library from http://commons.apache.org/proper/commons-math/download_math.cgi web site (tests were made with version 3.0).

The compiled CTBNCToolkit can be used as a stand-alone application, as showed in the next sections. On the same website where the `.jar` file is released, it is possible to find the published papers related to CTBNCs and a collection of free data sets to test CTBNCToolkit (see Section A.3).

CTBNCToolkit source code is released under GPL v2.0[4] license. The source code is available on GitHub at the following url: https://github.com/dcodecasa/CTBNCToolkit[3]. It can be freely used in accordance with the GPL v2.0 license.

A.1.2 *Run experiments from the command line*

Once downloaded the CTBNCToolkit jar file, or generated from the source code, it can be run as follows:

```
java -jar CTBNCToolkit.jar <parameters> <data>
```

where `<parameters>` are the CTBNCToolkit parameters (i.e modifiers), addressed in the following, and `<data>` is a directory

3 CTBNCToolkit is not available yet for publication reasons. It will be soon available as described.
4 GPL v2.0 license: http://www.gnu.org/licenses/gpl-2.0.html

containing a data set. This data set is used to generate both the training set and the test set (see Section A.1.2.4), unless `--training` or `--testset` modifier are used. In this case `<data>` is considered the test set, and it can refer to a single file (see Section A.1.2.19 and Section A.1.2.20). Hereafter the terms parameters and modifiers will be used interchangeably.

For sake of simplicity, the paths of the required libraries are defined in the manifest file contained in the `.jar` archive. Libraries are supposed to be saved in `lib/` directory. Tests are made using the Java virtual machine version 1.7.0_25 in ubuntu.

It is worthwhile to note that it is often necessary to add the Java virtual machine parameter `-Xmx` to increase the heap space and to avoid out of memory exceptions (for example `-Xmx2048m` increases the heap dimension to 2Gb).

CTBNCToolkit parameters can be without any arguments, if specified as `--modifier`; or with any number of arguments, if specified as `--modifier=arg1,arg2,..,argN`. Table 9 summarizes all the parameters while Table 10 shows parameter incompatibilities and dependencies. The next sections address each parameter separately.

A.1.2.1 *Help*

`--help` prints on the screen the help that shows the allowed parameters. For each parameter a short description is provided. When help is shown all the other parameters are ignored, and the program terminates after printing help.

```
java -jar CTBNCToolkit.jar --help
```

A.1.2.2 *Models to learn*

`--CTBNC` modifier allows to specify the list of CTBNCs to test.

The models allowed follow:

Parameter	Method	Arguments	Section
`--help`	`printHelp`	no	A.1.2.1
`--CTBNC`	`setCTBNCModels`	yes	A.1.2.2
`--model`	`setModels`	yes	A.1.2.3
`--validation`	`setValidationMethod`	yes	A.1.2.4
`--clustering`	`setClustering`	yes	A.1.2.5
`--1vs1`	`setModelToClass`	no	A.1.2.6
`--bThreshold`	`setBinaryDecider`	yes	A.1.2.7
`--testName`	`setTestName`	yes	A.1.2.8
`--ext`	`setFileExt`	yes	A.1.2.9
`--sep`	`setFileSeparator`	yes	A.1.2.10
`--className`	`setClassColumnName`	yes	A.1.2.11
`--timeName`	`setTimeColumnName`	yes	A.1.2.12
`--trjSeparator`	`setTrjSeparator`	yes	A.1.2.13
`--validColumns`	`setValidColumns`	yes	A.1.2.14
`--cvPartitions`	`setCVPartitions`	yes	A.1.2.15
`--cvPrefix`	`setCVPrefix`	yes	A.1.2.16
`--cutPercentage`	`setCutPercentage`	yes	A.1.2.17
`--timeFactor`	`setTimeFactor`	yes	A.1.2.18
`--training`	`setTrainingSet`	yes	A.1.2.19
`--testset`	`setTestSet`	yes	A.1.2.20
`--rPath`	`setResultsPath`	yes	A.1.2.21
`--confidence`	`setConfidence`	yes	A.1.2.22
`--noprob`	`disableProbabilities`	no	A.1.2.23
`--v`	`setVerbose`	no	A.1.2.24

Table 9.: It is shown for each modifier: the method of the `CommandLine` class that manages it, the presence or absence of arguments, and the Section in which the modifier is described.

- `CTNB`: Continuous Time Naive Bayes (CTNB) (Definition 3.2.2) (Stella and Amer, 2012; Codecasa and Stella, 2013);

Idx	Parameter	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	`--help`		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
2	`--CTBNC`	X																							
3	`--model`	X					X					D			D										
4	`--validation`	X				X																			
5	`--clustering`	X			X		X	X																	
6	`--1vs1`	X		X		X																			
7	`--bThreshold`	X				X																			
8	`--testName`	X																							
9	`--ext`	X																							
10	`--sep`	X																							
11	`--className`	X																							
12	`--timeName`	X																							
13	`--trjSeparator`	X																							
14	`--validColumns`	X																							
15	`--cvPartitions`	X			D																				
16	`--cvPrefix`	X			D											D									
17	`--cutPercentage`	X																							
18	`--timeFactor`	X																							
19	`--training`	X			D																				
20	`--testset`	X		D	D															D					
21	`--rPath`	X																							
22	`--confidence`	X																							
23	`--noprob`	X																							
24	`--v`	X																							

Table 10.: Incompatibilities (X marks) and dependencies (D marks) between the modifiers. Incompatibility relations are symmetric; this does not hold for dependency relations, i.e. a modifier requires a particular value for another modifier, but the opposite it is not necessarily true. Modifiers are indicated by an index for due to the problems of space (first column).

- `ACTNBk-f`: Max-k Augmented Continuous Time Naive Bayes (ACTNB) (Definition 4.2.2) (Codecasa and Stella, 2013) where `k` is the number of parents ($\geqslant 2$) and `f` is the scoring function used to learn the structure (`LL` or `CLL`);

- `CTBNCk-f`: Max-k Continuous Time Bayesian Network Classifier (CTBNC) (Definition 4.2.1) (Codecasa and Stella, 2013) where `k` is the number of parents ($\geqslant 1$) and `f` is the scoring function used to learn the structure (`LL` or `CLL`);

where

- `LL` stands for Marginal Log-likelihood scoring function (Section 3.1.3) (Nodelman et al., 2002b; Codecasa and Stella, 2013);
- `CLL` stands for Conditional Log-likelihood scoring function (Section 4.3.2) (Codecasa and Stella, 2013).

After each model definition, it is possible to specify the modifier parameters, which define the imaginary counts of the hyperparameters (Section 3.1.2):

- `Mk`: `k` are the imaginary counts related to the number of transitions for each variable (default value: `1.0`);
- `Tk`: `k` is the imaginary amount of time spent in a variable state (default value: `0.005`);
- `Pk`: `k` are the imaginary counts related to the class occurrences (default value: `1.0`).

It is a good habit to avoid zero values for the imaginary counts. Indeed, when the data set is not big enough, some model parameters can be 0 with the risk of a division by 0 error.

In addition to each model learned with the structural learning, it is possible to add the following parameter:

- `penalty`: adds the dimension penalty during the structural learning process; if omitted the penalty is disabled[5].

Here is an example:

5 The `penalty` flag is ignored if applied with the CTNB model.

```
java -jar CTBNCToolkit.jar --CTBNC=CTNB,M0.1,T0.001,ACTNB2-
   CLL,CTBNC4-LL,penalty <data>
```

The previous line enables the following tree models:

- a CTNB with 0.1 as prior for the variable counts, 0.001 as the time prior and the standard 1.0 as class imaginary counts;
- a Max-2 ACTNB learned maximizing the Conditional Log-likelihood score and with the default parameter priors;
- a Max-4 CTBNC learned maximizing the Marginal Log-likelihood, using the default priors and enabling the dimension penalty during the learning process.

A.1.2.3 *Model loading*

`--model` modifier allows to specify the file paths of CTBNC models to load[6]. The models must be in the `.ctbn` format (see Section A.4.3.3). If a training set is defined, the training set will be used only to learn models defined with the `--CTBNC` modifier (see Section A.1.2.2).

Here are some examples:

- tests the model stored in `model.ctbn` file:

  ```
  java -jar CTBNCToolkit.jar --model=models/model.ctbn <
  other_parameters> <data>
  ```

- tests the models stored in `model1.ctbn` and `model2.ctbn` files, and learn and test a CTNB model:

  ```
  java -jar CTBNCToolkit.jar --model=models/model1.ctbn,
  models/model2.ctbn
  --CTBNC=CTNB <other_parameters> <data>
  ```

6 The `--model` modifier will be soon implemented.

It is not possible to apply the `--1vs1` modifier or to load models generated using the `--1vs1` modifier (see Section A.1.2.6). The loaded models must be compatible with the input of `--className` modifier (see Section A.1.2.11) and `--validColumns` modifier (see Section A.1.2.14).

A.1.2.4 *Validation method*

The `--validation` modifier allows to specify the validation method to use. The CTBNCToolkit provides three validation methods: *hold-out*, *cross-validation* (Witten and Frank, 2005) and a validation method used for the clustering tests. The clustering validation method is automatically used when the clustering is enabled (see Section A.1.2.5). While hold-out and cross-validation can be enabled as follows:

- `--validation=HO,0.6`: enables the hold out validation method with a random partitioning of the data set in the training set (60%) and test set (40%) (default value: `0.7`);
- `--validation=CV,k`: enables the cross-validation with `k` folds (default value: `10`).

Here are some examples:

- 70%-30% hold-out partitioning:

```
java -jar CTBNCToolkit.jar --validation=HO <
    other_parameters> <data>
```

- 60%-40% hold-out partitioning:

```
java -jar CTBNCToolkit.jar --validation=HO,0.6 <
    other_parameters> <data>
```

- `10`-folds cross-validation:

```
java -jar CTBNCToolkit.jar --validation=CV <
    other_parameters> <data>
```

- 8-folds cross-validation:

```
java -jar CTBNCToolkit.jar --validation=CV,8 <
    other_parameters> <data>
```

No validation method can be specified in case of clustering (see Section A.1.2.5).

A.1.2.5 *Clustering*

The `--clustering` modifier disables the supervised learning and enables clustering (Hartigan, 1975). Tests require labeled data sets because the performances are calculated over the complete data set using external measures (see Section A.2.2), i.e. the *Rand index (R)* (Rand, 1971), *Jaccard's coefficient (J)* (Halkidi et al., 2001) and, the *Fowlkes–Mallows index (FM)* (Fowlkes and Mallows, 1983) (see Section 5.2.2).

CTBNC clustering is implemented using Expectation Maximization (EM) (Koller and Friedman, 2009). Both *soft-assignment* and *hard-assignment* clustering are implemented (Section 5.1.1). The parameters allow to specify the clustering method and the termination criterion as follows:

- `hard/soft`: enable the clustering method (default value: `soft`);
- an integer number (i.e. `15`): set to `15` the maximum number of iteration in the EM algorithm (default value: `10`);
- a `double` number (i.e. `0.1`): percentage of the data set trajectories; if less trajectories change class the EM algorithm is interrupted (default value: `0.01`).

All these parameter values are optional and can be inserted in any order.

Here are some examples:

- soft clustering, `10` iterations, 1% as the trajectory threshold:

```
java -jar CTBNCToolkit.jar --clustering <
    other_parameters> <data>
```

- soft clustering, `15` iterations, 10% as the trajectory threshold:

```
java -jar CTBNCToolkit.jar --clustering=soft,0.1,15 <
    other_parameters> <data>
```

- soft clustering, `10` iterations, 3% as the trajectory threshold:

```
java -jar CTBNCToolkit.jar clustering=0.03 <
    other_parameters> <data>
```

- hard clustering, `6` iterations, 5% as the trajectory threshold:

```
java -jar CTBNCToolkit.jar --clustering=6,0.05,hard <
    other_parameters> <data>
```

When the clustering is enabled a dedicated validation method is used. For this reason the validation modifier cannot be used (see Section A.1.2.4). Also the classification threshold for binary class problem cannot be used (see Section A.1.2.7).

A.1.2.6 *One model one class*

`--1vs1` modifier enables the one model one class modality. This modality generates for each model specified with the `--CTBNC` modifier (see Section A.1.2.2) a set of models, one for each class. For example, if a CTNB is required using the `--CTBNC` modifier over a 10 class data set, this modifier will force the generation of 10 CTNB models. Each one of the ten generated models will discriminate one class against the others. During the classification process each model returns the probability related to the class for which it is specialized. The classification class is the one with the highest probability.

```
java -jar CTBNCToolkit.jar --1vs1 <other_parameters> <data>
```

A.1.2.7 *Binary class threshold*

The `--bThreshold` modifier changes the probability threshold used for choosing the class in supervised classification of binary problems. The default value is `0.5` and indicates that the class with the highest probability will be chosen.

Here are some examples:

- a trajectory is classified in the first class only if its probability to be in that class is greater or equal to `0.4`:

  ```
  lstinline{java -jar CTBNCToolkit.jar --bThreshold=0.4
     <other_parameters> <data>
  ```

- a trajectory is classified in the first class only if its probability to be in that class is greater or equal to `0.6`:

  ```
  \lstinline{java -jar CTBNCToolkit.jar --bThreshold=0.6
     <other_parameters> <data>
  ```

Classes are ordered alphabetically. For example, if "A" and "B" are the classes, the first example gives advantage to class "A", while the second example gives advantage to class "B".

This modifier can be used only on data sets with two class states and cannot be used in the case of clustering (see Section A.1.2.5).

A.1.2.8 *Test name*

The `--testName` modifier specifies the name of the test. This name is used during the printing of the results to identify the particular test. The default value is specified by the current time using the `"yyMMddHHmm\_Test"` format.

```
java -jar CTBNCToolkit.jar --testName=CTNBTest1 <
   other_parameters> <data>
```

A.1.2.9 *File extension*

The `--ext` modifier allows to specify the extension of the files to load in the data set directory (default value: `.csv`).

With the following command all the files in the data set directory with `.txt` extension are loaded:

```
java -jar CTBNCToolkit.jar --ext=.txt <other_parameters> <
   data>
```

A.1.2.10 *Column separator*

The `--sep` modifier allows to specify the column separator of the file to load (default value: `,`).

```
java -jar CTBNCToolkit.jar --sep=; <other_parameters> <data>
```

A.1.2.11 *Class column name*

The `--className` modifier specifies the name of the class column in the file to load (default value: `class`).

```
java -jar CTBNCToolkit.jar --className=weather <
   other_parameters> <data>
```

A.1.2.12 *Time column name*

The `--timeName` modifier specifies the name of the time column in the file to load (default value: `t`).

```
java -jar CTBNCToolkit.jar --className=time <
   other_parameters> <data>
```

A.1.2.13 *Trajectory separator column name*

Many data sets provide multiple trajectories in the same file. The `--trjSeparator` enables trajectory separation using the information provided by a target column. For example, if the trajectories are indexed by an incremental number ("trjIndex" column) that indicates the trajectories in the file, it is possible to split the trajectories automatically, as follows:

```
java -jar CTBNCToolkit.jar --trjSeparator=trjIndex <
   other_parameters> <data>
```

By default this modifier is not used, and a one file - one trajectory matching is assumed.

A.1.2.14 *Data columns*

The `--validColumns` modifier allows to specify the columns of the file that correspond to the variables in the models to be generated. By default, i.e. when the modifier is not specified, all the columns except the time column (see Section A.1.2.12) and the trajectory separator column (see Section A.1.2.13) are considered variables.

It is possible to specify any number of columns as arguments of the modifier.

```
java -jar CTBNCToolkit.jar --validColumns=clmn1,clmn2,clmn3
   <other_parameters> <data>
```

A.1.2.15 *Cross-validation partitions*

The `--cvPartitions` modifier allows to specify a partitioning when the cross-validation method is enabled (see Section A.1.2.4). This modifier indicates a file in which a cross-validation partitioning is specified. The cross-validation method will follow the partitioning instead of generating a new one.

The partitioning file can be a result file, generated by the CTBNCToolkit (see Section A.2.1):

```
Test1
trj12.txt: True Class: 4, Predicted: 4, Probability:
   0.893885
...
trj87.txt: True Class: 2, Predicted: 2, Probability:
   0.494983
Test2
trj26.txt: True Class: 2, Predicted: 2, Probability:
   0.611254
...
trj96.txt: True Class: 2, Predicted: 3, Probability:
   0.637652
Test3
trj1.txt: True Class: 4, Predicted: 4, Probability:
   0.5697770
...
trj80.txt: True Class: 1, Predicted: 1, Probability:
   0.938935
Test4
trj15.txt: True Class: 4, Predicted: 4, Probability:
   0.624698
...
trj8.txt: True Class: 2, Predicted: 2, Probability:
   0.7586410
```

```
Test5
trj11.txt: True Class: 4, Predicted: 4, Probability:
   0.911368
...
trj99.txt: True Class: 4, Predicted: 4, Probability:
   0.413442
```

or a text file where the word `Test` identifies the cross-validation folders, and the following lines specify the trajectories to load for each folder[7]:

```
Test 1 of 5:
trj12.txt
...
trj87.txt
Test 2 of 5:
trj26.txt
...
trj96.txt
Test 3 of 5:
trj1.txt
...
trj80.txt
Test 4 of 5:
trj15.txt
...
trj8.txt
Test 5 of 5:
trj11.txt
...
trj99.txt
```

Here is an example where the file `partition.txt` in the `CV` directory is loaded:

7 The one trajectory - one file matching is supposed.

```
java -jar CTBNCToolkit.jar --cvPartitions=CV/partition.txt <
    other_parameters> <data>
```

This modifier requires to enable cross-validation (see Section A.1.2.4).

A.1.2.16 *Cross-validation prefix*

The `--cvPrefix` allows to specify a prefix to remove from the trajectory names in the partition file.

For example, if the partition file loaded with the `--cvPartitions` modifier has for some reasons the following form:

```
Test 1 of 5:
ex-trj12.txt
...
ex-trj87.txt
Test 2 of 5:
ex-trj26.txt
...
ex-trj96.txt
Test 3 of 5:
ex-trj1.txt
...
ex-trj80.txt
Test 4 of 5:
ex-trj15.txt
...
ex-trj8.txt
Test 5 of 5:
ex-trj11.txt
...
ex-trj99.txt
```

the `ex-` prefix can be removed automatically as follows:

```
java -jar CTBNCToolkit.jar --cvPrefix=ex- <other_parameters>
    <data>
```

This modifier requires the definition of a cross-validation partition file (see Section A.1.2.15).

A.1.2.17 *Data sets reduction*

The `--cutPercentage` modifier allows to reduce the data set dimension in terms of number of trajectories and trajectory length. With only one parameter (which is a percentage) it is possible to perform tests over reduced data sets in order to evaluate the ability of the models to work with a restricted amount of data.

The following line forces a random selection of 60% of the data set trajectories and reduces each selected trajectory to 60% of its original length:

```
java -jar CTBNCToolkit.jar --cutPercentage=0.6 <
   other_parameters> <data>
```

The default value is `1.0` that does not change the data amount.

A.1.2.18 *Time modifier*

The `--timeFactor` modifier specifies a time factor used to scale the trajectory timing.

Here are some examples:

- doubles the trajectory timing:

  ```
  java -jar CTBNCToolkit.jar --timeFactor=2.0 <
     other_parameters> <data>
  ```

- reduces the trajectory timing by a factor of ten:

```
java -jar CTBNCToolkit.jar --timeFactor=0.1 <
    other_parameters> <data>
```

The default value is `1.0`, which implies no time transformations.

A.1.2.19 *Training set*

The `--training` modifier specifies the directory that contains the training set. Using this modifier the `<data>` path in the command line is used as the test set. If the `--training` modifier is used, `<data>` can be a file or a directory.

Here are some examples:

- the files in the `trainingDir/` directory compose the training set, while the files in `testDir/` compose the test set:

```
java -jar CTBNCToolkit.jar --training=trainingDir/ <
    other_parameters> testDir/
```

- the files in the `training/` directory compose the training set, while the models are tested on the `testDir/trj.txt` file:

```
java -jar CTBNCToolkit.jar --training=training/ <
    other_parameters> testDir/trj.txt
```

This modifier requires the use of the hold out validation method.

A.1.2.20 *Test set*

The `--testset` modifier forces to consider `<data>` as the test set[8]. `<data>` can be a folder or a file. This modifier can be used with

8 `--testset` modifier will be soon implemented.

the `--model` modifier (see Section A.1.2.3) and the hold out validation method to avoid splitting the input data set into training and test set, when the definition of a training set is not necessary (i.e. when `--CTBNC` modifier is not used). This modifier can be omitted, if `--training` modifier (see Section A.1.2.19) is used.

Here are some examples:

- the files in the `trainingDir/` directory compose the training set, while the files in `testDir/` compose the test set:

```
java -jar CTBNCToolkit.jar --training=trainingDir/ <
    other_parameters> testDir/
```

- the files in the `trainingDir/` directory compose the training set, while the files in `testDir/` compose the test set:

```
java -jar CTBNCToolkit.jar --training=trainingDir/ --
    testset <other_parameters> testDir/
```

- no training set is specified, `model.ctbn` is loaded and tested on `testDir/trj.txt` file:

```
\lstinline{java -jar CTBNCToolkit.jar --testset --
    model=model.ctbn <other_parameters> testDir/trj.txt
```

This modifier requires the use of the hold out validation method.

A.1.2.21 *Results path*

The `--rPath` modifies specifies the directory where to store the results (see Section A.2). If it is not specified, the results are stored in a directory named as the test name (see Section A.1.2.8) under the directory of the data (i.e. `<data>`) specified in the CTBNCToolkit command line.

Here are some examples:

- save the results in `resultDir/` directory:

```
java -jar CTBNCToolkit.jar --rPath=resultDir --
    testName=Test1 <other_parameters> <data>
```

- save the results in `testDir/Test2/` directory:

```
java -jar CTBNCToolkit.jar --testName=Test2 <
    other_parameters> testDir/
```

- save the results in `testDir/Test3/` directory:

```
java -jar CTBNCToolkit.jar --testName=Test3 <
    other_parameters> testDir/file.txt
```

A.1.2.22 *Confidence interval*

The `--confidence` modifier allows to specify the confidence level to use in the model evaluation. The confidence level is used for model comparison in the case of supervised classification (see Section A.2.1.3). The allowed levels of confidence are: 99.9%, 99.8%, 99%, 98%, 95%, 90%, and 80%. 90% is the default value.

Here are some examples:

- 90% confidence:

```
java -jar CTBNCToolkit.jar <other_parameters> <data>
```

- 90% confidence:

```
java -jar CTBNCToolkit.jar --confidence=90% <
    other_parameters> <data>
```

- 99% confidence:

```
java -jar CTBNCToolkit.jar --confidence=99% <
    other_parameters> <data>
```

A.1.2.23 *Disable class probability*

The `--noprob` modifier disables the calculation of the class probability distribution all across the trajectory. The only probability values are calculated at the end of the trajectory. This allows to speed up the computation.

```
java -jar CTBNCToolkit.jar --noprob <other_parameters> <data
    >
```

A.1.2.24 *Verbose*

The `--v` modifier enables the verbose modality. This modality prints more information that helps to follow the execution of the CTBNCToolkit.

```
java -jar CTBNCToolkit.jar --v <other_parameters> <data>
```

A.2 READ THE RESULTS

Once a test is executed, the performances are printed in the results directory (see Section A.1.2.21). The directory contains the performances of all the tested models. The performances of each model are identified by its name. Model names start with `Mi`, where `i` is the index that identifies the model in the command line.

For example, the command line:

```
java -jar CTBNCToolkit.jar --CTBNC=CTNB,M0.1,T0.001,ACTNB2-
    CLL,CTBNC4-LL,penalty <data>
```

generates the following model names: `M0_CTNB`, `M1_ACTNB2-CLL`, `M2_CTBNC4-LL`.

In the case where external models are loaded (Section A.1.2.3), the `i` index first refers to the model defined with the

`--CTBNC` modifier (Section A.1.2.2), then it refers to the external models.

Here is another example. The command line:

```
java -jar CTBNCToolkit.jar --model=models/model1.ctbn,models
   /model2.ctbn --CTBNC=CTNB
<other_parameters> <data>
```

generates the following model names: `M0_CTNB`, `M1_model1`, `M2_model2`.

The following sections describe the performances calculated in the case of classification and clustering. In both cases the results directory contains a file that shows the modifiers used in the test.

A.2.1 *Classification*

A.2.1.1 *Results file*

For each tested model a results file is provided. It can be found in the test directory and contains the results for each classified data set instance. For each instance the name, the true class, the predicted class and the probability of the predicted class is shown.

Here is an example:

```
trj1: True Class: s2, Predicted: s2, Probability:
   0.9942336124116266
trj10: True Class: s4, Predicted: s2, Probability:
   0.9896614159579054
trj100: True Class: s1, Predicted: s1, Probability:
   0.9955018513870293
trj11: True Class: s4, Predicted: s4, Probability:
   0.977733953650775
trj12: True Class: s3, Predicted: s3, Probability:
   0.9997240904249852
```

```
...
```

A.2.1.2 *Metrics*

In the results directory the metric file is stored. It is the `.csv` file that contains the following performances for each tested model (Witten and Frank, 2005; Fawcett, 2006; Japkowicz and Shah, 2011):

- Accuracy: accuracy value with the confidence interval calculated in accordance with the defined level of confidence (Section A.1.2.22);
- Error: the percentage of wrong classified instances;
- Precision: precision value for each class;
- Recall: recall value for each class[9];
- F-Measure: balanced f-measure for each class (i.e. precision and recall weigh equally);
- PR AUC: precision-recall AUC (Area Under the Curve) for each class;
- Sensitivity: sensitivity for each class[9];
- Specificity: specificity for each class;
- TP-Rate: true positive rate for each class[9];
- FP-Rate: false positive rate for each class;
- ROC AUC: ROC Area Under the Curve for each class;
- Brier: brier value;
- Avg learning time: learning time or average learning time in case of multiple tests (i.e. cross-validation);

9 Sensitivity, TP-rate and Recall are the same.

- Var learning time: learning time variance in case of multiple tests (i.e. cross-validation);
- Avg inference time: average inference time between all the tested instances;
- Var inference time: inference time variance between all the tested instances;

Some information in the metric file depends on the validation method. In the case of cross validation (Section A.1.2.4), the file contains both the micro-averaging and the macro-averaging performances.

A.2.1.3 *Model comparison*

The model comparison file in the results directory contains the comparison matrices between the tested models.

A comparison matrix is a squared matrix that compares each pair of tested models by using their corresponding accuracy values. The comparison file contains the comparison matrices for the following confidence levels 99%, 95%, 90%, 80%, and 70%.

Here is an example of the comparison matrix with 99% confidence:

Comparison test	99%						
	M0	M1	M2	M3	M4	M5	M6
M0		UP	UP	UP	UP	UP	0
M1	LF		UP	0	UP	0	LF
M2	LF	LF		LF	0	LF	LF
M3	LF	0	UP		UP	0	LF
M4	LF	LF	0	LF		LF	LF
M5	LF	0	UP	0	UP		LF
M6	0	UP	UP	UP	UP	UP	

UP indicates that the upper model is statistically better than the left model, LF indicates that the left model is statistically better than the upper model, while 0 indicates that the models are indistinguishable .

A.2.1.4 *Single run data*

The results directory contains a directory for each model. In the case of cross-validation, the model directory stores a directory named "runs" that contains a metric file with the performances of each single run. In this directory all the models generated by the learning process are stored.

A.2.1.5 *ROC curve*

The model directory also contains a directory named "ROCs" which stores the ROC curves for each class (Fawcett, 2006). In the case of cross-validation the curves calculated in micro and macro averaging are stored. The macro averaging curves show the confidence interval due to the vertical averaging. The confidence interval can be set using the `--confidence` modifier (see Section A.1.2.22).

A.2.1.6 *Precision-Recall curve*

The model directory contains a directory named "Precision-Recall" which stores the precision-recall curves for each class (Fawcett, 2006). In the case of cross-validation the curves calculated in micro and macro averaging are stored. The macro averaging curves show the confidence interval due to the vertical averaging. The confidence interval can be set using the `--confidence` modifier (see Section A.1.2.22).

A.2.1.7 *Lift chart and cumulative response*

The directories of the models also contain a directory named "CumulativeResp&LiftChar" which stores the cumulative response curves and the lift charts for each class (Fawcett, 2006). In the case of cross-validation the curves calculated in micro and macro averaging are stored. The macro averaging curves show the confidence interval due to the vertical averaging. The confidence interval can be set using the `--confidence` modifier (see Section A.1.2.22).

A.2.2 *Clustering*

In the case of clustering, the results directory contains a directory for each tested model. Each directory contains a read me file with some information related to the test and the following files.

A.2.2.1 *Results file*

As in the case of supervised classification for each tested model a result file is provided. It contains the results for each classified data set instance. For each instance the name, the true class, the predicted cluster and the probability of the predicted cluster is shown. Of course there is no correspondence between class and cluster names.

Here is an example of the results file:

```
trj1.txt: True Class: 3, Predicted: 2, Probability:
   0.9998733687935368
trj10.txt: True Class: 3, Predicted: 2, Probability:
   0.9999799985527281
trj100.txt: True Class: 3, Predicted: 2, Probability:
   0.9998951904099552
```

```
trj11.txt: True Class: 3, Predicted: 2, Probability:
    0.9999999967591123
trj12.txt: True Class: 4, Predicted: 4, Probability:
    0.9999999983049304
...
```

A.2.2.2 *Performances*

The file performances contains the following external measures (Halkidi et al., 2001; Gan et al., 2007; Xu and Wunsch, 2008):

- the Rand index (R) (Rand, 1971);
- Jaccard's coefficient (J) (Halkidi et al., 2001);
- the Fowlkes–Mallows index (FM) (Fowlkes and Mallows, 1983).

In addition to these measures, association matrix, clustering-partition matrix, precision matrix, recall matrix, and f-measure matrix are shown. Learning time, average inference time and inference time variance are also shown.

A.2.2.3 *Model file*

The model file shows the CTBNC learned in the test. Since the clustering validator learns one model all over the data set, just one model file is generated. `.ctbn` is the format used (see Section A.4.3.3).

A.3 TUTORIAL EXAMPLES

This section provides replicable examples of CTBNCToolkit usage. To execute the tests follow the next steps:

- download the compiled version of the CTBNCToolkit or download and compile the source code (see Section A.1.1);
- create a directory for the tests where the compiled CTBNCToolkit must be copied (let's call the directory "tutorial");
- create a sub-directory named "lib" containing the required libraries (see Section A.1.1)[10];
- copy in the "tutorial" directory, for each of the following tests, the data sets used (see Section A.1.1).

A.3.1 *Classification*

In this section the tutorial examples of classification are given. For these tests "naiveBayes1" synthetic data set is used. The data set has to be downloaded and copied under the "tutorial" directory. The trajectories will be available in the "tutorial/naiveBayes1/dataset/" directory.

A.3.1.1 *Hold out*

Assume we are interested in making classifications using CTNB, ACTNB learned by maximizing Conditional Log-likelihood and CTBNC learned by maximizing Marginal Log-likelihood. The last two models learned setting to two the maximum number of parents. To evaluate the performances of the models we want to use the hold out validation method.

The data set is stored in `.txt` files. Trajectories use "Class" as the class column name and "t" as the time column name. To

10 Other directories can be used once the manifest file in the CTBNCToolkit `.jar`, is modified.

realize the test the following command must be used setting “tutorial” as the current directory[11]:

```
java -jar CTBNCToolkit.jar --CTBNC=CTNB,ACTNB2-CLL,CTBNC2-LL
    --validation=HO --ext=.txt --className=Class
   naiveBayes1/dataset/
```

where:

- `--CTBNC=CTNB,ACTNB2-CLL,CTBNC2-LL` specifies the models to test (Section A.1.2.2);
- `--validation=HO` specifies hold out as the validation method (Section A.1.2.4);
- `--ext=.txt` specifies that the data set files have `.txt` extension (Section A.1.2.9);
- `--className=Class` specifies that the class column has the name “Class” (“class” is the default value, see Section A.1.2.11);
- `naiveBayes1/dataset/` specifies the directory which contains the data set (Section A.1.2).

It is worthwhile to note that the time column name is not specified because “t” is the default name (Section A.1.2.12). Eventually the `--v` modifier can be added to enable the verbose modality in order to monitor the program execution (Section A.1.2.24).

The results are stored in a directory with a name similar to “131014_1041_Test”. To specify a different name use the `--testName` modifier (Section A.1.2.8).

The results follow the guidelines explained in Section A.2.1.

11 This test may takes a while.

A.3.1.2 *Cross validation loading a partitioning*

Assume we are interested in making classifications using just a CTNB. To evaluate the performances we use a 10 fold cross-validation, loading a pre-defined partitioning. The partitions are described by the `NB-results.txt` file in the "naiveBayes1/" directory.

Similarly to the previous example, here is the command line to execute the test:

```
java -jar CTBNCToolkit.jar --CTBNC=CTNB --validation=CV --
    cvPartitions=naiveBayes1/NB-results.txt --ext=.txt --
    className=Class naiveBayes1/dataset/
```

The difference from the classification example in Section A.3.1.1 relies on the following two points:

- `--validation=CV` enables cross-validation to validate the models (Section A.1.2.4);
- `--cvPartitions=naiveBayes1/NB-results.txt` specifies the file which contains the cross-validation partitioning (Section A.1.2.15).

`NB-results.txt` is the results file (Section A.2.2.1) of a CTNB model learned on the same data set during the experiment campaign realized for this dissertation (Section 4.4). The results file can be loaded as a partitioning file (Section A.1.2.15). This allows to replicate the executed tests.

To compare the results just obtained with those of this dissertation see Table 12 Section B.1.2.

A.3.2 *Clustering*

In this section a tutorial example of clustering is described. Also in this case the "naiveBayes1" synthetic data set is used. The data set has to be downloaded and copied under the "tutorial" directory. The trajectories will be contained in "tutorial/naive-Bayes1/dataset/" directory.

Assume we are interested in making clustering using CTNB and CTBNC learned by maximizing Marginal Log-likelihood when the maximum number of parents is set to two.

The data set is stored in `.txt` files. Trajectories use "Class" as the class column name and "t" as the time column name. To execute the test the following command must be used setting "tutorial" as the current directory[11]:

```
java -jar CTBNCToolkit.jar --CTBNC=CTNB,CTBNC2-LL --
    clustering --ext=.txt --className=Class naiveBayes1/
    dataset/
```

The command is similar to the one used in Section A.3.1.1, the main difference is the use of the `--clustering` modifier which substitutes the validation modifier (Section A.1.2.5).

The results are stored in a directory with a name similar to "131014_1252_Test". To specify a different name use the `--testName` modifier (Section A.1.2.8).

The results follow the guidelines explained in Section A.2.2. Since learning and testing in case of clustering are done using the whole data set, it is easy to replicate the clustering experiments shown in Section 5.2.3.

To compare the results just obtained with those of this dissertation see Table 20 Section B.2.2.1. The clustering results could be quite different due to the random instantiation of the EM algorithm (Section 5.1).

A.4 CTBNCTOOLKIT LIBRARY

CTBNCToolkit is released under the GPL v2.0[12] license. It is available on GitHub at the following url: `https://github.com/dcodecasa/CTBNCToolkit`. See section A.1.1 for more information regarding the CTBNCToolkit download.

The CTBNCToolkit is a library to manage CTBNCs. It provides:

- a CTBNC model representation which can be easily extended to define other types of models (Section A.4.3);
- a supervised learning algorithm (Section A.4.4);
- soft and hard assignment EM algorithms for clustering purposes (Section A.4.5);
- two different scoring functions to learn CTBNCs (Section A.4.4);
- CTBNCs inference algorithm for static classification (Section A.4.6);
- three different validation methods to realize experiments (Section A.4.7),
- a rich set of performance measures for supervised classification and clustering as well (Section A.4.8);
- a set of utilities for data set and experiment managing (Sections A.4.1 and A.4.9);
- an extendable command line front-end (Section A.4.10).

In the following part of this section the main software components are analyzed from the development point of view.

12 GPL v2.0 license: `http://www.gnu.org/licenses/gpl-2.0.html`

A.4.1 *Input trajectories*

Figure 21 depicts the simplified class diagram of interfaces and classes used for representing trajectories.

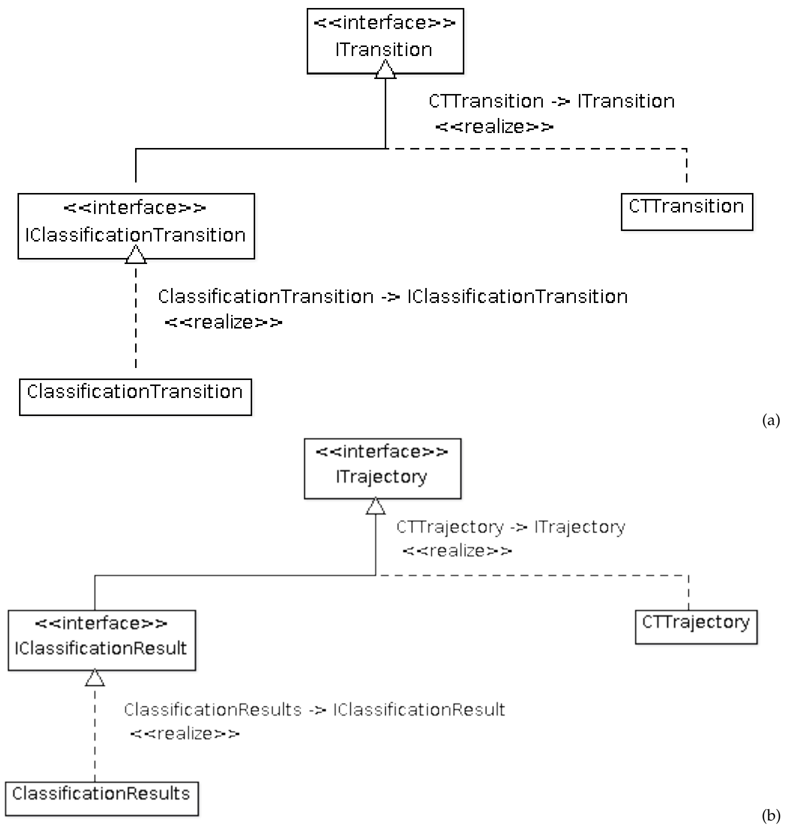


Figure 21.: Simplified class diagram of the trajectory managing components.

`ITrajectory` is the interface that defines the trajectories. Each trajectory consists of a sequence of transitions. Each transition (i.e. `ITransition`) occurs at a particular time. Transition time is generalized using `<TimeType extends Number>` generic. Since CTBNCs model continuous time trajectories, in the stand-alone CTBNCToolkit implementation the `TimeType` generic is filled with `Double` class. Nevertheless, each extension of `Number` and `Comparable` classes can be potentially used as time representation; i.e. the `Integer` class can be specified to represent discrete time trajectories.

`CTTransition` and `CTTrajectory` provide a standard implementation of `ITransition` and `ITrajectory`.

`IClassificationTransition` and `IClassificationResult` interfaces define an extension of `ITransition` and `ITrajectory` which allows to mange classified trajectories. They allow to insert the probability distribution for each transition into a trajectory.

`ClassificationTransition` and `ClassificationResults` implement a standard version of the interfaces for the classified trajectories and transitions.

Example A.4.1. Here is how to generate a trajectory.

```
0.0     Va12    Vb1     Vc123
0.2     Va12    Vb2     Vc123
0.6     Va3      Vb3    Vc123
1.3     Va4      Vb4    Vc4
```

The previous trajectory shows multiple changes of states for each time instant, even if theoretically this cannot happen. This can be implemented by the following code:

```
// Nodes-column name definition
String[] nodeNames = new String[3];
nodeNames[0] = "A"; nodeNames[1] = "B"; nodeNames[2] = "C";
NodeIndexing nodeIndexing = NodeIndexing.getNodeIndexing("
    IndexingName", nodeNames, nodeNames[0], null);

// Time jump definition
String[] v;
List<Double> times = new Vector<Double>();
List<String[]> values = new Vector<String[]>();
times.add(0.0);times.add(0.2);times.add(0.6);times.add(1.3);

// State definition
v = new String[3];
v[0] = "Va12"; v[1] = "Vb1"; v[2] = "Vc123";
values.add(v);
```

```
v = new String[3];
v[0] = "Va12"; v[1] = "Vb2"; v[2] = "Vc123";
values.add(v);
v = new String[3];
v[0] = "Va3"; v[1] = "Vb3"; v[2] = "Vc123";
values.add(v);
v = new String[3];
v[0] = "Va4"; v[1] = "Vb4"; v[2] = "Vc4";
values.add(v);

// Trajectory creation
CTTrajectory<Double> tr = new CTTrajectory<Double>(
    nodeIndexing, times, values);
```

Section A.4.2 provides a clarification about `NodeIndexing` class.

The `CTBNCTestFactory` class, a class used in test management (see Section A.4.9), provides a set of `static` methods which are useful for dealing with data sets:

- `loadDataset`: loads a data set;
- `partitionDataset`: partitions a data set in accordance with a partitioning file (i.e. cross-validation folding, Section A.1.2.15);
- `loadResultsDataset`: loads a data set composed of classified trajectories; for each transition the probability of the class has to be specified;
- `partitionResultDataset`: partitions a data set of results in accordance with a partitioning file (i.e. cross-validation folding);
- `permuteDataset`: randomly permutes a data set;
- `cutDataset`: reduces the original data set in terms of number and length of trajectories (Section A.1.2.17).

A.4.2 *Global node indexing*

Before introducing the models in section A.4.3, it is necessary to deal with the global node indexing system. Models nodes, variables, and columns in a trajectory use the same names. Indeed, objects like model nodes or state values in a trajectory have to be stored by using the same names. To allow a really efficient method to recover these objects the `NodeIndexing` class is developed. A model node can be recovered using its name, which corresponds to a column name in the trajectories, or using its index. Recovering objects by name uses a tree structure. This is efficient, but it is possible to improve it. The index solution is developed to be an $O(1)$ entity recovery system.

To guarantee the same indexing for all the trajectory columns and all the model nodes, the class `NodeIndexing` is used. The idea is that each trajectory and each model has to be synchronized with the same `NodeIndexing` instance. To ensure this, `NodeIndexing` provides a static method (i.e. `getNodeIndexing`) which allows to obtain or create a `NodeIndexing` instance using an unique name. This name usually corresponds to the test name.

Example A.4.2. Here is an example of how to create a new `NodeIndexing` class.

```
String[] nodeNames = new String[3]; nodeNames[0] = "class";
nodeNames[1] = "N1"; nodeNames[2] = "N2";
NodeIndexing nodeIndexing = NodeIndexing.getNodeIndexing("
    IndexingName", nodeNames, nodeNames[0], null);
```

The first argument is the unique key associated with the index. Using the same key in successive calls of the `getNodeIndexing` method allows to recover the same indexing instance. The second argument (i.e. `nodeNames`) is the array of all the node names, while the third argument is the class node name. The last argument is the set of all the node names in the `nodeNames` array to which an index will be associated. This argument allows to select just a subset of the names specified in the second argument. If it is left to `null`, all the names are loaded.

When two objects are synchronized with the same `NodeIndexing` instance, they use the same loaded indexing. This allows the direct communication index-to-index between all the objects with the same synchronization.

A.4.3 *CTBNCToolkit models*

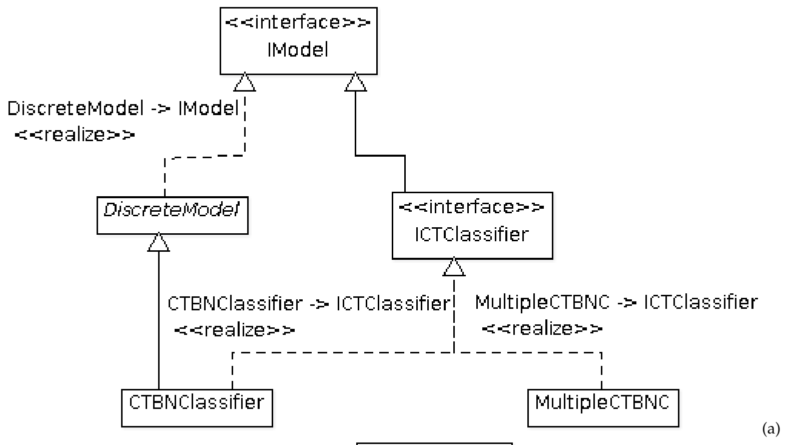
Figure 22 depicts the simplified class diagram of the interfaces and classes to be used to work with the models.

A.4.3.1 *Nodes*

A model consists of nodes which represent the model variables. Node objects are shown in Figure 22b. `INode` interface defines the generic properties of a node, while `IDiscreteNode` interface defines the properties of a discrete state space node.

`Node` class is an abstract class which implements the properties that all the nodes require. Those properties are mainly related to the naming and the dependency relations between nodes. `DiscreteNode` class implements all the requirements of a discrete node. It does not specify any quantitative component as Conditional Intensity Matrix (CIM) or Conditional Probability Tables (CPT), but only manages the discrete states of a variable.

`CTDiscreteNode` class implements all the properties which the CTBNC nodes require. It allows the CIMs definition and can implement both a continuous time node and a static node for continuous time models (i.e. the class node).



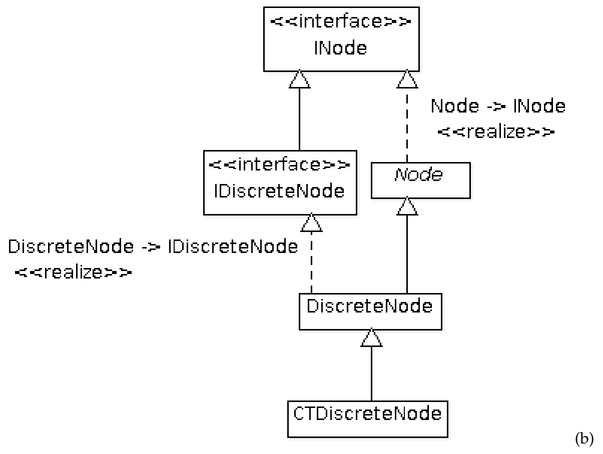


Figure 22.: Simplified class diagram of the model managing components.

Example A.4.3. Here is an example of how to create a continuous time discrete node (i.e. `CTDiscreteNode`).

```
// State definition
Set<String> states2;Set<String> states3;
states2 = new TreeSet<String>();
states2.add("n1_1");states2.add("n1_2");
states3 = new TreeSet<String>();
states3.add("n1_1");states3.add("n1_2");states3.add("n1_3");

// Node creation
CTDiscreteNode node = new CTDiscreteNode("node",states3,
    false);
```

```
CTDiscreteNode parent = new CTDiscreteNode("parent",states2,
    true);
parent.addChild(node);

// CIM definition
double[][] cim = new double[3][3];
cim[0][0] = -2; cim[0][1] = 1; cim[0][2] = 1;
cim[1][0] = 2; cim[1][1] = -4; cim[1][2] = 2;
cim[2][0] = 2; cim[2][1] = 1; cim[2][2] = -3;
node.setCIM(0, cim);
node.setCIM(1, cim);
assertTrue( node.checkCIMs() == -1);
```

A.4.3.2 *Models*

Generic model properties are defined by the `IModel` interface. Part of these properties are implemented by the `DiscreteModel` abstract class. Instead, the `ICTClassifier` interface defines the properties related to the classification process, which a continuous time classifier has to satisfy.

CTBNCs are implemented by `CTBNClassifier` class. While the `MultipleCTBNC` class implements a continuous time classifier composed of a set of binary CTBNCs, each one is specialized to recognize a class against the others (see Section A.1.2.6).

Models are simple Java classes used like containers of nodes. The real complexity related to the learning and the classification processes relies on the algorithm classes.

Example A.4.4. Here is an example that shows how to create a `CTBNClassifier`.

```
// Node names and indexing definition
String[] nodesNames = new String[4];
nodesNames[0] = "Class"; nodesNames[1] = "A";
nodesNames[2] = "B"; nodesNames[3] = "C";
```

```
NodeIndexing nodeIndexing = NodeIndexing.getNodeIndexing("
    IndexingName", nodesNames, nodesNames[0], null);

// State generation
CTDiscreteNode classNode, aNode, bNode, cNode;
Set<String> states2 = new TreeSet<String>();
Set<String> states3 = new TreeSet<String>();
Set<CTDiscreteNode> nodes = new TreeSet<CTDiscreteNode>();
states2.add("s1");states2.add("s2");
states3.add("s1");states3.add("s2");states3.add("s3");

// Node generation
nodes.add(classNode = new CTDiscreteNode(nodesNames[0],
    states2, true));
nodes.add(aNode = new CTDiscreteNode(nodesNames[1], states2,
     false));
nodes.add(bNode = new CTDiscreteNode(nodesNames[2], states3,
     false));
nodes.add(cNode = new CTDiscreteNode(nodesNames[3], states3,
     false));

// Model structure definition
classNode.addChild(aNode);
classNode.addChild(bNode);
classNode.addChild(cNode);

// Node CIM definition
double[][] cim;
cim = new double[1][2]; cim[0][0] = 0.5; cim[0][1] = 0.5;
classNode.setCIM(0, cim);
assertTrue(classNode.checkCIMs() == -1);

cim = new double[2][2];
cim[0][0] = -0.1; cim[0][1] = 0.1;
```

```
cim[1][0] = 0.1; cim[1][1] = -0.1;
aNode.setCIM(0, cim);
cim = new double[2][2];
cim[0][0] = -5; cim[0][1] = 5;
cim[1][0] = 5; cim[1][1] = -5;
aNode.setCIM(1, cim);
assertTrue(aNode.checkCIMs() == -1);

cim = new double[3][3];
cim[0][0] = -0.7; cim[0][1] = 0.5; cim[0][2] = 0.2;
cim[1][0] = 1.0; cim[1][1] = -1.6; cim[1][2] = 0.6;
cim[2][0] = 2; cim[2][1] = 1.3; cim[2][2] = -3.3;
bNode.setCIM(0, cim);cNode.setCIM(0, cim);
cim = new double[3][3];
cim[2][2] = -0.7; cim[2][1] = 0.5; cim[2][0] = 0.2;
cim[1][0] = 1.0; cim[1][1] = -1.6; cim[1][2] = 0.6;
cim[0][2] = 2; cim[0][1] = 1.3; cim[0][0] = -3.3;
bNode.setCIM(1, cim); cNode.setCIM(1, cim);
assertTrue(bNode.checkCIMs() == -1);
assertTrue(cNode.checkCIMs() == -1);

// Model generation
CTBNClassifier model = new CTBNClassifier(nodeIndexing, "
    classifier", nodes);
```

A.4.3.3 *.ctbn format*

.ctbn is the space-separated text format in which the CTBNC-Toolkit saves the learned models. The toString() method in the CTBNClassifier class is the method which provides the model text representation.

.ctbn format starts with the following lines:

```
-----------------------
BAYESIAN NETWORK
```

```
-----------------------
BBNodes N
-----------------------
```

where `N` is the number of nodes in the CTBNC (class node included).

Then each node is specified with the couple `<node_name states_number>` as follows:

```
Class   4
N01     2
N02     2
N03     2
...
N15     4
-----------------------
```

where data in the same line are separated by spaces or tabs, and the first node is the class.

The next lines in the `.ctbn` format define the Bayesian Network (BN) structure, which represents the initial probability distribution of the CTBNC. Each line starts with the considered node and then lists all its parents, using the space-separator format. Each line has to stop with a zero. Here is an example:

```
Class   0
N01     Class   0
N02     Class   N01     0
N03     Class   0
...
N15     Class   N07     0
-----------------------
```

The `.ctbn` format allows to define the initial probability distribution by using a Bayesian Network. On the contrary, the CTBNCToolkit does not support an initial probability distribution yet, but assumes an initial uniform distribution between the variables states. The only exception is for the class which has its own probability distribution, as specified later. For this reason, for the moment the initial distribution is represented as a disconnected Bayesian network.

```
Class 0
N01 0
N02 0
N03 0
...
N15 0
----------------------
```

Note that lines composed of minus signs separate the different file sections.

The next lines inform about the conditional probability distributions of the defined BN. Each CPT is written following the ordering of the parent nodes defined in the previous section. The CPTs are separated by a line composed by minus signs. Here is an example in case of uniform distribution for a disconnected BN:

```
Class
0 0 0 0
----------------------
N01
0.5 0.5
----------------------
...
----------------------
N15
0.25 0.25 0.25 0.25
----------------------
```

where the line of the class probability distribution contains zeros, due to the fact that the class prior will be defined in the next section of the file format. Since CTBNCToolkit currently supports only an initial uniform distribution, the CPTs shown are straightforward. Nevertheless, the `.ctbn` format supports complex CPTs, where each line is the probability distribution setting the value of the parent set. Considering the definition of the parent set of node `N02`:

```
N02     Class   N01     0
```

where `Class` node is binary, node `N01` is ternary, and `N02` has 4 states. Node `N02` CPTs can be defined by the probability distribution lines in the following order:

Class	N01	probability distribution line
0	0	`0.2 0.3 0.1 0.4`
1	0	`0.15 0.2 0.35 0.3`
0	1	`0.42 0.08 0.23 0.27`
1	1	`0.15 0.2 0.35 0.3`
0	2	`0.2 0.3 0.1 0.4`
1	2	`0.23 0.5 0.15 0.12`

With the previous lines we defined the model variables and then the initial probability distribution. In the next part of the file format the temporal model is defined.

First the graph structure is defined in the same way as previously done for the BN:

```
----------------------
DIRECTED GRAPH
----------------------
Class   0
N01     Class   N05     0
N02     Class   N01     0
N03     Class   N01     0
...
```

```
N15     Class   N11     0
-----------------------
```

For each node its parents are specified.

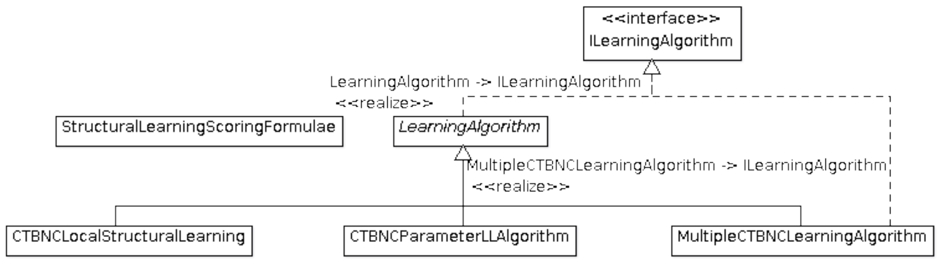
Then the CIMs are defined for each node. Each line represents a complete CIM given the parent set instantiation. The parent sets are defined iterating the parent values starting from the left as previously shown for the Bayesian Network CPTs.

```
-----------------------
CIMS
-----------------------
Class
0.2323008849557522 0.2488938053097345 0.2610619469026549
    0.2577433628318584
-----------------------
N01
-1.7437542178131549 1.7437542178131549 1.6677084125959616
    -1.6677084125959616
-1.1359184948768346 1.1359184948768346 1.468624833995604
    -1.468624833995604
-1.655441390834388 1.655441390834388 1.9083015334745217
    -1.9083015334745217
-1.992223211031885 1.992223211031885 1.128540290987483
    -1.128540290987483
-1.4597541436405608 1.4597541436405608 1.3875353532121044
    -1.3875353532121044
-1.8514888977211081 1.8514888977211081 1.2036550623637277
    -1.2036550623637277
-1.5536818371118852 1.5536818371118852 1.06784175008451
    -1.06784175008451
-1.2675007732387165 1.2675007732387165 1.8220901564788115
    -1.8220901564788115
-----------------------
```

It is worthwhile to note that for the class node the prior probability distribution is specified and not a temporal model since the class is a static node.

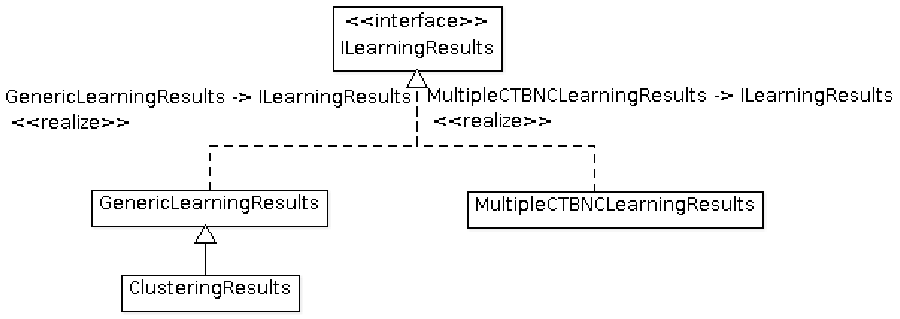
A.4.4 *Learning algorithms*

Figures 23 and 24 depict the simplified class diagram of the components used to provide learning algorithms.



<<interface>>
ILearningAlgorithm
LearningAlgorithm -> ILearningAlgorithm
<<realize>>
StructuralLearningScoringFormulae
LearningAlgorithm
MultipleCTBNCLearningAlgorithm -> ILearningAlgorithm
<<realize>>
CTBNCLocalStructuralLearning
CTBNCParameterLLAlgorithm
MultipleCTBNCLearningAlgorithm

Figure 23.: Learning algorithms simplified class diagram.



<<interface>>
ILearningResults
GenericLearningResults -> ILearningResults
<<realize>>
MultipleCTBNCLearningResults -> ILearningResults
<<realize>>
GenericLearningResults
MultipleCTBNCLearningResults
ClusteringResults

Figure 24.: Simplified class diagram of the learning result components.

Learning algorithms are modeled by the `ILearningAlgorithm` interface, which defines the parameter management and the learning methods.

`LearningAlgorithm` is an abstract class which provides an initial implementation of the parameter managing methods. While `CTBNCParameterLLAlgorithm`, `CTBNCLocalStructuralLearning`, and `MultipleCTBNCLearningAlgorithm` are the learning algorithm implementations.

`CTBNCParameterLLAlgorithm` implements the parameter learning Stella and Amer (2012). This algorithm has to define the CTBNC structure that specifies the dependencies between the variables in advance. The parameters learning algorithm is the basis for all the structural learning algorithms.

`CTBNCLocalStructuralLearning` implements the structural learning for CTBNCs using the local search (Nodelman et al., 2002b; Codecasa and Stella, 2013). The searching algorithm relies on the optimization classes shown in Figure 25.

`MultipleCTBNCLearningAlgorithm` is the implementation of the local searching algorithm for `MultipleCTBNC` models (see Section A.1.2.6). Its implementation is strictly related to the classical structural learning.

All the learning algorithms have to return the results of the learning process implemented by the `ILearningResults` interface.

`ILearningResults` defines the learning results as a container for the sufficient statistics (i.e. `SufficientStatistics` class).

`GenericLearningResults` and `MultipleCTBNCLearningResults` provide a standard implementation of learning results, respectively for CTBNCs and for `MultipleCTBNC` models. `ClusteringResults` extends the `GenericLearningResults` class to implement the learning results for the clustering algorithm (see Section A.4.5).

Figure 25 shows the simplified class diagram of the classes used for the local search in structural learning.

CTBNCs structural learning can be seen as an optimization problem. The CTBNCToolkit provides two scoring functions defined as `static` methods in `StructuralLearningScoringFormulae` class. The scoring functions rely on Marginal Log-likelihood

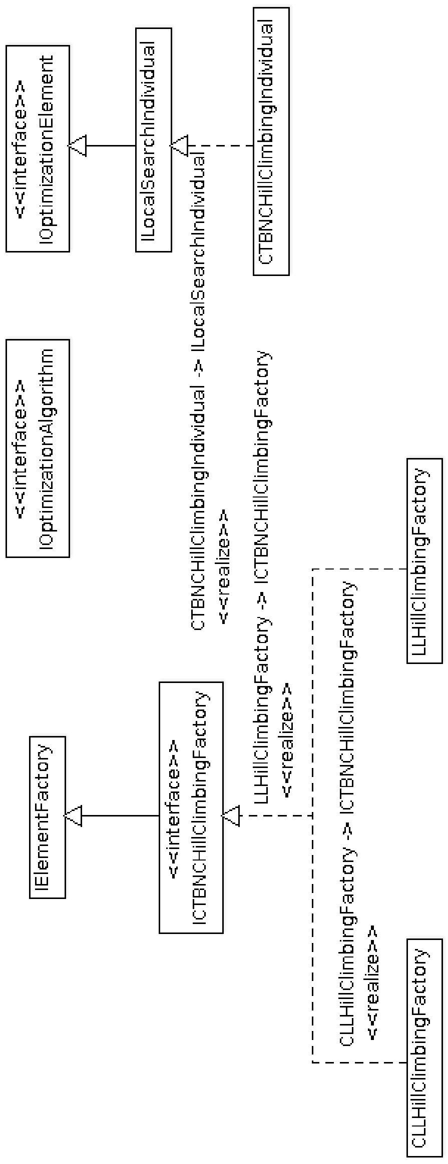

<<interface>>
IOptimizationElement
ILocalSearchIndividual
CTBNCHillClimbingIndividual
<<interface>>
IOptimizationAlgorithm
CTBNCHillClimbingIndividual -> ILocalSearchIndividual
<<realize>>
IElementFactory
<<interface>>
ICTBNCHillClimbingFactory
LLHillClimbingFactory -> ICTBNCHillClimbingFactory
<<realize>>
LLHillClimbingFactory
CLLHillClimbingFactory -> ICTBNCHillClimbingFactory
<<realize>>
CLLHillClimbingFactory

Figure 25.: Hill climbing simplified class diagram.

(MLL) (Nodelman et al., 2002b) and Conditional Log-likelihood (CLL) calculation (Codecasa and Stella, 2013).

Since the learning procedure is a maximization problem respect to a selected scoring function, the `IOptimizationElement` interface is defined. The interface provides a simple definition of an element that must be evaluated for optimization purposes. The `ILocalSearchIndividual` interface extends the `IOptimizationElement` interface, defining the properties of a local search element such as the ability to find the best neighbor (i.e. `getBestNeighbor` method).

`CTBNCHillClimbingIndividual` is the actual implementation of an individual in the local search procedure. It is the core of the local search algorithm implementation.

To generalize the local search algorithm, implemented in the `CTBNCLocalStructuralLearning` class, the factory pattern is used to generate local search individuals.

`IElementFactory` is the interface that defines the properties of a factory of `IOptimizationElement`. The use of Java generics helps the code generalization. The `ICTBNCHillClimbingFactory` interface extends the `IElementFactory` interface to define the requirements for generating `CTBNCHillClimbingIndividual`.

`LLHillClimbingFactory` and `CLLHillClimbingFactory` are the two factories that generate `CTBNCHillClimbingIndividual`. The first factory generates the individuals for the Marginal Log-likelihood score maximization, while the second factory generates the individuals for the Conditional Log-likelihood score maximization.

Example A.4.5. Here is an example of parameter learning.

```
// Model definition
CTBNClassifier clModel = new CTBNClassifier(nodeIndexing, "
    classifier", nodes);

// Structure definition
boolean[][] adjMatrix = new boolean[4][4];
for(int i = 0; i < adjMatrix.length; ++i)
        for(int j = 0; j < adjMatrix[i].length; ++j)
                adjMatrix[i][j] = false;
adjMatrix[0][1] = true;adjMatrix[0][2] = true;
adjMatrix[0][3] = true;

// Learning algorithm instantiation
```

```
CTBNCParameterLLAlgorithm  alg = new
    CTBNCParameterLLAlgorithm();

// Learning algorithm parameter (i.e. priors) definition
Map<String,Object> params = new TreeMap<String,Object>();
params.put("Mxx_prior", 1.0);
params.put("Tx_prior", 0.01);
params.put("Px_prior", 1.0);

// Learning algorithm parameters and structure setting
alg.setParameters(params);
alg.setStructure(adjMatrix);

// Parameter learning
Collection<ITrajectory<Double>> dataset = generateDataset();
alg.learn(clModel, dataset);
```

Example A.4.6. Here is an example of structural learning using the Marginal Log-likelihood scoring.

```
// Parameter learning algorithm to use in the structural
    learning
CTBNCParameterLLAlgorithm  paramsAlg = new
    CTBNCParameterLLAlgorithm();
Map<String,Object> params = new TreeMap<String,Object>();
params.put("Mxx_prior", 1.0);
params.put("Tx_prior", 0.01);
params.put("Px_prior", 1.0);
paramsAlg.setParameters(params);

// Hill climbing factory definition
LLHillClimbingFactory elemFactory = new
    LLHillClimbingFactory(paramsAlg, 3, false, false);

// Definition of the local search starting structure
```

```
int iClass = nodeIndexing.getClassIndex();
int iA = nodeIndexing.getIndex("A");
int iB = nodeIndexing.getIndex("B");
int iC = nodeIndexing.getIndex("C");
boolean[][] adjMatrix = new boolean[4][4];
for(int i = 0; i < adjMatrix.length; ++i)
        for(int j = 0; j < adjMatrix[i].length; ++j)
                adjMatrix[i][j] = false;
adjMatrix[iClass][iA] = true;adjMatrix[iClass][iB] = true;
adjMatrix[iClass][iC] = true;

// Definition of the structural learning algorithm and
   setting of the initial structure
CTBNCLocalStructuralLearning<String,
   CTBNCHillClimbingIndividual> alg = new
   CTBNCLocalStructuralLearning<String,
   CTBNCHillClimbingIndividual>(elemFactory);
alg.setStructure(adjMatrix);

// Structural learning of a target model
ICTClassifier<Double, CTDiscreteNode> model =
   generateClassifierModel();
alg.learn(model, trainingSet);
boolean[][] learnedStructure = model.getAdjMatrix();
```

A.4.5 *Clustering*

Figure 26 depicts the simplified class diagram of the clustering learning.

`IClusteringAlgorithm` is the interface that defines the clustering learning algorithms. It extends the `ILearningAlgorithm` interface (Section A.4.4).

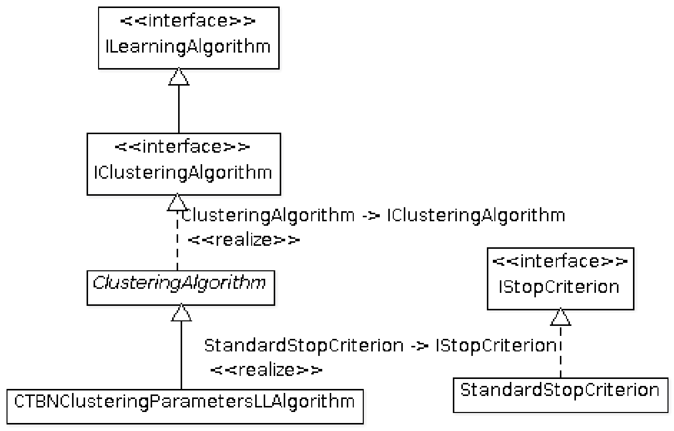


Figure 26.: Simplified class diagram of the clustering learning.

`ClusteringAlgorithm` is an abstract class which implements the basis functionality of the clustering learning algorithm, while `CTBNClusteringParametersLLAlgorithm` implements the soft and hard assignment EM algorithms for clustering purposes (see Section 5.1). The structural learning algorithm relies on the optimization algorithms, used also in the case of supervised learning (Section A.4.4).

`IStopCriterion` interface defines the stopping criterion for the EM iterative algorithm, and the `StandardStopCriterion` class provides a basic stop criterion implementation.

Example A.4.7. Here is an example of soft-assignment clustering.

```
// Parameter definition
Map<String, Object> params = new TreeMap<String,Object>();
params.put("Mxx_prior", 1.0);
params.put("Tx_prior", 0.005);
params.put("Px_prior", 1.0);
params.put("hardClustering", false);
StandardStopCriterion stopCriterion = new
    StandardStopCriterion(iterationNumber, 0.1);


// Classification algorithm definition
```

```
Map<String, Object> paramsClassifyAlg = new TreeMap<String,
    Object>();
paramsClassifyAlg.put("probabilities", true);
CTBNCClassifyAlgorithm classificationAlg = new
    CTBNCClassifyAlgorithm();
classificationAlg.setParameters(paramsClassifyAlg);

// Clustering algorithm initialization
CTBNClusteringParametersLLAlgorithm cAlg = new
    CTBNClusteringParametersLLAlgorithm();
cAlg.setParameters(params);
cAlg.setClassificationAlgorithm(classificationAlg);
cAlg.setStopCriterion( stopCriterion);

// Clustering learning
ClusteringResults<Double> results = cAlg.learn(model,
    dataSet);
```

A.4.6 *Inference algorithms*

Figure 27 depicts the simplified class diagram of the inference algorithm classes.

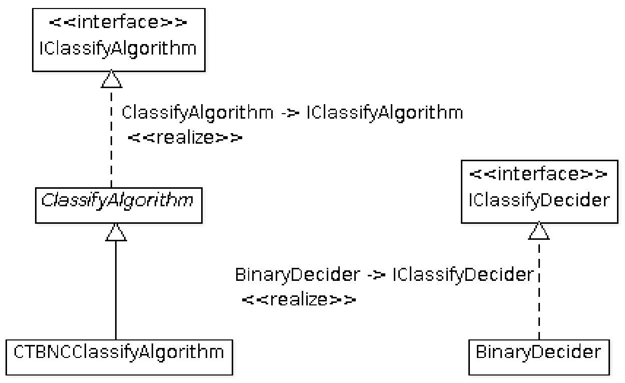


Figure 27.: Simplified class diagram of inference components.

`IClassifyAlgorithm` is the interface that defines the properties of an inference algorithm. It mainly defines properties related to parameter managing and classification methods. Classification methods return `IClassificationResult`. As for the learning algorithms (Section A.4.4), an abstract class is developed to manage the parameters, i.e. `ClassifyAlgorithm`. Generics are used to generalize the time representation and the nodes in the model to classify.

`CTBNCClassifyAlgorithm` implements the classification algorithm for CTBNCs, as described by Stella and Amer (2012).

Example A.4.8. Here is an example of classification inference.

```
// Generate the trajectory to classify
ITrajectory<Double> testTrajectory = new CTTrajectory<Double
    >(nodeIndexing, times, values);

// Classification
CTBNCClassifyAlgorithm clAlgorithm = new
    CTBNCClassifyAlgorithm();
IClassificationResult<Double> result =  clAlgorithm.
    classify(learnedModel, testTrajectory, 0.9);
```

Usually the classification relies on the most probable class, but defining a `IClassifyDecider` it is possible to use another classification criteria. `BinaryDecider` implements the `IClassifyDecider` interface to allow an unbalanced classification in the case of two class problems. The decider allows to define a threshold to use in the classification in order to give an advantage to one of the two classes.

A.4.7 *Validation methods*

Figure 28 depicts the simplified class diagram of the validation methods used to realize tests and to calculate performances.

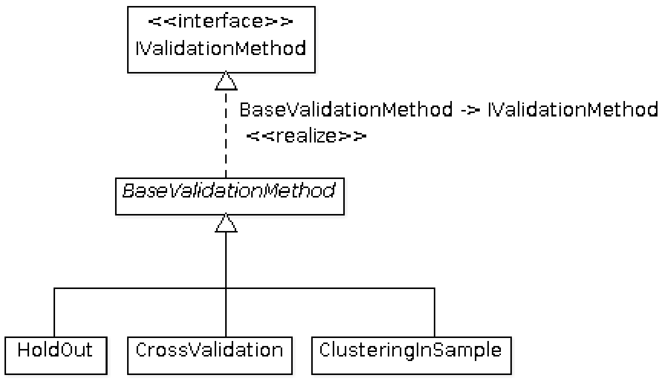


Figure 28.: Simplified class diagram of the validation methods.

To execute the tests over a data set, validation methods are implemented. `IValidationMethod` is the interface that defines the validation method properties. `BaseValidationMethod` is the abstract class that implements the base functions of the validation methods.

`HoldOut`, `CrossValidation` and `ClusteringInSample` are the classes that implement different validation methods (Section A.1.2.4). Each validation approach, one executed using `validate` method, returns the performances of the tested algorithm (Section A.4.8).

`HoldOut` class implements the hold out validation, where training set and test set are used to calculate the performances. `CrossValidation` class implements the cross validation method (Witten and Frank, 2005). `ClusteringInSample` class implements a validation method, where learning and testing are both realized over the same complete data set; this can be used to test clustering approaches (Section A.1.2.5).

Example A.4.9. Here is an example of classification inference.

```
// Instantiation of the cross validation method
CrossValidation<Double,CTDiscreteNode,CTBNClassifier,
    MicroMacroClassificationPerformances<Double,
    ClassificationStandardPerformances<Double>>,
    ClassificationStandardPerformances<Double>>
    validationMethod = new CrossValidation<...>(
    performanceFactory, 10, true);
validationMethod.setVerbose(true);

// Test execution using a validation method
performances = validationMethod.validate( model,
    learningAlgorithm, inferenceAlgorithm, dataset);
```

`performanceFactory` is a factory to generate new test performances. For more details about performances see Section A.4.8.

A.4.8 *Performances*

The CTBNCToolkit provides a rich set of performances to evaluate the experiments. Different performances are provided in the case of classification (Section A.2.1) and clustering (Section A.2.2). The simplified class diagrams of the classification and the clustering performances are depicted in Figures 29 and 30. In both cases the class hierarchy is the same. Different classes are provided to calculate single run and aggregate performances (i.e. for cross-validation multiple runs). To allow the best possible generalization, factory classes are provided to generate the performances. The simplified class diagram of the performance factories is depicted in Figure 31. A factory argument is required in all the validation methods in order to generate the performances during the tests (see Section A.4.7).

`IPerformances` is the interface which defines a generic performance, while `ISingleRunPerformances` and `IAggregatePerformances` respectively define the single run and the aggregate performances.

In the case of classification the `IClassificationPerformances` interface is provided. In the case of clustering the performances

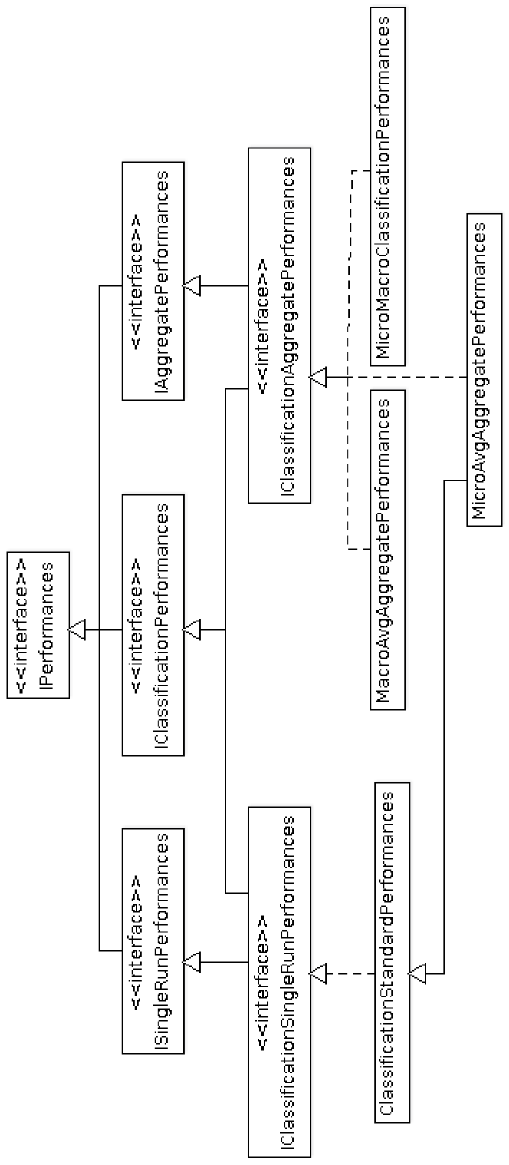

<<interface>>
IPerformances
<<interface>>
IAggregatePerformances
<<interface>>
IClassificationPerformances
<<interface>>
ISingleRunPerformances
<<interface>>
IClassificationAggregatePerformances
<<interface>>
IClassificationSingleRunPerformances
MicroMacroClassificationPerformances
MacroAvgAggregatePerformances
MicroAvgAggregatePerformances
ClassificationStandardPerformances

Figure 29.: Simplified class diagram of the supervised classification performances.

definitions relies on `IExternalClusteringPerformances`; only external clustering measures are provided (see Section 5.2.2).

`IClassificationSingleRunPerformances` is the interface which defines the single run classification performances, while the `IClassificationAggregatePerformances` interface defines the aggregate classification performances. Similarly, the clustering performances are de-

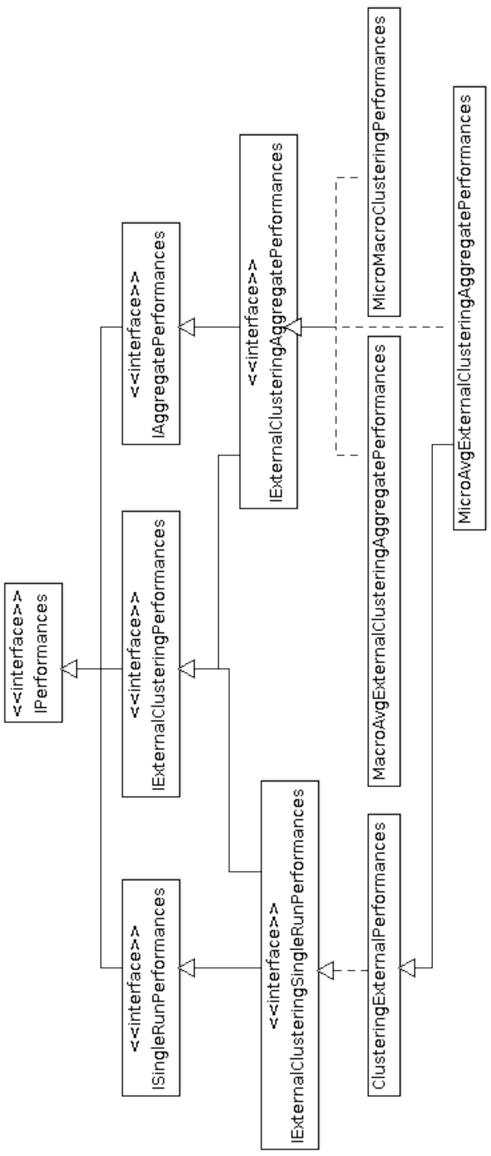

<<interface>>
IPerformances
<<interface>>
IAggregatePerformances
<<interface>>
IExternalClusteringPerformances
<<interface>>
ISingleRunPerformances
<<interface>>
IExternalClusteringAggregatePerformances
<<interface>>
IExternalClusteringSingleRunPerformances
MicroMacroClusteringPerformances
MacroAvgExternalClusteringAggregatePerformances
ClusteringExternalPerformances
MicroAvgExternalClusteringAggregatePerformances

Figure 30.: Simplified class diagram of the clustering performances.

fined by `IExternalClusteringSingleRunPerformances` and `IExternalClusteringAggregatePerformances` interfaces.

The `ClassificationStandardPerformances` class, together with the `ClusteringExternalPerformances` class, implements the performances for single runs in the case of classification and clustering. Aggregate performances are provided in micro and macro averaging. Micro averaging performances are calculated extending the single run performances. `MicroAvgAggregatePerformances` is the class that implements the micro averaging in the case of classification, while `MicroAvgExternalClusteringAggregatePerformances` implements the micro averaging performances in the case of clustering.

Macro averaging classification performances are provided by the `MacroAvgAggregatePerformances` class. Macro averaging performances for unsupervised learning (i.e. clustering) are provided by the `MarcoAvgExternalClusteringAggregatePerformances` class.

In order to have the possibility to calculate both micro and macro averaging performances, two classes that hide both the averaging approaches are provided in the case of classification (i.e. `MicroMacroClassificationPerformances`) and in the case of clustering (i.e. `MicroMacroClusteringPerformances`).

Figure 31 depicts the simplified class diagram of the performance factory classes.

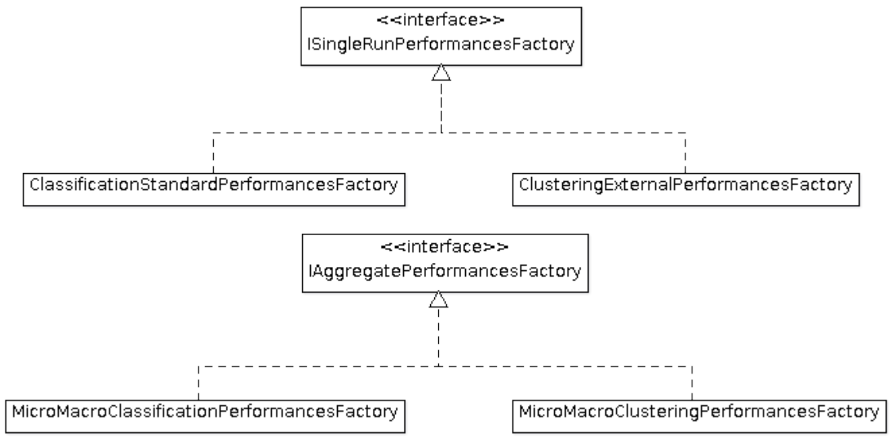

<<interface>>
ISingleRunPerformancesFactory
ClassificationStandardPerformancesFactory
ClusteringExternalPerformancesFactory
<<interface>>
IAggregatePerformancesFactory
MicroMacroClassificationPerformancesFactory
MicroMacroClusteringPerformancesFactory

Figure 31.: Simplified class diagram of the performance factories.

The `ISingleRunPerformancesFactory` interface defines the factory for single run performances. The factory for aggregate performances is defined by the `IAggregatePerformancesFactory` interface. In the case of classification and in the case of clustering two factories are provided: one for the single run performances and one for the combination of micro and macro averaging aggregate performances. In the case of classification `ClassificationStandardPerformancesFactory` is provided for single runs, and `MicroMacroClassificationPerformancesFactory` is provided for aggregate performances. In the case of clustering the provided classes are: `ClusteringExternalPerformancesFactory` and `MicroMacroClusteringPerformancesFactory`.

A.4.9 *Tests*

To perform the test experiments a set of utilities have been developed. Figure 32 depicts the simplified class diagram of these utilities.

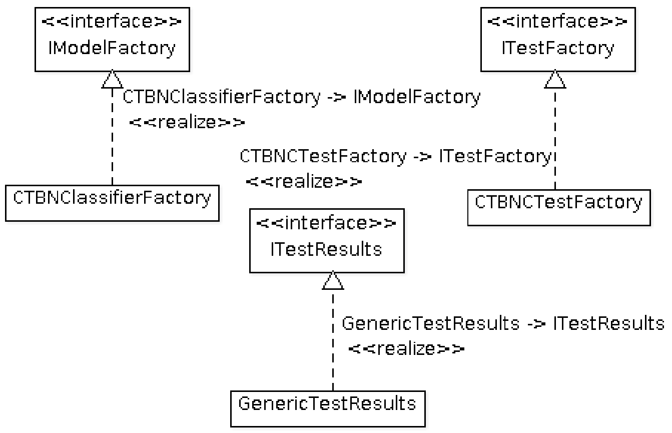


Figure 32.: Simplified class diagram of the classes used to test CTBNCs.

`IModelFactory` interface defines a factory to generate new models. `CTBNClassifierFactory` implements a simple way to generate synthetic CTBNCs. This class has been used to generate the models tested in the synthetic test campaigns.

Example A.4.10. Here is an example of factory for the CTNB models.

```
// Number of variables
int N = 16;
```

```
// Number of states for each variable
int[] nStates = new int[N];
nStates[0] = 10;                // class variable
nStates[1] = 2;  ...;  nStates[15] = 4;

// Range in which sample the lambda values of the
   exponential distribution
double[][] lambdaRanges = new double[2][N];
lambdaRanges[0][1] = 10; lambdaRanges[1][1] = 20;
...
lambdaRanges[0][15] = 40; lambdaRanges[1][15] = 80;

// Initialize the factory
CTBNClassifierFactory modelFactory = CTBNClassifierFactory(
   "naiveBayes", nStates, lambdaRanges);

// Generate a model
CTBNClassifier model = modelFactory.newInstance();
```

ITestFactory is an interface which defines a factory for tests. The idea is to provide a factory instance with the parameters of the tests, for example the learning and inference algorithms, and then to run a test calling a method. CTBNCTestFactory implements a test factory for CTBNCs. It provides the possibility to use a model factory to generate and test different data sets created from different model instances, and it provides the possibility to execute the tests over a data set in input.

In both cases GenericTestResults class is returned. This class implements a test result defined by the ITestResults interface and contains the performances of the tests.

Example A.4.11. Here is an example of CTBNCTestFactory use. Generics are widely used to have a complete generalization.

```
// Factory instantiation
```

```
CTBNCTestFactory<Double,CTDiscreteNode,CTBNClassifier,
    MicroMacroClassificationPerformances<Double,
    ClassificationStandardPerformances<Double>>> testFactory
    ;
testFactory = new CTBNCTestFactory<Double,CTDiscreteNode,
    CTBNClassifier,MicroMacroClassificationPerformances<
    Double,ClassificationStandardPerformances<Double>>>(
    modelFactory, validationMethod, nbParamsLearningAlg,
    classificationAlg);

// Test execution
GenericTestResults<Double,CTDiscreteNode,CTBNClassifier,
    MicroMacroClassificationPerformances<Double,
    ClassificationStandardPerformances<Double>>> resultNB =
    testFactory.newTest("NB", datasetDim);

// Performances recovering
String resultNB.performancesSummary(testName,
    confidenceLevel, datasetDim, kFolds);
```

In addition to the basic methods, `GenericTestResults` and `CTBNCTestFactory` classes provide a set of general utilities to manage the tests. `GenericTestResults` provides a set of `static` methods to print the classification and clustering performances. `CTBNCTestFactory` provides a set of `static` methods to load and to manage the input data sets (see Section A.4.1).

A.4.10 *Command line front-end*

`CTBNCToolkit.frontend` is the package that contains the front-end. Currently, only a command line front-end is developed. The package consist of two classes: `Main` and `CommandLine`. `Main` class is due to start the command line front-end implemented by the `CommandLine` class.

A.4.10.1 *Command line parameters*

The `CommandLine` class was developed to easily add new command line parameters (i.e. modifiers).

The command line parameters use an array of `Strings` that gathers the information necessary to manage the modifiers (i.e. `modifiersList`).

```
private String[][] modifiersList = {
   {"help", "printHelp", "print the CTBNCToolkit help", "
       enabled"},
   ... ,
   {"validation", "setValidationMethod", "specify the
       validation method:\n" +
      "\t—validation=CV,k  \tk–folds cross validation is
         used\n" +
      "\t—validation=HO,0.6\thold out is used", "enabled"},
   ... ,
   {"v", "setVerbose", "enable the verbose comments", "
       enabled"}
};
```

Each line consists of four columns:

i. name of the modifier, i.e. `"help"` is the modifier name that can be called from the command line writing `--help`;

ii. name of the function called to manage the modifier, i.e. when the `--help` modifier is inserted, the `printHelp()` function is called;

iii. modifier description, automatically printed when the `--help` modifier is inserted;

iv. flag to enable or not the modifier: if the fourth column contains `"enabled"` the modifier is enabled, otherwise it is completely ignored, even in the printing of the help.

When the command line is started, the input parameters are analyzed in order to call the appropriate methods. It is possible to define a method without parameters, i.e. `printHelp()`:

```
public void printHelp() {
   System.out.println("CTBNCToolkit for classification");
   ....
}
```

or a method with parameters, i.e. `setValidationMethod( vMethod )`:

```
public void setValidationMethod( LinkedList<String> vMethod)
     {
   ....
}
```

The inputs are passed to the method through a linked list automatically generated from the command line parameters. The command line parameters with arguments have to be in the following form:

```
--modifier=arg1,arg2,..,argN
```

this generates a linked list of the following `Strings`: `"arg1"`, `"arg2"`, ..., `"argN"`. Using the example of the validation method (A.1.2.4): the command line input `--validation=HO,0.6` generates a linked list where `"HO"` is the first argument and `"0.6"` is the second.

Using the `modifiersList` and the automatic managing of the modifier parameters it is possible to add a new modifier just adding in the code a line in the `modifiersList` and the corresponding method, which can have in input any number of arguments.

A.4.10.2 *Program execution*

The command line parameters are used to set the parameters required during the program execution, while the real CTBNC-Toolkit starter relies on the `CommandLine()` constructor.

The constructor first initializes the modifier methods using `Java` reflexivity. This is done in the `initModifiers()` method. It then analyzes the command line parameters, calling the right methods and checking the compatibilities between the parameters.

Once the modifiers are managed, the CTBNCToolkit starts with the following steps:

- data set loading and generation of a CTNB, provided by the `loadDatasets(..)` method (see Section A.4.1);
- loading of the learning algorithms to test, provided by the `loadLearningAlgorithms(..)` method (see Section A.4.4);
- loading of the classification algorithm, provided by the `loadInferenceAlgorithm()` method (see Section A.4.6);
- test generation, provided by the `generateTestFactories (..)` method (see Section A.4.9);
- test execution, provided by the `executeTests(..)` method (see Section A.4.9).

B

MORE RESULTS

This sections shows some of the test results obtained that for reasons of brevity were not inserted in the main chapters of the dissertation.

B.1 CLASSIFICATION

B.1.1 *DBN synthetic data sets*

Test	CTNB	$k=2$ ACTNB (MLL)	$k=2$ ACTNB (CLL)	$k=2$ CTBNC (MLL)	$k=2$ CTBNC (CLL)	$k=3$ CTBNC (MLL)	$k=3$ CTBNC (CLL)	$k=4$ CTBNC (MLL)	$k=4$ CTBNC (CLL)	DBN-NB1	DBN-NB2	DBNC1	DBNC2
DBNTest1	**1**	**1**	**1**	**.99**	.37	**.99**	.35	**.99**	.35	**1**	**1**	**1**	**1**
DBNTest2	.86	.91	**.98**	.85	.26	.85	.26	.85	.26	.90	.91	**.95**	.91
DBNTest3	**1**	**1**	**1**	**.98**	.67	**.98**	.67	**.98**	.67	**.98**	**.98**	**.99**	.95
DBNTest4	.82	.82	**.96**	.81	.35	.81	.35	.81	.35	.80	.83	.87	.74
DBNTest5	.64	.64	**.73**	.64	.2	.64	.2	.64	.2	.63	**.68**	.62	.64

Table 11.: Average accuracy value for each DBN test and for each model. The bold characters indicate the best models with 90% confidence. Tests are made using 10 fold cross validation.

B.1.2 *CTBNC synthetic data sets*

		k = 2	k = 2	k = 2	k = 2	k = 3	k = 3	k = 4	k = 4		
Test	CTNB	ACTNB	ACTNB	CTBNC	CTBNC	CTBNC	CTBNC	CTBNC	CTBNC	DBN-	DBN-
		(MLL)	(CLL)	(MLL)	(CLL)	(MLL)	(CLL)	(MLL)	(CLL)	NB1	NB2
CTNB1	**.972**	**.972***	.965	**.968***	.961	**.968***	.913	**.968***	.772	.847	.849
CTNB2	**.973**	**.973***	.961	**.970***	.956	**.970***	.895	**.970***	.779	.858	.866*
CTNB3	**.984**	**.984***	.979	**.983**	.979	**.983***	.920	**.983***	.806	.890	.892
CTNB4	**.906**	**.906***	.854	.824	.845*	.824*	.666	.824*	.394	.697	.704
CTNB5	**.933**	**.933***	.881	.889	.881	.889*	.720	.889*	.466	.712	.725*
k2ACTNB1	.823	.914	**.965***	.796	.960*	.796	.940*	.796	.839*	.666	.666
k2ACTNB2	.804	**.967**	**.968**	.958	**.968***	.958*	.920	.958*	.794	.659	.681*
k2ACTNB3	.589	.654	**.739***	.198	**.739***	.198	.510*	.198	.309*	.408	.418
k2ACTNB4	.814	.937	**.966***	.877	**.966***	.877	.930*	.877*	.844	.669	.674
k2ACTNB5	.846	**.959**	**.960**	**.951**	**.960**	**.951***	.926	**.951***	.799	.675	.694*
k2CTBNC1	.627	.807	**.840***	.781	**.840***	.781	.755	.781*	.592	.492	.508*
k2CTBNC2	.690	.890	**.924***	.873	**.924***	.873*	.852	.873*	.691	.541	.568*
k2CTBNC3	.733	.832*	.809	**.859***	.809	**.859***	.708	**.859***	.557	.544	.559
k2CTBNC4	.757	.858*	.826	**.872***	.826	**.872***	.694	**.872***	.555	.520	.535
k2CTBNC5	.578	.813	**.868***	.823	**.868***	.823*	.777	.823*	.597	.397	.399
k2CTBNC6	.665	.837	**.865***	**.878**	**.865**	**.878***	.762	**.878***	.599	.412	.440*
k3CTBNC1	.508	.670	.701*	.629	.701*	.898	**.923***	.898*	.780	.326	.354*
k3CTBNC2	.565	.732	.735	.730	.735	**.909**	**.927**	**.909***	.795	.373	.415*
k3CTBNC3	.602	.787*	.757	.789*	.757	**.934***	.877	**.934***	.744	.366	.363
k3CTBNC4	.622	.784	.793	.809	.793	**.942***	.915	**.942***	.806	.384	.384
k3CTBNC5	.539	.803*	.674	.808*	.674	**.887***	.805	**.887***	.637	.363	.371
k3CTBNC6	.563	.776	.778	.822*	.778	**.927***	.834	**.927***	.697	.405	.420*
k4CTBNC1	.426	.436	.560*	.236	.560*	.236	**.864***	.236	**.875***	.290	.297
k4CTBNC2	.399	.501	.633*	.462	.633*	.594	.848*	.773	**.876***	.293	.279
k4CTBNC3	.870	.937	.924	.951*	.926	**.984***	.967	**.984***	.887	.445	.435
k4CTBNC4	.885	.952*	.931	.958*	.931	**.994***	.977	**.994***	.901	.644	.641
k4CTBNC5	.584	.718	.830*	.648	.831*	.736	**.975***	.736	.911*	.363	.383
k4CTBNC6	.643	.873	.869	.886	.869	**.996**	**.995**	**.996***	.947	.388	.390

Table 12.: Average accuracy value for each CTBNC test and for each model. Tests are made using 10 fold cross validation. The bold characters indicate the best models with 90% confidence. For the CTBNCs the asterisk indicates which scoring function (i.e. MLL or CLL) is better with 90% confidence, once the structure constraints (i.e. number of parents) are set. For the DBNs the asterisk indicates which DBN is the best with 90% confidence.

Test	data set %	CTNB	k = 2 ACTNB (MLL)	k = 2 ACTNB (CLL)	k = 2 CTBNC (MLL)	k = 2 CTBNC (CLL)	k = 3 CTBNC (MLL)	k = 3 CTBNC (CLL)	k = 4 CTBNC (MLL)	k = 4 CTBNC (CLL)
CTNB	100%	**.954**	**.954**	.928	.927	.924	.927	.823	.927	.643
	80%	**.918**	**.918**	.859	.725	.857	.725	.718	.725	.550
	60%	**.847**	**.847**	.758	.596	.758	.596	.586	.596	.441
	40%	**.703**	**.703**	.566	.184	.561	.184	.406	.184	.333
	20%	**.444**	**.444**	.337	.187	.340	.187	.287	.187	.242
K2ACTNB	100%	.775	.886	**.920**	.756	**.919**	.756	.845	.756	.717
	80%	.722	.793	**.826**	.568	**.818**	.568	.729	.568	.582
	60%	.642	.662	**.697**	.336	**.695**	.336	.564	.336	.427
	40%	**.515**	**.518**	.497	.247	.488	.247	.373	.247	.297
	20%	**.341**	**.341**	.300	.197	.302	.197	.290	.197	.228
K2CTBNC	100%	.675	.840	**.855**	**.848**	**.855**	**.848**	.758	**.848**	.599
	80%	.619	**.781**	**.784**	**.774**	**.781**	**.774**	.674	**.774**	.519
	60%	.538	.654	**.673**	.567	**.664**	.567	.541	.567	.406
	40%	.478	.514	**.542**	.358	**.538**	.358	.425	.358	.329
	20%	**.329**	**.329**	**.322**	.228	**.312**	.228	.246	.228	.255
K3CTBNC	100%	.486	.650	.634	.655	.634	**.785**	.754	**.785**	.637
	80%	.445	.550	.579	.480	.579	.568	**.669**	.568	.509
	60%	.387	.442	.479	.362	.477	.381	**.527**	.381	.388
	40%	.317	.320	**.362**	.213	.345	.213	.325	.213	.249
	20%	**.272**	**.270**	**.280**	.233	**.267**	.233	.249	.233	.220
K4CTBNC	100%	.635	.736	.791	.690	.792	.757	**.938**	.787	.900
	80%	.589	.679	.727	.642	.727	.726	**.869**	.755	.767
	60%	.542	.601	.648	.559	.648	.592	**.765**	.592	.610
	40%	.461	.465	.493	.360	.480	.360	**.518**	.360	.392
	20%	**.341**	**.341**	.299	.258	.297	.258	.308	.258	.260

Table 13.: Average accuracy value for each test model category and for each CTBNC model, changing the dimension of the data set used. The first column indicates the model category that generates the data sets used in the tests. The second column indicates the percentage of the original data set used (i.e. X% indicates that only X% of the data set trajectories are used, and each trajectory is cut in order to be X% of the original length). Tests are made using 10 fold cross validation. The bold characters indicate the best models with 90% confidence.

B.1.2.1 *CTNB tests*

Test	data set %	CTNB	k = 2 ACTNB (MLL)	k = 2 ACTNB (CLL)	k = 2 CTBNC (MLL)	k = 2 CTBNC (CLL)	k = 3 CTBNC (MLL)	k = 3 CTBNC (CLL)	k = 4 CTBNC (MLL)	k = 4 CTBNC (CLL)
CTNB1	100%	**.972**	**.972***	.965	**.968***	.961	**.968***	.913	**.968***	.772
	80%	**.954**	**.954***	.920	.904	.929*	.904*	.815	.904*	.684
	60%	**.918**	**.918***	.873	.813	.882*	.813*	.722	.813*	.573
	40%	**.82**	**.82***	.7	.255	.683*	.255	.530*	.255	.463*
	20%	**.645**	**.645***	.52	.26	.545*	.26	.465*	.26	.375*
CTNB2	100%	**.973**	**.973***	.961	**.970***	.956	**.970***	.895	**.970***	.779
	80%	**.955**	**.955***	.929	.916	.920	.916*	.818	.916*	.676
	60%	**.897**	**.897***	.837	.733	.835*	.733*	.680	.733*	.538
	40%	**.793**	**.793***	.695	.225	.698*	.225	.528*	.225	.418*
	20%	**.525**	**.525***	.365	.23	.36*	.23	.32*	.23	.265
CTNB3	100%	**.984**	**.984***	.979	**.983**	.979	**.983***	.920	**.983***	.806
	80%	**.96**	**.96***	.936	**.955**	.936	**.955***	.8725	**.955***	.695
	60%	**.918**	**.918***	.860	.878	.860	.878*	.732	.878*	.537
	40%	**.808**	**.808***	.683	.250	.683*	.250	.540*	.250	.458*
	20%	**.59**	**.59***	.49	.275	.49*	.275	.415*	.275	.335
CTNB4	100%	**.906**	**.906***	.854	.824	.845*	.824*	.666	.824*	.394
	80%	**.855**	**.855***	.761	.085	.751*	.085	.518*	.085	.331*
	60%	**.727**	**.727***	.563	.092	.555*	.092	.378*	.092	.232*
	40%	**.523**	**.523***	.378	.080	.368*	.080	.233*	.080	.170*
	20%	**.24**	**.24***	.165	.075*	.16	.075	.105*	.075	.085
CTNB5	100%	**.933**	**.933***	.881	.889	.881	.889*	.720	.889*	.466
	80%	**.868**	**.868***	.751	.765	.751	.765*	.564	.765*	.363
	60%	**.773**	**.773***	.657	.465	.657*	.465*	.420	.465*	.325
	40%	**.570**	**.570***	.373	.110	.373*	.110	.198*	.110	.155
	20%	**.220**	**.220***	.145	.095	.145*	.095	.13	.095	**.150***

Table 14.: Average accuracy value for each CTNB test and for each CTBNC model, changing the dimension of the data set used. The second column indicates the percentage of the original data set used (i.e. X% indicates that only X% of the data set trajectories are used, and each trajectory is cut in order to be X% of the original length). Tests are made using 10 fold cross validation. The bold characters indicate the best models with 90% confidence. The asterisk indicates which scoring function (i.e. MLL or CLL) is better with 90% confidence once set the structure constraints (i.e. number of parents).

B.1.2.2 k2*ACTNB tests*

Test	data set %	CTNB	k = 2 ACTNB (MLL)	k = 2 ACTNB (CLL)	k = 2 CTBNC (MLL)	k = 2 CTBNC (CLL)	k = 3 CTBNC (MLL)	k = 3 CTBNC (CLL)	k = 4 CTBNC (MLL)	k = 4 CTBNC (CLL)
k2ACTNB1	100%	.823	.914	**.965***	.796	.960*	.796	.940*	.796	.839*
	80%	.766	.791	**.926***	.359	.911*	.359	.854*	.359	.673*
	60%	.703	.703	**.858***	.210	**.837***	.210	.728*	.210	.565*
	40%	**.595**	**.595**	**.56**	.275	.553*	.275	.46*	.275	.353*
	20%	**.325**	**.325**	**.34**	.22	**.340***	.22	**.345***	.22	.275
k2ACTNB2	100%	.804	**.967**	**.968**	.958	**.968***	.958*	.920	.958*	.794
	80%	.778	**.908**	**.911**	.849	**.911***	.849	.839	.849*	.683
	60%	.723	.748	**.797***	.480	**.797***	.480	.658*	.480	.470
	40%	**.565**	**.565**	**.573**	.380	**.573***	.380	.390	.380	.323
	20%	**.415**	**.415**	**.380**	.225	**.380***	.225	.340*	.225	.235
k2ACTNB3	100%	.589	.654	**.739***	.198	**.739***	.198	.510*	.198	.309*
	80%	**.516**	**.524**	**.489**	.099	**.489***	.099	.310*	.099	.213*
	60%	**.397**	**.397**	**.367**	.087	**.367***	.087	.212*	.087	.192*
	40%	**.243**	**.243**	**.245**	.100	**.245***	.100	.140*	.100	.130
	20%	**.160**	**.160**	**.160**	**.160**	**.160**	**.160**	**.170**	**.160***	.115
k2ACTNB4	100%	.814	.937	**.966***	.877	**.966***	.877	.930*	.877*	.844
	80%	.758	.819	**.939***	.653	.915*	.653	.860*	.653	.716*
	60%	.675	.675	**.813***	.273	**.822***	.273	.697*	.273	.527*
	40%	**.550**	**.550**	**.585**	.270	**.550***	.270	.458*	.270	.335*
	20%	**.345**	**.345**	**.295**	.165	**.305***	.165	**.290***	.165	.250*
k2ACTNB5	100%	.846	**.959**	**.960**	**.951**	**.960**	**.951***	.926	**.951***	.799
	80%	.794	**.921***	.864	.881	.864	.881*	.781	.881*	.623
	60%	.713	**.788***	.650	.628	.650	.628*	.523	.628*	.383
	40%	**.620**	**.635***	.520	.208	.520*	.208	.418*	.208	.343*
	20%	**.460**	**.460***	.325	.215	.325*	.215	.305*	.215	.265

Table 15.: Average accuracy value for each k2ACTNB test and for each CTBNC model, changing the dimension of the data set used. The second column indicates the percentage of the original data set used (i.e. X% indicates that only X% of the data set trajectories are used, and each trajectory is cut in order to be X% of the original length). Tests are made using 10 fold cross validation. The bold characters indicate the best models with 90% confidence. The asterisk indicates which scoring function (i.e. MLL or CLL) is better with 90% confidence once set the structure constraints (i.e. number of parents).

B.1.2.3 k2*CTBNC tests*

Test	data set %	CTNB	k = 2 ACTNB (MLL)	k = 2 ACTNB (CLL)	k = 2 CTBNC (MLL)	k = 2 CTBNC (CLL)	k = 3 CTBNC (MLL)	k = 3 CTBNC (CLL)	k = 4 CTBNC (MLL)	k = 4 CTBNC (CLL)
	100%	.627	.807	**.840***	.781	**.840***	.781	.755	.781*	.592
	80%	.566	.732	**.770***	.703	**.765***	.703*	.675	.703*	.549
k2CTBNC1	60%	.505	.590	**.700***	.252	.672*	.252	.545*	.252	.403*
	40%	.453	.453	**.555***	.208	**.533***	.208	.408*	.208	.353*
	20%	**.285**	**.285**	**.270**	.220	**.245***	.220	**.250***	.220	**.260***
	100%	.690	.890	**.924***	.873	**.924***	.873*	.852	.873*	.691
	80%	.649	**.839**	**.846**	.805	**.846***	.805*	.738	.805*	.551
k2CTBNC2	60%	.547	**.747***	.708	.695	.708	.695*	.605	.695*	.445
	40%	**.453**	**.453**	**.468**	.283	**.468***	.283	.375*	.283	.270
	20%	**.355**	**.355***	.285	.255	.285	.255*	.195	.255	.230
	100%	.733	.832*	.809	**.859***	.809	**.859***	.708	**.859***	.557
	80%	.675	**.768***	.744	.745	.736	.745*	.635	.745*	.503
k2CTBNC3	60%	**.605**	**.605**	**.632**	.415	**.610***	.415	.528*	.415	.385
	40%	**.490**	**.490**	**.503**	.280	**.485***	.280	.398*	.280	.303
	20%	**.285**	**.285**	**.325**	.190	**.285***	.190	**.305***	.190	.225
	100%	.757	.858*	.826	**.872***	.826	**.872***	.694	**.872***	.555
	80%	.679	**.825***	.750	**.833***	.750	**.833***	.646	**.833***	.501
k2CTBNC4	60%	.572	.698*	.648	**.763***	.648	**.763***	.495	**.763***	.367
	40%	.550	**.640***	.545	.573	.545	.573*	.450	.573*	.378
	20%	**.430**	**.430**	**.405**	.225	**.405***	.225	.275*	.225	.360*
	100%	.578	.813	**.868***	.823	**.868***	.823*	.777	.823*	.597
	80%	.532	.761	**.796***	.759	**.795***	.759*	.678	.759*	.508
k2CTBNC5	60%	.460	.568	**.692***	.532	**.692***	.532	.545	.532*	.448
	40%	.438	.438	**.578***	.198	**.593***	.198	.453*	.198	.323*
	20%	**.355**	**.355**	**.355**	.210	**.360***	.210	.220	.210	.210
	100%	.665	.837	**.865***	**.878**	**.865**	**.878***	.762	**.878***	.599
	80%	.614	.759	**.795***	**.800**	**.795**	**.800***	.670	**.800***	.499
k2CTBNC6	60%	.538	.717*	.655	**.745***	.655	**.745***	.528	**.745***	.388
	40%	.483	**.610**	**.605**	**.603**	**.605**	**.603***	.468	**.603** *	.345
	20%	**.265**	**.265**	**.290**	**.270**	**.290**	**.270**	**.230**	**.270**	**.245**

Table 16.: Average accuracy value for each k2CTBNC test and for each CTBNC model, changing the dimension of the data set used. The second column indicates the percentage of the original data set used (i.e. X% indicates that only X% of the data set trajectories are used, and each trajectory is cut in order to be X% of the original length). Tests are made using 10 fold cross validation. The bold characters indicate the best models with 90% confidence. The asterisk indicates which scoring function (i.e. MLL or CLL) is better with 90% confidence once set the structure constraints (i.e. number of parents).

B.1.2.4 k3*CTBNC tests*

Test	data set %	CTNB	k = 2 ACTNB (MLL)	k = 2 ACTNB (CLL)	k = 2 CTBNC (MLL)	k = 2 CTBNC (CLL)	k = 3 CTBNC (MLL)	k = 3 CTBNC (CLL)	k = 4 CTBNC (MLL)	k = 4 CTBNC (CLL)
	100%	.508	.670	.701*	.629	.701*	.898	**.923***	.898*	.780
	80%	.489	.479	.663*	.255	.663*	.255	**.846***	.255	.684*
k3CTBNC1	60%	.428	.428	.553*	.235	.547*	.235	**.708***	.235	.512*
	40%	.330	.330	**.398***	.168	.380*	.168	**.430***	.168	.308*
	20%	**.295**	**.295**	**.255**	**.295**	**.25**	**.295***	.215	**.295***	.220
	100%	.565	.732	.735	.730	.735	**.909**	**.927**	**.909***	.795
	80%	.490	.618	.644	.648	.644	**.814**	**.829**	**.814***	.618
k3CTBNC2	60%	.438	.467	.528*	.330	.528*	.330	**.625***	.330	.427*
	40%	**.325**	**.325**	**.355**	.183	**.355***	.183	**.348***	.183	.268*
	20%	**.275**	**.275**	**.280**	**.225**	**.280**	**.225**	**.255**	**.225**	**.275**
	100%	.602	.787*	.757	.789*	.757	**.934***	.877	**.934***	.744
	80%	.551	.753*	.704	.729	.704	**.884***	.790	**.884***	.620
k3CTBNC3	60%	.460	**.672***	.603	.598	.607	.598	**.687***	.598*	.505
	40%	.363	.363	**.445***	.218	**.430***	.218	**.413***	.218	.313*
	20%	**.290**	**.290**	**.325**	.210	**.290***	.210	.220	.210	.160
	100%	.622	.784	.793	.809	.793	**.942***	.915	**.942***	.806
	80%	.559	.750*	.725	.776*	.725	**.923***	.749	**.923***	.563
k3CTBNC4	60%	.527	.692*	.577	.697*	.577	**.827***	.530	**.827***	.398
	40%	.410	**.443***	.383	.398	.383	.400*	.348	.400*	.260
	20%	**.305**	**.305***	.240	.250	.240	.250	.215	.250	.230
	100%	.539	.803*	.674	.808*	.674	**.887***	.805	**.887***	.637
	80%	.489	.555	.606*	.271	.606*	.271	**.714***	.271	.536*
k3CTBNC5	60%	.420	.408	**.540***	.257	**.527***	.257	**.557***	.257	.428*
	40%	.405	.408	**.513***	.278	.430*	.278	.378*	.278	.303
	20%	**.300**	**.300**	**.305**	.230	**.255**	.230	**.250**	.230	.210
	100%	.563	.776	.778	.822*	.778	**.927***	.834	**.927***	.697
	80%	.536	.693	.709	.678	.709	**.826***	.758	**.826***	.543
k3CTBNC6	60%	.438	.427	**.552***	.418	**.552***	.418	**.585***	.418	.448
	40%	**.385**	.368	**.440***	.243	**.440***	.243	.355*	.243	.293*
	20%	**.235**	**.235**	**.245**	**.280**	**.245**	**.280**	**.260**	**.280***	.220

Table 17.: Average accuracy value for each k3CTBNC test and for each CTBNC model, changing the dimension of the data set used. The second column indicates the percentage of the original data set used (i.e. X% indicates that only X% of the data set trajectories are used, and each trajectory is cut in order to be X% of the original length). Tests are made using 10 fold cross validation. The bold characters indicate the best models with 90% confidence. The asterisk indicates which scoring function (i.e. MLL or CLL) is better with 90% confidence once set the structure constraints (i.e. number of parents).

B.1.2.5 k4*CTBNC tests*

Test	data set %	CTNB	k = 2 ACTNB (MLL)	k = 2 ACTNB (CLL)	k = 2 CTBNC (MLL)	k = 2 CTBNC (CLL)	k = 3 CTBNC (MLL)	k = 3 CTBNC (CLL)	k = 4 CTBNC (MLL)	k = 4 CTBNC (CLL)
k4CTBNC1	100%	.426	.436	.560*	.236	.560*	.236	**.864***	.236	**.875***
	80%	.369	.369	.500*	.236	.500*	.236	**.771***	.236	**.774***
	60%	.337	.337	.420*	.273	.420*	.273	**.593***	.273	**.548***
	40%	.338	.338	**.358**	.273	**.358***	.273	**.403***	.273	.290
	20%	**.270**	**.270**	**.300**	**.290**	**.310**	**.290**	**.275**	**.290**	.250
k4CTBNC2	100%	.399	.501	.633*	.462	.633*	.594	.848*	.773	**.876***
	80%	.379	.448	.529*	.420	.529*	.544	**.689***	**.718***	.583
	60%	.348	.427	.465*	.268	.465*	.268	**.575***	.268	.418*
	40%	.270	.270	.293	.195	.293*	.195	**.363***	.195	**.298***
	20%	**.235**	**.235**	**.280**	**.250**	**.280**	**.250**	**.265**	**.250**	**.285**
k4CTBNC3	100%	.870	.937	.924	.951*	.926	**.984***	.967	**.984***	.887
	80%	.819	.896	.885	.895	.889	**.963***	.929	**.963***	.795
	60%	.768	**.827**	**.828**	.813	**.832**	.813	**.845***	.813*	.628
	40%	**.658**	**.645**	**.653**	.248	**.623***	.248	**.610***	.248	.448*
	20%	**.445**	**.445***	.275	.210	.285*	.210	.295*	.210	.295*
k4CTBNC4	100%	.885	.952*	.931	.958*	.931	**.994***	.977	**.994***	.901
	80%	.848	.906*	.869	.925*	.869	**.980***	.926	**.980***	.796
	60%	.763	.857*	.788	.868*	.788	**.955***	.825	**.955***	.665
	40%	**.713**	**.690**	**.668**	**.703**	**.668**	**.703***	.578	**.703***	.428
	20%	**.470**	**.470***	.340	.300	.340	.300	.295	.300*	.230
k4CTBNC5	100%	.584	.718	.830*	.648	.831*	.736	**.975***	.736	.911*
	80%	.540	.663	.763*	.619	.760*	.693	**.944***	.693	.809*
	60%	.505	.580	.673*	.535	.672*	.535	**.865***	.535	.703*
	40%	.348	.348	.485*	.285	.438*	.285	**.568***	.285	.425*
	20%	.315	.315	**.345**	.270	.310*	.270	**.385***	.270	.250
k4CTBNC6	100%	.643	.873	.869	.886	.869	**.996**	**.995**	**.996***	.947
	80%	.581	.794	.814	.759	.814*	**.940**	**.955**	**.940***	.844
	60%	.530	.575	.712*	.595	.712*	.707	**.885***	.707	.695
	40%	.438	.500	.498	.458	.498	.458	**.588***	.458	.460
	20%	**.310**	**.310**	.255	.225	.255	.225	**.330***	.225	.250

Table 18.: Average accuracy value for each k4CTBNC test and for each CTBNC model, changing the dimension of the data set used. The second column indicates the percentage of the original data set used (i.e. X% indicates that only X% of the data set trajectories are used, and each trajectory is cut in order to be X% of the original length). Tests are made using 10 fold cross validation. The bold characters indicate the best models with 90% confidence. The asterisk indicates which scoring function (i.e. MLL or CLL) is better with 90% confidence once set the structure constraints (i.e. number of parents).

B.2 CLUSTERING

B.2.1 *DBN synthetic data sets*

Test	Measure	CTNB	$k=2$ ACTNB (MLL)	$k=2$ ACTNB (CLL)	$k=2$ CTBNC (MLL)	$k=2$ CTBNC (CLL)	$k=3$ CTBNC (MLL)	$k=3$ CTBNC (CLL)	$k=4$ CTBNC (MLL)	$k=4$ CTBNC (CLL)	DBN-NB1	DBN-NB2
DBNTest1	R	1	1	.872	.260	.616	.924	.594	.903	.613	.705	.547
	J	1	1	.624	.260	.148	.775	.164	.712	.150	.271	.185
	FM	1	1	.770	.510	.258	.880	.282	.835	.261	.426	.319
DBNTest2	R	.683	.762	.675	.248	.622	.248	.622	.248	.633	.617	.498
	J	.256	.356	.216	.248	.146	.248	.141	.248	.168	.150	.206
	FM	.409	.525	.355	.498	.255	.498	.247	.498	.287	.261	.365
DBNTest3	R	.991	1	.839	.907	.248	.865	.598	.853	.248	.607	.443
	J	.962	1	.516	.689	.248	.588	.147	.563	.248	.150	.219
	FM	.981	1	.681	.816	.498	.742	.257	.722	.498	.262	.398
DBNTest4	R	.727	.695	.658	.266	.631	.266	.623	.266	.636	.617	.546
	J	.306	.277	.208	.266	.162	.266	.161	.266	.175	.149	.184
	FM	.469	.434	.345	.516	.280	.516	.278	.516	.298	.259	.317
DBNTest5	R	.687	.672	.665	.256	.630	.256	.631	.256	.622	.617	.325
	J	.238	.218	.212	.256	.148	.256	.149	.256	.139	.148	.243
	FM	.384	.357	.350	.506	.258	.506	.260	.506	.244	.257	.464

Table 19.: Clustering performances for each DBN test and for each model. R stands for the Rand index, J stands for Jaccard's coefficient and FM for the Fowlkes–Mallows index.

B.2.2 *CTBNC synthetic data sets*

B.2.2.1 *CTNB tests*

Test	Measure	CTNB	k = 2 ACTNB (MLL)	k = 2 ACTNB (CLL)	k = 2 CTBNC (MLL)	k = 2 CTBNC (CLL)	k = 3 CTBNC (MLL)	k = 3 CTBNC (CLL)	k = 4 CTBNC (MLL)	k = 4 CTBNC (CLL)	DBN-NB1	DBN-NB2
CTNB1	R	.972	.973	.830	.917	.964	.914	.784	.913	.627	.566	.509
	J	.896	.900	.526	.717	.866	.710	.402	.703	.147	.175	.199
	FM	.945	.947	.692	.835	.928	.830	.573	.826	.256	.304	.349
CTNB2	R	.971	.971	.958	.936	.959	.933	.805	.936	.625	.495	.491
	J	.889	.892	.845	.772	.848	.765	.439	.774	.144	.201	.203
	FM	.941	.943	.916	.871	.918	.867	.610	.873	.252	.355	.360
CTNB3	R	.983	.985	.973	.952	.973	.950	.714	.959	.626	.431	.293
	J	.935	.942	.896	.826	.896	.818	.273	.849	.145	.219	.245
	FM	.966	.970	.945	.905	.945	.900	.429	.918	.253	.399	.478
CTNB4	R	.958	.953	.846	.100	.858	.100	.821	.100	.820	.741	.509
	J	.653	.624	.137	.100	.173	.100	.056	.100	.053	.071	.090
	FM	.790	.769	.241	.316	.295	.316	.107	.316	.100	.140	.219
CTNB5	R	.926	.970	.848	.886	.866	.846	.822	.927	.819	.740	.259
	J	.480	.739	.151	.284	.205	.150	.060	.479	.053	.071	.099
	FM	.649	.850	.262	.442	.341	.260	.113	.648	.101	.140	.285

Table 20.: Clustering performances for each CTNB test and for each model. R stands for the Rand index, J stands for Jaccard's coefficient and FM for the Fowlkes–Mallows index.

B.2.2.2 k2*ACTNB tests*

Test	Measure	CTNB	$k = 2$ ACTNB (MLL)	$k = 2$ ACTNB (CLL)	$k = 2$ CTBNC (MLL)	$k = 2$ CTBNC (CLL)	$k = 3$ CTBNC (MLL)	$k = 3$ CTBNC (CLL)	$k = 4$ CTBNC (MLL)	$k = 4$ CTBNC (CLL)	DBN-NB1	DBN-NB2
k2ACTNB1	R	.828	.917	.780	.675	.707	.740	.634	.758	.626	.553	.580
	J	.488	.716	.389	.219	.265	.324	.155	.352	.144	.180	.172
	FM	.656	.834	.560	.360	.418	.490	.268	.520	.252	.312	.298
k2ACTNB2	R	.813	.967	.695	.892	.800	.920	.627	.894	.626	.534	.249
	J	.456	.875	.242	.645	.430	.723	.145	.649	.143	.189	.249
	FM	.626	.933	.390	.784	.601	.839	.253	.787	.250	.330	.499
k2ACTNB3	R	.830	.838	.824	.100	.822	.100	.820	.100	.820	.604	.133
	J	.093	.108	.065	.100	.061	.100	.056	.100	.052	.086	.100
	FM	.171	.196	.122	.317	.115	.317	.105	.317	.099	.194	.311
k2ACTNB4	R	.774	.945	.726	.760	.662	.763	.635	.781	.626	.429	.349
	J	.384	.802	.300	.352	.194	.376	.156	.398	.143	.220	.235
	FM	.555	.890	.462	.520	.325	.547	.270	.570	.250	.401	.447
k2ACTNB5	R	.845	.957	.702	.923	.738	.902	.626	.938	.625	.617	.275
	J	.529	.840	.258	.733	.319	.672	.143	.778	.144	.148	.246
	FM	.692	.913	.410	.846	.483	.804	.251	.875	.251	.258	.487

Table 21.: Clustering performances for each k2ACTNB test and for each model. R stands for the Rand index, J stands for Jaccard's coefficient and FM for the Fowlkes–Mallows index.

B.2.2.3 k2*CTBNC tests*

Test	Measure	CTNB	$k=2$ ACTNB (MLL)	$k=2$ ACTNB (CLL)	$k=2$ CTBNC (MLL)	$k=2$ CTBNC (CLL)	$k=3$ CTBNC (MLL)	$k=3$ CTBNC (CLL)	$k=4$ CTBNC (MLL)	$k=4$ CTBNC (CLL)	DBN-NB1	DBN-NB2
k2CTBNC1	R	.687	.786	.638	.743	.738	.784	.627	.776	.627	.576	.452
	J	.233	.408	.161	.330	.318	.400	.145	.384	.145	.170	.213
	FM	.378	.580	.278	.497	.483	.571	.254	.555	.253	.295	.385
k2CTBNC2	R	.672	.787	.637	.892	.631	.840	.628	.798	.626	.589	.578
	J	.212	.418	.159	.646	.152	.516	.149	.448	.143	.166	.170
	FM	.350	.590	.275	.785	.265	.680	.260	.620	.251	.287	.294
k2CTBNC3	R	.744	.849	.749	.764	.655	.853	.626	.845	.627	.453	.490
	J	.326	.535	.339	.359	.194	.546	.144	.525	.144	.213	.202
	FM	.491	.697	.506	.528	.325	.707	.252	.689	.252	.384	.359
k2CTBNC4	R	.723	.762	.676	.879	.668	.886	.636	.881	.627	.581	.537
	J	.307	.355	.213	.610	.206	.628	.157	.616	.145	.167	.187
	FM	.470	.524	.352	.758	.342	.772	.271	.762	.253	.290	.326
k2CTBNC5	R	.646	.758	.753	.822	.642	.836	.626	.819	.625	.556	.337
	J	.171	.347	.342	.474	.165	.505	.145	.471	.143	.180	.237
	FM	.292	.515	.510	.643	.284	.671	.253	.640	.250	.312	.454
k2CTBNC6	R	.690	.826	.637	.905	.634	.842	.626	.902	.627	.616	.267
	J	.236	.483	.158	.681	.155	.520	.145	.673	.145	.147	.247
	FM	.381	.651	.273	.810	.268	.684	.253	.805	.253	.257	.491

Table 22.: Clustering performances for each k2CTBNC test and for each model. R stands for the Rand index, J stands for Jaccard's coefficient and FM for the Fowlkes–Mallows index.

B.2.2.4 k3*CTBNC tests*

Test	Measure	CTNB	k = 2 ACTNB (MLL)	k = 2 ACTNB (CLL)	k = 2 CTBNC (MLL)	k = 2 CTBNC (CLL)	k = 3 CTBNC (MLL)	k = 3 CTBNC (CLL)	k = 4 CTBNC (MLL)	k = 4 CTBNC (CLL)	DBN-NB1	DBN-NB2
k3CTBNC1	R	.635	.677	.634	.250	.632	.874	.636	.868	.626	.573	.439
	J	.156	.217	.158	.250	.152	.596	.157	.581	.143	.173	.217
	FM	.270	.356	.273	.500	.264	.747	.271	.735	.251	.299	.395
k3CTBNC2	R	.642	.674	.632	.685	.628	.813	.625	.838	.629	.547	.432
	J	.173	.225	.152	.231	.146	.456	.144	.511	.147	.184	.218
	FM	.295	.368	.265	.376	.255	.626	.252	.677	.257	.320	.398
k3CTBNC3	R	.664	.711	.664	.785	.627	.934	.626	.897	.626	.579	.342
	J	.199	.268	.198	.398	.146	.767	.144	.662	.143	.171	.236
	FM	.332	.423	.330	.569	.255	.868	.251	.797	.250	.295	.451
k3CTBNC4	R	.645	.728	.640	.831	.657	.942	.626	.930	.625	.600	.327
	J	.169	.306	.163	.495	.190	.793	.144	.756	.142	.158	.239
	FM	.289	.469	.281	.662	.320	.884	.251	.861	.249	.273	.460
k3CTBNC5	R	.646	.672	.638	.693	.632	.770	.628	.899	.625	.594	.250
	J	.172	.212	.162	.249	.156	.391	.145	.665	.142	.161	.250
	FM	.294	.350	.279	.399	.270	.563	.254	.799	.249	.279	.500
k3CTBNC6	R	.639	.666	.631	.827	.629	.809	.627	.868	.627	.593	.440
	J	.160	.202	.151	.486	.151	.485	.149	.583	.144	.162	.216
	FM	.276	.336	.263	.654	.263	.657	.259	.737	.252	.281	.393

Table 23.: Clustering performances for each k3CTBNC test and for each model. R stands for the Rand index, J stands for Jaccard's coefficient and FM for the Fowlkes–Mallows index.

B.2.2.5 k4*CTBNC tests*

Test	Measure	CTNB	$k = 2$ ACTNB (MLL)	$k = 2$ ACTNB (CLL)	$k = 2$ CTBNC (MLL)	$k = 2$ CTBNC (CLL)	$k = 3$ CTBNC (MLL)	$k = 3$ CTBNC (CLL)	$k = 4$ CTBNC (MLL)	$k = 4$ CTBNC (CLL)	DBN-NB1	DBN-NB2
k4CTBNC1	R	.628	.632	.628	.250	.627	.250	.634	.250	.626	.607	.444
	J	.148	.153	.146	.250	.148	.250	.154	.250	.144	.155	.215
	FM	.257	.265	.256	.500	.258	.500	.267	.500	.251	.268	.390
k4CTBNC2	R	.628	.627	.624	.249	.626	.249	.626	.767	.625	.571	.574
	J	.147	.145	.143	.249	.145	.249	.144	.368	.142	.172	.170
	FM	.256	.253	.250	.499	.253	.499	.251	.538	.249	.298	.295
k4CTBNC3	R	.876	.937	.790	.880	.780	.983	.692	.984	.629	.617	.362
	J	.602	.778	.419	.620	.400	.934	.239	.938	.148	.147	.233
	FM	.752	.875	.591	.766	.572	.966	.385	.968	.257	.257	.439
k4CTBNC4	R	.884	.945	.829	.955	.871	.994	.662	.993	.627	.603	.477
	J	.623	.801	.493	.834	.589	.976	.194	.972	.144	.155	.206
	FM	.768	.889	.660	.910	.742	.988	.326	.986	.252	.270	.368
k4CTBNC5	R	.644	.812	.631	.691	.640	.778	.626	.788	.625	.556	.495
	J	.168	.453	.151	.255	.162	.385	.144	.408	.142	.180	.201
	FM	.288	.624	.262	.406	.280	.556	.252	.579	.249	.312	.356
k4CTBNC6	R	.672	.791	.635	.804	.639	.997	.629	.997	.626	.528	.267
	J	.206	.411	.157	.440	.160	.988	.148	.988	.143	.190	.247
	FM	.342	.583	.272	.611	.276	.994	.258	.994	.250	.333	.490

Table 24.: Clustering performances for each k4CTBNC test and for each model. R stands for the Rand index, J stands for Jaccard's coefficient and FM for the Fowlkes–Mallows index.

b.2.3 *Traffic profiling tests*

Road network	Trajectories length	Measure	CTNB	k = 2 ACTNB (MLL)	k = 2 ACTNB (CLL)	k = 2 CTBNC (MLL)	k = 2 CTBNC (CLL)	k = 3 CTBNC (MLL)	k = 3 CTBNC (CLL)	k = 4 CTBNC (MLL)	k = 4 CTBNC (CLL)	DBN-NB1	DBN-NB2
Toy netwok	100 seconds	R	.932	.930	.933	.931	.867	.923	.933	.860	.931	.476	.647
		J	.700	.693	.704	.700	.500	.667	.704	.489	.696	.193	.184
		FM	.823	.818	.826	.823	.667	.800	.827	.658	.821	.375	.321
	300 seconds	R	.819	.874	.877	.898	.950	.968	.912	.957	.885	.486	.306
		J	.397	.532	.535	.568	.771	.847	.664	.798	.544	.164	.183
		FM	.571	.696	.698	.725	.871	.917	.805	.888	.705	.318	.393
Monza's netwok*	100 seconds	R	.797	.806	.796	.805	.805	.795	.800	.788	.805	.253	.344
		J	.287	.304	.288	.301	.296	.280	.281	.274	.309	.176	.171
		FM	.446	.466	.447	.463	.457	.438	.438	.431	.472	.399	.367
	300 seconds	R	.819	.817	.805	.811	.832	.823	.810	.821	.820	.535	.540
		J	.341	.357	.376	.328	.364	.333	.321	.321	.336	.244	.252
		FM	.508	.527	.554	.494	.534	.500	.486	.486	.503	.461	.474
Monza's network real loops*	100 seconds	R	.772	.776	.759	.774	.757	.766	.772	.781	.759	.588	.502
		J	.213	.243	.207	.242	.225	.223	.229	.236	.218	.145	.156
		FM	.352	.391	.343	.389	.368	.365	.372	.382	.358	.270	.305
	300 seconds	R	.790	.807	.771	.810	.808	.797	.805	.803	.813	.607	.573
		J	.260	.297	.245	.305	.289	.272	.295	.307	.315	.193	.198
		FM	.413	.458	.394	.467	.448	.428	.456	.471	.479	.349	.366

Table 25.: Clustering performances on the traffic profiling prediction data sets. R stands for the Rand index, J stands for Jaccard's coefficient and FM for the Fowlkes–Mallows index. The asterisk indicates that for computational reasons a random subset of trajectories was used for the experiments.

BIBLIOGRAPHY

Angulo, E., Romero, F. P., García, R., Serrano-Guerrero, J., and Olivas, J. A. (2011). An adaptive approach to enhanced traffic signal optimization by using soft-computing techniques. *Expert Systems with Applications*, 38(3):2235–2247. (Cited on pages 84 and 85.)

Arnott, R. and Small, K. (1994). The economics of traffic congestion. *American Scientist*, 82(5):446–455. (Cited on page 83.)

Barber, D. and Cemgil, A. (2010). Graphical models for time-series. *Signal Processing Magazine, IEEE*, 27(6):18–28. (Cited on page 1.)

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press. (Cited on page 8.)

Boudali, H. and Dugan, J. (2006). A continuous-time bayesian network reliability modeling, and analysis framework. *Reliability, IEEE Trans. on*, 55(1):86–97. (Cited on page 21.)

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32. (Cited on page 8.)

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167. (Cited on page 8.)

Caliński, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27. (Cited on page 69.)

Codecasa, D. and Stella, F. (2013). Conditional log-likelihood for continuous time bayesian network classifiers. In *International Workshop NFMCP held at ECML-PKDD 2013*. (Cited on pages 39, 42, 79, 98, 99, 100, 141, and 142.)

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27. (Cited on page 12.)

Dacorogna, M. (2001). *An introduction to high-frequency finance*. AP. (Cited on page 1.)

Daigle, G., Krueger, G. D., and Clark, J. (1997). Tsis: advanced traffic software tools for the user. In *Traffic Congestion and Traffic Safety in the 21st Century: Challenges, Innovations, and Opportunities*. (Cited on page 85.)

Dasarathy, B. V. (1991). *Nearest Neighbor ({NN}) Norms:{NN} Pattern Classification Techniques*. IEEE Computer Society Press. (Cited on page 12.)

Davies, D. L. and Bouldin, D. W. (1979). A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):224–227. (Cited on page 69.)

Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational intelligence*, 5(2):142–150. (Cited on pages 2, 3, and 13.)

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157. (Cited on page 8.)

Duda, R. O., Hart, P. E., et al. (1973). *Pattern classification and scene analysis*, volume 3. Wiley New York. (Cited on page 9.)

El-Hay, T., Friedman, N., and Kupferman, R. (2008). Gibbs sampling in factorized continuous-time markov processes. In McAllester, D. A. and Myllym, P., editors, *Proc. of the 24th Conf. on Uncertainty in Artificial Intelligence*, pages 169–178. AUAI Press. (Cited on pages 16 and 27.)

Fan, Y. and Shelton, C. (2008). Sampling for approximate inference in continuous time bayesian networks. In *10th Int. Symposium on Artificial Intelligence and Mathematics*. (Cited on pages 16 and 27.)

Fan, Y. and Shelton, C. (2009). Learning continuous-time social network dynamics. In *Proc. of the 25th Conf. on Uncertainty in Artificial Intelligence*, pages 161–168. AUAI Press. (Cited on page 21.)

Favilla, J., Machion, A., and Gomide, F. (1993). Fuzzy traffic control: adaptive strategies. In *Fuzzy Systems, 1993., Second IEEE International Conference on*, pages 506–511. IEEE. (Cited on page 84.)

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874. (Cited on pages 117, 119, and 120.)

Felici, G., Rinaldi, G., Sforza, A., and Truemper, K. (2006). A logic programming based approach for on-line traffic control. *Transportation Research Part C: Emerging Technologies*, 14(3):175–189. (Cited on page 84.)

Forster, A. and Young, J. (2002). *The clinical and cost effectiveness of physiotherapy in the management of elderly people following a stroke*. London, UK: The Chartered Society Of Physiotherapy. (Cited on page 76.)

Fowlkes, E. B. and Mallows, C. L. (1983). A method for comparing two hierarchical clusterings. *Journal of the American statistical association*, 78(383):553–569. (Cited on pages 70, 103, and 121.)

Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine learning*, 29(2):131–163. (Cited on pages 7, 9, 10, 36, 37, 38, and 41.)

Friedman, N., Murphy, K., and Russell, S. (1998). Learning the structure of dynamic probabilistic networks. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 139–147. Morgan Kaufmann Publishers Inc. (Cited on page 13.)

Gan, G., Ma, C., and Wu, J. (2007). *Data clustering: theory, algorithms, and applications*, volume 20. Siam. (Cited on pages 69 and 121.)

Gatti, E., Luciani, D., and Stella, F. (2011). A continuous time bayesian network model for cardiogenic heart failure. *Flexible Services and Manufacturing Journal*, pages 1–20. (Cited on page 21.)

Gershenson, C. (2004). Self-organizing traffic lights. *arXiv preprint nlin/0411066*. (Cited on page 84.)

Ghahramani, Z. (1998). Learning dynamic bayesian networks. In *Adaptive processing of sequences and data structures*, pages 168–197. Springer. (Cited on page 13.)

Gopalratnam, K., Kautz, H., and Weld, D. S. (2005). Extending continuous time bayesian networks. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 981. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. (Cited on pages 16 and 93.)

Granger, C. W. (1969). Investigating causal relations by econometric models and cross-spectral methods. *Econometrica: Journal of the Econometric Society*, pages 424–438. (Cited on page 46.)

Greiner, R. and Zhou, W. (2002). Structural extension to logistic regression: Discriminative parameter learning of belief net classifiers. In *Proc. of the National Conf. on Artificial Intelligence*, pages 167–173. Menlo Park, CA. (Cited on page 10.)

Grossman, D. and Domingos, P. (2004). Learning bayesian network classifiers by maximizing conditional likelihood. In *Proc. of the 21st Int. Conf. on Machine Learning*, pages 361–368. ACM Press. (Cited on pages 10 and 44.)

Gualtieri, M., Rigamonti, L., Galeotti, V., and Camatini, M. (2005). Toxicity of tire debris extracts on human lung cell line a549. *Toxicology in vitro*, 19(7):1001–1008. (Cited on page 83.)

Gunawardana, A., Meek, C., and Xu, P. (2011). A model for temporal dependencies in event streams. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 24*, pages 1962–1970. (Cited on pages 2 and 93.)

Haijema, R. and van der Wal, J. (2008). An mdp decomposition approach for traffic control at isolated signalized intersections. *Probability in the Engineering and Informational Sciences*, 22(04):587–602. (Cited on page 84.)

Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2001). On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145. (Cited on pages 69, 70, 103, and 121.)

Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2002). Cluster validity methods: part i. *ACM Sigmod Record*, 31(2):40–45. (Cited on page 70.)

Hartigan, J. A. (1975). *Clustering algorithms*. John Wiley & Sons, Inc. (Cited on pages 17 and 103.)

Hazewinkel, M. (1993). *Encyclopaedia of Mathematics (9)*, volume 9. Springer. (Cited on pages 13 and 78.)

Hirankitti, V. and Krohkaew, J. (2007). An agent approach for intelligent traffic-light control. In *Modelling & Simulation, 2007. AMS'07. First Asia International Conference on*, pages 496–501. IEEE. (Cited on page 84.)

Itakura, F. (1975). Minimum prediction residual principle applied to speech recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 23(1):67–72. (Cited on page 12.)

Japkowicz, N. and Shah, M. (2011). *Evaluating learning algorithms: a classification perspective*. Cambridge University Press. (Cited on page 117.)

Joachims, T. (1998). *Text categorization with support vector machines: Learning with many relevant features*. Springer. (Cited on page 8.)

Keogh, E. and Ratanamahatana, C. A. (2005). Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386. (Cited on pages 12 and 77.)

Keogh, E. J. and Pazzani, M. J. (2000). Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 285–289. ACM. (Cited on page 11.)

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. The MIT Press. (Cited on pages 18, 62, 65, and 103.)

Kwakkel, G., van Peppen, R., Wagenaar, R. C., Dauphinee, S. W., Richards, C., Ashburn, A., Miller, K., Lincoln, N., Partridge, C., Wellwood, I., et al. (2004). Effects of augmented exercise therapy time after stroke a meta-analysis. *Stroke*, 35(11):2529–2539. (Cited on page 76.)

Lämmer, S. and Helbing, D. (2008). Self-control of traffic lights and vehicle flows in urban road networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(04):P04019. (Cited on page 84.)

Langley, P., Iba, W., and Thompson, K. (1992). An analysis of bayesian classifiers. In *AAAI*, volume 90, pages 223–228. (Cited on page 9.)

Langseth, H. and Nielsen, T. D. (2005). Latent classification models. *Machine Learning*, 59(3):237–265. (Cited on page 15.)

Mantecca, P., Gualtieri, M., Andrioletti, M., Bacchetta, R., Vismara, C., Vailati, G., and Camatini, M. (2007). Tire debris organic extract affects< i> xenopus</i> development. *Environment international*, 33(5):642–648. (Cited on page 83.)

Murphy, K. et al. (2001). The bayes net toolbox for matlab. *Computing science and statistics*, 33(2):1024–1034. (Cited on pages 45 and 68.)

Murphy, K. P. (2002). *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California. (Cited on page 13.)

Nodelman, U., Koller, D., and Shelton, C. (2005). Expectation propagation for continuous time bayesian networks. In *Proc. of the 21st Conf. on Uncertainty in Artificial Intelligence*, pages 431–440, Edinburgh, Scotland, UK. (Cited on pages 2, 16, 26, and 31.)

Nodelman, U., Shelton, C., and Koller, D. (2002a). Continuous time bayesian networks. In *Proc. of the 18th Conf. on Uncertainty in Artificial Intelligence*, pages 378–387. Morgan Kaufmann Publishers Inc. (Cited on pages 15, 16, 20, 22, 23, 25, 26, 32, and 91.)

Nodelman, U., Shelton, C., and Koller, D. (2002b). Learning continuous time bayesian networks. In *Proc. of the 19th Conf. on Uncertainty in Artificial Intelligence*, pages 451–458. Morgan Kaufmann Publishers Inc. (Cited on pages 3, 4, 16, 28, 29, 40, 44, 91, 100, 141, and 142.)

Nodelman, U., Shelton, C. R., and Koller, D. (2012). Expectation maximization and complex duration distributions for continuous time bayesian networks. *arXiv preprint arXiv:1207.1402*. (Cited on page 93.)

Nodelman, U. D. (2007). *Continuous time Bayesian networks*. PhD thesis, Stanford University. (Cited on pages 15 and 57.)

Oates, T., Firoiu, L., and Cohen, P. R. (1999). Clustering time series with hidden markov models and dynamic time warping. In *Proceedings of the IJCAI-99 workshop on neural, symbolic and reinforcement learning methods for sequence learning*, pages 17–21. Citeseer. (Cited on page 18.)

Owen, L. E., Zhang, Y., Rao, L., and McHale, G. (2000). Street and traffic simulation: traffic flow simulation using corsim. In *Proceedings of the 32nd conference on Winter simulation*, pages 1143–1147. Society for Computer Simulation International. (Cited on page 85.)

Paolucci, S., Antonucci, G., Grasso, M. G., Morelli, D., Troisi, E., Coiro, P., and Bragoni, M. (2000). Early versus delayed inpatient stroke rehabilitation: a matched comparison conducted in italy. *Archives of physical medicine and rehabilitation*, 81(6):695–700. (Cited on page 76.)

Papageorgiou, M., Diakaki, C., Dinopoulou, V., Kotsialos, A., and Wang, Y. (2003). Review of road traffic control strategies. *Proceedings of the IEEE*, 91(12):2043–2067. (Cited on page 84.)

Park, B. and Messer, C. (1998). A genetic algorithm-based signal optimization program for oversaturated intersections. In *Towards the new horizon together. Proceeding of the 5th world congress of intelligent transport system.*, number 1026, Seoul, Korea. (Cited on page 84.)

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann. (Cited on page 9.)

Pękalska, E., Duin, R. P., and Paclík, P. (2006). Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, 39(2):189–208. (Cited on page 12.)

Pelleg, D., Moore, A. W., et al. (2000). X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, pages 727–734. (Cited on page 17.)

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106. (Cited on page 8.)

Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16. (Cited on page 14.)

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286. (Cited on pages 2 and 14.)

Rajaram, S., Graepel, T., and Herbrich, R. (2005). Poisson-networks: A model for structured point processes. In *Proc. of the 10th Int. Workshop on Artificial Intelligence and Statistics*. (Cited on page 2.)

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850. (Cited on pages 70, 103, and 121.)

Ratanamahatana, C. A. and Keogh, E. (2004). Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*, pages 22–25. (Cited on page 11.)

Ripley, B. D. (2007). *Pattern recognition and neural networks*. Cambridge university press. (Cited on page 8.)

Robertson, D. I. (1969). " transyt" method for area traffic control. *Traffic Engineering & Control*, 11(6). (Cited on page 84.)

Robertson, D. I. and Bretherton, R. D. (1991). Optimizing networks of traffic signals in real time-the scoot method. *Vehicular Technology, IEEE Transactions on*, 40(1):11–15. (Cited on page 84.)

Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43–49. (Cited on page 12.)

Saria, S., Nodelman, U., and Koller, D. (2007). Reasoning at the right time granularity. In *UAI*, pages 326–334. (Cited on pages 16 and 26.)

Simma, A., Goldszmidt, M., MacCormick, J., Barham, P., Black, R., Isaacs, R., and Mortier, R. (2008). Ct-nor: Representing and reasoning about events in continuous time. In McAllester, D. A. and Myllym, P., editors, *Proc. of the 24th Conf. on Uncertainty in Artificial Intelligence*, pages 484–493. AUAI Press. (Cited on page 2.)

Simma, A. and Jordan, M. (2010). Modeling events with cascades of poisson processes. In *Proc. of the 26th Conf. on Uncertainty in Artificial Intelligence*, pages 546–555. AUAI Press. (Cited on pages 1 and 2.)

Sims, A. G. and Dobinson, K. (1980). The sydney coordinated adaptive traffic (scat) system philosophy and benefits. *Vehicular Technology, IEEE Transactions on*, 29(2):130–137. (Cited on page 84.)

Sokal, R. R. and Rohlf, F. J. (1962). The comparison of dendrograms by objective methods. *Taxon*, 11(2):33–40. (Cited on page 69.)

Spall, J. C. and Chin, D. C. (1997). Traffic-responsive signal timing for system-wide traffic control. *Transportation Research Part C: Emerging Technologies*, 5(3-4):153–163. (Cited on page 84.)

Stella, F. and Amer, Y. (2012). Continuous time bayesian network classifiers. *Journal of Biomedical Informatics*, 45(6):1108 – 1119. (Cited on pages 3, 16, 17, 29, 30, 31, 32, 33, 36, 91, 98, 141, and 148.)

Stella, F., Viganò, V., Bogni, D., and Benzoni, M. (2006). An integrated forecasting and regularization framework for light rail transit systems. *Journal of Intelligent Transportation Systems*, 10(2):59–73. (Cited on page 84.)

Thorpe, T. L. and Anderson, C. W. (1996). Tra c light control using sarsa with three state representations. Technical report, Citeseer. (Cited on page 84.)

Tormene, P. and Giorgino, T. (2008). Upper-limb rehabilitation exercises acquired through 29 elastomer strain sensors placed on fabric. release 1.0. (Cited on page 77.)

Tormene, P., Giorgino, T., Quaglini, S., and Stefanelli, M. (2009). Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation. *Artificial Intelligence in Medicine*, 45(1):11–34. (Cited on pages 12, 76, 77, 78, 79, and 80.)

Truccolo, W., Eden, U., Fellows, M., Donoghue, J., and Brown, E. (2005). A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects. *Journal of neurophysiology*, 93(2):1074–1089. (Cited on page 1.)

Vapnik, V. N. and Kotz, S. (1982). *Estimation of dependences based on empirical data*, volume 41. Springer-Verlag New York. (Cited on page 8.)

Voit, E. (2012). *A First Course in Systems Biology*. Garland Science: NY. (Cited on page 1.)

Weiss, J. C. and Page, D. (2013). Forest-based point process for event prediction from electronic health records. In *Machine Learning and Knowledge Discovery in Databases*, pages 547–562. Springer. (Cited on page 93.)

Welch, G. and Bishop, G. (1995). An introduction to the kalman filter. (Cited on page 14.)

Wiering, M. (2000). Multi-agent reinforcement learning for traffic light control. (Cited on page 84.)

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann. (Cited on pages 17, 102, 117, and 149.)

Xi, X., Keogh, E., Shelton, C., Wei, L., and Ratanamahatana, C. A. (2006). Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, pages 1033–1040. ACM. (Cited on page 12.)

Xu, J. and Shelton, C. (2008). Continuous time bayesian networks for host level network intrusion detection. *Machine Learning and Knowledge Discovery in Databases*, pages 613–627. (Cited on page 21.)

Xu, R. and Wunsch, D. (2008). *Clustering*, volume 10. Wiley. com. (Cited on pages 69 and 121.)

Yilmaz, A., Javed, O., and Shah, M. (2006). Object tracking: A survey. *Acm Computing Surveys (CSUR)*, 38(4). (Cited on page 1.)

Yu, X.-H. and Recker, W. W. (2006). Stochastic adaptive control model for traffic signal systems. *Transportation Research Part C: Emerging Technologies*, 14(4):263–282. (Cited on page 84.)

Zhang, N. L., Nielsen, T. D., and Jensen, F. V. (2004). Latent variable discovery in classification models. *Artificial Intelligence in Medicine*, 30(3):283–299. (Cited on page 15.)

Zhong, S., Langseth, H., and Nielsen, T. D. (2012). Bayesian networks for dynamic classification. Technical report, http://idi.ntnu.no/˜shket/dLCM.pdf. (Cited on pages 3 and 15.)

Zhong, S., Martinez, A. M., Nielsen, T. D., and Langseth, H. (2010). Towards a more expressive model for dynamic classification. In *FLAIRS Conference*. (Cited on page 15.)

INDEX

Milano, Italia (Milan, Italy)

October MMXIII