DEPARTAMENTO DE INTELIGENCIA ARTIFICIAL

Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

PhD THESIS

Nonparametric Models and Bayesian Networks. Applications to Anomaly Detection

Author

David Atienza
MS Artificial Intelligence

PhD supervisors

Pedro Larrañaga
PhD Computer Science

Concha Bielza
PhD Computer Science

2021

Thesis Committee

President:

Member:

Member:

Member:

Secretary:

A mis padres, Ángel y Casilda,
a mi hermana Aída y a María

Agradecimientos

Este ha sido un largo (y emocionante) camino que no habría sido posible recorrer sin el apoyo de muchas personas. Estas líneas son para mostrar mi gratitud al menos a algunas de ellas.

Mis supervisores, Pedro Larrañaga y Concha Bielza, por su confianza, orientación y motivación. He aprendido mucho de ellos durante todos estos años.

Una parte importante de esta tesis se ha desarrollado gracias a la colaboración de Etxe-Tar y Aingura IIoT. Especialmente quiero agradecérselo a Javier Díaz, Patxi Samaniego y José Juan Gabilondo. No sólo me proporcionaron datos valiosos, sino que también dedicaron su tiempo para introducirme en el mundo de la industria.

Esta tesis ha sido financiada por la beca FPU16/00921 del Ministerio de Educación, Cultura y Deporte, el proyecto TIN2016-79684-P del Ministerio de Economía y Competitividad, el proyecto TERMPROB (IDI-20160715) del Centro para el Desarrollo Tecnológico Industrial (CDTI), y los proyectos PID2019-109247GB-I00 y RTC2019-00687-7 del Ministerio de Ciencia e Innovación.

Mis compañeros del *Computational Intelligence Group* fueron un apoyo en el día a día creando un fantástico ambiente de trabajo. Siempre recordaré estos años con ellos como una gran experiencia.

Gracias a todos los profesores que me han enseñado tantas cosas desde el día que decidí estudiar informática. Quiero hacer también un agradecimiento especial a Juan José Rodríguez Díez y César García Osorio de la Universidad de Burgos, que con sus interesantísimas lecciones me hicieron disfrutar de la inteligencia artificial y la algoritmia.

Mi familia ha sido una fuente infinita de amor y motivación. Mi mayor agradecimiento es para mis padres (Ángel y Casilda), mi hermana (Aída) y mis sobrinos (Héctor y Marcos) que me han apoyado más de lo que las palabras pueden expresar. Esta tesis es tan suya como mía.

Finalmente, mi más profundo agradecimiento a María, cuyo amor y cariño es lo mejor que me ha pasado en los últimos años. Espero que nuestro camino continue muchos años más.

Abstract

Nowadays, machine learning has become an important tool to create models from the large amount of available data. These models are usually useful to solve many different tasks such as classification, clustering, probability density estimation, anomaly detection, etc.

This thesis is primarily concerned with dealing with uncertainty, which is usually present in data. Analyzing this uncertainty can be helpful to better understand the proccess under study. A commonly used technique is to estimate the underlying probability distribution that generated the data, which is unknown for most real world data. This estimation can be performed with two different types of models: parametric and nonparametric. Parametric models make assumptions about the class of the underlying probability distribution and the objective is to find the best parameter values that provide the best fit to the data. In contrast, nonparametric models alleviate the assumptions on the underlying probability distribution, and generate the estimate direclty from data. However, nonparametric models do not provide a good performance when dealing with high-dimensional data, a problem often referred to as the curse of dimensionality in the literature.

Bayesian networks are a probabilistic graphical model that factorizes a joint probability distribution into the product of multiple conditional probability distributions, taking advantage of the conditional independences in the probability distribution. This is helpful for converting the estimation of a high-dimensional probability distribution into the estimation of several low-dimensional conditional probability distributions. Thus, in this thesis we propose the class of semiparametric Bayesian networks, which model the low-dimensional conditional probability distributions using either parametric or nonparametric models. This novel class of Bayesian networks generalizes two common classes of Bayesian networks in the state of the art. Moreover, the semiparametric Bayesian networks can be learned using an adaptation of standard learning algorithms for Bayesian networks. In addition, an extension to semiparametric Bayesian networks is proposed which can model hybrid data containing both discrete and continuous data.

Anomaly detection is the proccess of detecting events that differ significantly from the normal behavior of the system. This task is often approached by detecting low-probability events, since anomalies are rare. This has many applications, particularly in industry where errors in production must be identified as early as possible. In this thesis, we perform anomaly detection in a real laser heat-treatment process used in the automative industry. Two different approaches are proposed to detect anomalies. In the first approach, the laser movement is tracked, so the source high-dimensional data is transformed into low-dimensional data. Then, a grid of nonparametric models is used to detect anomalies. The second approach models the source high-dimensional data using semiparametric Bayesian networks. Both approaches take into account the temporal characteristics of the data and exhibit promising capabilities to detect anomalies.

Resumen

En la actualidad, el aprendizaje automático se ha convertido en una herramienta importante para crear modelos a partir de la gran cantidad de datos disponibles. Estos modelos suelen ser útiles para resolver muchas tareas diferentes, como la clasificación, el *clustering*, la estimación de la densidad de probabilidad, la detección de anomalías, etc.

Esta tesis se ocupa principalmente de tratar la incertidumbre que suele estar presente en los datos. El análisis de esta incertidumbre puede ser útil para comprender mejor el proceso estudiado. Una técnica comúnmente utilizada es estimar la distribución de probabilidad subyacente que generó los datos, que es desconocida para la mayoría de los datos del mundo real. Esta estimación puede realizarse con dos tipos de modelos diferentes: paramétricos y no paramétricos. Los modelos paramétricos hacen asunciones sobre la clase de la distribución de probabilidad subyacente y el objetivo es encontrar los valores de los parámetros que proporcionen el mejor ajuste a los datos. Por el contrario, los modelos no paramétricos reducen las asunciones sobre la distribución de probabilidad subyacente y generan la estimación directamente a partir de los datos. Sin embargo, los modelos no paramétricos no obtienen buen rendimiento cuando se trabaja con datos de alta dimensionalidad, un problema que a menudo se conoce como la maldición de la dimensionalidad en la literatura.

Las redes Bayesianas son un modelo gráfico probabilístico que factoriza una distribución de probabilidad conjunta en el producto de múltiples distribuciones de probabilidad condicionales, aprovechando las independencias condicionales en la distribución de probabilidad. Esto es útil para convertir la estimación de una distribución de probabilidad de alta dimensión en la estimación de varias distribuciones de probabilidad condicional de baja dimensión. Así, en esta tesis proponemos la clase de redes Bayesianas semiparamétricas, que modelan las distribuciones de probabilidad condicionales de baja dimensión utilizando modelos paramétricos o no paramétricos. Esta nueva clase de redes Bayesianas generaliza dos clases comunes de redes Bayesianas en el estado del arte. Además, las redes Bayesianas semiparamétricas pueden aprenderse utilizando una adaptación de los algoritmos de aprendizaje estándar para redes Bayesianas. Adicionalmente, también se propone una extensión de las redes Bayesianas semiparamétricas que puede modelar datos híbridos formados por datos discretos y continuos.

La detección de anomalías es el proceso de detección de eventos que difieren significativamente del comportamiento normal del sistema. Esta tarea se suele abordar detectando eventos de baja probabilidad, ya que las anomalías son poco frecuentes. Esto tiene muchas aplicaciones, especialmente en la industria, donde los errores en la producción deben ser identificados lo antes posible. En esta tesis, realizamos la detección de anomalías en un proceso real de tratamiento térmico por láser utilizado en la industria automovilística. Se proponen dos enfoques diferentes para detectar las anomalías. En el primer enfoque, se rastrea el movimiento del láser, por lo que los datos originales se transforman de una alta dimensionalidad a baja dimensionalidad. A continuación, se utiliza una retícula de modelos no paramétricos para detectar las anomalías. El segundo enfoque modela directamente los datos originales de alta dimensionalidad utilizando redes Bayesianas semiparamétricas. Ambos enfoques tienen en cuenta las características temporales de los datos y muestran capacidades

prometedoras para detectar anomalías.

Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

δ Dirac delta function. 16

γ Minimum number of positions required to generate a KDE model. 126

λ Patience parameter. 68

$\boldsymbol{\nabla}^r f$ r-th partial derivatives of function f. 35

$\hat{\boldsymbol{\Sigma}}$ Sample covariance. 33

$\boldsymbol{\theta}$ Bayesian network parameters. 40

A Set of Bayesian network arcs. 11

$\mathcal{B}$ Bayesian network. 40

C Number of columns for the KDE-AMD. 125

c Number of continuous variables/nodes. 11

$\mathcal{D}$ The representation of a dataset. 11

d Number of discrete variables/nodes. 11

$\mathbf{E}$ Set of movements. 125

$f(x)$ A (usually unknown) probability density function. 14

$\mathcal{G}$ A directed acyclic graph. 11

$\hat{f}(x)$ An estimation of the underlying probability density function. 14

h Bandwidth. 24

$\mathbf{H}$ Bandwidth matrix. 24

$I_R(c)$ Indicator function. 16

$\mathcal{I}$ Cross-validation indices. 67

K Kernel function. 24

K^P Product kernel function. 24

K^S Spherically symmetric kernel function. 25

$\mathcal{L}$ Log-likelihood. 44

M A movement. 124

$\mathcal{N}$ Normal probability density function. 16, 92

N Number of instances of a data set. 11

n Number of variables/nodes. 11

R Number of rows for the KDE-AMD. 125

s Laser spot position. 124

$\mathcal{S}$ Score function. 48

Sig Sigmoid function. 74

T Number of temporal slices. 103

V Set of Bayesian network nodes. 11

W Number of temporal windows. 125

AE Absolute error. 14

AKDE Adaptive kernel density estimation. 38

AMISE Asymptotic mean integrated squared error. 15

AMSE Asymptotic mean squared error. 35

ASH Average shifted histogram. 20

AUC Area under the curve. 134

BCV Biased cross-validation. 34

BIC Bayesian information criterion. 48

BPFI Ball pass frequency inner. 82

BPFO Ball pass frequency outer. 82

BSF Ball spin frequency. 82

CKDE Conditional kernel density estimation. 62

CLG Conditional linear Gaussian. 44

CLGBN Conditional linear Gaussian Bayesian network. 42

CPC Candidate parent children. 51

CPD Conditional probability distribution. 40

CPT Conditional probability table. 41

DAG Directed acyclic graph. 11

DMMHC Dynamic max-min hill-climbing. 108

DSPBN Dynamic semiparametric Bayesian network. 140

FP-ASH Average shifted frequency polygon. 20

fps Frames per second. 117

FTF Fundamental train frequency. 82

GAN Generative adversarial network. 57

GBN Gaussian Bayesian network. 42

HC Greedy hill-climbing. 49

HCKDE Hybrid conditional kernel density estimation. 89

HMD Hamming distance. 74

HMM Hidden Markov model. 82

HSPBN Hybrid semiparametric Bayesian network. 88

IAE Integrated absolute error. 15

IIoT Industrial internet of things. 117

ISE Integrated squared error. 15

KDE Kernel density estimation. 23

KDE-AMD KDE-anomaly movement detector. 122

KDEBN Kernel density estimation Bayesian network. 43

LG Linear Gaussian. 42

MAE Mean absolute error. 14

MDL Minimum description length. 48

MGDAG Mixtures of Gaussian DAG. 42

MIAE Mean integrated absolute error. 15

MISE Mean integrated squared error. 15

MIT Mutual information tests. 48

MLE Maximum likelihood estimate. 44

MMHC Max-min hill-climbing. 51

MMPC Max-min parent & children. 51

MoP Mixture of polynomials. 43

MoTBF Mixture of truncated basis functions. 43

MSE Mean squared error. 14

MTE Mixture of truncated exponentials. 43

PDAG Partially directed acyclic graph. 70

PDF Probability density function. 14

PLC Partial linear correlation. 50

RCoT Randomized conditional correlation test. 71

ROC Receiver operating characteristic. 129

ROI Region of interest. 140

SE Squared error. 14

SHD Structural Hamming distance. 74

SPBN Semiparametric Bayesian network. 61

TDB Tweedie Bayesian network. 42

THMD Node type Hamming distance. 74

UCV Unbiased cross-validation. 33

Part I

INTRODUCTION

Chapter 1

Introduction

Machine learning is focused on creating models from data. These models try to capture the important patterns existing on the data. The data usually comes from real world events that we are interested in studying and an underlying objective is also to obtain information about the real world. For this reason, machine learning has drawn the attention of many other fields of science [Bielza and Larrañaga, 2020] and technology [Larrañaga et al., 2018]. There are many objectives that can be pursued while using machine learning: clustering [Xu and Tian, 2015], classification [Kotsiantis, 2007], feature subset selection [Li et al., 2017; Chandrashekar and Sahin, 2014], anomaly detection [Chandola et al., 2009], density estimation [Scott, 2015], etc.

Many techniques have been developed to solve these tasks and they include a variety of models and algorithms. Many of these techniques combine contributions from other fields such as probability, statistics, optimization; and are also inspired from such different fields as biology [Holland, 1992], neuroscience [Rosenblatt, 1958], metallurgy [Kirkpatrick et al., 1987] and physics [Ackley et al., 1985]. An important problem while dealing with data is the existence of uncertainty, which is specially common in real world data. Much research has been carried out in models that manage the uncertainty.

Bayesian networks are a probabilistic graphical model that uses concepts of the probability theory, so it has a sound foundation to deal with uncertainty. In addition, Bayesian networks have an intuitive representation in form of a graph. This representation is useful to make explicit the patterns that describe the data. Moreover, a versatile inference procedure can be executed in a Bayesian network, so the model can answer multiple different types of queries. In that way, Bayesian networks can be very flexible models that have been adapted to solve many different tasks: clustering [Pham and Ruz, 2009], classification [Bielza and Larrañaga, 2014] or feature subset selection [Inza et al., 2001]. In this thesis, we take advantage of Bayesian networks to represent probability distributions in a sparser way than other alternative representations. This sparsity is a consequence of the factorization of the joint probability distribution into a set of conditional probability distributions.

In most cases, these conditional probability distributions are represented using parametric models [Koller and Friedman, 2009; Geiger and Heckerman, 1994]. A parametric model has

a fixed set of parameters, which usually reduces the complexity of the representation. Nevertheless, a parametric model is restricted to characterize a family of probability distributions. This approach might be suboptimal if no member of that family can model correctly the uncertainty of the data.

Nonparametric models [Silverman, 1986; Scott, 2015; Wand and Jones, 1994] do not have a fixed set of parameters, and usually make a fewer number of assumptions than parametric models. In many cases, the probability distribution emerges directly from the training data. Thus, it can define a much richer set of probability distributions. Nonparametric models are an interesting alternative when the data is too complex to be represented by a parametric model. For this reason, they are an important tool to make more robust models. Parametric models usually have other drawbacks, such as the curse of dimensionality [Scott, 2015]. That is, the performance of the model decreases rapidly when the number of variables increases. In this thesis, we combine parametric with nonparametric models in the Bayesian network framework to try to overcome the weaknesses of both parametric and nonparametric models.

In the last few years, there have been many advances in the anomaly detection task [Chandola et al., 2009]. In anomaly detection, we are interested in finding the subsets of the data that do not have the expected or usual characteristics in the normal data. Anomaly detection has been successfully applied in many different contexts [Martí et al., 2015; Ahmed et al., 2016; Fernando et al., 2021] and many different specific techniques have been developed [Chandola et al., 2009]. For many probabilistic models the anomaly detection task is solved naturally. They estimate the probability distribution of the expected or normal data. Then, the anomalies can be easily found because the probability assigned to these anomalous data is extremely low. In this thesis, we propose models that deal with anomaly detection in industrial productive processes. In this sense, this work is framed within the Industry 4.0 paradigm [Drath and Horch, 2014; Larrañaga et al., 2018], where many new technologies are applied to improve the productivity and reliability of the industrial processes.

1.1 Hypotheses and Objectives

This thesis is motivated by the following research hypotheses:

- It is possible to combine nonparametric models and Bayesian networks to define a new class of Bayesian networks that mix parametric and nonparametric models to represent their conditional probability distributions. Thus, this class of Bayesian networks has the possibility of discarding parametric assumptions when they are not satisfied, using nonparametric models.
- The previously defined class of Bayesian networks can be automatically learned from data, and provides competitive results with respect to state-of-the-art Bayesian networks using standard criteria of comparison in the literature.
- Nonparametric models and Bayesian networks can be used to detect anomalies in data extracted from industrial applications.

Based on the stated hypotheses, the main objectives of this thesis are as follows:

- To define a new class of Bayesian networks, and the form of their conditional probability distributions.
- To propose learning algorithms that automatically find the structure and the parameters of this new class of Bayesian networks.
- To implement the new class of Bayesian networks and the state-of-the-art Bayesian networks in a Python package.
- To develop new methodologies for discovering anomalies in a real laser heat-treatment process, which take advantage of the work in the previous objectives.

1.2 Document Organization

The thesis is divided into five parts and a total of 11 chapters, organized as follows:

- **Part I. Introduction**

  This part serves as an introduction to the thesis.

  - Chapter 1 introduces the thesis and the research hypotheses and objectives.

- **Part II. Background**

  This part presents the notation and the necessary background with a description of the state-of-the-art techniques.

  - Chapter 2 presents the notation and terminology that will be used throughout all the dissertation.
  - Chapter 3 introduces nonparametric models and details some of the most important density estimation techniques.
  - Chapter 4 describes Bayesian networks and the most relevant related topics used in the thesis.
  - Chapter 5 presents the anomaly detection task, and the techniques most related to this thesis.

- **Part III. Contributions to Bayesian Networks**

  This part describes the contributions on Bayesian networks and their relation with nonparametric models.

  - Chapter 6 introduces a new class of Bayesian networks called semiparametric Bayesian networks. This class generalizes other classes of parametric and nonparametric Bayesian networks in the state of the art.

  - Chapter 7 proposes an improvement to the semiparametric Bayesian networks that allows modeling hybrid data, i.e., continuous and discrete data.
  - Chapter 8 presents a Python package called `PyBNesian` for Bayesian networks, which can be easily extended so new research can be quickly developed. It implements many different types of Bayesian network models, including the previously proposed new class of Bayesian networks.

- **Part IV. Contributions to Anomaly Detection**

  This part includes the contributions to anomaly detection during the development of the thesis.

  - Chapter 9 presents an application of nonparametric models to anomaly detection on a laser heat-treatment process. For this, it proposes a model called Kernel Density Estimation-Anomaly Movement Detection (KDE-AMD) that takes advantage of the spatio-temporal characteristics of the data.
  - Chapter 10 proposes dynamic semiparametric Bayesian networks to detect anomalies in the same laser heat-treatment process as Chapter 9. For this purpose, dynamic semiparametric Bayesian networks model the distribution of pixel values from video recordings of the laser heat-treatment process.

- **Part V. Conclusions**

  This part concludes the dissertation and presents future research lines.

  - Chapter 11 summarizes the main contributions in this thesis and outlines the possible future work.

Part II

BACKGROUND

Chapter 2

Notation and Terminology

We denote with capital letters, e.g., X, a random variable, while using the boldface version to represent random variable vectors, e.g., $\mathbf{X} = (X_1, \ldots, X_n)$. A subscript is used to index an element or set of elements, e.g., X_i is the i-th element of $\mathbf{X}$ and $\mathbf{X}_S$, with $S \subseteq \{1, \ldots, n\}$, selects the set S of indices from a vector $\mathbf{X}$ of n variables. The instantation of random variables is denoted using lowercase letters, e.g., $\mathbf{x}$, x_i or $\mathbf{x}_S$. The letter Y is used to represent a continuous random variable, while $\mathbf{Y}$ denotes a vector of continuous random variables. Similarly, the letter Z represents a discrete random variable, and $\mathbf{Z}$ a vector of discrete random variables. $\mathbf{X}$ may contain a combination of continuous and discrete random variables, so $\mathbf{X} = (\mathbf{Y}, \mathbf{Z})$. The domain of the discrete variable Z_i is denoted $\mathbf{\Omega}_i$, and the domain of the discrete variable $\mathbf{Z}_S$ is $\mathbf{\Omega}_S = \bigtimes_{i \in S} \mathbf{\Omega}_i$. A dataset is denoted with $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$, which contains N different instances. A dataset can also be indexed by indices, $\mathcal{D}_i = \{x_i^1, \ldots, x_i^N\}$, and sets of indices, $\mathcal{D}_S = \{\mathbf{x}_S^1, \ldots, \mathbf{x}_S^N\}$, to select a subset of random variables. Also, a subset of the data with the instances having a discrete configuration $\mathbf{z}$ for the variables $\mathbf{Z}$ is denoted $\mathcal{D}_{\downarrow \mathbf{z}} = \{\mathbf{x}^i \mid \mathbf{z}^i = \mathbf{z}\}$. Multiple subset operators can be applied to a dataset, separated by commas, e.g., $\mathcal{D}_{S,\downarrow \mathbf{z}}$. The number of instances in a dataset $\mathcal{D}$ that satisfies the condition $\downarrow \mathbf{z}$ is represented $N[\mathbf{z}]$, i.e., $N[\mathbf{z}] = |\mathcal{D}_{\downarrow \mathbf{z}}|$.

For Bayesian networks and datasets $\mathcal{D}$, the number of variables is n, the number of discrete variables is d and the number of continuous variables is c, so $n = d + c$.

A directed acyclic graph (DAG) is represented by $\mathcal{G} = (V, A)$ with a set of nodes $V = \{1, \ldots, n\}$ and a set of arcs $A \subseteq V \times V$. The set of nodes V can index a vector of random variables, so $\mathbf{X}_V = \mathbf{X}$. The set of parents of a node i are denoted $\mathrm{Pa}(i)$, so the respective parent random variables are $\mathbf{X}_{\mathrm{Pa}(i)}$. In a similar way, $\mathrm{Pa}_{\mathbf{Y}}(i)$ and $\mathrm{Pa}_{\mathbf{Z}}(i)$ denote the parents of i which index continuous and discrete random variables, respectively. Thus, $\mathbf{X}_{\mathrm{Pa}_{\mathbf{Y}}(i)}$ is a vector of continuous random variables and $\mathbf{X}_{\mathrm{Pa}_{\mathbf{Z}}(i)}$ is a vector of discrete random variables, and $\mathrm{Pa}(i) = \mathrm{Pa}_{\mathbf{Y}}(i) \cup \mathrm{Pa}_{\mathbf{Z}}(i)$.

For temporal data, $\mathbf{X}^{(t)}$ denotes the random variable $\mathbf{X}$ at the temporal moment t. Also, the trajectory of random variables $\mathbf{X}$ from the temporal moment t_1 to t_2 is represented $\mathbf{X}^{(t_1:t_2)}$. The instantation of $\mathbf{X}^{(t)}$ can be expressed as $\mathbf{x}^{(t)}$. Note that in a dataset with

temporal data $\mathcal{D}$ this is usually denoted as $\mathbf{x}^t$, but we use the parentheses to emphasize the temporal characteristics of the data.

Chapter 3

Nonparametric Models

3.1 Introduction

Nonparametric estimation is a set of statistical techniques used to obtain nonparametric models that are fitted to some data. The nonparametric models differ from the parametric ones because the latter assume that the data follow a parametric probability distribution. Usually, a parametric probability distribution can be uniquely defined with a fixed set of parameters. Therefore, the aim in parametric estimation is to obtain a joint estimate for all the set of parameters that provides the best fit to the data. Contrary to what its name might suggest, a nonparametric model is not necessarily a model without parameters, but a model which does not have a fixed number of parameters. In fact, the number of parameters might be infinite. Usually, the complexity of the nonparametric model representation increases together with the increase of the data availability. Therefore, the functional form of a nonparametric model may adapt to the complexity of the data. In this sense, the nonparametric estimation is devoted to directly estimate the best fit to the data without being restricted to a parametric probability distribution and the form of its parameters.

These nonparametric models might be optimized to solve different types of problems: classification, regression, probability (density) estimation, etc. However, in this thesis we will focus our attention on the probability density estimation problem. The density estimation can also be closely related to the anomaly detection problem (see Chapter 5), which is also a topic of interest in this thesis.

The rest of the chapter is organized as follows: Section 3.2 describes the problem of probability density estimation. Section 3.3 presents kernel density estimation, which is the main nonparametric model used in this thesis, and the most important relevant concepts are discussed. The analysis of the error and some interesting asymptotic properties are provided in Section 3.3.1. Next, the possible kernel functions and some of its properties are analyzed in Section 3.3.2. One of the key aspects is the selection of the bandwidth parameter, which is introduced in Section 3.3.3. Finally, a variant of kernel density estimation is described in Section 3.4.

3.2 Probability Density Estimation

Probability density estimation, usually abbreviated as just density estimation, is the process of creating an estimate of the underlying probability density function (PDF) of some given data. Therefore, given a dataset $\mathcal{D}$ drawn from the underlying PDF $f(x)$, the objective is to obtain an estimate $\hat{f}(x)$ that minimizes the error with respect to $f(\mathbf{x})$.

3.2.1 Error Criteria

The error of the estimate can be measured with different error criteria. We can distinguish between pointwise error criteria and error criteria over the whole space.

There are two common pointwise error criteria:

- Mean absolute error (MAE) or L_1 error criterion:

$$\text{MAE}\{\hat{f}(\mathbf{x})\} = \mathbb{E}\left[\left|\hat{f}(\mathbf{x}) - f(\mathbf{x})\right|\right]. \tag{3.1}$$

- Mean squared error (MSE) or L_2 error criterion:

$$\text{MSE}\{\hat{f}(\mathbf{x})\} = \mathbb{E}\left[\left(\hat{f}(\mathbf{x}) - f(\mathbf{x})\right)^2\right]. \tag{3.2}$$

Note that $\hat{f}(\mathbf{x})$ is constructed with a particular set of N instances. Therefore, MAE and MSE average out over all the possible estimates $\hat{f}(\mathbf{x})$ constructed with all the possible realizations of N instances. The absolute error (AE), $\left|\hat{f}(\mathbf{x}) - f(\mathbf{x})\right|$, or the squared error (SE), $\left(\hat{f}(\mathbf{x}) - f(\mathbf{x})\right)^2$, can also be used as error criteria. However, MAE and MSE are usually more interesting to evaluate the goodness of an estimate $\hat{f}(\mathbf{x})$ because they are not dependent on the training dataset $\mathcal{D}$. Usually, it is preferable to work with MSE because there are many interesting results in the state of the art which assumes MSE. One of the interesting mathematical properties shows the relation of the MSE with the bias-variance decomposition [Hastie et al., 2009]:

$$\text{MSE}\{\hat{f}(\mathbf{x})\} = \left(\text{Bias}\{\hat{f}(\mathbf{x})\}\right)^2 + \text{Var}\{\hat{f}(\mathbf{x})\}, \tag{3.3}$$

where

$$\text{Bias}\{\hat{f}(\mathbf{x})\} = \mathbb{E}\left[\hat{f}(\mathbf{x})\right] - f(\mathbf{x}), \tag{3.4}$$

and

$$\text{Var}\{\hat{f}(\mathbf{x})\} = \mathbb{E}\left[\left(\hat{f}(\mathbf{x}) - \mathbb{E}\left[\hat{f}(\mathbf{x})\right]\right)^2\right]. \tag{3.5}$$

As we will show in Section 3.3, a good estimator usually finds the best tradeoff between bias and variance because usually it is hard to reduce both at the same time.

The drawback of both MAE and MSE is that they only evaluate the estimate $\hat{f}(\mathbf{x})$ pointwise, i.e., in a specific point $\mathbf{x}$ in the space. However, we are usually interested in evaluating the estimate $\hat{f}(\mathbf{x})$ in the whole space. To achieve this, a common procedure is to integrate the error over the whole space. This gives rise to many additional common error criteria:

- Integrated absolute error (IAE):

$$\text{IAE}\{\hat{f}\} = \int_{\mathbb{R}^n} \left|\hat{f}(\mathbf{x}) - f(\mathbf{x})\right| d\mathbf{x}. \tag{3.6}$$

- Integrated squared error (ISE):

$$\text{ISE}\{\hat{f}\} = \int_{\mathbb{R}^n} \left(\hat{f}(\mathbf{x}) - f(\mathbf{x})\right)^2 d\mathbf{x}. \tag{3.7}$$

- Mean integrated absolute error (MIAE):

$$\text{MIAE}\{\hat{f}\} = \mathbb{E}\left[\text{IAE}\{\hat{f}\}\right] = \int_{\mathbb{R}^n} \text{MAE}\{\hat{f}(\mathbf{x})\} d\mathbf{x}. \tag{3.8}$$

- Mean integrated squared error (MISE):

$$\text{MISE}\{\hat{f}\} = \mathbb{E}\left[\text{ISE}\{\hat{f}\}\right] = \int_{\mathbb{R}^n} \text{MSE}\{\hat{f}(\mathbf{x})\} d\mathbf{x}. \tag{3.9}$$

- L_∞ error criterion:

$$L_\infty\{\hat{f}\} = \sup_{\mathbf{x}} \left|\hat{f}(\mathbf{x}) - f(\mathbf{x})\right|. \tag{3.10}$$

The IAE and ISE error criteria are the integrated versions of AE and SE. As with the AE and SE, usually it is more interesting to obtain the expected value to take into account all the possible estimates $\hat{f}(\mathbf{x})$, which gives rise to the MIAE and MISE error criteria. Lastly, the L_∞ error criterion is an alternative to the L_1 and L_2 norms. Note that there are many other error criteria, such as the visual error estimate [Marron and Tsybakov, 1995], the Kullback-Leibler divergence [Hall, 1987] or the Hellinger distance [Ahmad and Mugdadi, 2006].

In some instances, an asymptotic approximation to the MISE, which is called AMISE (asymptotic MISE) is easier to calculate. The AMISE is an approximation to the MISE when $N \to \infty$. Most of the times the AMISE is constructed keeping only the dominant terms of the MISE expression.

3.2.2 Probability Density Estimation Techniques

The most basic form of probability estimation is the empirical probability distribution. The estimation for an univariate distribution function is equal to:

$$\hat{F}_{\text{empirical}}(x) = \frac{1}{N}\sum_{i=1}^{N} I_{(-\infty,x]}(x^i), \tag{3.11}$$

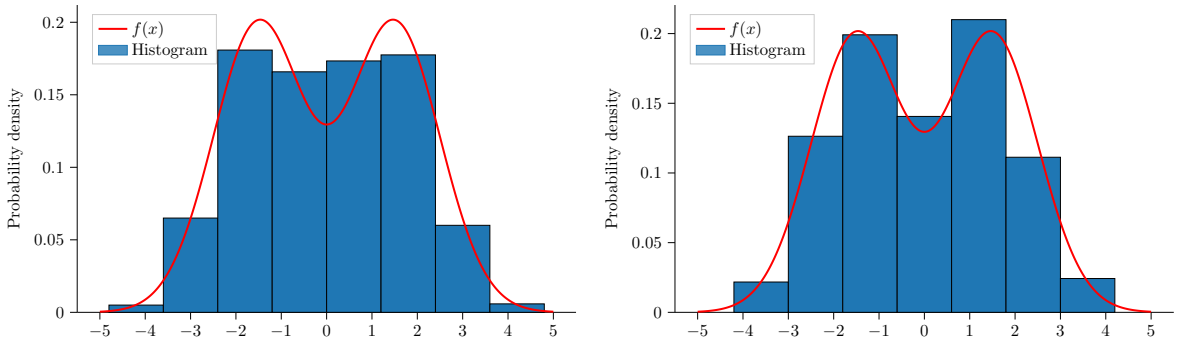
where $I_R(c)$ is the indicator function, which is equal to 1 if $c \in R$, otherwise it is equal to 0. Then, the empirical density estimate can be obtained deriving Equation (3.11):

$$\hat{f}_{\text{empirical}}(x) = \frac{1}{N}\sum_{i=1}^{N} \delta(x - x^i), \tag{3.12}$$

where δ is the Dirac delta function. Usually, this estimate is not very useful because the probability density is spread only over the training points. When estimating a continuous distribution, the probability that a test point x sampled from $f(x)$ being one of the training points is 0, $P_f(x = x^i) = 0, 1 \leq i \leq N$, so almost surely the empirical density function will return a 0 probability density estimate. Therefore, the empirical density function does not provide a meaningful estimate in most cases.

One of the oldest methods of probability density estimation is the histogram. The histogram groups the data into different nonoverlapping intervals of the space, which are usually called bins. When the histogram is normalized by the total number of training instances, the histogram can estimate the probability of some data. Grouping data dates back to at least 1662 in a work by John Graunt discussing the mortality and diseases in London. However, the Graunt's work was not devoted to probability density estimation. The term histogram was coined by Karl Pearson in some of his lectures on statistics in 1891 [Pearson, 1895].

To determine the estimation, the histogram needs to define two parameters: the bin origin point, t_0, and the bin width $h > 0$. The bin origin point defines the point where there is a bin boundary. Usually, $t_0 = 0$, but other models may use different bin origin points. The bin width defines the size of each bin, which is inversely proportional to the number of bins. Asymptotically, the bin origin point is known to have a negligible effect. However, this is not necessarily true for finite sample sizes. Figure 3.1 shows the histogram estimation of 1,000 instances of a normal mixture model $f(x) = 0.5 \cdot \mathcal{N}(-1.5, 1) + 0.5 \cdot \mathcal{N}(1.5, 1)$ using $t_0 = 0$ (in (a)) and $t_0 = 0.6$ (in (b)) with a fixed $h = 1.2$. In the former case, the estimated density appears to be unimodal, while in the latter case it shows a clear bimodal density. Later, we will review some techniques used to reduce the variability of the estimation produced by t_0. Figure 3.2 illustrates the effect of the bin width fixing t_0 and using different values for h: 0.1, 0.4, 0.7, 1, 1.3 and 1.8. Note that a lower h increases the variance of the estimation. It can clearly seen in Figure 3.2(a) where there is a lot of variation in the estimation between contiguous bins, so that points very close in space are given very different estimates. However, the estimate usually has a lower bias because the estimation can adapt to even abrupt changes of $f(x)$. This effect is called undersmoothing. However, when h is too large the histogram



(a) $t_0 = 0$ (b) $t_0 = 0.6$

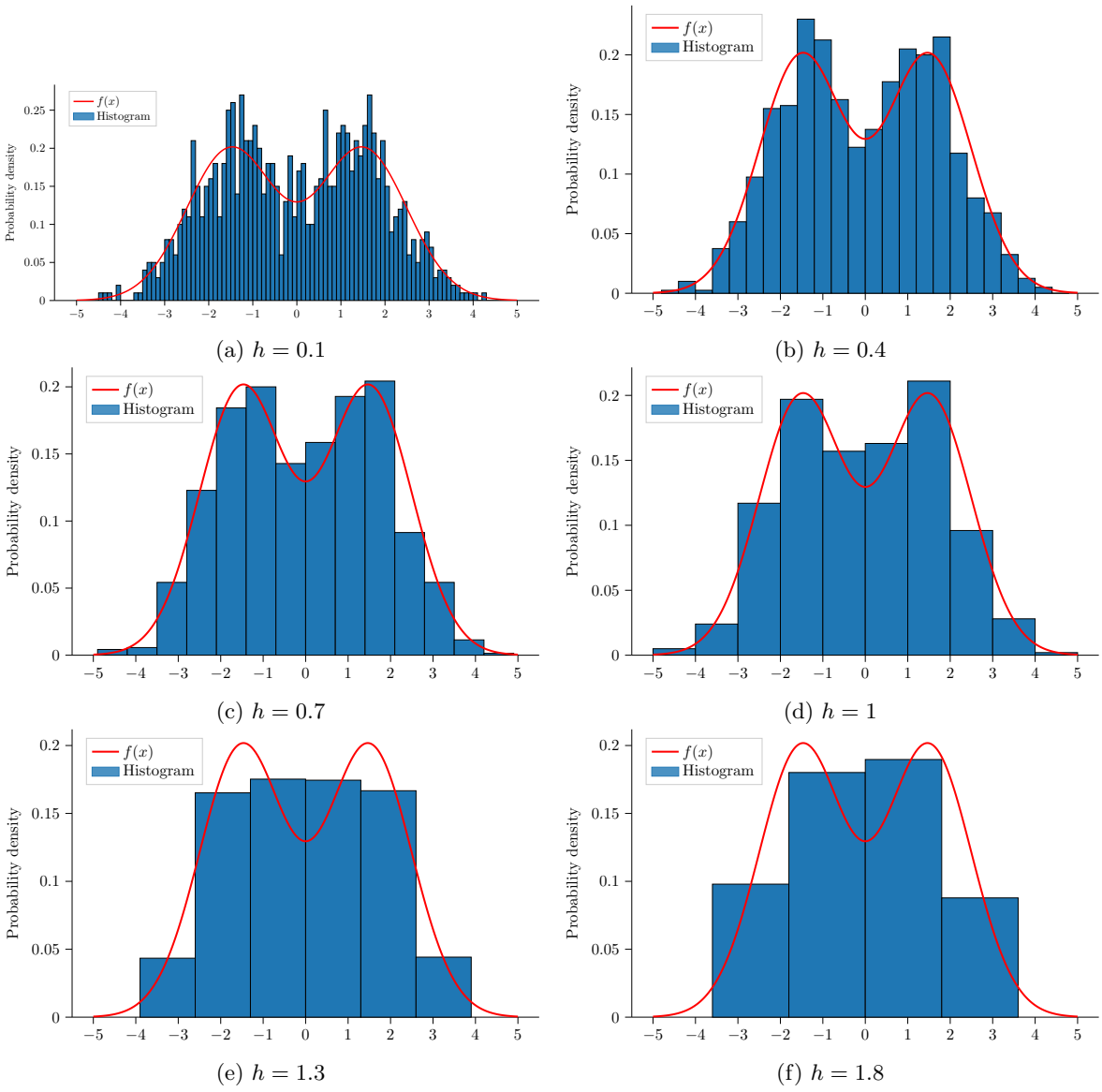
Figure 3.1: Histogram estimation for 1,000 instances of the normal mixture model $f(x) = 0.5 \cdot \mathcal{N}(-1.5, 1) + 0.5 \cdot \mathcal{N}(1.5, 1)$ (red line). The histogram is estimated with $h = 1.2$ for both figures and (a) $t_0 = 0$ or (b) $t_0 = 0.6$.

estimate cannot adapt to the $f(x)$ function trend, because the bins are spread over a large portion of the space. This is the case for Figure 3.2(e) and Figure 3.2(f), where the trend at the extremes and the middle of the space is not captured. Therefore, a large h reduces the variance and increases the bias of the estimation. This effect is called oversmoothing. Usually, the best estimation can be found in a tradeoff between very low and very high bin widths, which is also a good tradeoff between the variance and bias of the estimation.

The histogram estimation presents discontinuities in the bin borders, where the estimation is not very smooth, i.e., close points in the space can have very different estimated probability density. These discontinuities can be removed applying a linear interpolation between the bin estimates. This density estimation model is called frequency polygon. Therefore, a frequency polygon can be constructed by connecting the density estimate of the mid point of each bin. Figure 3.3 illustrates the construction of a frequency polygon and its corresponding histogram. The frequency polygon shows a smoother estimate than the histogram.

The histograms and frequency polygons can also model multivariate distributions by partitioning the space along all the different variables, usually into different hyperrectangles. For the bivariate case, other types of meshes have been proposed such as triangular and hexagonal partitions [Scott, 1988]. For the general multivariate case, a different bin width is used for each variable to take into account the different variability of each variable. The estimation of a multivariate distribution is much more difficult that the univariate case, especially when the dimensionality of the distribution is high. This is a problem called curse of dimensionality that we will also find for other models in Section 3.3.1.

For the histogram and frequency polygons, expressions to calculate the AMISE with respect to h, i.e., AMISE(h), have been developed. These expressions are different for histogram and frequency polygons. This also allows to find the optimal parameter h that minimizes the AMISE, i.e., $h^* = \arg\max_h \text{AMISE}(h)$. Also, the best possible AMISE can be found, i.e., $\text{AMISE}^* = \text{AMISE}(h^*)$. Moreover, if the unknown density function $f(x)$ satisfies some conditions, the frequency polygon can be shown to reduce the error faster than the histogram [Scott, 2015] as we will show below. The expressions for the univariate histogram



(a) $h = 0.1$ (b) $h = 0.4$

(c) $h = 0.7$ (d) $h = 1$

(e) $h = 1.3$ (f) $h = 1.8$

Figure 3.2: Histogram estimation for 1,000 instances of the normal mixture model $f(x) = 0.5 \cdot \mathcal{N}(-1.5, 1) + 0.5 \cdot \mathcal{N}(1.5, 1)$ (red line). The histogram is estimated with $t_0 = 0$ for all figures and (a) $h = 0.1$, (b) $h = 0.4$, (c) $h = 0.7$, (d) $h = 1$, (e) $h = 1.3$ and (f) $h = 1.8$.

are:

$$\begin{aligned} \text{AMISE}_{\text{hist}}(h) &= \frac{1}{Nh} + \frac{1}{12}h^2 R(f'), \\ h^*_{\text{hist}} &= \left(\frac{6}{R(f')}\right)^{1/3} N^{-1/3}, \\ \text{AMISE}^*_{\text{hist}} &= (3/4)^{2/3}\, R(f')^{1/3} N^{-2/3}, \end{aligned} \tag{3.13}$$

with $R(g) = \int_{\mathbb{R}} g(x)^2 dx$, and f' is the first derivative of f. The corresponding expressions for the frequency polygons are:

$$\begin{aligned} \text{AMISE}_{\text{FP}}(h) &= \frac{2}{3Nh} + \frac{49}{2880}h^4 R(f''), \\ h^*_{\text{FP}} &= 2\left(\frac{15}{49R(f'')}\right)^{1/5} N^{-1/5}, \\ \text{AMISE}^*_{\text{FP}} &= (5/12)\left(\frac{49R(f'')}{15}\right)^{1/5} N^{-4/5}, \end{aligned} \tag{3.14}$$

where f'' is the second derivative of f.

The AMISE* expressions in the univariate case for the histogram and the frequency polygon are of the order $O(N^{-2/3})$ [Scott, 1979] and $O(N^{-4/5})$ [Scott, 1985], respectively. This result shows that the AMISE* of the frequency polygon decreases faster than the AMISE* of the histogram with respect to the number N of instances. Note that in the AMISE(h), h^* and AMISE* expressions the ground truth density function derivatives $f'(x)$ and $f''(x)$ are needed. For this reason, these developments cannot be often used directly to compute the best h parameter because $f(x)$ and its derivatives are unknown. However, some estimation techniques of h have been proposed, by estimating the unknown terms in AMISE* [Scott and Terrell, 1987] or assuming a common density function $f(x)$ such as the Gaussian [Scott, 1979, 1985]. Previously, other methods were developed to calculate the bin width, such as the Sturges' rule [Sturges, 1926], which is a simple rule to estimate the best number of bins:

$$k = 1 + \log_2 N, \tag{3.15}$$

where k is the number of bins. Therefore, the assumed bin width is equal to $h = Q/k$, where Q is the range of the data. The Sturges' rule is also based on the assumption that the underlying PDF $f(x)$ is normal and using the binomial distribution as an approximation to the normal distribution.

The selection of the t_0 is known to have a negligible effect asymptotically. This can be seen easily in Equation (3.13) and Equation (3.14), where t_0 is not included in any of these expressions. However, as was demonstrated earlier in Figure 3.1, the effect is not negligible for finite sample sizes. An alternative technique to reduce the effect of t_0 is to construct multiple histogram/frequency polygons with different t_0 parameters and average the estimation of all of them. Usually, the t_0 is selected splitting the bin width length h into segments of equal size. Therefore, if m different histogram/frequency polygons are constructed, the separation

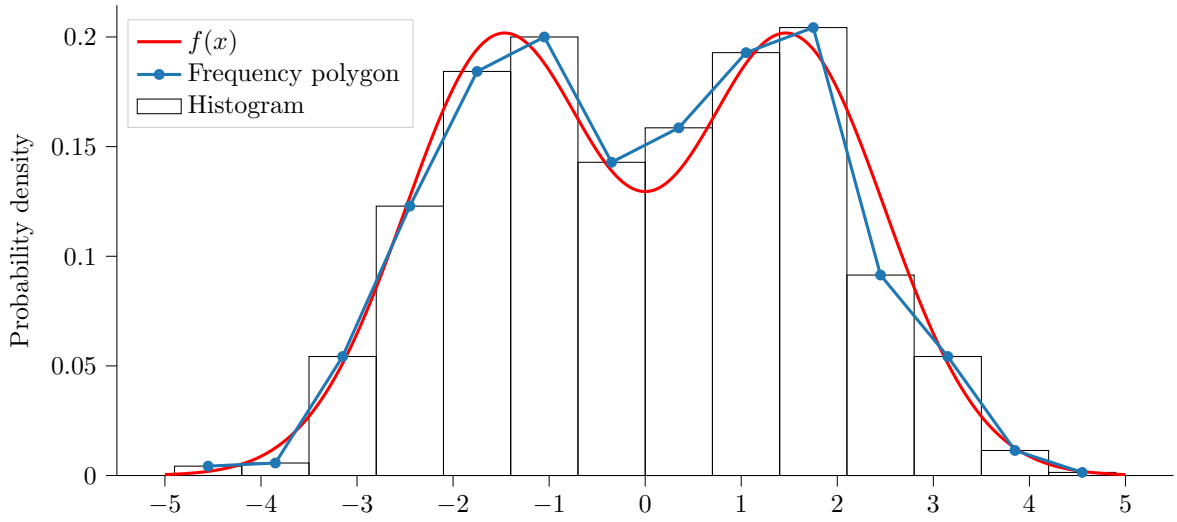


Figure 3.3: Frequency polygon estimation for 1,000 instances of the normal mixture model $f(x) = 0.5{\cdot}\mathcal{N}(-1.5, 1){+}0.5{\cdot}\mathcal{N}(1.5, 1)$ (red line). The frequency polygon is shown along with the corresponding histogram used to construct it ($t_0 = 0$ and $h = 0.7$).

between each t_0 is $\Delta = h/m$, and the selected t_0 points are $0, \Delta, 2\Delta, \ldots, (m-1)\Delta$. These type of models are called average shifted histogram (ASH) and average shifted frequency polygon (FP-ASH), respectively. Then, the expressions that define the estimate for ASH and FP-ASH are:

$$\begin{aligned} \hat{f}_{\text{ASH}}(x; h, m) &= \frac{1}{m} \sum_{i=0}^{m-1} \hat{f}_{\text{hist}}(x; i\Delta, h), \\ \hat{f}_{\text{FP-ASH}}(x; h, m) &= \frac{1}{m} \sum_{i=0}^{m-1} \hat{f}_{\text{FP}}(x; i\Delta, h), \end{aligned} \tag{3.16}$$

where $\hat{f}_{\text{hist}}(\cdot; t_0, h)$ and $\hat{f}_{\text{FP}}(\cdot; t_0, h)$ are the estimates for the histogram and frequency polygon constructed with parameters t_0 and h, respectively. Figure 3.4 illustrates the construction of an ASH from a set of 6 instances with $m = 3$ and $h = 0.3$. Each histogram is represented using dashed lines and the ASH estimate is represented with a red line.

All the nonparametric models described earlier estimate the density taking into account the locality of the data. To estimate the probability density of point x, the histogram only takes into account the instances that are at most $\frac{h}{2}$ away. The frequency polygon constructs a linear interpolation with the contiguous bins, so to estimate the probability density of point x, only the instances that are at most $\frac{3h}{2}$ away are considered because the height of the contiguous bins affect the linear interpolation. For the ASH, the estimate of a point x is constructed with the instances at a distance of at most h. However, the contribution of a training instance is weighted by the distance to x naturally, so the closer instances contribute more to the density estimate. Figure 3.5 illustrates this effect with the same data and ASH as in Figure 3.4. If x is the test point whose $f(x)$ is being estimated, the bin of histogram $t_0 = 0$ contains the instances x^4, x^5 and x^6; the bin of histogram $t_0 = 0.1$ contains the

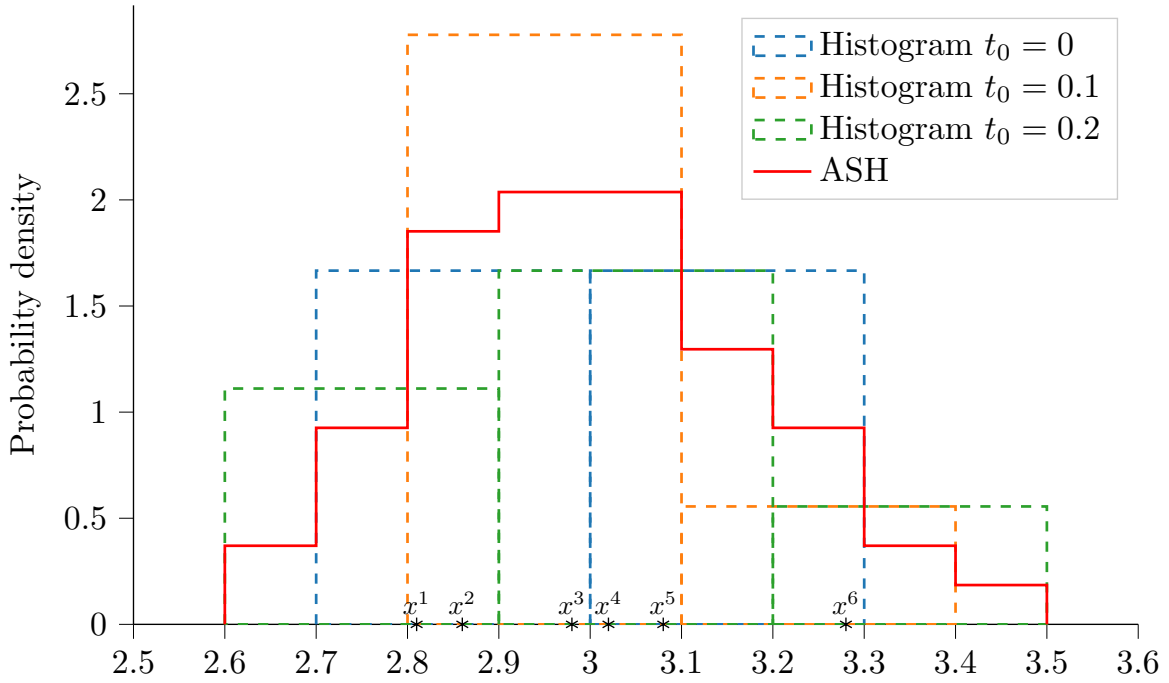


Figure 3.4: ASH constructed with $m = 3$ histograms with bin width $h = 0.3$. The training set is composed of 6 instances represented with markers on the horizontal axis. Each individual histogram is represented with dashed lines, and the ASH estimate is represented with a red line.

instances x^1, x^2, x^3, x^4 and x^5; and the bin of histogram $t_0 = 0.2$ contains the instances x^3, x^4 and x^5. Therefore, the instances x^4 and x^5, which are in the space Δ_0 (at a distance of at most Δ from any x), are contained in three histograms. The instance x^3, which is in the space Δ_{-1} (at a distance of at most 2Δ from any x), is contained in two histograms. Also, the instances x^1, x^2, which are in the space Δ_{-2} and the instance x^6 in Δ_2 (at a distance of at most $3\Delta = h$ from any x), are contained in just one histogram. All the instances that might be located outside the range $[2.8, 3.3]$ (corresponding to Δ_{-2} up to Δ_2) do not have any influence in the probability density estimation of x.

With this analysis, we could write the expression for an ASH as:

$$\begin{aligned}
\hat{f}_{\text{ASH}}(x; h, m) &= \frac{1}{m}\frac{1}{Nh}\sum_{j=1-m}^{m-1}(m - |j|)\cdot|\Delta_j| \\
&= \frac{1}{Nh}\sum_{j=1-m}^{m-1}\left(1 - \frac{|j|}{m}\right)\cdot|\Delta_j| \\
&= \frac{1}{Nh}\sum_{j=1-m}^{m-1}\left(1 - \frac{|j|\Delta}{m\Delta}\right)\cdot|\Delta_j| \\
&= \frac{1}{Nh}\sum_{j=1-m}^{m-1}\left(1 - \frac{|j|\Delta}{h}\right)\cdot|\Delta_j|,
\end{aligned} \tag{3.17}$$

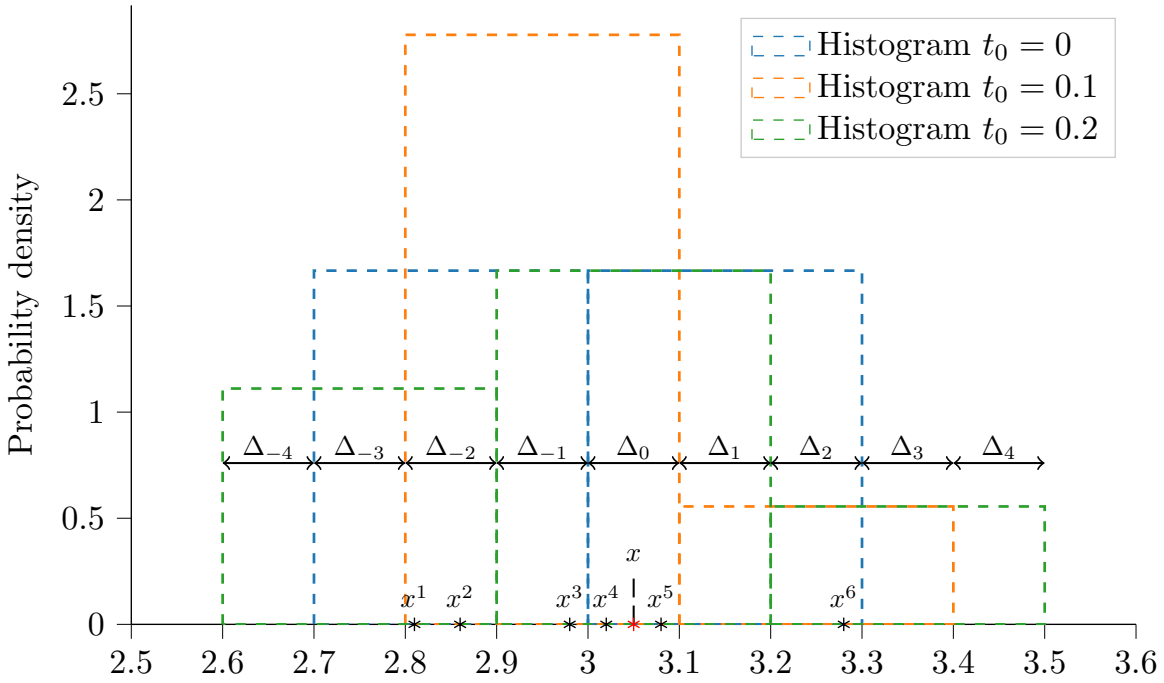


Figure 3.5: Local contribution of each instance to estimate the probability density of x in the ASH model of Figure 3.4. Points closer to x have a greater contribution to the estimate because they are contained in more bins in the individual histograms.

where $|\Delta_j|$ is the number of instances in the interval Δ_j.

This expression shows that each instance in the interval Δ_j contributes with a weight $(1 - |j|\Delta/h)$, where $|j|$ is the absolute value of j and $|j|\Delta$ is equal to the distance between the mid points of Δ_j and Δ_0. Therefore, we can also express the ASH estimate summing the contribution of every instance in the training set. For a training instance x^k in Δ_j, the distance between the mid points can be approximated by the following expression: $|j|\Delta = |x - x^k| + O(\Delta)$. Note that the approximation error $O(\Delta)$ is justified because there can be a maximum distance of $\Delta/2$ between x and the mid point of Δ_0, and a maximum distance of $\Delta/2$ between x^k and the mid point of Δ_j. In the limiting case where $m \to \infty$, then $\Delta \to 0$, so the approximation error tends to 0. Thus, the ASH for the limiting case $m \to \infty$ can be expressed as:

$$\begin{aligned} \lim_{m\to\infty} \hat{f}_{\text{ASH}}(x; h, m) &= \frac{1}{Nh} \sum_{i=1}^{N} \left(1 - \frac{|x - x^i|}{h}\right) I_{[0,h]}(|x - x^i|) \\ &= \frac{1}{Nh} \sum_{i=1}^{N} \left(1 - \frac{|x - x^i|}{h}\right) I_{[-1,1]}\left(\frac{x - x^i}{h}\right). \end{aligned} \tag{3.18}$$

The expression of the summand can be simplified using the following function:

$$K(t) = (1 - |t|)\, I_{[-1,1]}(t), \tag{3.19}$$

where K is known as the triangular kernel function. Then, the ASH estimate with $m \to \infty$ can be expressed as:

$$\lim_{m\to\infty} \hat{f}_{\text{ASH}}(x; h, m) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x^i}{h}\right). \tag{3.20}$$

This expresion is equal to a nonparametric model called kernel density estimation.

3.3 Kernel Density Estimation

The kernel density estimation (KDE) model, also called Parzen windows or kernel smoothers, is a type of nonparametric density estimation model that estimates the probability density taking into account the contribution of each training instance. The KDE models are also called Parzen windows because they are named after one of the first authors that introduced the KDE models [Parzen, 1962]. Rosenblatt [1956] estimates the density deriving the empirical distribution function (Equation (3.11)) using a finite central difference:

$$\hat{f}_{\text{difference}}(x) = \frac{\hat{F}_{\text{empirical}}(x + h) - \hat{F}_{\text{empirical}}(x - h)}{2h}, \tag{3.21}$$

where $h = h(N)$ is a parameter selected from a function $h(N)$ that depends on N, such that when $N \to \infty$, then $h \to 0$. Then, Rosenblatt [1956] showed that the estimator in Equation (3.21) can be represented with the following expression:

$$\hat{f}_{\text{difference}}(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x^i}{h}\right), \tag{3.22}$$

where K is the PDF of the uniform distribution ranging from -1 to 1:

$$K(x) = \begin{cases} \frac{1}{2} & \text{when } |x| \leq 1 \\ 0 & \text{otherwise.} \end{cases} \tag{3.23}$$

The $\hat{f}_{\text{difference}}$ estimator counts the number of training instances at a distance of at most h and normalizes the result dividing by N. It is easy to see the parallelism between the limiting case of an ASH, $\lim_{m\to\infty} \hat{f}_{\text{ASH}}(x; h, m)$, in Equation (3.20) and the $\hat{f}_{\text{difference}}(x)$ estimator in Equation (3.22). In both cases, only training instances at a distance from x of at most h have a positive contribution to the estimate. However, the K function in Equation (3.19) weights the contribution of a training instance x^i to the estimate by the distance between x and x^i, but the K function in Equation (3.23) does not. This example shows that the kernel function K can take different shapes.

Therefore, the probability density estimate of the univariate KDE model can be defined

as:

$$\hat{f}_{\text{KDE}}(x) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{x - x^i}{h}\right) = \frac{1}{N}\sum_{i=1}^{N} K_h\left(x - x^i\right), \tag{3.24}$$

where $K : \mathbb{R} \to \mathbb{R}$ is a kernel function whose integral in $\mathbb{R}$ is equal to 1, and $K_h(t) = \frac{1}{h}K(t/h)$ is the scaled kernel version of K. Moreover, h is a parameter called bandwidth, which has a role similar to the histogram bin width parameter, so the same letter is used to represent both parameters to highlight the parallelism.

Moreover, it is easy to see the similarity between the empirical density function in Equation (3.12) and the KDE estimate in Equation (3.24). The difference between these two expressions is that the KDE does not concentrate the contribution of a training instance x^i just to the point x^i, but it spreads this contribution in a local neighborhood around x^i of a size proportional to h. In this sense, we can understand the KDE as a kernelized form of the empirical distribution, which substitutes the Dirac delta function by a function K. Thus, the KDE model is sometimes called kernel smoother because it provides a smooth estimate, unlike the empirical density function.

The KDE model can be easily extended to deal with multivariate data, using a kernel function with a multivariate domain. The general multivariate KDE density estimate is:

$$\hat{f}_{\text{KDE}}(\mathbf{x}) = \frac{1}{N|\mathbf{H}|^{1/2}}\sum_{i=1}^{N} K\left(\mathbf{H}^{-1/2}(\mathbf{x} - \mathbf{x}^i)\right) = \frac{1}{N}\sum_{i=1}^{N} K_{\mathbf{H}}\left(\mathbf{x} - \mathbf{x}^i\right), \tag{3.25}$$

where $K : \mathbb{R}^n \to \mathbb{R}$ is a multivariate kernel function whose integral in $\mathbb{R}^n$ is equal to 1, and $K_{\mathbf{H}}(\mathbf{x}) = |\mathbf{H}|^{-1/2}K\left(\mathbf{H}^{-1/2}\mathbf{x}\right)$ is the scaled kernel version of K. $\mathbf{H}$ is a symmetric positive definite $n \times n$ matrix called bandwidth matrix, and is analogous to the bandwidth parameter h in the univariate case. The bandwidth matrix can be selected from different classes of matrices with different restrictions. The most basic form of bandwidth matrix is the scalar class, $\mathcal{S} = \{h^2\mathbf{I} : h > 0\}$, which is a diagonal matrix with the same smoothing value for each dimension ($\mathbf{I}$ is the identity matrix). This is a special case of the class of diagonal matrices $\mathcal{D} = \{\text{diag}(h_1^2, \ldots, h_n^2) : h_1, \ldots, h_n > 0\}$, where $\text{diag}(h_1^2, \ldots, h_n^2)$ is a diagonal matrix constructed with the parameters $h_1^2, \ldots, h_n^2$ in the diagonal. The class $\mathcal{D}$ can apply a different smoothing parameter to each dimension. Finally, the unconstrained class of symmetric positive definite matrices is denoted $\mathcal{F}$. This last class of bandwidth matrices can contain non-zero off-diagonal values, which can improve the estimate by taking into account the relationship between the variables.

There are two main ways to generate a kernel with multivariate domain from a univariate kernel. The product kernel is constructed in this way:

$$K^P(\mathbf{x}) = \prod_{i=1}^{n} K(x_i), \tag{3.26}$$

where the scaled version of the kernel is $K_{\mathbf{H}}^P(\mathbf{x}) = \prod_{j=1}^{n} \frac{1}{h_j}K\left(\frac{x_j}{h_j}\right)$. This type of kernel gives

rise to a product kernel density estimator:

$$\hat{f}_{\text{product}}(\mathbf{x}) = \frac{1}{N|\mathbf{H}|^{1/2}} \sum_{i=1}^{N} K^P(\mathbf{H}^{-1/2}(\mathbf{x} - \mathbf{x}^i)) = \frac{1}{N \prod_{j=1}^{n} h_j} \sum_{i=1}^{N} \prod_{j=1}^{n} K\left(\frac{x_j - x_j^i}{h_j}\right), \quad (3.27)$$

where $\mathbf{H}$ belongs to $\mathscr{S}$ or $\mathscr{D}$. The latter equality in Equation (3.27) is a specific expression when $\mathbf{H} \in \mathscr{D}$. When $\mathbf{H}$ belongs to $\mathscr{S}$, the expression can be simplified as:

$$\hat{f}_{\text{product}}(\mathbf{x}) = \frac{1}{Nh^n} \sum_{i=1}^{N} \prod_{j=1}^{n} K\left(\frac{x_j - x_j^i}{h}\right). \quad (3.28)$$

A multivariate kernel can be alternatively constructed using the following expression:

$$K^S(\mathbf{x}) = \kappa K\left(\left[\mathbf{x}^T\mathbf{x}\right]^{1/2}\right), \quad (3.29)$$

where $\kappa^{-1} = \int_{\mathbb{R}^n} K^S(\mathbf{x})d\mathbf{x}$ is a normalization constant that makes K^S integrate to 1. The K^S kernels are called spherically symmetric kernels. The scaled version of the kernel is $K^S_{\mathbf{H}}(\mathbf{x}) = \kappa|\mathbf{H}|^{-1/2}K(\mathbf{H}^{-1/2}\mathbf{x})$. The spherically symmetric kernels admits bandwith matrices of classes $\mathscr{S}$, $\mathscr{D}$ and $\mathscr{F}$.

Note that, in general, K^P is not equal to K^S even when a bandwidth from the classes $\mathscr{S}$ and $\mathscr{D}$ is selected. The Gaussian kernel is a notable exception, where the product kernel and the spherically symmetric kernel with bandwidth $\mathbf{H}$ in $\mathscr{S}$ or $\mathscr{D}$ return the same estimate.

3.3.1 Consistency and Asymptotic Analysis

From the first works of KDE, the authors tried to estimate how good the $\hat{f}_{\text{KDE}}$ can estimate f. In this sense, some work has been devoted to demonstrate the consistency of the KDE estimator. Wied and Weißbach [2012] offer a good summary of the consistency properties that have been derived for KDEs. The different forms of consistency that have been studied in the literature are as follows:

Definition 3.1. (Weak pointwise consistency). Let f be a continuous PDF. Then, at each x of the f domain, let also $\hat{f}_N(x)$ be an estimator of $f(x)$ based on N samples. The estimator $\hat{f}_N(x)$ is weakly pointwise consistent if $\hat{f}_N(x) \to f(x)$ in probability as $N \to \infty$, i.e., for each $\epsilon > 0$,

$$\lim_{N\to\infty} P\left(|\hat{f}_N(x) - f(x)| > \epsilon\right) = 0. \quad (3.30)$$

Definition 3.2. (Mean square consistency). Let f be a continuous PDF. Then, at each x of the f domain, let also $\hat{f}_N(x)$ be an estimator of $f(x)$ based on N samples. The estimator $\hat{f}_N(x)$ is consistent in the mean square if $\text{MSE}\{\hat{f}_N(x)\} \to 0$ as $N \to \infty$.

If an estimator is consistent in the mean square, then it is weakly pointwise consistent.

Definition 3.3. (Strong pointwise consistency). Let f be a continuous PDF. Then, at each x of the f domain, let also $\hat{f}_N(x)$ be an estimator of $f(x)$ based on N samples. The estimator $\hat{f}_N(x)$ is strongly pointwise consistent if $\hat{f}_N(x) \to f(x)$ almost surely as $N \to \infty$, i.e.,

$$P\left(\lim_{N\to\infty} \hat{f}_N(x) = f(x)\right) = 1. \tag{3.31}$$

Definition 3.4. (Strong uniform consistency). Let f be a continuous PDF, and $\hat{f}_N(x)$ be an estimator of $f(x)$ based on N samples, respectively. The estimator $\hat{f}_N(x)$ is strongly uniformly consistent if:

$$P\left(\lim_{N\to\infty} \sup_{-\infty<x<\infty} |\hat{f}_N(x) - f(x)| = 0\right) = P\left(\lim_{N\to\infty} \left\|\hat{f}_N(x) - f(x)\right\|_\infty = 0\right) = 1, \tag{3.32}$$

where $\|\cdot\|_\infty$ is the Chebyshev norm.

Parzen [1962] demonstrated that a KDE model is consistent in the mean square (and weakly pointwise consistent). To obtain this result, he introduced the following set of assumptions on the kernel function K:

$$\begin{aligned} &\text{(i)} \quad \int_{-\infty}^{\infty} |K(x)|dx < \infty, \\ &\text{(ii)} \quad \int_{-\infty}^{\infty} K(x)dx = 1, \\ &\text{(iii)} \quad \lim_{x\to\infty} |xK(x)| = 0. \end{aligned} \tag{3.33}$$

In addition, the bandwidth selection function must fullfil the following set of conditions:

$$\begin{aligned} &\text{(i)} \quad \lim_{N\to\infty} h(N) = 0, \\ &\text{(ii)} \quad \lim_{N\to\infty} Nh(N) = \infty. \end{aligned} \tag{3.34}$$

Strong pointwise consistency of a KDE model was demonstrated by Ryzin [1969] and Devroye and Penrod [1984]. The proofs for strong pointwise consistency requires adding more assumptions to the weakly pointwise consistency assumptions. At the same time, Devroye and Penrod [1984] and Devroye [1983] demonstrated the convergence of the IAE (Equation (3.6)), that is, $\int_{\mathbb{R}^n} \left|\hat{f}(x) - f(x)\right| dx \to 0$ almost surely as $N \to \infty$.

The strong uniform consistency of the KDE model has been studied by many authors: Parzen [1962], Nadaraya [1965], Ryzin [1969] and Silverman [1978]. These proofs also require additional assumptions about the kernel function and the bandwidth selection function.

These results show that the KDE model performs well in the limiting case where $N \to \infty$. Although these properties are not demonstrated for finite sample sizes, they highlight the usefulness of the KDE model and suggest that it is especially well suited for large sample sizes.

For the KDE model, expressions for the MISE and the AMISE have been developed.

Rosenblatt [1956] found the expression for these error measures assuming the uniform kernel function of Equation (3.23). For the general univariate KDE model, the AMISE(h), h^*, and AMISE* expressions can be found in Scott [2015]:

$$\begin{aligned} \text{AMISE}_{\text{KDE}}(h) &= \frac{R(K)}{Nh} + \frac{1}{4}m_2(K)^2 h^4 R(f''), \\ h^*_{\text{KDE}} &= \left(\frac{R(K)}{m_2(K)^2 R(f'')}\right)^{1/5} N^{-1/5}, \\ \text{AMISE}^*_{\text{KDE}} &= \frac{5}{4}\left(m_2(K)^{1/2} R(K)\right)^{4/5} R(f'')^{1/5} N^{-4/5}. \end{aligned} \tag{3.35}$$

where $m_p(K) = \int_{\mathbb{R}} x^p K(x)dx$. These expressions assume that the kernel function K satisfies (ii) in Equation (3.33) and also the following expressions:

$$\begin{aligned} &\text{(i)} \quad \int_{-\infty}^{\infty} xK(x)dx = 0, \\ &\text{(ii)} \quad m_2(K) > 0. \end{aligned} \tag{3.36}$$

The assumption (i) in Equation (3.36) is usually satisfied using a symmetric function, so it is common the use of symmetric kernel functions although the symmetry is not a necessary condition. The AMISE$^*_{\text{KDE}}$ expression in Equation (3.35) is of order $O(N^{-4/5})$, similar as the frequency polygon. As with the histogram and frequency polygon, the optimal bandwidth h^*_{KDE} contains an unknown term ($R(f'')$) related to the unknown function f. For this reason, the optimal bandwidth cannot be directly used to compute the best bandwidth estimate, unless f is known and $R(f'')$ can be calculated. In Section 3.3.3 we will review some of the most common techniques to select the bandwidth.

In the multivariate case, the AMISE for a bandwidth matrix $\mathbf{H}$ is equal to:

$$\text{AMISE}_{\text{KDE}}(\mathbf{H}) = N^{-1}|\mathbf{H}|^{1/2} R(K) + \frac{1}{4}m_2(K)^2 \int_{\mathbb{R}^n} \text{tr}\{\mathbf{H}\boldsymbol{\nabla}^2 f(\mathbf{x})\}^2 d\mathbf{x}, \tag{3.37}$$

where $R(K) = \int_{\mathbb{R}^n} K(\mathbf{x})^2 d\mathbf{x}$, tr$\{M\}$ is the trace of matrix M, and $\boldsymbol{\nabla}^2 f(\mathbf{x})$ is the Hessian matrix of $f(\mathbf{x})$. This expression assumes that the kernel function K has the following properties:

$$\begin{aligned} &\text{(i)} \quad \int_{\mathbb{R}^n} K(\mathbf{x})d\mathbf{x} = 1, \\ &\text{(ii)} \quad \int_{\mathbb{R}^n} \mathbf{x}K(\mathbf{x})d\mathbf{x} = 0, \\ &\text{(iii)} \quad \int_{\mathbb{R}^n} \mathbf{x}\mathbf{x}^T K(\mathbf{x})d\mathbf{x} = m_2(K)\mathbf{I}_n, \end{aligned} \tag{3.38}$$

where:

$$m_p(K) = \int_{\mathbb{R}^n} x_i^p K(\mathbf{x})d\mathbf{x} \quad \text{for all } i = 1, \ldots, n. \tag{3.39}$$

The AMISE$_{\text{KDE}}$ expression and the kernel function assumptions are analogous to the

univariate case. However, no closed form expression exists for the optimal bandwidth matrix $\mathbf{H}^*_{\text{KDE}}$ or the $\text{AMISE}^*_{\text{KDE}}$. Nevertheless, the $\mathbf{H}^*_{\text{KDE}}$ and the $\text{AMISE}^*_{\text{KDE}}$ are known to be of order $O\left(N^{-2/(n+4)}\right)$ and $O\left(N^{-4/(n+4)}\right)$, respectively [Chacón and Duong, 2018]. For the special case $n = 1$, these rates are the same as the univariate expressions in Equation (3.35). These results show that the convergence rate of the multivariate KDE model worsens with the increase in the dimensionality. This is a problem called the curse of dimensionality [Scott, 2015], which means that estimating multiviariate densities usually requires a larger sample size.

3.3.2 Kernel Selection

The kernel function K plays an important role in the performance of the KDE model. As we showed in Section 3.3.1, there are several properties that are desirable for the kernel function because they provide information about the consistency and the AMISE of the resulting KDE model.

Analyzing the $\text{AMISE}^*_{\text{KDE}}$ expression of Equation (3.35) it can be checked that the kernel function K contributes to the AMISE proportionally to:

$$\text{AMISE}^*_{\text{KDE}} \propto \left(m_2(K)^{1/2} R(K)\right)^{4/5}. \tag{3.40}$$

Therefore, optimizing the term in Equation (3.40) with respect to the kernel form of K, the optimal kernel function can be found. The kernel that minimizes this expression among the nonnegative kernels is known as the Epanechnikov kernel [Epanechnikov, 1969]:

$$K^*(x) = \frac{3}{4}(1 - x^2) I_{[-1,1]}(x). \tag{3.41}$$

This function has a finite support in $[-1, 1]$. Tsybakov [2008] argued against the claim that the Epanechnikov kernel is the optimal kernel, because other not nonnegative kernels can be considered in the optimization. Using kernels that might return negative values, can return a negative density estimate, which is not adequate for density estimation. In his exposition, Tsybakov [2008] proposed dealing with these negative values by clipping them, so the density estimate is $\hat{f}^+_{\text{KDE}} = \max\{0, \hat{f}_{\text{KDE}}\}$. However, this clipping may cause the integral of the KDE to be greater than 1. Solving this problem would require substituting the $\frac{1}{N}$ normalization factor in Equation (3.25) by $\left[\int_{\mathbb{R}^n} \max\{0, \sum_{i=1}^N K_h(x - x^i)\} dx\right]^{-1}$. As solving these problems can be cumbersome, most of the literature limits its analysis to nonnegative kernels, so the Epanechnikov kernel is considered the optimal kernel for most of the works.

Since we know the kernel that minimizes Equation (3.40), we can compare all the kernels with the optimal kernel:

$$\text{eff}(K) = \frac{m_2(K^*)^{1/2} R(K^*)}{m_2(K)^{1/2} R(K)} = \frac{3/(5\sqrt{5})}{m_2(K) R(K)}, \tag{3.42}$$

and this expression is known as the (relative) efficiency of kernel K. Table 3.1 shows several common univariate kernel functions used in KDE along with some of their properties,

including the efficiency. An interesting insight into the efficiency of a kernel K is that the number of instances needed to obtain the same $\text{AMISE}_{\text{KDE}}$ as the optimal kernel is $\text{eff}(K)^{-1}$ times higher. For example, the Gaussian kernel requires 1.051 times as many instances as the Epanechnikov kernel to converge to the same $\text{AMISE}_{\text{KDE}}$.

The analysis of the multivariate case is more complicated because the $\text{AMISE}^*_{\text{KDE}}$ is not available. However, it can be shown that the optimal kernel is the spherically symmetric Epanechnikov kernel and the efficiencies can be calculated for some kernels [Wand and Jones, 1994; Chacón and Duong, 2018].

In addition to the efficiency of the kernel, other aspects are taken into account. For example, the cosine kernel is infinitely differentiable, while the Epanechnikov kernel is not. If this property is needed, the difference in efficiency can be considered negligible. Also, some kernel functions involve less computational effort. The Gaussian kernel is equal to a Gaussian density function. Therefore, a KDE model with a Gaussian kernel is equivalent to a normal mixture model with equiprobable components where each component is located on each training instance. Since the Gaussian density function has many interesting properties, e.g., fast calculation of marginal and conditional distributions, or being infinitely differentiable, it is a common kernel function in the literature.

Previously, we discussed how some not nonnegative kernels could improve the error compared with the Epanechnikov kernel. This is the case for some type of kernels called higher order kernels. The order of a kernel can be defined as follows:

Definition 3.5. (Order of a kernel). A kernel K is of order p if the functions $x \to x^j K(x)$, $j = 0, \ldots, p$ are integrable and satisfy:

$$\begin{aligned} &\text{(i)} \quad \int_{\mathbb{R}} K(x)dx = 1, \\ &\text{(ii)} \quad m_j(K) = 0, \quad j = 1, \ldots, p-1, \\ &\text{(iii)} \quad m_p(K) \neq 0. \end{aligned} \tag{3.43}$$

The definition for multivariate kernels is analogous, so the kernel is integrated over $\mathbb{R}^n$ instead of $\mathbb{R}$, and uses the $m_p(K)$ in Equation (3.39).

All the kernels that we listed in Table 3.1 are of order 2. Reviewing the $\text{AMISE}_{\text{KDE}}$, we can see that the second term (corresponding to the asymptotic integrated squared bias) can be cancelled out using kernels with order $p > 2$. Note that the bias is not completely removed, but the dominant term of the asymptotic integrated squared bias changes. For a KDE model construted with kernels of higher order p (HO_p), the AMISE, h^* and AMISE^*

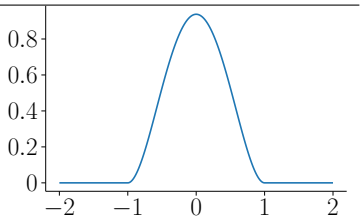
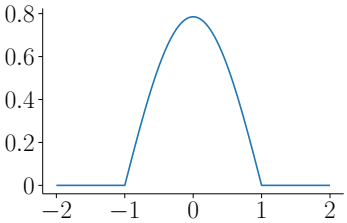
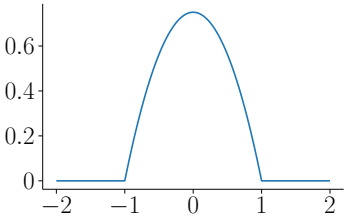
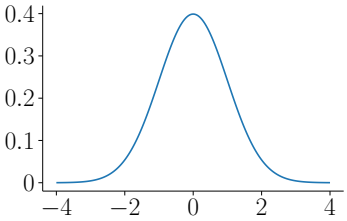
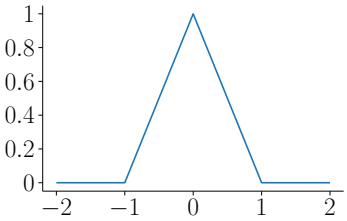
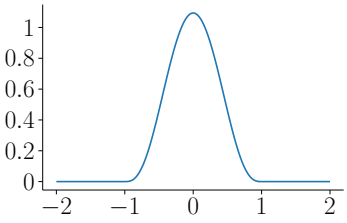
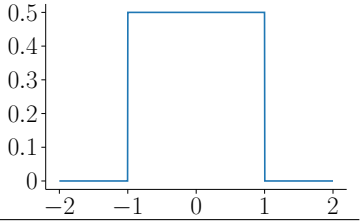
Name	$K(x)$	$m_2(K)$	$R(K)$	eff(K)	Shape
Biweight	$\frac{15}{16}(1-x^2)^2 I_{[-1,1]}(x)$	$\frac{1}{7}$	$\frac{5}{7}$	0.994	
Cosine	$\frac{\pi}{4}\cos\left(\frac{\pi}{2}x\right) I_{[-1,1]}(x)$	$1-\frac{8}{\pi^2}$	$\frac{\pi^2}{16}$	0.999	
Epanechnikov	$\frac{3}{4}(1-x^2) I_{[-1,1]}(x)$	$\frac{1}{5}$	$\frac{3}{5}$	1	
Gaussian	$\frac{1}{(2\pi)^{1/2}}\exp(-\frac{1}{2}x^2)$	1	$\frac{1}{2\pi^{1/2}}$	0.951	
Triangular	$(1-\lvert x\rvert) I_{[-1,1]}(x)$	$\frac{1}{6}$	$\frac{2}{3}$	0.986	
Triweight	$\frac{35}{32}(1-x^2)^3 I_{[-1,1]}(x)$	$\frac{1}{9}$	$\frac{350}{429}$	0.987	
Uniform	$\frac{1}{2} I_{[-1,1]}(x)$	$\frac{1}{3}$	$\frac{1}{2}$	0.930	

Table 3.1: Common kernels and their properties.

are:

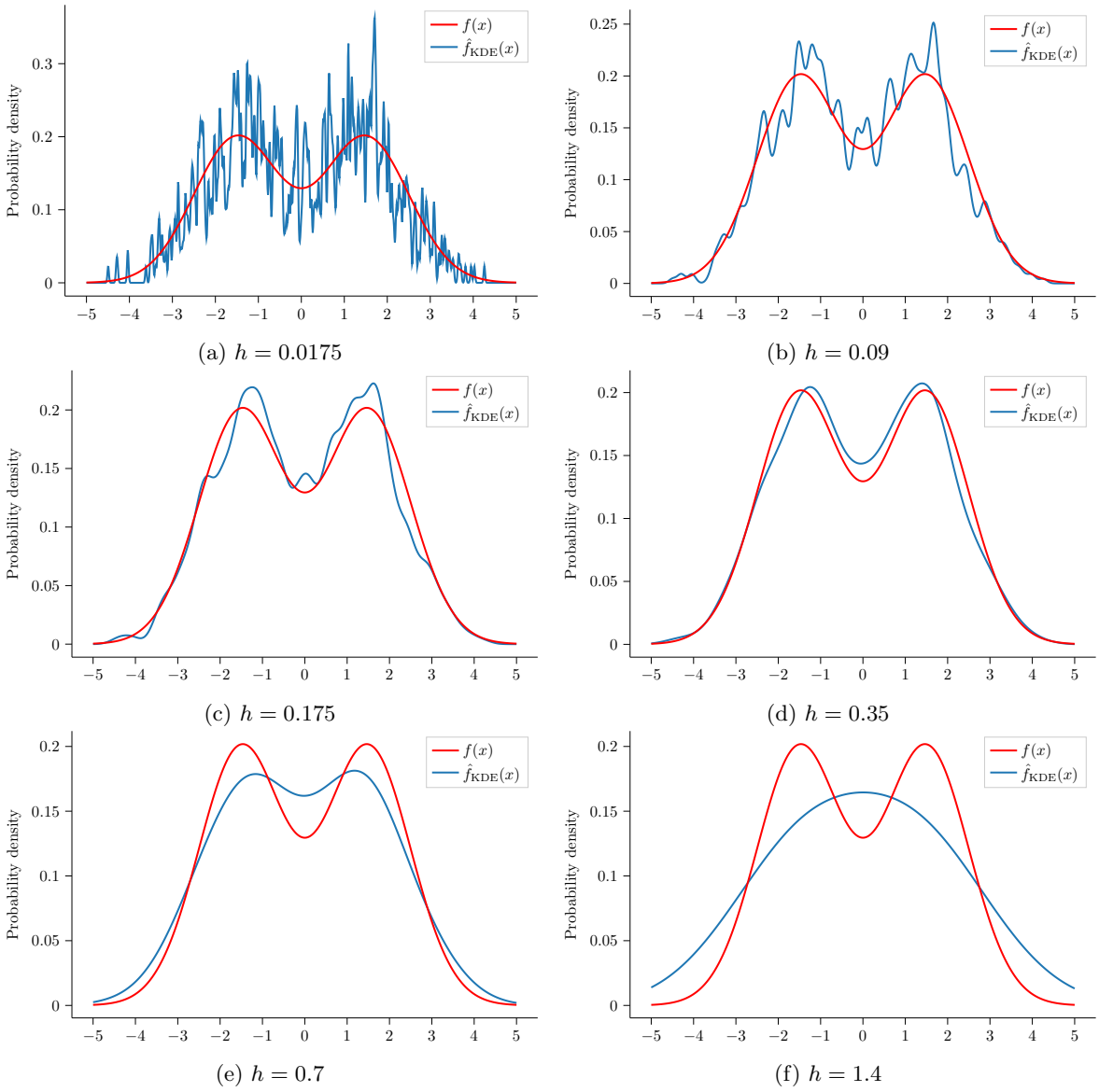
$$\begin{aligned}
\text{AMISE}_{\text{HO}_p}(h) &= \frac{R(K)}{Nh} + \frac{1}{(p!)^2} m_p(K)^2 h^{2p} R(f^{(p)}), \\
h^*_{\text{HO}_p} &= \left(\frac{(p!)^2 R(K)}{2p m_p(K)^2 R(f^{(p)})} \right)^{1/(2p+1)} N^{-1/(2p+1)}, \\
\text{AMISE}^*_{\text{HO}_p} &= \frac{2p+1}{2p} \left[2p m_p(K)^2 R(K)^{2p} R(f^{(p)}/(p!)^2) \right]^{1/(2p+1)} N^{-2p/(2p+1)}.
\end{aligned} \tag{3.44}$$

respectively. The $\text{AMISE}^*_{\text{HO}_p}$ is of order $O(N^{-2p/(2p+1)})$, which improves with the increase of the order p. In the limiting case $p \to \infty$, the best AMISE is of the order $O(N^{-1})$, which is a common rate of convergence for parametric models. However, these results are asymptotic and in practice the gains are not so important [Scott, 2015]. Moreover, this kernel may return negative values as commented above, with the drawbacks that we discussed.

3.3.3 Bandwidth Selection

As with the histogram, the bandwidth parameter h, has an important role in the resulting density estimate. A small bandwidth generates a density estimate that may fluctuate abruptly because each training instance only contributes to a small neighborhood. If the bandwidth is too small, this effect is very pronounced and it is also called undersmoothing. Conversely, a large bandwidth generates more stable estimates because the neighborhood of the kernel is larger. However, the KDE adapts more slowly to abrupt changes of f. This effect is also called oversmoothing. The undersmoothing is usually related to a low bias and high variance, while the oversmoothing causes high bias and low variance. This can be verified in the $\text{AMISE}_{\text{KDE}}$ expression of Equation (3.35). The first term corresponds to the asymptotic integrated variance, while the second term is the asymptotic integrated squared bias. It is easy to see that reducing the bandwidth generates larger variance and lower bias, while increasing the bandwidth generates the opposite effect. For this reason, the best error reduction is usually achieved by a compromise between variance and bias. This effect is illustrated in Figure 3.6 using a Gaussian kernel function for the KDE, and the same data generating function as in Figure 3.2, i.e., $f(x) = 0.5 \cdot \mathcal{N}(-1.5, 1) + 0.5 \cdot \mathcal{N}(1.5, 1)$. Figure 3.6(a) is an example of a clear undersmoothing, which makes density interpretation difficult because it is too noisy. Figure 3.6(f) exhibits a strong oversmoothing as the bimodality of f is completely lost. Figure 3.6(d) shows a pretty balanced bandwidth selection. Therefore, the bandwidth selection is a key component to obtain a good density estimate.

In practice, the true $f(x)$ is not known, so viable bandwidth selection methods have been developed in the literature without relying on its knowledge. Jones et al. [1996]; Scott [2015]; Wand and Jones [1994] and Chacón and Duong [2018] provide a comprehensive discussion on the most common bandwidth selection estimators in the literature. Next will review some of the most important ones.



(a) $h = 0.0175$ (b) $h = 0.09$

(c) $h = 0.175$ (d) $h = 0.35$

(e) $h = 0.7$ (f) $h = 1.4$

Figure 3.6: KDE for 1,000 instances of the normal mixture model $f(x) = 0.5 \cdot \mathcal{N}(-1.5, 1) + 0.5 \cdot \mathcal{N}(1.5, 1)$ (red line). The KDE is estimated with (a) $h = 0.0175$, (b) $h = 0.09$, (c) $h = 0.175$, (d) $h = 0.35$, (e) $h = 0.7$ and (f) $h = 1.4$ using a Gaussian kernel.

3.3.3.1 Normal Reference Rule

The h^*_{KDE} expression in Equation (3.35), has only an unknown term, which is $R(f'')$. If f is assumed to be normal, the bandwidth selection can be reduced to the following expression known as the normal reference rule:

$$\hat{h}_{\text{NR}} = (4/3)^{1/5} N^{-1/5} \hat{\boldsymbol{\Sigma}}, \tag{3.45}$$

where $\hat{\sigma}$ is the sample standard deviation. Wand [1992] extended this rule to the multivariate case as a special case of the assumption that f belongs to the class of distributions defined by a normal mixture model. Therefore, if f is a multivariate Gaussian distribution (which can be considered an special case of a normal mixture model):

$$\hat{\mathbf{H}}_{\text{NR}} = \left(\frac{4}{n+2}\right)^{2/(n+4)} N^{-2/(n+4)} \hat{\boldsymbol{\Sigma}}, \tag{3.46}$$

where $\hat{\boldsymbol{\Sigma}}$ is the sample covariance. Note that the univariate case is a special case of the multivariate formula. The first factor always takes values very close to 1. Therefore, Scott [2015] proposed the Scott's rule, which is a simplification of the normal reference rule:

$$\hat{\mathbf{H}}_{\text{Scott}} = N^{-2/(n+4)} \hat{\boldsymbol{\Sigma}}. \tag{3.47}$$

The normal reference bandwidths usually provide oversmoothed estimates. This seems reasonable as the Gaussian is the maximum entropy continuous distribution with support in $(-\infty, \infty)$.

3.3.3.2 Unbiased Cross-Validation

The unbiased cross-validation (UCV) [Rudemo, 1982], also known as least squares cross-validation, is based on minimizing the ISE, which can be expressed as follows:

$$\begin{aligned} \text{ISE} &= \int_{\mathbb{R}^n} \left(\hat{f}_{\text{KDE}}(\mathbf{x}) - f(\mathbf{x})\right)^2 d\mathbf{x} \\ &= \int_{\mathbb{R}^n} \hat{f}_{\text{KDE}}(\mathbf{x})^2 d\mathbf{x} - 2\int_{\mathbb{R}^n} \hat{f}_{\text{KDE}}(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} + \int_{\mathbb{R}^n} f(\mathbf{x})^2 d\mathbf{x}. \end{aligned} \tag{3.48}$$

The third term is not relevant for the optimization of the bandwidth, so it can be ignored. The second term is the expected value of $\hat{f}(\mathbf{x})$, i.e., $\mathbb{E}\left[\hat{f}(\mathbf{x})\right]$. Rudemo [1982] proposed estimating this term using a leave-one-out cross-validation. The KDE model defined without the $\mathbf{x}^k$ instance is:

$$\hat{f}^{-k}_{\text{KDE}}(\mathbf{x}) = \frac{1}{N-1} \sum_{i=1:i\neq k}^{N} K_{\mathbf{H}}(\mathbf{x} - \mathbf{x}^i). \tag{3.49}$$

Then, the UCV can be defined as:

$$\text{UCV}(\mathbf{H}) = R(\hat{f}_{\text{KDE}}) - \frac{2}{N}\sum_{i=1}^{N}\hat{f}_{\text{KDE}}^{-i}(\mathbf{x}^i). \tag{3.50}$$

The first term of this expression can be easily expanded as follows:

$$\begin{aligned} R(\hat{f}_{\text{KDE}}) &= \frac{1}{N^2}\int_{\mathbb{R}^n}\left[\sum_{i=1}^{N}K_{\mathbf{H}}(\mathbf{x}-\mathbf{x}^i)\right]^2 d\mathbf{x} \\ &= \frac{1}{N^2}\sum_{i=1}^{N}\int_{\mathbb{R}^n}K_{\mathbf{H}}(\mathbf{x}-\mathbf{x}^i)^2 d\mathbf{x} + \frac{1}{N^2}\sum_{i,j=1:i\neq j}^{N}\int_{\mathbb{R}^n}K_{\mathbf{H}}(\mathbf{x}-\mathbf{x}^i)K_{\mathbf{H}}(\mathbf{x}-\mathbf{x}^j)d\mathbf{x} \\ &= \frac{R(K)}{N|\mathbf{H}|^{1/2}} + \frac{1}{N^2}\sum_{i,j=1:i\neq j}^{N}\int_{\mathbb{R}^n}K_{\mathbf{H}}(\mathbf{w}+\mathbf{x}^j-\mathbf{x}^i)K_{\mathbf{H}}(\mathbf{w})d\mathbf{w} \\ &= \frac{R(K)}{N|\mathbf{H}|^{1/2}} + \frac{1}{N^2}\sum_{i,j=1:i\neq j}^{N}(K_{\mathbf{H}}*K_{\mathbf{H}})(\mathbf{x}^i-\mathbf{x}^j), \end{aligned} \tag{3.51}$$

where we applied a change of variables in the third equality and $(f*g)(\mathbf{x}) = \int_{\mathbb{R}^n} f(\mathbf{x}-\mathbf{y})g(\mathbf{y})d\mathbf{y}$ is the convolution operator. Note that the last equality is obtained assuming the symmetry of the kernel, i.e, $K_{\mathbf{H}}(\mathbf{w}+\mathbf{x}^j-\mathbf{x}^i) = K_{\mathbf{H}}(\mathbf{x}^i-\mathbf{x}^j-\mathbf{w})$.

If the kernel function K is Gaussian, the convolution can be simplified [Wand and Jones, 1993; Chacón and Duong, 2018], so the UCV score is equal to:

$$\begin{aligned} \text{UCV}(\mathbf{H}) = &\frac{1}{N|\mathbf{H}|^{1/2}(4\pi)^{n/2}} \\ &+ \frac{1}{N(N-1)}\sum_{i,j=1:i\neq j}\left[(1-N^{-1})K_{2\mathbf{H}}(\mathbf{x}^i-\mathbf{x}^j) - 2K_{\mathbf{H}}(\mathbf{x}^i-\mathbf{x}^j)\right]. \end{aligned} \tag{3.52}$$

The UCV is an unbiased estimator of the MISE (without taking into account the constant $R(f)$), hence its name. The bandwidth selection can be completed finding the bandwidth matrix that minimizes the UCV score:

$$\hat{\mathbf{H}}_{\text{UCV}} = \arg\min_{\mathbf{H}} \text{UCV}(\mathbf{H}). \tag{3.53}$$

3.3.3.3 Biased Cross-Validation

The biased cross-validation (BCV) [Scott and Terrell, 1987], also known as asymptotic cross-validation, tries to minimize the AMISE instead of the MISE. Therefore, it is called biased because the AMISE can be considered a biased approximation to the MISE.

In the AMISE expressions of Equation (3.35) and Equation (3.37), the unique unknown quantities are $R(f'')$ and $R(\text{tr}\{\mathbf{H}\nabla^2 f\})$. With some transformations, $R(\text{tr}\{\mathbf{H}\nabla^2 f\})$ can

be calculated estimating only the unknown $R(\nabla^2 f)$. This term can be estimated differentiating the KDE model $\hat{f}_{\text{KDE}}$. For the univariate case, the derivative can be found easily differentiating the kernel function:

$$\hat{f}''_{\text{KDE}}(x) = \frac{1}{N}\sum_{i=1}^{N} K''_h(x - x^i), \tag{3.54}$$

and $R(\hat{f}''_{\text{KDE}})$ can be calculated with an expression similar to Equation (3.51) adapted to remove the asymptotic bias (see Scott and Terrell [1987]). Therefore, the BCV score is:

$$\begin{aligned} \text{BCV}(h) &= \frac{R(K)}{Nh} + \frac{1}{4}m_2(K)^2 h^4 R(\hat{f}''_{\text{KDE}}), \\ \text{BCV}(\mathbf{H}) &= N^{-1}|\mathbf{H}|^{1/2}R(K) + \frac{1}{4}m_2(K)^2 \int_{\mathbb{R}^n} \text{tr}\{\mathbf{H}\nabla^2 \hat{f}_{\text{KDE}}(\mathbf{x})\}^2 d\mathbf{x}. \end{aligned} \tag{3.55}$$

An alternative formulation to compute $R(\hat{f}''_{\text{KDE}})$ is proposed in Hall and Marron [1987]. This gives rise to a modified expression for the BCV [Jones and Kappenman, 1992]. These algorithms have also been adapted to the multivariate context [Sain et al., 1994]. The estimation of $R(\nabla^2 f)$ in the multivariate case can be more complicated because all partial derivatives need to be calculated. In most cases, these derivatives cannot be calculated in an exact way, but there are approximations developed in the literature. Chacón and Duong [2015] propose an efficient algorithm to compute the derivatives of a Gaussian kernel.

Finally, the BCV selects the bandwidth that minimizimes Equation (3.55):

$$\hat{\mathbf{H}}_{\text{BCV}} = \arg\min_{\mathbf{H}} \text{BCV}(\mathbf{H}). \tag{3.56}$$

3.3.3.4 Plug-in Bandwidth Selection

The motivation for the plug-in bandwidth selector is similar to the BCV because it tries to estimate $R(f'')$ (or $R(\nabla^2 f)$ in the multivariate case) and then "plug-in" this estimate into the AMISE formula. Unlike the BCV criterion, the estimate of $R(f'')$ is not constructed differentiating $\hat{f}_{\text{KDE}}$, but it constructs a new KDE model that estimates the derivatives $\hat{g}''_{\text{KDE}}$. The $\hat{g}''_{\text{KDE}}$ model is constructed using a kernel function L_1, which can be different from the kernel K used in $\hat{f}_{\text{KDE}}$, and a bandwidth $\mathbf{G}_1$, which is different from $\mathbf{H}$, and is called pilot bandwidth.

In this way, we can see that the plug-in bandwidth selection method transforms the problem of selecting $\mathbf{H}$ into the problem of selecting $\mathbf{G}_1$. First, note that the estimation error of $\mathbf{G}_1$ affects the final estimate of $\mathbf{H}$ only in the estimation of $R(f'')$ or $R(\nabla^2 f)$. This is opposed to BCV, where $\mathbf{H}$ is used both to estimate the AMISE (Equation (3.55)) and $R(f'')$ or $R(\nabla^2 f)$. Therefore, the selection of $\mathbf{G}_1$ is less critical than the selection of $\mathbf{H}$ in BCV. Moreover, the $\mathbf{G}_1$ can be selected with an additional criterion. The selection of $\mathbf{G}_1$ can be driven by the asymptotic mean squared error (AMSE), which depends on $R(f''')$ or $R(\nabla^3 f)$, where $\nabla^3 f$ contains all the partial derivatives of order 3 of f [Chacón and Duong,

2010]. Note that while working with derivatives of order greater than 2 we need to organize the arrangement of the partial derivatives. A common convention is the use of the vec($\mathbf{M}$) operator which transforms matrix $\mathbf{M}$ into a vector stacking up all the columns.

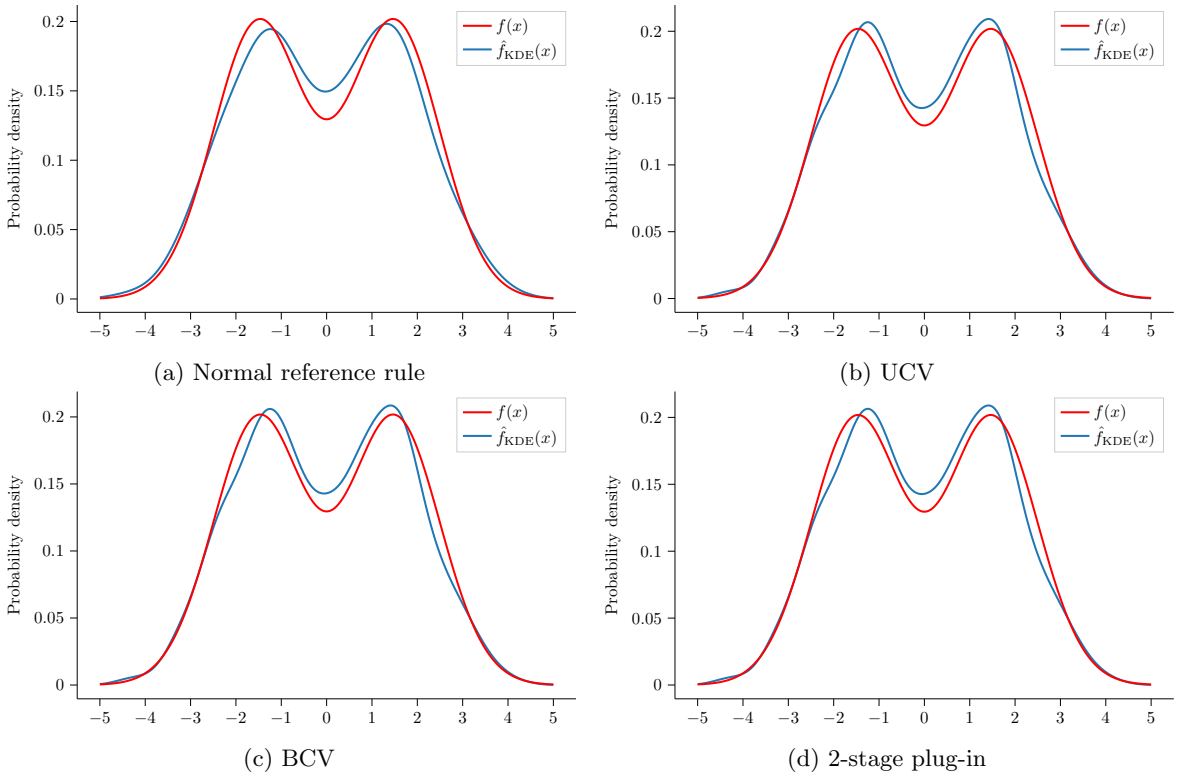
If matrix $\mathbf{G}_1$ is selected with the AMSE criterion, the unknown terms to select it, $R(f''')$ or $R(\boldsymbol{\nabla}^3 f)$, can be estimated with a different KDE model $\hat{v}'''_{\text{KDE}}$, with a kernel function L_2 and a different pilot bandwidth matrix $\mathbf{G}_2$. Now, the problem of selecting $\mathbf{H}$ is transformed into the problem of selecting $\mathbf{G}_2$. This process can be repeated several times and several pilot bandwidths can be estimated, so a pilot bandwidth is used to estimate the next, until $\mathbf{G}_1$ is finally estimated. This is because the AMSE of the pilot bandwidth $\mathbf{G}_{r-1}$ used to estimate $R(f^{(r)})$ or $R(\boldsymbol{\nabla}^r f)$ always depends on $R(f^{(r+1)})$ or $R(\boldsymbol{\nabla}^{r+1} f)$. Finally, the last pilot bandwidth needs to be estimated without using the AMSE criterion (or else, the chain of pilot bandwidths would be infinite). A common option is to use a variation of the normal reference rule which is specially tailored to estimate derivatives, for the last pilot bandwidth. By adding more pilot bandwidths, the contribution of the last pilot bandwidth in $\mathbf{H}$ selection diminishes, so the error produced by the normal reference rule is not so important. Each pilot bandwidth estimation is called stage, so a plug-in method with r pilot bandwiths $\mathbf{G}_1, \ldots, \mathbf{G}_r$ is called an r-stage plug-in estimation model. Note that a 0-stage plug-in estimation is equal to the normal reference rule estimation method. Usually, 1-stage or 2-stage plugin bandwidth selectors are used in practice. Algorithm 3.1 outlines the procedure to estimate r-stage plug-in bandwidth selection in a multivariate context. The univariate case is constructed equivalently replacing $R(\boldsymbol{\nabla}^{i+1} f)$ by $R(f^{(i+1)})$.

Algorithm 3.1 r-stage plug-in bandwidth selector

Require: Training data $\mathcal{D}$. Number of stages $r > 1$
Ensure: Bandwidth matrix $\mathbf{H}$
1: Compute $\mathbf{G}_r$ by the normal reference rule. // [Chacón and Duong, 2010, Equation (8)]
2: **for** $i = r-1, \ldots, 1$ **do**
3: $\quad \hat{R}(\boldsymbol{\nabla}^{i+1} f) \leftarrow$ Estimate $R(\boldsymbol{\nabla}^{i+1} f)$ with the KDE constructed with $\mathbf{G}_{i+1}$ and $\mathcal{D}$
4: $\quad \mathbf{G}_i \leftarrow \arg\max_{\mathbf{G}} \widehat{\text{AMSE}}(\mathbf{G}, \hat{R}(\boldsymbol{\nabla}^{i+1} f))$ // [Chacón and Duong, 2010, Theorem 1]
5: **end for**
6: $\hat{R}(\boldsymbol{\nabla}^2 f) \leftarrow$ Estimate $R(\boldsymbol{\nabla}^2 f)$ with the KDE model constructed with $\mathbf{G}_1$ and $\mathcal{D}$
7: $\mathbf{H} \leftarrow \arg\max_{\mathbf{H}} \widehat{\text{AMISE}}(\mathbf{H}, \hat{R}(\boldsymbol{\nabla}^2 f))$ // Equation (3.37)
8: **return H**

3.3.3.5 Bandwidth Selection Criteria Optimization

The UCV, BCV and plug-in selection methods require optimizing some criteria. This optimization is usually performed with standard optimization algorithms. Usually, derivative-free optimization algorithms, e.g., golden-section search, are used because expressions for the gradients are not available. Alternatively, a Newton method can be used if the derivatives are estimated using finite differences. For the multivariate case, a common derivative-free algorithm is the Nelder-Mead method [Nelder and Mead, 1965].



(a) Normal reference rule

(b) UCV

(c) BCV

(d) 2-stage plug-in

Figure 3.7: KDE for 1,000 instances of the normal mixture model $f(x) = 0.5 \cdot \mathcal{N}(-1.5, 1) + 0.5 \cdot \mathcal{N}(1.5, 1)$ (red line). The KDE models are estimated with a bandwidth selected using the (a) normal reference rule, (b) UCV, (c) BCV and (d) a 2-stage plug-in using a Gaussian kernel.

Furthermore, in the multivariate case the optimization must be performed over the space of positive definite matrices. A common solution to guarantee that the algorithm returns a positive definite matrix is optimizing the square root of the matrix, i.e., $\mathbf{H} = \mathbf{L}\mathbf{L}^T$, which is guaranteed to be positive definite.

3.3.3.6 Example: Normal Mixture Data

In this section we compare in practice all the bandwidth selection methods. Figure 3.7 illustrates the density estimates obtained by selecting the bandwidth with the previously discussed selection methods. The ground truth model is a normal mixture model with the following expression: $f(x) = 0.5 \cdot \mathcal{N}(-1.5, 1) + 0.5 \cdot \mathcal{N}(1.5, 1)$ as above. All the selection methods return very similar bandwidths. The normal reference rule is the most distinct estimation method, as it returns a slightly oversmoothed estimate. The remaining three methods provide an estimate that does not show under- or oversmoothing.

3.4 Adaptive Kernel Density Estimator

The bandwith matrix is constant in $\hat{f}_{\text{KDE}}$, so the same amount of smoothing is applied for all the training instances. If the distribution of the data differs in different parts of the

space, this could be suboptimal. For this reason, some works have proposed to use different bandwidth matrices in different parts of the space. This type of density estimator has been called adaptive kernel density estimation (AKDE). This idea has been implemented in two different ways. The bandwidth matrix could depend on a function of $\mathbf{x}$ [Loftsgaarden and Quesenberry, 1965], and then:

$$\hat{f}_{\text{adaptive}_1}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} K_{\mathbf{H}(\mathbf{x})}\left(\mathbf{x} - \mathbf{x}^i\right), \tag{3.57}$$

where $\mathbf{H}(\mathbf{x})$ is a function that returns a bandwidth matrix. Alternatively, each training dataset can have a different bandwidth matrix [Breiman et al., 1977]:

$$\hat{f}_{\text{adaptive}_2}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} K_{\mathbf{H}(\mathbf{x}^i)}\left(\mathbf{x} - \mathbf{x}^i\right). \tag{3.58}$$

Chapter 4

Bayesian Networks

4.1 Introduction

Real world data usually exhibit uncertainty from many sources. The data can be difficult to interpret with a deterministic model given the complexity of the phenomenon under study; some data sources, e.g., different types of sensors, may introduce uncertainty in the collection of data; or sometimes it is not possible to extract data from all relevant elements of the system, i.e., there is partial information available. A powerful and sound tool to manage uncertainty is probability theory, commonly by identifying or estimating a probability distribution that best explains the data. This probability distribution allows us to perform different types of queries, e.g., how probable is a given event?

However, representing and estimating a probability distribution can be a daunting task. This is notably frequent in a multivariate context. In Chapter 3, we introduced some non-parametric models to estimate a probability density, focusing especially on the KDE model. As we discussed in Section 3.3.1, the error convergence deteroriates as the dimensionality of the distribution increases. Removing this effect, which we called the curse of dimensionality, can be a problem very difficult to solve. A common approach is to harness the structure of the probability distribution to mitigate the curse of dimensionality.

Bayesian networks take advantage of the conditional independences present in the probability distribution to model it in a factorized form. Moreover, Bayesian networks make explicit the different relationships between the random variables. In this sense, these conditional independences between variables are represented graphically, so they can easily identified by humans.

The chapter is organized as follows. Section 4.2 introduces Bayesian networks and reviews some of the Bayesian networks proposed in the literature. Section 4.3 describes the learning process of the Bayesian network parameters. Finally, Section 4.4 concludes the chapter outlining the most common automatic procedures to estimate a Bayesian network structure.

4.2 Bayesian Networks

As introduced before, the Bayesian network takes advantage of the conditional independences in the probability distribution. Therefore, we define a conditional independence as:

Definition 4.1. (Conditional independence). Let $\mathbf{X}_A$, $\mathbf{X}_B$ and $\mathbf{X}_C$ be three different random variables with the sets A, B and C disjoint. Then, $\mathbf{X}_A$ and $\mathbf{X}_B$ are conditionally independent given $\mathbf{X}_C$ in a probability distribution P if:

$$P(\mathbf{X}_A, \mathbf{X}_B \mid \mathbf{X}_C) = P(\mathbf{X}_A \mid \mathbf{X}_C)P(\mathbf{X}_B \mid \mathbf{X}_C),$$

or equivalently:

$$P(\mathbf{X}_A \mid \mathbf{X}_B, \mathbf{X}_C) = P(\mathbf{X}_A \mid \mathbf{X}_C),$$

We will use the notation $(\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B \mid \mathbf{X}_C)_P$ to denote the conditional independence of $\mathbf{X}_A$ and $\mathbf{X}_B$ given $\mathbf{X}_C$ in the probability distribution P.

Formally, a Bayesian network is a pair $\mathcal{B} = (\mathcal{G}, \boldsymbol{\theta})$, where $\mathcal{G} = (V, A)$ is a DAG with a set of nodes $V = \{1, \ldots, n\}$ and a set of arcs $A \subseteq V \times V$. $\boldsymbol{\theta} = \{P\left(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)}\right), i = 1, \ldots, n\}$ is a set of parameters that defines a conditional probability distribution (CPD) for each random variable. A Bayesian network factorizes a joint probability distribution $P(\mathbf{x})$ of a vector of random variables $\mathbf{X} = (X_1, \ldots, X_n)$. The set of nodes V indexes the vector of random variables, so $\mathbf{X}_V = \mathbf{X}$. The graph $\mathcal{G}$ encodes a set of conditional independences among triplets of variables. That is, each variable $\mathbf{X}_i$ is conditionally independent of its nondescendants variables given $\mathbf{X}_{\mathrm{Pa}(i)}$. This is called the local Markov property. The set of descendant nodes of i is the set of all reachable nodes following a direct path from i. Thus, the set of conditional independences encoded by the Bayesian network depends on the set of arcs A of $\mathcal{G}$. Taking advantage of these conditional independences, the joint probability distribution $P(\mathbf{x})$ is factorized as:

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)}), \tag{4.1}$$

where $\mathrm{Pa}(i)$ are the parent indices of node i in the graph $\mathcal{G}$. The set of independences defined by the Bayesian network can be directly read from the graph $\mathcal{G}$ using the d-separation criterion [Geiger et al., 1990]. The d-separation criterion is based on the concept of active trails:

Definition 4.2. (Trail). A trail is a sucession of nodes $V_1, \ldots, V_k$ for which either $V_i \rightarrow V_{i+1}$ or $V_i \leftarrow V_{i+1}$, i.e., there is a succession of arcs connecting V_1 and V_k, ignoring the arc directions.

Definition 4.3. (Active trail). A trail $V_1, \ldots, V_k$ is active given a set of nodes C if:

- For every $V_{i-1} \rightarrow V_i \leftarrow V_{i+1}$, then V_i or one of its descendants is in C.
- Every other node in the trail is not in C.

Then, the d-separation can be defined as:

Definition 4.4. (d-separation). Let A, B and C be three different disjoint sets of nodes. A and B are d-separated by C if no active trail exists between any node in A and any node in B given the set of nodes C. We will use the notation $\text{d-sep}_{\mathcal{G}}(A; B \mid C)$ to denote the d-separation of A and B given C in the graph $\mathcal{G}$.

According to the relationship between the conditional independences of a probability distribution P and the d-separation criterion of a graph $\mathcal{G}$, several concepts are defined:

Definition 4.5. (I-map). Let $\mathcal{G}$ be a graph and P a probability distribution. Then $\mathcal{G}$ is an independence map, I-map, of P if:

$$\text{d-sep}_{\mathcal{G}}(A; B \mid C) \implies (\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B \mid \mathbf{X}_C)_P. \tag{4.2}$$

That is, the set of conditional independences defined by $\mathcal{G}$ is contained in the set of conditional independences of P. However, some additional conditional independences could be found in P. For every distribution P that factorizes according to the graph $\mathcal{G}$ (Equation (4.1)), $\text{d-sep}_{\mathcal{G}}(A; B \mid C)$, also implies $(\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B)_P$, i.e., if P factorizes according to $\mathcal{G}$, then $\mathcal{G}$ is an I-map of P.

Definition 4.6. (D-map). Let $\mathcal{G}$ be a graph and P a probability distribution. Then $\mathcal{G}$ is a dependence map, D-map, of P if:

$$(\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B \mid \mathbf{X}_C)_P \implies \text{d-sep}_{\mathcal{G}}(A; B \mid C). \tag{4.3}$$

That is, the set of conditional independences of P is contained in the set of conditional independences defined by $\mathcal{G}$. However, some additional conditional independences could be found in $\mathcal{G}$.

Definition 4.7. (P-map). Let $\mathcal{G}$ be a graph and P a probability distribution. Then $\mathcal{G}$ is a perfect map, P-map, of P if:

$$\text{d-sep}_{\mathcal{G}}(A; B \mid C) \iff (\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B \mid \mathbf{X}_C)_P. \tag{4.4}$$

That is, the set of conditional independences defined by $\mathcal{G}$ and the set of conditional independences of P are the same. A P-map is both an I-map and a D-map. If the graph $\mathcal{G}$ is a P-map of the probability distribution P, then P is said to be faithful to $\mathcal{G}$.

The number of parameters of an unfactorized distribution $P(\mathbf{x})$ is usually larger than $O(n)$, e.g., it is quadratic, $O(n^2)$, for a multivariate normal distribution, or exponential, $\prod_{i=1}^{n} |\mathbf{\Omega}_i|$, for a conditional probability table (CPT) of a categorical distribution. Since usually $|\text{Pa}(i)| \ll n$, the set $\boldsymbol{\theta}$ contains n CPDs with a very small number of parameters. Thus, the number of parameters needed to define a Bayesian network is usually much lower than the unfactorized representation of $P(\mathbf{x})$.

A Bayesian network can manage categorical and continuous data. This gives rise to three types of Bayesian networks: discrete Bayesian networks, which only supports categorical variables; continuous Bayesian networks, which only supports continuous variables; and hybrid

Bayesian networks, which supports a mix of categorical and continuous variables. Note that for continuous Bayesian networks, the factorization in Equation (4.1) computes a probability density because each local CPD is a PDF:

$$f(\mathbf{x}) = \prod_{i=1}^{n} f(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)}). \tag{4.5}$$

The Bayesian network model is very general because it admits different types of CPDs for each node. Depending on these types, we can clasify Bayesian networks into three different groups: parametric Bayesian networks, nonparametric Bayesian networks and semiparametric Bayesian networks. Parametric Bayesian networks use only parametric models to define the CPDs of each variable. Conversely, nonparametric Bayesian networks only define the CPDs using nonparametric models. The semiparametric Bayesian networks define the set of CPDs combining parametric and nonparametric elements. Similarly as the discussion in Chapter 3, parametric Bayesian networks make assumptions about the underlying distribution of the data. On the contrary, nonparametric Bayesian networks remove these assumptions. Combining both approaches, semiparametric Bayesian networks uses parametric models when these assumptions are suitable, and nonparametric models otherwise.

4.2.1 Parametric Bayesian Networks

Most of the Bayesian networks defined in the literature are parametric, i.e., they are defined using parametric CPDs with a fixed number of parameters. The most common type of Bayesian networks are discrete Bayesian networks, that only can be used to model discrete domains. Usually, discrete Bayesian networks are defined using a CPT that specifies a conditional probability for each discrete configuration of X_i and $\mathbf{X}_{\mathrm{Pa}(i)}$. Other types of CPDs are available such as tree-structured CPDs [Friedman and Goldszmidt, 1996], which can take advantage of the conditional independences in the CPD to reduce the needed number of parameters.

Gaussian Bayesian networks (GBNs) [Shachter and Kenley, 1989; Geiger and Heckerman, 1994] are a specific continuous Bayesian network type. A GBN uses a linear Gaussian (LG) CPD for each node of the Bayesian network. The LG CPD assumes that there is a linear relationship between X_i and $\mathbf{X}_{\mathrm{Pa}(i)}$, and there is a normal conditional distribution given $\mathbf{x}_{\mathrm{Pa}(i)}$. A GBN assumes that the random variables are distributed according to a multivariate normal distribution. Indeed, a GBN can be considered an alternative representation of a multivariate Gaussian distribution. Since the normality assumption can be inadequate to model non-normal distributions, other types of Bayesian networks have been developed. Mixtures of Gaussian DAG (MGDAG) models [Thiesson et al., 1998] are analogous to the normal mixture model but they use GBNs to represent the base components of the mixture. Tweedie Bayesian networks (TDB) [Masmoudi and Masmoudi, 2019], restrict the CPDs to be a special class of distributions from the exponential family. The Tweedie class of distributions include the Gaussian, the inverse-Gaussian, the gamma and the Poisson distributions.

A conditional linear Gaussian Bayesian network (CLGBN) [Lauritzen and Wermuth,

1989], is an hybrid Bayesian network that assumes that the discrete random variables can be represented with a CPT, and the continuous random variables are assummed to be conditionally distributed with an LG CPD or a set of LG CPDs. A CLGBN restricts the structure of the network, so arcs from continuous random variables to discrete random variables are not allowed. Also, the assumption of an LG CPD implies, for each possible configuration of the discrete parents, a normal conditional distribution. Some efforts have been made to increase the flexibility of the CLGBN model. The augmented CLGBN [Lerner et al., 2001] addresses the restriction on the continuous parents for discrete variables using a softmax function as CPD for the discrete variables. Mixture of truncated exponentials (MTE) networks [Moral et al., 2001] describe piecewise-defined exponential functions that are used as CPD. The MTE networks overcome the normality distribution by dividing the domain in multiple hypercubes and assigning a different exponential function to each domain partition. This idea inspired the mixture of truncated polynomials (MoP) networks [Shenoy and West, 2011] and mixture of truncated basis functions (MoTBF) networks [Langseth et al., 2012], which use polynomials and basis functions, respectively, in each domain partition.

4.2.2 Nonparametric Bayesian networks

A nonparametric Bayesian network is a Bayesian network where all the CPDs of a Bayesian network are estimated with nonparametric methods. Hofmann and Tresp [1995] define a type of nonparametric Bayesian network where the CPDs are defined as the ratio of two KDE models. We will denote this type of Bayesian network as kernel density estimation Bayesian networks (KDEBN). The use of KDE models was extended to create flexible Bayesian network classifiers. A flexible naive Bayes is described in John and Langley [1995], and other flexible Bayesian network classifiers with fewer structure constraints are proposed in Pérez et al. [2009]. Gonzalez et al. [2015] apply a flexible Bayesian network classifier to detect anomalies in an industrial context.

Friedman and Nachman [2000] define Bayesian networks using Gaussian processes, which are also nonparametric methods. These type of Bayesian networks are designed to detect functional dependencies between the variables. Ickstadt et al. [2012] introduce a continuous nonparametric Bayesian network that uses an infinite mixture model to avoid the normality assumption. This Bayesian network is learned taking into account the uncertainty of the parameters and the graph of a GBN and using Bayesian priors over the parameters.

4.2.3 Semiparametric Bayesian networks

A semiparametric model combines parametric and nonparametric components. They try to get the advantages of both types of models. Not much work has been done on semiparametric Bayesian networks. Boukabour and Masmoudi [2020] define a continuous semiparametric Bayesian network that rely on semiparametric regression. The semiparametric regression combines a linear regression component with a Nadaraya-Watson estimator [Nadaraya, 1964; Watson, 1964], that uses a diagonal bandwidth matrix for the nonparametric component.

In this work, all the CPDs are semiparametric regressions. The graph structure of these networks is learned using a novel algorithm based on statistical hypothesis tests of conditional independence that requires to know the correct ancestral ordering for the nodes.

To the best of our knowledge, no previous approach supported hybrid data using semiparametric Bayesian networks.

4.3 Parameter Learning

In this section, we describe how the parameters of a Bayesian network are learned automatically from data. The parameters of a Bayesian network are located in the CPDs $\boldsymbol{\theta}$. Therefore, the parameter learning procedure for each CPD depends on its type. The two most common CPDs used in the literature are the categorical, LG and conditional linear Gaussian (CLG). For this reason, we include a description of the usual learning procedures for these CPDs. We assume that the structure of the Bayesian network $\mathcal{G}$, is known. Section 4.4 deals with the structure learning issue.

A standard technique used to estimate the parameters is the maximum likelihood estimate (MLE) criterion. The likelihood of a dataset $\mathcal{D}$ given a graph structure $\mathcal{G}$ and parameters $\boldsymbol{\theta}$ is derived from Equation (4.1):

$$P(\mathcal{D} \mid \mathcal{G}, \boldsymbol{\theta}) = \prod_{i=1}^{n} \prod_{j=1}^{N} P(x_i^j \mid \mathbf{x}_{\mathrm{Pa}(i)}^j), \tag{4.6}$$

which assumes that the samples in $\mathcal{D}$ are independent and identically distributed. The MLE selects the set of parameters $\boldsymbol{\theta}$ that maximizes Equation (4.6). For convenience, most of the times the log of the likelihood, the log-likelihood, is used because the maximization of the likelihood and the log-likelihood returns the same set of parameters. The log-likelihood of some data $\mathcal{D}$ given a graph structure $\mathcal{G}$ and parameters $\boldsymbol{\theta}$ is:

$$\mathcal{L}(\mathcal{G}, \boldsymbol{\theta} : \mathcal{D}) = \sum_{i=1}^{n} \sum_{j=1}^{N} \log P\left(x_i^j \mid \mathbf{x}_{\mathrm{Pa}(i)}^j\right) = \sum_{i=1}^{n} \mathcal{L}(X_i \mid \mathbf{X}_{\mathrm{Pa}(i)}, \boldsymbol{\theta}_i : \mathcal{D}), \tag{4.7}$$

where $\mathcal{L}(X_i \mid \mathbf{X}_{\mathrm{Pa}(i)}, \boldsymbol{\theta}_i : \mathcal{D})$ is the local log-likelihood of variable X_i, and $\boldsymbol{\theta}_i$ are the set of parameters for the $P(X_i \mid \mathbf{X}_{\mathrm{Pa}(i)})$ CPD. Then, the set of the MLE parameters are equal to:

$$\begin{aligned} \hat{\boldsymbol{\theta}}^{\mathrm{MLE}} &= \arg\max_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\mathcal{G}, \boldsymbol{\theta} : \mathcal{D}) \\ &= \{\arg\max_{\boldsymbol{\theta}_1 \in \Theta_1} \mathcal{L}(X_1 \mid \mathbf{X}_{\mathrm{Pa}(1)}, \boldsymbol{\theta}_1 : \mathcal{D}), \ldots, \arg\max_{\boldsymbol{\theta}_n \in \Theta_n} \mathcal{L}(X_n \mid \mathbf{X}_{\mathrm{Pa}(n)}, \boldsymbol{\theta}_n : \mathcal{D})\}, \end{aligned} \tag{4.8}$$

where Θ is the parameter space, namely, the set of allowable parameters, and Θ_i is the parameter space for the $P(X_i \mid \mathbf{X}_{\mathrm{Pa}(i)})$ CPD. This optimization is performed finding the MLE parameters of each CPD independently because there are no shared parameters between the CPDs. This property is called global decomposability.

An alternative to MLE is the Bayesian estimation, which tries to take into account the uncertainty in the selection of $\boldsymbol{\theta}$. Thus, instead of performing a single point estimate like the MLE, multiple parameter values are considered in the estimation, calculating the prior probability of each $\boldsymbol{\theta}$ estimate. In this way, the set of parameters $\boldsymbol{\theta}$ can be seen as a random variable that can be included in the model:

$$P(\mathcal{D}, \boldsymbol{\theta} \mid \mathcal{G}) = P(\mathcal{D} \mid \mathcal{G}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{G}), \tag{4.9}$$

where $P(\mathcal{D} \mid \mathcal{G}, \boldsymbol{\theta})$ is the likelihood defined in Equation (4.6) and $P(\boldsymbol{\theta} \mid \mathcal{G})$ is a prior probability function for the parameters $\boldsymbol{\theta}$ given the graph $\mathcal{G}$. The prior distribution defines our beliefs about the distribution of the parameters $\boldsymbol{\theta}$ before seeing any data. This prior distribution over the parameters is updated when data is available to take into account the evidence, generating the posterior distribution:

$$P(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{G}) = \frac{P(\mathcal{D} \mid \mathcal{G}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{G})}{P(\mathcal{D} \mid \mathcal{G})} = \frac{P(\mathcal{D} \mid \mathcal{G}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{G})}{\int_{\Theta} P(\mathcal{D} \mid \mathcal{G}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{G}) d\boldsymbol{\theta}}. \tag{4.10}$$

Once the posterior distribution has been computed, the likelihood of a new dataset $\mathcal{D}'$ can be calculated taking into account the uncertainty about the parameters after seeing the dataset $\mathcal{D}$:

$$P(\mathcal{D}' \mid \mathcal{D}, \mathcal{G}) = \int_{\Theta} P(\mathcal{D}' \mid \mathcal{G}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{G}) d\boldsymbol{\theta}, \tag{4.11}$$

where this is usually called the posterior predictive distribution.

The integrals in Equation (4.10) and Equation (4.11) can be difficult to calculate. However, for some likelihood functions there exist some prior distributions for which the posterior distribution is of the same family and can be easily calculated. These prior distributions are called conjugate priors of the likelihood function. Furthermore, for many conjugate priors the integral in Equation (4.11) can also be easily calculated with a closed formula. Therefore, a common technique is to choose the prior distribution on the parameters $\boldsymbol{\theta}$ using conjugate priors to simplify the learning process.

4.3.1 Categorical Distribution

The categorical distribution $P(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)})$ defines a parameter $\theta_{x_i|\mathbf{x}_{\mathrm{Pa}(i)}}$ for each discrete configuration of X_i and $\mathbf{X}_{\mathrm{Pa}(i)}$, so $P(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)}) = \theta_{x_i|\mathbf{x}_{\mathrm{Pa}(i)}}$. The MLE estimator for the parameter $\theta_{x_i|\mathbf{x}_{\mathrm{Pa}(i)}}$ from a dataset $\mathcal{D}$ is equal to:

$$\hat{\theta}^{\mathrm{MLE}}_{x_i|\mathbf{x}_{\mathrm{Pa}(i)}} = \frac{N[x_i, \mathbf{x}_{\mathrm{Pa}(i)}]}{N[\mathbf{x}_{\mathrm{Pa}(i)}]}, \tag{4.12}$$

where $N[x_i, \mathbf{x}_{\mathrm{Pa}(i)}]$ is the number of instances in $\mathcal{D}$ where $X_i = x_i$ and $\mathbf{X}_{\mathrm{Pa}(i)} = \mathbf{x}_{\mathrm{Pa}(i)}$. $N[\mathbf{x}_{\mathrm{Pa}(i)}]$ is defined similarly.

The conjugate prior of the categorical likelihood is the Dirichlet distribution. Dirichlet distribution has a set of parameters $\alpha_1, \ldots, \alpha_K, \alpha_i > 0$ where K is the number of categories.

These parameters of the prior distribution are called hyperparameters. Then, to define the prior for the conditional categorical distribution, a Dirichlet prior is assumed for each discrete configuration of $\mathbf{X}_{\text{Pa}(i)}$. Therefore, the Bayesian estimator of the categorical distribution with a Dirichlet prior is taken from the posterior predictive distribution:

$$\hat{\theta}^{\text{Bayes}}_{x_i|\mathbf{x}_{\text{Pa}(i)}} = \frac{\alpha_{x_i|\mathbf{x}_{\text{Pa}(i)}} + N[x_i, \mathbf{x}_{\text{Pa}(i)}]}{\alpha_{\mathbf{x}_{\text{Pa}(i)}} + N[\mathbf{x}_{\text{Pa}(i)}]}, \tag{4.13}$$

where $\alpha_{x_i|\mathbf{x}_{\text{Pa}(i)}}$ is the hyperparameter for the category $X_i = x_i$ in the Dirichlet prior for $\mathbf{x}_{\text{Pa}(i)}$, and $\alpha_{\mathbf{x}_{\text{Pa}(i)}} = \sum_{x_i \in \mathbf{\Omega}_i} \alpha_{x_i|\mathbf{x}_{\text{Pa}(i)}}$.

A commonly used technique to define the Dirichlet prior hyperparameters is to use a uniform Dirichlet, where all the hyperparameters are equal. Then, the Bayes estimation can be expressed as:

$$\hat{\theta}^{\text{Bayes}}_{x_i|\mathbf{x}_{\text{Pa}(i)}} = \frac{\alpha / \left|\mathbf{\Omega}_{\{i\}\cup\text{Pa}(i)}\right| + N[x_i, \mathbf{x}_{\text{Pa}(i)}]}{\alpha / \left|\mathbf{\Omega}_{\text{Pa}(i)}\right| + N[\mathbf{x}_{\text{Pa}(i)}]}, \tag{4.14}$$

where $\alpha_{\mathbf{x}_{\text{Pa}(i)}}$ is the same for all discrete configurations of $\mathbf{X}_{\text{Pa}(i)}$ and $\alpha = |\mathbf{\Omega}_{\text{Pa}(i)}|\alpha_{\mathbf{x}_{\text{Pa}(i)}}$ is the equivalent sample size. The equivalent sample size determines the strength with which the prior affects the estimate.

4.3.2 Linear Gausian Distribution

Definition 4.8. (LG CPD). Let X be a continuous random variable and $\mathbf{Y} = (Y_1, \ldots, Y_r)$ a vector of continuous random variables. The conditional distribution of X given $\mathbf{Y}$ follows a normal distribution and its mean is calculated with a linear regression:

$$f_{\text{LG}}(x \mid \mathbf{y}) = \mathcal{N}\left(\beta_0 + \sum_{j=1}^{r} \beta_j y_j, \sigma^2\right), \tag{4.15}$$

where β_0 is the intercept of the linear regression, β_j $(j = 1, \ldots, r)$ is the regression coefficient corresponding to Y_j, and σ^2 is the variance of the conditional distribution. The LG CPD is derived from the assumption of a linear relationship between X and Y_j:

$$X = \beta_0 + \sum_{j=1}^{r} \beta_j Y_j + \epsilon, \text{ with } \epsilon \sim \mathcal{N}(0, \sigma^2), \tag{4.16}$$

where $Y_1, \ldots, Y_r$ are independent of ϵ.

The parameters of an LG CPD are $\boldsymbol{\theta} = \{\beta_0, \beta_1, \ldots, \beta_r, \sigma^2\}$. The MLE estimate for the LG CPD can be found by ordinary least squares [Fox, 1997]. This is the most commonly used technique to fit the parameters.

The conjugate prior of the LG CPD is the normal inverse-gamma distribution. However, the Bayesian estimation is not used frequently for LG CPDs. This is probably because the posterior predictive distribution in Equation (4.11) is a Student-t distribution. Since

the Gaussian distribution has many interesting theoretical properties, the MLE is probably preferred because it preserves the normality.

4.3.3 Conditional Linear Gausian Distribution

Definition 4.9. (CLG CPD). Let X be a continuous random variable, $\mathbf{Y} = (Y_1, \ldots, Y_r)$ a vector of continuous random variables, and $\mathbf{Z} = (Z_1, \ldots, Z_p)$ a vector of discrete random variables. The conditional distribution of X given $\mathbf{Y}$ and $\mathbf{Z}$ is:

$$f_{\text{CLG}}(x \mid \mathbf{y}, \mathbf{z}) = \mathcal{N}\left(\beta_{\mathbf{z},0} + \sum_{j=1}^{r} \beta_{\mathbf{z},j} y_j, \sigma^2_{\mathbf{z}}\right), \tag{4.17}$$

where $\beta_{\mathbf{z},0}$ is the intercept for the discrete configuration $\mathbf{z}$, $\beta_{\mathbf{z},j}$ $(j = 1, \ldots, r)$ is the regression coefficient corresponding to Y_j for the discrete configuration $\mathbf{z}$, and $\sigma^2_{\mathbf{z}}$ is the variance of the conditional distribution for discrete configuration $\mathbf{z}$.

The parameters of a CLG CPD are $\boldsymbol{\theta} = \{\boldsymbol{\beta}_{\mathbf{z}}, \sigma^2_{\mathbf{z}} \mid \mathbf{z} \in \boldsymbol{\Omega}_{\mathbf{Z}}\}$, where $\boldsymbol{\beta}_{\mathbf{z}} = (\beta_{\mathbf{z},0}, \beta_{\mathbf{z},1}, \ldots, \beta_{\mathbf{z},r})$ is the combination of the intercept and the regressor coefficients. The CLG CPD defines an independent LG CPD for each discrete configuration. Therefore, the parameter estimation of a CLG can be performed as the parameter estimation of a LG CPD (Section 4.3.2) for each discrete configuration.

4.4 Structure Learning

The structure of the Bayesian network, $\mathcal{G}$, can be learned automatically from data. In the first works, the learning structure algorithms could only learn structures with some restrictions, such as learning graphs that were trees [Chow and Liu, 1968] or polytrees [Dasgupta, 1999]. Other algorithms could learn general graphs, but it required to know an ancestral ordering of the nodes, such as the K2 algorithm [Cooper and Herskovits, 1992].

In this section we focus on algorithms which can learn general graphs without requiring additional assumptions, such as knowing the ancestral ordering of the nodes. There are three main approaches to learn the structure of a Bayesian network: score and search, constraint-based and hybrid procedures. However, other common optimization methods such as evolutionary algorithms [Larrañaga et al., 2013] or ant colony optimization [de Campos et al., 2002].

4.4.1 Score and Search

The score and search approaches are based on defining a score function that evaluates how good a structure captures the patterns in the data. Then, learning the structure is transformed into an optimization process of the structure with respect to the score function.

4.4.1.1 Score Function

A key element of the learning procedure is the score function. One of the first score functions is the maximum log-likelihood function, which is the log-likelihood of the data (Equation (4.7)) when the parameters are estimated using MLE, i.e., $\mathcal{L}(\mathcal{G}, \hat{\boldsymbol{\theta}}^{\text{MLE}} : \mathcal{D})$. However, the maximum log-likelihood score almost always returns complete structures even when some conditional independences exist due to the statistical noise in the data [Koller and Friedman, 2009]. A commonly used alternative is the Bayesian information criterion (BIC) [Schwarz, 1978] score:

$$\mathcal{S}_{\text{BIC}}(\mathcal{D}, \mathcal{G}) = \mathcal{L}(\mathcal{G}, \hat{\boldsymbol{\theta}}^{\text{MLE}} : \mathcal{D}) - \frac{\log N}{2}\text{Dim}(\mathcal{G}), \tag{4.18}$$

where $\text{Dim}(\mathcal{G})$ counts the number of free parameters in the structure $\mathcal{G}$. The BIC score penalizes the maximum log-likelihood score with the number of parameters in the structure, so sparser structures are usually found. This avoids that spurious arcs are added by the statistical noise. The BIC score is equivalent to the minimum description length (MDL) [Rissanen, 1978] score, which is derived from the information theory. In addition, other scores derived from the information theory have been proposed, such as the mutual information tests (MIT) [de Campos, 2006] score.

Under particular assumptions, including $N \to \infty$, the BIC score can be considered as an approximation to the Bayesian score [Koller and Friedman, 2009]. The Bayesian score considers the uncertainty in the structure selection. Therefore, it first defines a prior distribution over the possible structures $P(\mathcal{G})$. Then, the posterior distribution of the structure, given the data $\mathcal{D}$, can be calculated as:

$$P(\mathcal{G} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \mathcal{G})P(\mathcal{G})}{P(\mathcal{D})}. \tag{4.19}$$

Following a full Bayesian approach, all the possible structures should be taken into account to estimate the likelihood of a new dataset $\mathcal{D}'$:

$$P(\mathcal{D}' \mid \mathcal{D}) = \sum_{\mathcal{G} \in \mathscr{G}} P(\mathcal{D}' \mid \mathcal{D}, \mathcal{G})P(\mathcal{G} \mid \mathcal{D}), \tag{4.20}$$

where $\mathscr{G}$ is the set of all possible structures. This technique is usually called Bayesian model averaging. In practice this is intractable because the cardinality of the set $\mathscr{G}$ is super-exponential on the number of nodes n [Robinson, 1977]. An alternative is to select the structure that maximizes Equation (4.19). The Bayesian score applying a logarithm function is as follows:

$$\log P(\mathcal{G} \mid \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{G}) + \log P(\mathcal{G}) - \log P(\mathcal{D}). \tag{4.21}$$

Note that the third term does not affect the optimization of the structure, so it can be ignored. Then, the Bayesian score can be defined as:

$$\mathcal{S}_{\text{Bayes}}(\mathcal{D}, \mathcal{G}) = \log P(\mathcal{D} \mid \mathcal{G}) + \log P(\mathcal{G}). \tag{4.22}$$

The contribution of the prior over the structures is asymptotically negligible. For this reason, a commonly used prior is a uniform prior, i.e., all the structures are assigned the same prior probability. In this case, the prior distribution over the structures can be ignored in the score as it does not introduce any bias to the learning process. However, other methods to construct nonuniform priors exist [Castelo and Siebes, 2000].

The term $P(\mathcal{D} \mid \mathcal{G})$, called the marginal likelihood, is calculated as:

$$P(\mathcal{D} \mid \mathcal{G}) = \int_{\Theta} P(\mathcal{D} \mid \mathcal{G}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{G}) d\,\boldsymbol{\theta}, \tag{4.23}$$

which is similar to the expression in Equation (4.11), so the same conclusions about conjugate priors hold. Using conjugate priors, this gives rise to the BDe score for discrete Bayesian networks [Heckerman et al., 1995], and the BGe score for GBNs [Geiger and Heckerman, 1994, 2002].

An important property for the score is the decomposability. A score is decomposable if it can be expressed as a sum of local terms, one for each family (a node and its parents) of the graph:

$$\mathcal{S}_{\text{decomposable}}(\mathcal{D}, \mathcal{G}) = \sum_{i=1}^{n} \mathcal{S}_{\text{local}}(X_i \mid \mathbf{X}_{\text{Pa}(i)} : \mathcal{D}, \mathcal{G}). \tag{4.24}$$

The decomposability of the score is important to reduce the computational cost of the search procedure. The maximum log-likelihood, BIC, BDe and BGe scores are all decomposable.

4.4.1.2 Search Procedure

Once the score function has been defined, all that remains is to search for the best structure. Since the number of possible structures is super-exponential in the number of nodes, in practice all structures cannot be evaluated to find the best structure.

A commonly used search algorithm is greedy hill-climbing (HC), which tries to optimize the structure applying small and local changes to the structure that improve the score function. These small changes to the structure are called operators. The most common operators are a single arc addition, a single arc removal and a single arc reversal. On each iteration of the HC, the operator that improves the score the most is selected. To find the best operator, it is necessary to calculate how much the score changes after applying each operator, which is called the delta score of an operator.

Calculating all the delta scores of all operators on each iteration can be very computationally expensive. Therefore, a common technique is to cache the delta scores at the start of the algorithm. If the score function is decomposable, when an operator is applied, only a few delta scores change because the majority of the local score terms keep unaltered. This reduces significantly the number of score function executions. Algorithm 4.1 outlines a basic HC algorithm assuming a decomposable score. The delta score cache is initialized in line 3. Then, on each iteration the best operator is found (line 5) and applied (line 6). If the

operator improves the score of the structure, the structure and the delta scores are updated (lines 7-9). Otherwise, the algorithm converged and it returns the best structure (line 10-11 and line 14).

Algorithm 4.1 Greedy hill-climbing

Require: Training data $\mathcal{D}$, starting structure $\mathcal{G}_0$, decomposable score function $\mathcal{S}$, set of operators $\mathcal{O}$
Ensure: Best found structure $\mathcal{G}$
1: Converged $\leftarrow$ false
2: $\mathcal{G} \leftarrow \mathcal{G}_0$
3: Cache-Delta-Score($\mathcal{D}, \mathcal{G}, \mathcal{O}$)
4: **repeat**
5: $o \leftarrow$ Find-Max-Delta-Score($\mathcal{G}$)
6: $\mathcal{G}_{\text{new}} \leftarrow o(\mathcal{G})$ // Apply the operator
7: **if** $\mathcal{S}(\mathcal{D}, \mathcal{G}_{\text{new}}) > \mathcal{S}(\mathcal{D}, \mathcal{G})$ **then**
8: $\mathcal{G} \leftarrow \mathcal{G}_{\text{new}}$
9: Update-Delta-Scores($\mathcal{D}, \mathcal{G}, o$)
10: **else**
11: Converged $\leftarrow$ true
12: **end if**
13: **until** not Converged
14: **return** $\mathcal{G}$

4.4.2 Constraint-Based Methods

The constraint-based methods try to find the structure that best represents the set of conditional independences present in the data. A constraint-based method consists of two elements: a conditional independence test that confirms the existence of a conditional independence in the data, and an algorithm that constructs the structure with this information.

The selection of the independence test usually depends on the type of data. For discrete data, a common conditional independence test is the χ^2 test. For continuous data, a commonly used test is the partial linear correlation (PLC) test [Fisher, 1915, 1921]. Moreover, for both types of data a conditional test based on mutual information can be used.

Several algorithms have been developed to construct the structure. One of the popular algorithms is the PC algorithm [Spirtes et al., 2000]. The PC algorithm first finds the skeleton of the structure, which is an undirected graph in which there is an edge where there must be an arc in the final structure (in any direction). Then, the algorithm tries to orient all the possible v-structures $X_i \rightarrow X_k \leftarrow X_j$ with X_i and X_j nonadjacent in the skeleton. Finally, Meek rules [Meek, 1995] are applied to orient all undirected edges where the direction can be deduced. The PC algorithm is known to return the correct structure if the conditional independence test makes perfect decisions on the conditional independences in the distribution, and the true distribution of $\mathbf{X}$ is faithful to $\mathcal{G}$, i.e., the conditional independences in the distribution can be expressed in the structure $\mathcal{G}$ with d-separation. In practice, the

conditional independence tests do not make perfect decisions, and are the main cause of inaccuracies in structure estimation.

The max-min parent & children (MMPC) algorithm [Tsamardinos et al., 2003b] is based on estimating the set of parents and children of a node. The MMPC algorithm (Algorithm 4.2) first estimates a candidate parent children (CPC) set for node G using the $\overline{\text{MMPC}}$ algorithm (Algorithm 4.3). Then, it applies a symmetric correction so if H is in the CPC of G, then G should be in the CPC of H. If this condition is not satisfied, H might be a false positive so it is removed from the CPC of G. The $\overline{\text{MMPC}}$ algorithm is structured in two phases to compute the CPC of node G. In the forward phase, it adds the node which maximizes the minimum association with G taking into account all the possible subsets of the CPC. As a measure association, it is common to use the negative p-value. In the backward phase, all the false positives in the CPC are removed performing conditional independence tests. Lines 10-12 in Algorithm 4.3 are justified because if it is true that $V \to T$ or $V \leftarrow T$, then there cannot exist a set S, such that $(V \perp\!\!\!\perp T \mid S)$.

Algorithm 4.2 MMPC algorithm

Require: Association function *Assoc*, target random variable G, candidate variables to the parent children set $\mathbf{C}$
Ensure: Candidate parent children (CPC) set
1: CPC $\leftarrow \overline{\text{MMPC}}(G, \mathbf{C})$
2: **for** $H \in$ CPC **do**
3: **if** $G \notin \overline{\text{MMPC}}(H, \mathbf{C} \cup \{G\} \setminus \{H\})$ **then**
4: CPC $\leftarrow$ CPC $\setminus H$
5: **end if**
6: **end for**
7: **return** CPC

Other constraint-based algorithms have been developed in the literature such as the incremental association Markov blanket algorithm [Tsamardinos et al., 2003a], the grow-shrink algorithm [Margaritis and Thrun, 1999; Margaritis, 2003] or the interleaved HITON-PC algorithm [Aliferis et al., 2010].

4.4.3 Hybrid Methods

The score and search algorithm drives the optimization process based on a score function that is related to the likelihood. On its part, the constraint-based algorithm drives the search process following the conditional independence information. The main consequence of these decisions is that usually the structures learned with score and search methods offer better log-likelihood performance, and the constraint-based algorithms return structures with better structural accuracy, i.e., they return structures that represent the set of conditional independences in the distribution better [Scutari et al., 2019].

The hybrid methods try to combine the advantages of both types of methods. A commonly used hybrid algorithm is max-min hill-climbing (MMHC) algorithm [Tsamardinos et al., 2006]. The algorithm is described in Algorithm 4.4. The MMHC algorithm first runs the

Algorithm 4.3 $\overline{\text{MMPC}}$ algorithm

Require: Association function $Assoc$, target random variable G, candidate variables to the parent children set $\mathbf{C}$
Ensure: Candidate parent children (CPC) set
// Forward phase
CPC $\leftarrow \emptyset$
repeat
  $F \leftarrow \arg\max_{H \in \mathbf{C}} \min_{S \subseteq \text{CPC}} Assoc(H, G \mid S)$
  $AssocF \leftarrow \max_{H \in \mathbf{C}} \min_{S \subseteq \text{CPC}} Assoc(H, G \mid S)$
  if $AssocF \neq 0$ **then**
    CPC $\leftarrow$ CPC $\cup F$
  end if
until CPC has not changed
// Backward phase
for $H \in$ CPC **do**
  if $\exists S \subseteq$ CPC such that $(H \perp\!\!\!\perp G \mid S)$ **then**
    CPC $\leftarrow$ CPC $\setminus \{H\}$
  end if
end for
return CPC

MMPC algorithm to detect a set of allowed arcs. These arcs are the result of executing an MMPC algorithm on each node. Then, the structure is learned executing an HC that only admits the set of allowed arcs. Gasse et al. [2012] propose a similar approach changing the constraint-based algorithm to be used at the start of the process.

Algorithm 4.4 MMHC algorithm

Require: A set of random variables $\mathbf{X}$
Ensure: Best found structure $\mathcal{G}$
for $G \in \mathbf{X}$ **do**
  $\text{CPC}_G \leftarrow \text{MMPC}(G, \mathbf{X} \setminus G)$
end for
$\mathcal{G} \leftarrow$ Run an HC algorithm (Algorithm 4.1) where the arc addition $X \rightarrow G$ is only allowed if $X \in \text{CPC}_G$ (and $G \in \text{CPC}_X$ by the symmetric correction)
return $\mathcal{G}$

Chapter 5

Anomaly Detection

5.1 Introduction

Anomaly detection is a task which consists of finding events in the data that differ significantly from the normal behavior of the system under study. This task is related to other well-known techniques called novelty detection and outlier detection. These terms are often used as synonyms to anomaly detection. Pimentel et al. [2014] highlight a subtle difference in the use of these terms: the novelty detection is dedicated to finding new patterns in the data, but these patterns do not mean that the system is not behaving normally. This usually happens when the system under study is dynamic, so the system changes its behavior over time and we want the model to adapt automatically to these changes. On the contrary, anomalies or outliers are events that are considered transitory, so they are never considered part of the normal behavior of the system. However, the techniques used in anomaly detection and novelty detection are very similar, so many authors use these two terms interchangeably.

Anomalies can ocurr in many different domains: industry [Martí et al., 2015], data networks [Ahmed et al., 2016; Moustafa et al., 2019], medicine [Fernando et al., 2021], fraud detection [Adewumi and Akinyelu, 2017], etc.

There are many different surveys on anomaly detection. Although there are earlier surveys of merit [Hodge and Austin, 2004; Markou and Singh, 2003a,b], Chandola et al. [2009] provide a specially comprehensive survey on anomaly detection, discussing many different aspects, such as the different types of anomalies, types of data available, and types of techniques used in the literature. Posterior surveys follow the same scheme to categorize the different types of techniques [Boukerche et al., 2020; Smiti, 2020]. Moreover, in the last years new interesting topics such as deep learning anomaly detection [Pang et al., 2021; Ruff et al., 2021], and anomaly detection in data streams [Boukerche et al., 2020] have arisen. Finally, many specific applications have been published in the last years as we showed above.

The chapter is organized as follows. Section 5.2 describes the most important characteristics of the anomaly detection problem. Section 5.3 introduces statistical anomaly detection, which is the main focus of this thesis.

5.2 Anomaly Detection

Chandola et al. [2009] classify the anomalies into three different types:

- *Point anomalies.* This type of anomaly takes place when an individual instance shows a different pattern to the normal data. Figure 5.1 illustrates the point anomaly concept. There are two classes of normal behavior C_1 and C_2. The points A_1, A_2 and the points in A_3 are far away from the normal behavior, so they are considered anomalies. Note that the determination of A_3 as an anomaly depends on the system under study. It can sometimes be difficult to automatically assess whether a cluster of points such as A_3 is an anomaly or a third normal pattern. Typically, these ambiguities must be resolved by an expert in the field.

- *Contextual anomalies.* Contextual anomalies are anomalous only in a specific context. For this purpose, data usually contain two types of attributes: contextual and behavioral. Contextual attributes specify the context of each instance. Behavioral attributes include all noncontextual information of each instance. Figure 5.2 illustrates an example of contextual anomalies. In this example the contextual attributes are the position components x and y. The behavioral attributes are the marker shape and color. The anomalous instances (A_1 and A_2) exhibit normal behavioral attributes but in the wrong context.

- *Collective anomalies.* A collective anomaly occurs when a set of related data is anomalous with respect to the rest of the data. Each instance of the anomalous set can be considered normal, but their ocurrence together is what makes the set anomalous. A collective anomaly usually takes place in structured data such as temporal data, graph data or spatial data. Figure 5.3 illustrates a collective anomaly highlighted in red. Note that the individual instances of the anomaly set take values that can be considered normal. However, their ocurrence in sequence can be considered anomalous.

The data used to train anomaly detection models can take different forms:

- *Supervised data.* The supervised data label each instance as normal or anomalous. Sometimes, the label also includes information about the specific pattern of normal behavior or the specific anomaly that ocurred. This type of data is not common because obtaining labels for each instance can be costly and usually requires human supervision. In addition, anomalies are rare in most applications, so the data is too unbalanced because there are only a few anomalous instances.

- *Semisupervised data.* In many contexts of machine learning, semisupervised data refers to partially labeled data. This alleviates the cost of labelling each instance. In the context of anomaly detection, semisupervised data also refer to data containing only instances of the normal pattern. This class of learning process is also known as one-class classification. This type of data is quite common in anomaly detection due to the scarcity of anomalous data.

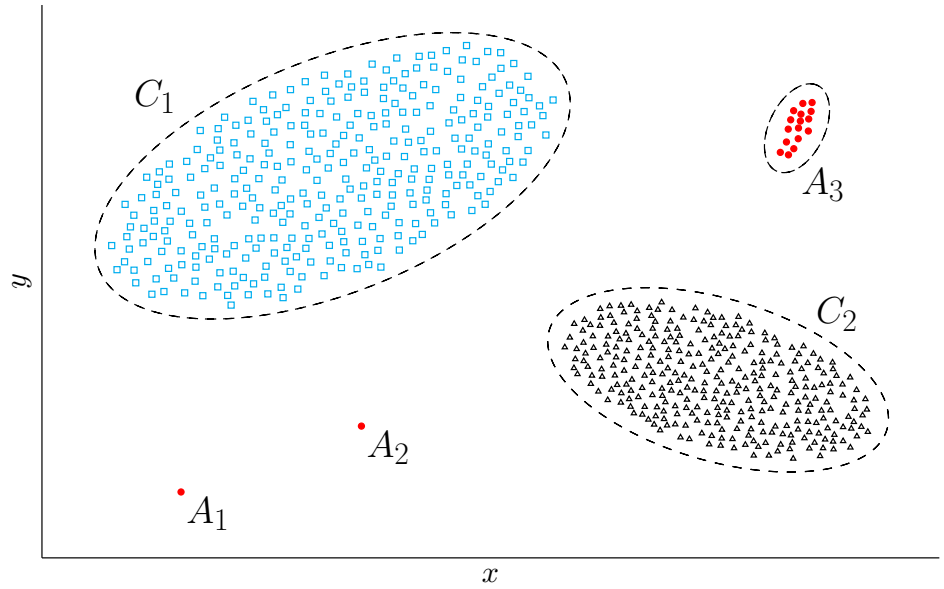


Figure 5.1: Illustration of point anomalies. The normal behavior patterns are represented as the points in C_1 and C_2. The points A_1, A_2 and the points in A_3 are considered anomalies.

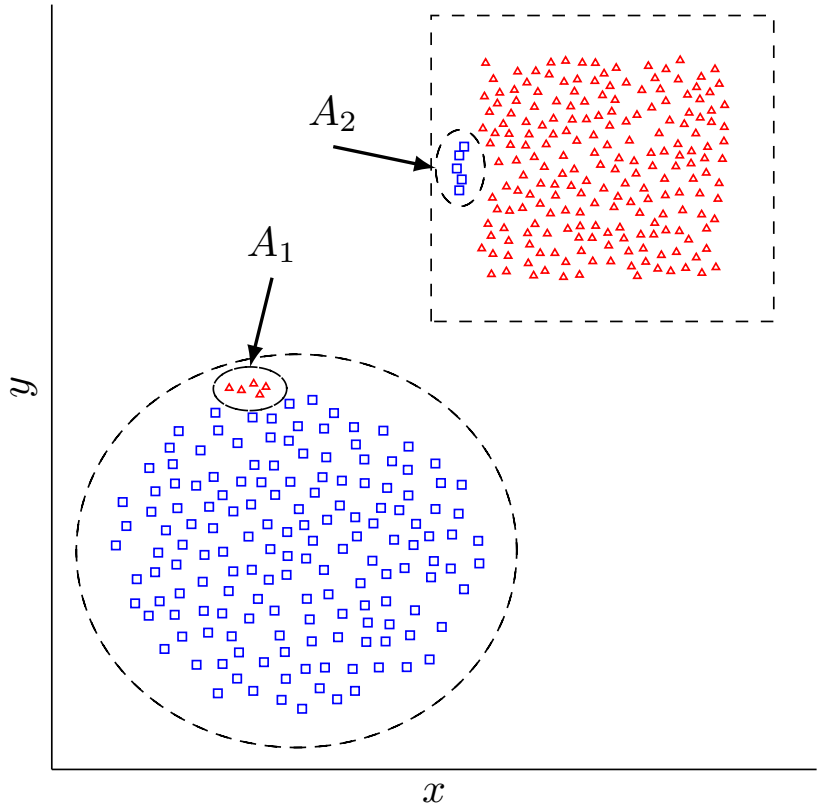


Figure 5.2: Illustration of contextual anomalies. The contextual attributes are the positional attributes (x and y), and the behavioral attributes are the marker shape and color.

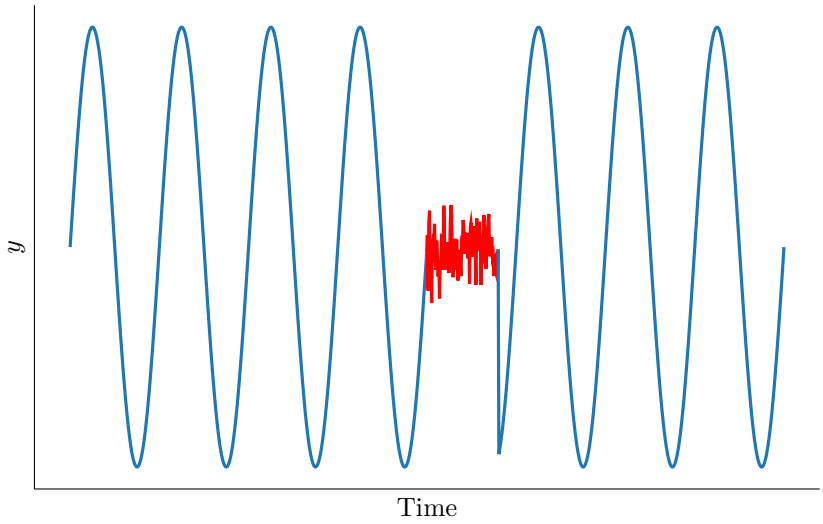


Figure 5.3: Illustration of collective anomalies. The collective anomaly is highlighted in red.

- *Unsupervised data.* In the unsupervised setting, the instances are not labelled. Therefore, we cannot assume that the data is either normal or anomalous. A commonly used assumption is that the data contain only a few instances, which are far away from the normal instance clusters.

An anomaly detection model has to discriminate between normal and anomalous data. Some models emit a label indicating if the instance is normal or anomalous. However, other models output an anomaly score. The anomaly score quantifies the degree of anomaly of an instance. Then, to classify an instance, the anomaly score is compared to an anomaly threshold. If the anomaly score is greater than the anomaly threshold, the instance is considered anomalous; otherwise the instance is considered normal.

There are many different models devoted to anomaly detection. Chandola et al. [2009] classified them into six categories:

- *Classification-based anomaly detection.* A classifier is learned from labelled data to discriminate between normal and anomalous data. Usually, these techniques require supervised data.

- *Nearest neighbor-based anomaly detection.* These techniques assume that normal data is concentrated on very dense areas, while the anomalies are far away from these areas.

- *Clustering-based anomaly detection.* These techniques assume that the normal data are part of some cluster, while the anomalies are not. These techniques are commonly used with unsupervised data.

- *Statistical anomaly detection.* The normal data is located in regions of high probability, while the anomalous data is located in low probability regions. In many cases, this assumption is true, because the anomalies are low probability events.

- *Information theoretic anomaly detection.* These techniques are based on the detection of changes in some information theory criteria, such as entropy, produced by the anomalies.

- *Spectral anomaly detection.* The spectral techniques project the data into a lower dimensional space. In this new space, the anomaly instances are very different from the normal instances.

Boukerche et al. [2020] added a new category of techniques:

- *Deep learning anomaly detection.* These techniques are based on some deep learning models such as robust deep autoencoders [Zhou and Paffenroth, 2017] and generative adversarial networks (GANs) [Goodfellow et al., 2014].

In this chapter we focus on statistical anomaly detection, as it is the category most closely related to the contributions proposed in this thesis.

5.3 Statistical Anomaly Detection

As noted above, statistical anomaly detection assumes that normal data are in high probability regions, while anomalous instances have a low probability. In many cases, taking advantage of this assumption requires building a model that estimates the probability of each instance. This aligns with the discussion of Chapter 3 on probability density estimation for continuous data. Furthermore, it is also related to the Bayesian network model (Chapter 4) which factorizes the distribution to improve the probability estimation.

As in probability density estimation, the anomaly detection can be addressed using parametric or nonparametric models. The parametric models assume that the distribution of the data is known, including the distribution of normal and anomalous data. On the contrary, nonparametric models do not make assumptions about the underlying distribution. Normally, we need at least to perform probability estimation for the normal data. Therefore, this estimation can be difficult if there are anomalous unlabeled data. Then, statistical anomaly detection is a specially good suit for labeled or one-class data. If the anomalous data are unlabeled, these techniques are only feasible if there are only a few anomalies.

5.3.1 Parametric Techniques

A common choice for the parametric model is assuming that the data is Gaussian distributed. Then, all the instances at a distance larger than 3σ from the mean can be considered anomalous. Moreover, other models assume Gaussianity of the data and then perform a statistical test such as Grubb's test [Grubbs, 1969] or χ^2 test [Ye and Chen, 2001] to detect data not distributed according to the Gaussian.

Alternatively, the data can be assumed to be generated from a mixture of distributions (usually, the Gaussian distribution). There are two main ways of learning a mixture depend-

ing on whether the data contains anomalies or is one-class. In the former case, the data is assumed to be generated from the following combination of two mixtures [Eskin, 2000]:

$$\hat{f}(\mathbf{x}) = (1-\lambda)\hat{f}_{\text{normal}}(\mathbf{x}) + \lambda\hat{f}_{\text{anomalous}}(\mathbf{x}), \tag{5.1}$$

where $\hat{f}_{\text{normal}}$ is a mixture learned with normal data, $\hat{f}_{\text{anomalous}}$ is a mixture learned with anomalous data, and λ is the a priori probability that an instance is anomalous. Then, the probability of $\mathbf{x}$ being anomalous can be calculated easily as:

$$P(\mathbf{x} \text{ is anomalous}) = \frac{\lambda\hat{f}_{\text{anomalous}}(\mathbf{x})}{(1-\lambda)\hat{f}_{\text{normal}}(\mathbf{x}) + \lambda\hat{f}_{\text{anomalous}}(\mathbf{x})}. \tag{5.2}$$

In the latter case, $\hat{f}_{\text{normal}}$ models the probability of $\mathbf{x}$ being normal, and a threshold is specified to classify the instance as anomalous.

Many probabilistic graphical models [Koller and Friedman, 2009] have been used in anomaly detection. Bayesian networks are used in many different works [Mascaro et al., 2014; Codetta-Raiteri and Portinale, 2015; Mori and Yu, 2013]. The anomalies are detected because a low probability is assigned to them by the Bayesian network. In particular, all these works use dynamic Bayesian networks because the data have a temporal component. Other simpler probabilistic graphical models have been used, such as the Kalman filter [Jäger et al., 2008a; van Wyk et al., 2020] and the HMM [Jäger and Hamprecht, 2009; Jäger et al., 2008b; Baruah and Chinnam, 2005; Zhou et al., 2004]. These models are also particularly suitable for temporal data.

Another technique suitable for temporal data is the symbolic anomaly detection [Ray, 2004; Rao et al., 2009]. The symbolic anomaly detection first assigns a symbol to each instance, which represents the state of the system. Then, it constructs a probabilistic finite state machine that estimates the probability of transition to a new symbol taking into account the last D instances. This model is called D-Markov machine. The anomaly detection is performed by comparing the state probability vector of the training and the test data.

5.3.2 Nonparametric Techniques

In nonparametric techniques no assumptions about the underlying distribution are performed. The two most usual models are histograms and KDEs, which we reviewed in Chapter 3. These techniques are especially suited for one-class data because they just model the distribution of the normal data.

The histogram has been used in many works, despite its apparent simplicity [Chandola et al., 2009]. This is important because working with histograms is computationally cheap, so they are mainly used for applications that require a fast response. This is the case, for example, for intrusion detection in data networks [Kind et al., 2009; Kim et al., 2006; Goldstein and Dengel, 2012].

On its part, the KDE models have been used in many other works such as Desforges et al. [1998]; Yeung and Chow [2002]; Zhang et al. [2018] and Hu et al. [2020].

Part III

CONTRIBUTIONS TO BAYESIAN NETWORKS

Chapter 6

Semiparametric Bayesian Networks

6.1 Introduction

Bayesian networks can be used to model uncertainty in domains containing continuous random variables. A common approach to process continuous random variables is to discretize them and learn the structure of a discrete Bayesian network from data, which can be done without assuming any underlying continuous distribution. However, this can be suboptimal due to the loss of information caused by discretization, meaning that different continuous values can be assigned to the same discrete category.

In the related literature, several methods have been introduced to model continuous random variables without discretizing data (Section 4.2). There are two main approaches to estimate continuous probability distributions: parametric and nonparametric models. Both types of estimations have been used to construct Bayesian networks as we showed in Section 4.2. In parametric models, a specific known probability distribution from a particular family is assumed concerning a given dataset. A distribution has a finite number of parameters that can be estimated from data. Such models are notably efficient if the specified assumptions hold with regard to a given dataset. Nonparametric models do not assume any specific probability distribution; however, generally, an estimated probability distribution emerges from training data. Nonparametric models are more flexible, as they can represent almost any probability distribution. However, they typically have worse error convergence rates with respect to the number of instances (i.e., their estimation error decreases at a slower rate than parametric models as the sample size increases) and are associated with higher computational costs in the cases when inferences over a distribution are performed.

In this chapter we propose to combine parametric and nonparametric estimation models to define a new class of continuous Bayesian networks. Hereinafter, we call these models semiparametric Bayesian networks (SPBNs). SPBNs are intended to combine the advantages of both parametric and nonparametric models. For this purpose, we define two types of CPDs: parametric and nonparametric. A CPD assigned to each node depends on its type. A parametric CPD can be used to represent linear relationships between random variables

using a LG distribution. A nonparametric CPD can be considered to represent nonlinear relationships given the flexibility of nonparametric models.

The contributions of the chapter are as follows: (1) definition of a new class of continuous Bayesian networks, SPBNs, that generalize GBNs and KDEBNs; (2) modification of the HC and PC algorithms to learn SPBN structures, which detect automatically the best type of CPD for each node. These modifications illustrate how a score and search algorithm, or a constraint-based learning algorithm can be adapted to learn SPBNs; (3) creation of a new learning operator for the score and search learning algorithms; (4) definition of a learning score inspired by cross-validation, which is score decomposable; and (5) a real use case of bearing degradation monitoring.

The chapter is organized as follows. In Section 6.2, the proposed approach is explained in detail, including the definition of a model with several useful theoretical propositions and the adaptation of two learning algorithms. Section 6.3 provides the discussion on the experimental results obtained by testing on artificial datasets, on datasets sampled from GBNs, on datasets extracted from the UCI repository, and on bearing degradation datasets. Section 6.4 concludes the chapter and outlines future research directions.

6.2 Semiparametric Bayesian Networks

In this section, we introduce SPBNs that combine the characteristics of parametric and nonparametric Bayesian networks. First, the representation of SPBN is detailed in Section 6.2.1. Then, two algorithms are proposed to automatically learn the structure of an SPBN from data in Section 6.2.2. Finally, the asymptotic time complexity of all learning procedures is analyzed in Section 6.2.3.

6.2.1 Representation of Semiparametric Bayesian Networks

SPBNs are composed of parametric and nonparametric CPDs. In this section, we describe their representation.

For the parametric CPDs, we used the well-known LG CPDs used in GBNs, as they are easy to train and usually offer good performance when there is a linear relationship between variables. The nonparametric CPDs are represented as the ratio of two joint KDE models, as in the KDEBNs. We denote this type of CPDs as conditional kernel density estimation (CKDE) distributions.

6.2.1.1 Linear Gaussian

The LG CPD is defined in Section 4.3.2. For a node X_i, the evidence variables in a Bayesian network are $\mathbf{X}_{\mathrm{Pa}(i)}$, so the LG CPDs are of the form:

$$\hat{f}_{\mathrm{LG}}(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)}) = \mathcal{N}\left(\beta_{i0} + \sum_{k \in \mathrm{Pa}(i)} \beta_{ik} x_k, \sigma_i^2\right). \tag{6.1}$$

Note that the LG CPD assumes that the random variable X_i is a linear combination of its parent random variables plus a normal error ϵ_i (Equation (4.16)). In addition, we assume that error variable ϵ_i is conditionally independent of all regressor variables X_k, for $k \in \text{Pa}(i)$.

In GBNs, random variable X_i follows an unconditional normal distribution, similarly as each parent random variable. This can be easily derived from Equation (4.16), because the linear combination of normal random variables is also normally distributed. Moreover, the unconditional distribution of multivariate random variables $(X_i, \mathbf{X}_{\text{Pa}(i)})$ and $\mathbf{X}$ is also multivariate normal distribution.

It should be noted that SPBNs do not make assumptions about the normality of parent random variables (see below). Therefore, in an SPBN, the unconditional distribution of random variables following the LG conditional distribution X_i and $(X_i, \mathbf{X}_{\text{Pa}(i)})$ may not be necessarily normal.

However, if the assumption of the normality of parents holds, then the unconditional distribution of X_i is exactly the same as in GBNs with the same unconditional distribution for parent random variables and the same parameter values.

Proposition 6.1. An SPBN in which all the nodes are LG CPDs is equivalent to a GBN (Section 4.2.1) with the same arcs and parameter values.

Proof. The proof is straightforward, as, by definition, a GBN is a Bayesian network in which all CPDs are LG CPDs. □

Based on Proposition 6.1, we can easily deduce that every possible GBN is contained in the class of SPBNs.

6.2.1.2 Conditional Kernel Density Estimation

Definition 6.1. (CKDE CPD). Let X_i be a random variable following a CKDE conditional distribution; then, the conditional distribution of X_i given $\mathbf{X}_{\text{Pa}(i)}$ is defined as:

$$\hat{f}_{\text{CKDE}}(x_i \mid \mathbf{x}_{\text{Pa}(i)}) = \frac{\hat{f}_{\text{KDE}}(x_i, \mathbf{x}_{\text{Pa}(i)})}{\hat{f}_{\text{KDE}}(\mathbf{x}_{\text{Pa}(i)})} = \frac{\sum_{j=1}^{N} K_{\mathbf{H}}\left(\begin{bmatrix} x_i \\ \mathbf{x}_{\text{Pa}(i)} \end{bmatrix} - \begin{bmatrix} x_i^j \\ \mathbf{x}_{\text{Pa}(i)}^j \end{bmatrix}\right)}{\sum_{j=1}^{N} K_{\mathbf{H}_{-i}}\left(\mathbf{x}_{\text{Pa}(i)} - \mathbf{x}_{\text{Pa}(i)}^j\right)}, \tag{6.2}$$

where $\hat{f}_{\text{KDE}}(x_i, \mathbf{x}_{\text{Pa}(i)})$ and $\hat{f}_{\text{KDE}}(\mathbf{x}_{\text{Pa}(i)})$ are KDE models, as defined in Equation (3.25), x_i^j and $\mathbf{x}_{\text{Pa}(i)}^j$ are the values in the j-th training instance for the variables X_i and $\mathbf{X}_{\text{Pa}(i)}$ respectively. $\mathbf{H}$ and $\mathbf{H}_{-i}$ are the bandwidth matrices for the KDE models $\hat{f}_{\text{KDE}}(x_i, \mathbf{x}_{\text{Pa}(i)})$ and $\hat{f}_{\text{KDE}}(\mathbf{x}_{\text{Pa}(i)})$, respectively.

This CPD does not assume any underlying distribution in data by modeling the multivariate random variable $(X_i, \mathbf{X}_{\text{Pa}(i)})$ using a nonparametric model. In this study, we use a Gaussian kernel for each CKDE; however, any kernel can be applied if a valid bandwidth matrix can be estimated (see Section 6.2.2.1).

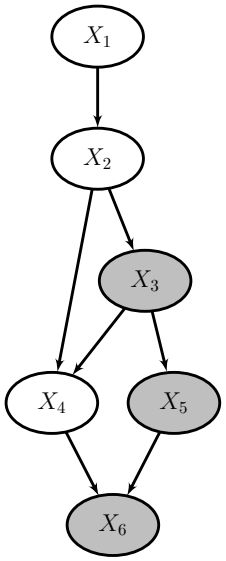


Figure 6.1: Structure of an example of SPBN. White nodes are of the LG type, and gray shaded nodes are of the CKDE type.

Therefore, the following proposition holds:

Proposition 6.2. An SPBN in which all variables follow a CKDE CPD is equivalent to a KDEBN model (Section 4.2.2) with the same arcs and bandwidth matrices and trained on the same data.

Proof. The proof is straightforward, as, by definition, a KDEBN is a Bayesian network in which all CPDs are CKDE CPDs. □

Based on Proposition 6.2, we can easily deduce that every possible KDEBN is contained in the class of SPBNs.

6.2.1.3 Graph Structure

In the SPBN model, the graph contains the type of each node, which determines the type of the corresponding CPD. There are no restrictions on the arcs, so the parent sets of each variable can be of different types: only LG parents, only CKDE parents or a mix of both options. Figure 6.1 illustrates an example of SPBN. Here, the LG and CKDE nodes are depicted using white and gray shaded nodes, respectively. As can be seen, there are different combinations of parent node types. Analyzing this structure, we can guarantee that the unconditional probability distribution of random variables X_1, X_2 and (X_1, X_2) is Gaussian. However, the unconditional probability distribution of remaining random variables cannot be known from the structure. The conditional distribution of variables X_1, X_2 and X_4 is known to be Gaussian, and their relationship with their parents is linear. The interpretability of the structure can serve as a useful tool to extract knowledge from an SPBN learned from data.

6.2.2 Learning of Semiparametric Bayesian Networks

In this section we present a procedure to learn the structure and parameters of a SPBN from data. As we introduced in Section 4.4 there are two main types of methods to learn the structure of a Bayesian network from data plus the hybrid approach. The constraint-based approaches are based on performing conditional independence tests and reconstructing a Bayesian network structure by representing the same tested conditional independences as accurately as possible [Spirtes et al., 2000]. The score and search approaches rely on defining a scoring function that measures how well the Bayesian network structure fits to the training data. Then, the structure learning problem turns into the search for the Bayesian network structure that scores the best.

In this chapter, we adapt a score and search procedure (HC), and also a constraint-based procedure (PC) to illustrate how both types of methods can be used to learn an SPBN.

6.2.2.1 Parameter Learning

Let us assume that the structure of a SPBN is fixed. That is, the set of arcs of a graph and the type of CPD of each node are known. Then, the parameters of each node CPD need to be estimated to complete the model. We will use standard techniques in the literature to learn the parameters, taking advantage of the locality of each CPD.

For the LG CPDs, the parameters can be learned using ordinary least squares (Section 4.3.2). Recall that the ordinary least square parameters are the MLE parameters.

The CKDE conditional distribution is composed of two nonparametric distributions: $\hat{f}_{\text{KDE}}(x_i, \mathbf{x}_{\text{Pa}(i)})$ and $\hat{f}_{\text{KDE}}(\mathbf{x}_{\text{Pa}(i)})$. For each nonparametric model, two bandwidth matrices $\mathbf{H}_i$ (for $\hat{f}_{\text{KDE}}(x_i, \mathbf{x}_{\text{Pa}(i)})$) and $\mathbf{H}_{-i}$ (for $\hat{f}_{\text{KDE}}(\mathbf{x}_{\text{Pa}(i)})$) need to be estimated. However, both bandwidth matrices cannot be computed independently, as the conditional distribution $\hat{f}_{\text{CKDE}}(x_i \mid \mathbf{x}_{\text{Pa}(i)})$ must integrate to 1:

$$\int_{-\infty}^{\infty} \hat{f}_{\text{CKDE}}(x_i \mid \mathbf{x}_{\text{Pa}(i)}) dx_i = 1, \forall \mathbf{x}_{\text{Pa}(i)},$$

Then, a valid selection of $\mathbf{H}_i$ and $\mathbf{H}_{-i}$ ensures that:

$$\hat{f}_{\text{KDE}}(\mathbf{x}_{\text{Pa}(i)}) = \int_{-\infty}^{\infty} \hat{f}_{\text{KDE}}(x_i, \mathbf{x}_{\text{Pa}(i)}) dx_i, \ \forall \mathbf{x}_{\text{Pa}(i)}.$$

Expanding the expression (without constant terms $\frac{1}{N}$) in both KDE models, we can formulate the following statement:

$$\int_{-\infty}^{\infty} \sum_{j=1}^{N} K_{\mathbf{H}} \left(\begin{bmatrix} x_i \\ \mathbf{x}_{\text{Pa}(i)} \end{bmatrix} - \begin{bmatrix} x_i^j \\ \mathbf{x}_{\text{Pa}(i)}^j \end{bmatrix} \right) dx_i = \sum_{j=1}^{N} K_{\mathbf{H}_{-i}} \left(\mathbf{x}_{\text{Pa}(i)} - \mathbf{x}_{\text{Pa}(i)}^j \right).$$

Using Fubini's theorem to switch the integral and the summation we have:

$$\sum_{j=1}^{N} \int_{-\infty}^{\infty} K_{\mathbf{H}} \left(\begin{bmatrix} x_i \\ \mathbf{x}_{\mathrm{Pa}(i)} \end{bmatrix} - \begin{bmatrix} x_i^j \\ \mathbf{x}_{\mathrm{Pa}(i)}^j \end{bmatrix} \right) dx_i = \sum_{j=1}^{N} K_{\mathbf{H}_{-i}} \left(\mathbf{x}_{\mathrm{Pa}(i)} - \mathbf{x}_{\mathrm{Pa}(i)}^j \right).$$

If $K_{\mathbf{H}}$ is a normal multivariate kernel, the integral can be easily computed. If the $\mathbf{H}$ matrix is defined by blocks:

$$\mathbf{H} = \begin{bmatrix} a & \mathbf{b}^T \\ \mathbf{b} & \mathbf{C} \end{bmatrix}, \tag{6.3}$$

the integral of $K_{\mathbf{H}}$ is a kernel $K_{\mathbf{C}}$ verifying:

$$\sum_{j=1}^{N} K_{\mathbf{C}} \left(\mathbf{x}_{\mathrm{Pa}(i)} - \mathbf{x}_{\mathrm{Pa}(i)}^j \right) = \sum_{j=1}^{N} K_{\mathbf{H}_{-i}} \left(\mathbf{x}_{\mathrm{Pa}(i)} - \mathbf{x}_{\mathrm{Pa}(i)}^j \right). \tag{6.4}$$

The expression in Equation (6.4) is true for any dataset $\mathcal{D}$ if and only if $\mathbf{H}_{-i} = \mathbf{C}$. Thus, to select the bandwidths of the CKDE, we only need to select $\mathbf{H}$ and assign $\mathbf{H}_{-i} = \mathbf{C}$.

Here, $\mathbf{H}_i$ cannot be estimated using the MLE because the training data constitute a part of the KDE model. From Equation (3.25), when calculating the likelihood for instance $\mathbf{x}^i$, there is a term $K_{\mathbf{H}}(\mathbf{0})$. This is the largest term in the sum of Equation (3.25) and is maximized for bandwidth $\mathbf{H}_i$ with determinant $|\mathbf{H}_i| \to 0$, leading to a likelihood approaching infinity which clearly overestimates the goodness of $\mathbf{H}$.

The bandwidth $\mathbf{H}$ can be selected with any of the bandwidth selection methods of Section 3.3.3. In this chapter, we employ the Scott's rule (Section 3.3.3.1) [Scott, 2015], which is a fast rule-of-thumb estimator:

$$\mathbf{H}_i = N^{-2/(|\mathrm{Pa}(i)|+5)} \hat{\mathbf{\Sigma}}, \tag{6.5}$$

where $\hat{\mathbf{\Sigma}}$ is the sample covariance matrix of random variables X_i and $\mathbf{X}_{\mathrm{Pa}(i)}$. In this study, we use a full covariance matrix to estimate $\mathbf{H}_i$. Other previous works [Hofmann and Tresp, 1995; Pérez et al., 2009] used diagonal matrices (also called product kernels for the Gaussian kernel [Scott, 2015]), i.e., $\mathbf{H} = h \cdot \mathrm{diag}(s_1, \ldots, s_n)$, where diag() defines a diagonal matrix, h is a smoothing parameter, and s_i is the standard deviation of X_i. Thus, we consider our model more flexible than previous approaches. Testing other bandwidth selection methods is left as a future research direction.

6.2.2.2 Greedy Hill-Climbing

In this section, we adapt the HC algorithm in Section 4.4.1 to learn the structure of an SPBN. Recall that the HC algorithm is aimed at optimizing the structure of a network by moving through the space of structures applying operators that, generally, make small and local changes on a candidate structure. The set of operators defines a neighborhood set of candidate structures. At each step, the operator that produces the largest improvement in score is applied to generate a new candidate structure. The algorithm runs until a local

optimum (which can be a global optimum) is reached.

Generally, three operators are utilized in HC: arc addition, arc removal, and arc reversal. In SPBNs, the structure is composed of arcs in a graph and the types of nodes, namely, LG or CKDE conditional distributions. Then, a new operator is added into the HC algorithm to learn SPBNs: node type change. This operator is denoted by TYPE-CHANGE(i), where i is a node index. The TYPE-CHANGE operator can change the type of a single node in a graph. That is, an LG node can be changed to be a CKDE node, and vice versa.

The definition of a score function is an important part of the score and search algorithms. We discussed some interesting properties of the score function in Section 4.4.1.1. The most common scores for GBNs are BIC (Equation (4.18)) and BGe.

However, for an SPBN, any score including the log-likelihood of the training data, such as the maximum likelihood score or BIC, are inappropriate because the training data constitute part of the KDE model. As shown in Section 6.2.2.1, for each training instance, there would be a term $K_{\mathbf{H}}(\mathbf{0})$ defined in Equation (3.25). Considering that the maximum argument of the kernel function is often $\mathbf{0}$, i.e., $\arg\max_{\mathbf{x}} K_{\mathbf{H}}(\mathbf{x}) = \mathbf{0}$, the log-likelihood of the training data overestimates the goodness of a model on unseen data. This is explained because the probability that the unseen data is exactly the same as the training data is 0, so there will not be $K_{\mathbf{H}}(\mathbf{0})$ terms almost surely while evaluating the log-likelihood of the unseen data. In addition, the BIC score cannot be calculated because a nonparametric model does not have a fixed (and countable) number of parameters.

From these observations, it is reasonable that the data used to fit the CPDs must be different than the data to evaluate the goodness of the model. Applying a k-fold cross-validation to the data can split the data to achieve this objective, while ensuring all the data are used. A k-fold cross-validation splits the data into k disjoint subsets. We will denote $\mathcal{I}^i$ the instance indices for the i-th fold, so $\mathcal{D}_{\downarrow\mathcal{I}^i}$ is the i-th fold data. The indices not in the i-th fold will be denoted $\mathcal{I}^{-i} = \bigcup_{m=1:m\neq i}^{k} \mathcal{I}^m$. The set of indices is represented as $\mathcal{I} = \{\mathcal{I}^1, \ldots, \mathcal{I}^k\}$. A k-fold cross-validated likelihood score is then:

$$\mathcal{S}^k_{\mathrm{CV}}(\mathcal{D}, \mathcal{G}) = \sum_{m=1}^{k} \mathcal{L}(\mathcal{G}, \boldsymbol{\theta}^{\mathcal{I}^{-m}} : \mathcal{D}_{\downarrow\mathcal{I}^m}), \tag{6.6}$$

where $\boldsymbol{\theta}^{\mathcal{I}^{-m}}$ are the parameters trained with the data $\mathcal{D}_{\downarrow\mathcal{I}^{-m}}$ using the parameter learning procedure described in Section 4.3. We note that the likelihood function is computed by Equation (4.7), where the contribution of each node depends on its type, as in Definition 4.8 for LG nodes and in Definition 6.1 for CKDE nodes.

The $\mathcal{S}^k_{\mathrm{CV}}$ is a valid score to train SPBNs because the log-likelihood is calculated over data that were not seen while estimating the parameters. In addition, the score $\mathcal{S}^k_{\mathrm{CV}}$ can decrease by the addition of some arcs as opposed to the maximum likelihood score (Section 4.4.1.1). This event takes place when the arc addition does not improve the model performance in unseen data, thus avoiding the overfitting. Therefore, the score can control the complexity of the model automatically. $\mathcal{S}^k_{\mathrm{CV}}$ can be understood as an estimator of the expected log-likelihood of the model, i.e., the expected log-likelihood for the new and unseen data.

Some scores have the property of decomposability (see Section 4.4.1.1). A score $\mathcal{S}$ is decomposable if it can be expressed as the sum of local score terms related to each node. The decomposability of a score is important, as a local change in a structure only modifies a limited number of local score terms. Then, during the structure search process, the delta score provided by each operator can be efficiently cached. It has been demonstrated that log-likelihood, BIC, and BGe scores are decomposable [Koller and Friedman, 2009; Geiger and Heckerman, 1994]. The cross-validated score in Equation (6.6) is also decomposable given a selection of disjoint sets of indices $\mathcal{I}$, as it is just summing k log-likelihood scores, which are themselves decomposable.

Note that the score of Equation (6.6) would return different results for differents sets of indices $\mathcal{I}$ and $\mathcal{I}'$, as the training and test data used to estimate the score of each fold is different. For this reason, we need to fix a specific set of indices $\mathcal{I}$ during the learning process to take advantage of the decomposability of the score. Otherwise, if the set of indices $\mathcal{I}$ is changed, the cached delta scores are no longer valid. However, fixing the set of indices for the cross-validated log-likelihood can induce searching for solutions that are only optimal for the specific set of indices $\mathcal{I}$. This effect can be understood as overfitting the set of indices $\mathcal{I}$. This is a type of overfitting that can be solved using the early-stopping criterion [Prechelt, 2012] that randomly splits the data $\mathcal{D}$ into two disjoint datasets called the training and validation sets, $\mathcal{D} = \mathcal{D}^{\text{train}} \cup \mathcal{D}^{\text{val}}$. Now, the learning process will be guided by the subset $\mathcal{D}^{\text{train}}$ and a fixed set of indices $\mathcal{I}$ over the data $\mathcal{D}^{\text{train}}$, while the subset $\mathcal{D}^{\text{val}}$ controls the overfitting to the set of indices $\mathcal{I}$. Thus, the selection of new operators in HC is performed using the score $\mathcal{S}^k_{\text{CV}}(\mathcal{D}^{\text{train}}, \mathcal{G})$, while the overfitting is controlled using $\mathcal{D}^{\text{val}}$ measuring the goodness of the new structure of each iteration as:

$$\mathcal{S}_{\text{validation}}(\mathcal{D}^{\text{train}}, \mathcal{D}^{\text{val}}, \mathcal{G}) = \mathcal{L}(\mathcal{G}, \boldsymbol{\theta}^{\text{train}} : \mathcal{D}^{\text{val}}), \tag{6.7}$$

where $\boldsymbol{\theta}^{\text{train}}$ are the parameters estimated using the full training set $\mathcal{D}^{\text{train}}$. If the structure overfits $\mathcal{I}$ with a given operator, the $\mathcal{S}^k_{\text{CV}}$ delta score will be positive, but the $\mathcal{S}_{\text{validation}}$ delta score will be negative because the operator deteriorates the generalization ability of the model. However, if the delta of both scores are positive, the operator is clearly useful to increase the performance of the model.

The HC algorithm selects and applies the best operator found on each iteration until no operator provides an improvement. The HC usually ends up finding a local optimum. By definition, the local optimum structure does not contain in the neighborhood of the search space a better structure, i.e., there is no operator that improves the current structure. In our HC implementation, we try to relax this restriction. Therefore, we allow the structure not to improve the score for a maximum of λ iterations. Here, λ is a parameter called patience. If λ is greater than 0, we allow exploration beyond the local neighborhood trying to escape the local optimum. To improve the exploration, we implemented a tabu search [Glover and Laguna, 1993] that is executed while trying to escape from local optimum. The tabu search forbids applying operators that reverse recently applied ones, e.g., an arc addition is not allowed if the same arc removal was executed recently. If the algorithm manages to escape

the local optimum, the tabu search is disabled. Thus, when exploration of the search space is not needed, the search is dedicated to finding the optimum as fast as possible.

Algorithm 6.1 Greedy hill-climbing for SPBNs

Require: Training data $\mathcal{D}$, starting structure $\mathcal{G}_0$, the set of operators $\mathcal{O}$, patience $\lambda \in \mathbb{N}$, the number of folds $k \geq 2$ (and $k < N$), minimum delta $\epsilon \geq 0$

1: $\mathcal{G}_{\text{best}} \leftarrow \mathcal{G}_0$
2: $\mathcal{G}_{\text{new}} \leftarrow \mathcal{G}_0$
3: $p \leftarrow 0$
4: Tabu $\leftarrow \emptyset$
5: $\mathcal{D}^{\text{train}}, \mathcal{D}^{\text{val}} \leftarrow \text{Split}(\mathcal{D})$
6: $\mathcal{I} \leftarrow$ Generate k sets of disjoint indices for $\mathcal{D}^{\text{train}}$
7: Assign $\mathcal{I}$ to $\mathcal{S}^k_{\text{CV}}$
8: **repeat**
9: $\quad\mathcal{G} \leftarrow \mathcal{G}_{\text{new}}$
10: $\quad$**for** o in $\mathcal{O}$ **do**
11: $\quad\quad$**if** o does not reverse $o' \in$ Tabu **then**
12: $\quad\quad\quad\mathcal{G}_{\text{candidate}} \leftarrow o(\mathcal{G})$
13: $\quad\quad\quad$**if** $\mathcal{S}^k_{\text{CV}}(\mathcal{D}^{\text{train}}, \mathcal{G}_{\text{candidate}}) > \mathcal{S}^k_{\text{CV}}(\mathcal{D}^{\text{train}}, \mathcal{G}_{\text{new}})$ **and** $\mathcal{S}^k_{\text{CV}}(\mathcal{D}^{\text{train}}, \mathcal{G}_{\text{candidate}}) - \mathcal{S}^k_{\text{CV}}(\mathcal{D}^{\text{train}}, \mathcal{G}) > \epsilon$ **then**
14: $\quad\quad\quad\quad o_{\text{new}} \leftarrow o$
15: $\quad\quad\quad\quad\mathcal{G}_{\text{new}} \leftarrow \mathcal{G}_{\text{candidate}}$
16: $\quad\quad\quad$**end if**
17: $\quad\quad$**end if**
18: $\quad$**end for**
19: $\quad$**if** $\mathcal{S}_{\text{validation}}(\mathcal{D}^{\text{train}}, \mathcal{D}^{\text{val}}, \mathcal{G}_{\text{new}}) > \mathcal{S}_{\text{validation}}(\mathcal{D}^{\text{train}}, \mathcal{D}^{\text{val}}, \mathcal{G}_{\text{best}})$ **then**
20: $\quad\quad\mathcal{G}_{\text{best}} \leftarrow \mathcal{G}_{\text{new}}$
21: $\quad\quad$Tabu $\leftarrow \emptyset$
22: $\quad\quad p \leftarrow 0$
23: $\quad$**else**
24: $\quad\quad$Tabu $\leftarrow$ Tabu $\cup\, o_{\text{new}}$
25: $\quad\quad p \leftarrow p + 1$
26: $\quad$**end if**
27: $\quad$UPDATE_SCORE_CACHE$(\mathcal{G}, o_{\text{new}})$
28: **until** $p < \lambda$
29: **return** $\mathcal{G}_{\text{best}}$

Algorithm 6.1 details the implementation of our HC strategy combined with tabu search. The algorithm starts by doing some basic initializations (lines 1-4), splitting the data into training and validation sets (line 5), and assigning a set of k-fold cross-validation indices $\mathcal{I}$ (lines 6-7). Then, the algorithm starts optimizing the structure in the main loop (lines 8-28). First, the algorithm finds the best available operator (lines 10-18). We include the requirement that the delta score $\mathcal{S}^k_{\text{CV}}(\mathcal{D}^{\text{train}}, \mathcal{G}_{\text{candidate}}) - \mathcal{S}^k_{\text{CV}}(\mathcal{D}^{\text{train}}, \mathcal{G})$ must be greater than a threshold $\epsilon \geq 0$. In this work, we always use $\epsilon = 0$ because it guarantees that the selected operator improves the $\mathcal{S}^k_{\text{CV}}$ score. Then, the algorithm checks that the best operator improves the validation score $\mathcal{S}_{\text{validation}}$ (line 19). If it improves the validation score (lines 20-22), the

best structure so far has been found, the tabu search is disabled and the patience counter, p, is reset to 0. Otherwise, the tabu search set is updated and the patience counter is increased (lines 24-25). Exploiting the decomposability property of the cross-validated log-likelihood, the algorithm updates the cached delta scores, as described in line 27. This procedure allows modifying only the delta scores affected by applying operator o_{new}. This update function for the arc operators is well-known in the literature [Koller and Friedman, 2009], so we do not include it here. The delta score for the operator TYPE-CHANGE(i) is:

$$\Delta\text{TYPE-CHANGE}(i) = \mathcal{S}^k_{\text{CV}}(X_i \mid \mathbf{X}_{\text{Pa}(i)}, \neg\text{Type}(i)) - \mathcal{S}^k_{\text{CV}}(X_i \mid \mathbf{X}_{\text{Pa}(i)}, \text{Type}(i)) \tag{6.8}$$

where $\mathcal{S}^k_{\text{CV}}(X_i \mid \mathbf{X}_{\text{Pa}(i)}, \text{Type}(i))$ is the local score of variable X_i, with parents $\mathbf{X}_{\text{Pa}(i)}$, when the type of CPD for X_i is determined by the function Type(i). The complement of function Type(i) is denoted as $\neg$Type(i), that is, $\neg$CKDE is LG, and $\neg$LG is CKDE. On each iteration of the HC algorithm, only a small amount of delta scores need to be updated depending on the last operator applied. If the last operator applied was an addition or removal of an arc $s \to d$, only ΔTYPE-CHANGE(d) needs to be updated. For the reversal of the same arc, both ΔTYPE-CHANGE(s) and ΔTYPE-CHANGE(d) need the update. Lastly, if an operator TYPE-CHANGE(i) is applied, only ΔTYPE-CHANGE(i) changes.

6.2.2.3 PC algorithm

The PC algorithm learns the structure of the Bayesian network by performing conditional independence tests to construct the graph that best captures the conditional independence relationships. The PC algorithm assumes that the underlying distribution is faithful to the Bayesian network graph, so that if two variables X_i and X_j are conditionally independent given a separating set of variables $\mathbf{S}_{ij}$, then the variables X_i and X_j must be d-separated in the graph $\mathcal{G}$ given $\mathbf{S}_{ij}$. Therefore, if no separating set $\mathbf{S}_{ij}$ can be found that makes X_i and X_j conditionally independent, then the nodes must be adjacent in the graph $\mathcal{G}$. The PC algorithm conducts the search for the separating sets $\mathbf{S}_{ij}$ that make all pairs of variables conditionally independent in an efficient manner. Once a skeleton is found that identifies which nodes are adjacent, the PC algorithm tries to orient the v-structures $X_i \to X_k \leftarrow X_j$ with X_i and X_j nonadjacent. A v-structure can be oriented if $X_k \notin \mathbf{S}_{ij}$. In this work, we used the MPC version of the PC stable algorithm [Colombo and Maathuis, 2014], which is guaranteed to always return the same structure, even if the order in which the variables are presented to the algorithm changes. The MPC version performs a new search of the possible separating sets $\mathbf{S}_{ij}$ for every v-structure candidate, and only orients a v-structure if the majority of the separating sets do not contain X_k. We omit the details of the algorithm here and we refer the reader to [Colombo and Maathuis, 2014] for more details. The end product of the PC algorithm is a partially directed acyclic graph (PDAG) that represents the skeleton of an equivalence class. This PDAG can be converted into a DAG of that equivalence class by using a simple algorithm [Dor and Tarsi, 1992].

One of the key components in a constraint-based algorithm is the type of conditional independence test. A common choice is the use of the PLC test [Fisher, 1915, 1921], which assumes that all the variables are distributed with a multivariate Gaussian. Since the SPBN model does not assume the distribution of any variable, we also used nonparametric conditional independence tests. In particular, we tested the CMIknn [Runge, 2018] conditional independence test, that is based on the estimation of the mutual information with K-nearest neighbors. However, this is a permutation conditional independence test and it was too slow to finish the high quantity number of tests needed for the PC algorithm in a reasonable time. Therefore, we also tested the randomized conditional correlation test (RCoT) [Strobl et al., 2019], which is faster because its distribution under the null hypothesis can be approximated with less computational resources.

Finally, to learn an SPBN, it is needed to establish the best type of CPD for each variable given the DAG learned by PC. An appealing approach would be to perform a statistical normality test, such as Shapiro-Wilks, on the regression residuals of the LG CPD. However, most of the normality tests have too much power when the sample size is too large (Rahman and Govindarajulu [1997] sets the limit at 5000 instances for Shapiro-Wilks), thus easily rejecting the null distribution of normality.

For this reason, we select the best node types with the execution of the HC algorithm described in Algorithm 6.1, but allowing only the operator Type-Change. This ensures that the arc selection returned by PC is not modified by HC.

6.2.3 Asymptotic Time Complexity

In this section we analyze the asymptotic time complexity of the different learning procedures. For both HC and PC algorithm, the execution time depends on the number of iterations needed. Usually, this number of iterations cannot be known in advance, since it depends on the starting model, the global optimum, the local optima present in the search path, the possible innacuracies caused by the score function or the conditional independence tests, and many other factors. The score function and the conditional independence tests are always the most computationally demanding elements of a learning algorithm as we will show in the following analysis.

In the HC algorithm, the set of arc operators contains $n(n-1)$ different operators for each graph (although some of them may be innaplicable because of the acyclicity constraint). Moreover, there are always n different Type-Change operators. The delta score of all these operators can be calculated with $n(n+1)$ evaluations of the score function, by caching the local score of each node in advance. Therefore, at the start of the HC algorithm, the number of score evaluations is quadratic on the number of nodes because the delta score of all operators is needed. The update of the delta scores after each iteration depends on the number of affected local scores, which can be 1 (arc addition, arc removal and node type change) or 2 (arc reversal). In the former case, only $n + |\text{Pa}(i)| - 1$ arc operators and 1 Type-Change operator change their delta score, where X_i is the affected local score node. This update can be completed with n score function evaluations (taking advantage of cached delta scores). In

the latter case, $2n + |\text{Pa}(i)| + |\text{Pa}(j)| - 3$ arc operators and 2 TYPE-CHANGE operator change their delta score, where X_i and X_j are the involved nodes in the arc reversal. This update can be completed with $2n$ score function evaluations. This analysis shows that, thanks to the decomposability of the score, the complexity decreases from quadratic to linear in the number of nodes for each iteration of the HC. In addition, to update the validation score, we only need 1 (arc addition, arc removal and node type change) or 2 (arc reversal) evaluations for each iteration.

We now present an analysis of the complexity of the score functions to compute the local score of node X_i with parents $\mathbf{X}_{\text{Pa}(i)}$. The complexity of the cross-validated score function (Equation (6.6)) is of the form $O(kF)$, where F is the cost of parameter learning and log-likelihood evaluation for each fold. This complexity is different for the LG and CKDE CPDs. Let $L = N/k$ and $J = N - L$ be the number of test and train instances on each fold respectively. For the LG CPD, it is necessary to find a least squares estimate, which has a complexity of $O\left(J|\text{Pa}(i)|^2\right)$. Once the least squares estimate is found, the log-likelihood of the test instances can be computed with complexity $O(L|\text{Pa}(i)|)$. Since $J \geq L$ and usually $J \gg L$, the complexity of the least squares estimate dominates the complexity of the log-likelihood evaluation. Therefore, the complexity of the cross-validated score function for LG CPDs is $O(kJ|\text{Pa}(i)|^2)$.

The CKDE CPD requires evaluating LJ multivariate Gaussian PDF (with dimensionality $(|\text{Pa}(i)|+1)$) for each fold. Each Gaussian evaluation has a complexity of $O\left((|\text{Pa}(i)|+1)^2\right)$ if the inverse and the determinant of the bandwidth matrix $\mathbf{H}$ are calculated in advance, which takes $O\left((|\text{Pa}(i)|+1)^3\right)$. Therefore, the complexity of the cross-validated score function for CKDE CPDs is $O(kLJ|\text{Pa}(i)|^2)$ or $O(NJ|\text{Pa}(i)|^2)$. This can be expressed with a looser bound as $O(N^2|\text{Pa}(i)|^2)$, which suggests that the complexity is quadratic with respect to the number of instances and the number of parents. Furthermore, we know that the complexity increases with k, so the less demanding setting is $k = 2$ and the most costly setting is $k = N-1$. In this work, we use $k = 10$ on all the experiments. This complexity might be probably reduced using an approximation such as random Fourier features [Rahimi and Recht, 2007], but we leave that approach as future work. In practice, this is an embarranssingly parallel problem [Herlihy and Shavit, 2008] because each multivariate Gaussian PDF can be executed independently. `PyBNesian` (Chapter 8) implements this parallel problem using OpenCL [Stone et al., 2010] to enable GPU acceleration, which significantly speeds up the execution.

The complexity of the BIC score is dominated by the least squares estimation of parameter learning, so it is equal to $O(N|\text{Pa}(i)|^2)$. The BGe score has a complexity of $O(N(|\text{Pa}(i)| + 1)^2 + (|\text{Pa}(i)| + 1)^3)$. The first term of the sum is because the sample sum of squared error of variables $\{X_i\} \cup \mathbf{X}_{\text{Pa}(i)}$ needs to be calculated [Kuipers et al., 2014]. The second term is the cost of calculating the determinant of a square matrix of size $|\text{Pa}(i)| + 1$. However, we can cache the sample sum of squared errors at the start of the HC algorithm in $O(Nn^2)$. Then, each score function evaluation has a cost of $O((|\text{Pa}(i)|+1)^3)$, which is usually preferable because, as we described before, the HC algorithm performs many score function evaluations.

The complexity of the PC algorithm is difficult to analyze because the number of iter-

ations depends on the size of the largest separating set $\mathbf{S}_{ij}$ (assuming a perfect conditional independence test). In the first iteration, $\frac{n(n-1)}{2}$ unconditional independence tests are performed. In the subsequent iterations, the number of independence test executions depends on the conditional independences found in the previous iterations, so the number of expected conditional independence tests cannot be calculated. In addition, the number of possible separating sets $\mathbf{S}_{ij}$ between a pair of variables X_i and X_j is equal to $\binom{|\text{adj}_l(X_i)|-1}{l} + \binom{|\text{adj}_l(X_j)|-1}{l}$, where $\text{adj}_l(X_i)$ is the set of adjacent nodes to node X_i at iteration l. Furthermore, in the worst case, it is necessary to perform a conditional independence test for each possible separating set. Thus, the number of conditional independence test executions can grow quickly in the worst case.

We analyze now the complexity of a conditional independence test between X_i and X_j given a separator set $\mathbf{S}_{ij}$. The PLC independence test can be calculated using the precision matrix of the set of variables $\{X_i\} \cup \{X_j\} \cup \mathbf{S}_{ij}$. Thus, the complexity of the PLC independence test is $O(N(|\mathbf{S}_{ij}| + 2)^2 + (|\mathbf{S}_{ij}| + 2)^3)$, where the first term comes from the calculation of the covariance matrix and the second term from the complexity of its inversion. As in BGe, we can cache the covariance matrix information at the start of the algorithm in $O(Nn^2)$, and then each evaluation of the conditional independence test can be performed in $O((|\mathbf{S}_{ij}|+2)^3)$. The RCoT independence test has a complexity of $O(N|\mathbf{S}_{ij}|)$, assuming the number of random Fourier features is fixed. This is the complexity of the computation of the random Fourier feature matrices, which is the most demanding procedure of the independence test. The authors provide a description of the complexity of RCoT [Strobl et al., 2019], so we do not include more details here.

6.3 Experimental Results

In this section, we discuss the results of the experiments conducted on SPBNs and the comparison with alternative methods. We conducted four types of experiments depending on the input data source: synthetic data sampled by mixing linear and nonlinear functions, data sampled from GBNs, data from the UCI repository [Dua and Graff, 2017], and bearing degradation data. Finally, the execution times of all algorithms are shown in Section 6.3.5.

The experiments were performed using the `PyBNesian`[1] package (Chapter 8). The source code of the experiments is available at `https://github.com/davenza/SPBN-Experiments`.

6.3.1 Synthetic Data

In this section, we discuss the results of applying the learning algorithms introduced in Section 6.2.2 to the artificial data. We sampled the data from the following probabilistic

[1] `https://github.com/davenza/PyBNesian`

model:

$$\begin{aligned} f(a) &= \mathcal{N}(\mu_A = 0, \sigma_A^2 = 1) \\ f(b) &= 0.5 \cdot \mathcal{N}(\mu_{B_1} = -2, \sigma_{B_1}^2 = 2) + 0.5 \cdot \mathcal{N}(\mu_{B_2} = 2, \sigma_{B_2}^2 = 2) \\ f(c|a,b) &= a \cdot b + \epsilon_C, \text{ where } \epsilon_C \sim \mathcal{N}(\mu_{\epsilon_C} = 0, \sigma_{\epsilon_C}^2 = 1) \\ f(d|c) &= \mathcal{N}(\mu_D = 10 + 0.8 \cdot c, \sigma_D^2 = 0.5) \\ f(e|d) &= \text{Sig}(d) + \epsilon_E, \text{ where } \epsilon_E \sim \mathcal{N}(\mu_{\epsilon_E} = 0, \sigma_{\epsilon_E}^2 = 0.5) \end{aligned} \tag{6.9}$$

where $\text{Sig}(x) = 1/(1+\exp(-x))$, is the sigmoid function. The set of conditional independences of the probabilistic model can be represented with an SPBN as in Figure 6.2. We selected this structure as it contained CKDE nodes with different types of parents. We sampled three training datasets with the different number of instances: 200, 2,000, and 10,000. In addition, we sampled another test dataset with 1,000 instances to evaluate the log-likelihood of each learned model and the ground truth model on unseen data. In such a way, we can quantitatively compare all models. To compare the learned structures, we also calculated the Hamming distance (HMD)[2] between the graphs corresponding to the learned models and the ground truth model. As HMD does not consider the direction of arcs, we also employed the structural Hamming distance [Tsamardinos et al., 2006] introduced to calculate the number of additions, removals, and reversals of arcs required to transform the DAG of the learned model into that of the ground truth model. Moreover, we computed a node type Hamming distance (THMD) measuring the number of nodes with a different node type in the learned and ground truth models. We ran the HC and PC algorithms with two different values of patience λ: 0 and 5. However, both options learned the same model, so we omit the λ parameter in this analysis. We tested HC using a starting graph $\mathcal{G}_0$ without arcs and with two configurations for the type nodes: all the nodes were LG (SPBN-LG) or all the nodes were CKDE (SPBN-CKDE). We observed that in the latter case, the resulting graphs tend to be more sparse. This is reasonable because the CKDE CPDs are more flexible than LG CPDs, so it does not need as much parents to obtain a good fit to the data. For this reason, we present only the results for $\mathcal{G}_0$ with CKDE nodes. The PC algorithm was executed using a PLC test (PC-PLC) and RCoT (PC-RCoT).

We present the results in Table 6.1. As expected, the ground truth demonstrated better log-likelihood compared with the learned models. Moreover, the log-likelihood and the structural accuracy improved with an increase in the number of training instances. With 10,000 instances both HC and PC-RCoT returned the ground truth structure. We can see that PC-RCoT always behaved better than PC-PLC. This is because the PLC test requires that all the variables are multivariate Gaussian, but the ground truth contains nonlinearities and some non-Gaussian distributions (such as the bimodal distribution in variable B). Furthermore, for 2,000 and 10,000 instances all the algorithms recovered the node type correctly for all the nodes.

[2]HMD between two graphs is used to evaluate the number of arcs that are present in a graph but not in the other graph ignoring arc directions

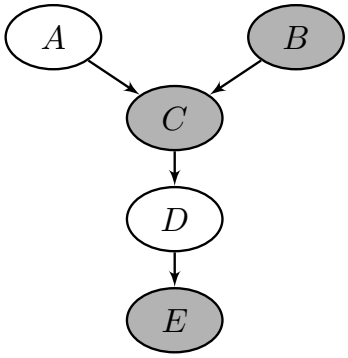
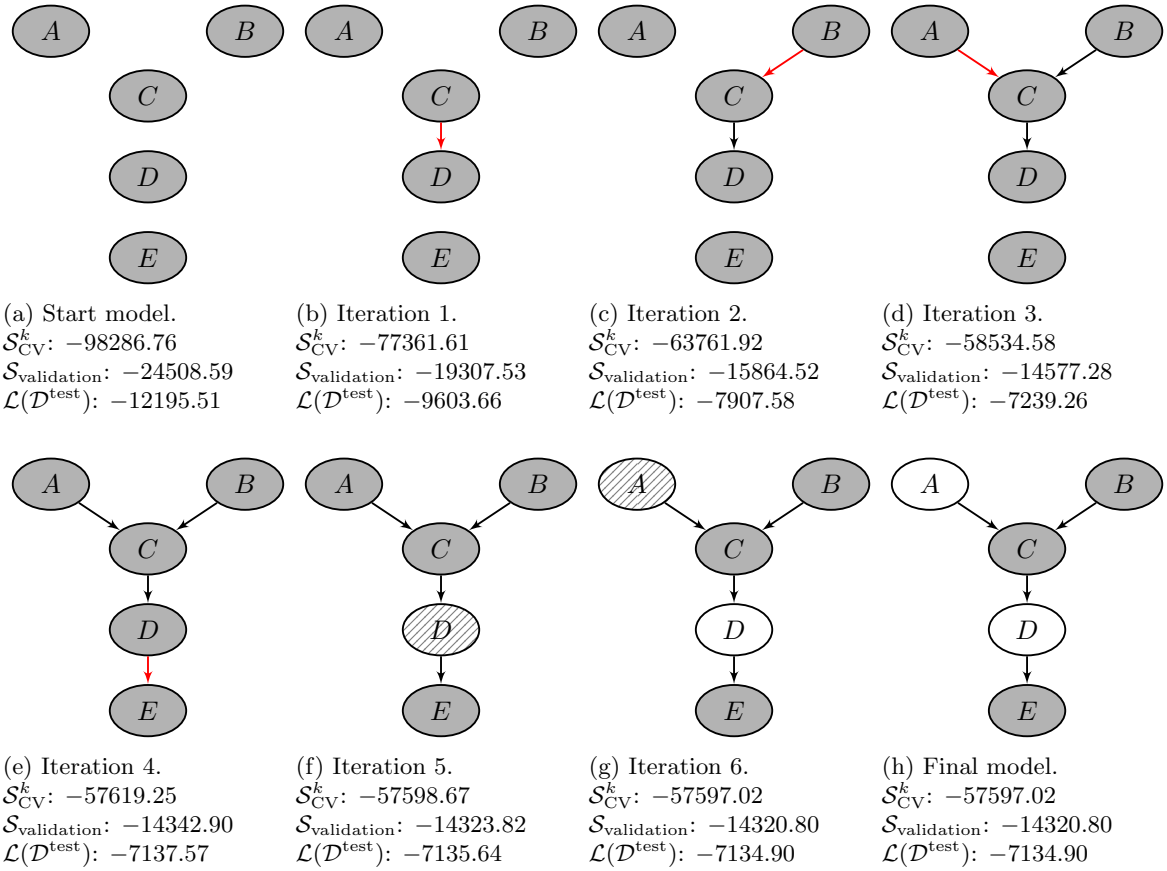


Figure 6.2: Ground truth SPBN. White nodes denote the LG type, and gray shaded nodes correspond to the CKDE type.

Model		Log-likelihood	HMD	SHD	THMD
Max possible value			10	10	5
Ground truth		−6982.23	0	0	0
	HC	−7479.48	0	0	2
200 instances	PC-PLC	−8034.62	3	3	2
	PC-RCoT	−8031.09	2	2	2
	HC	−7217.31	2	2	0
2,000 instances	PC-PLC	−7827.65	4	4	0
	PC-RCoT	−7316.59	1	1	0
	HC	−7134.90	0	0	0
10,000 instances	PC-PLC	−7817.06	3	4	0
	PC-RCoT	−7134.90	0	0	0

Table 6.1: Results of training using the synthetic data of Equation (6.9). HMD stands for the Hamming distance, SHD denotes the structural Hamming distance, and THMD corresponds to the node type Hamming distance computed between the learned model and the ground truth model.



(a) Start model.
$\mathcal{S}^k_{\text{CV}}$: -98286.76
$\mathcal{S}_{\text{validation}}$: -24508.59
$\mathcal{L}(\mathcal{D}^{\text{test}})$: -12195.51

(b) Iteration 1.
$\mathcal{S}^k_{\text{CV}}$: -77361.61
$\mathcal{S}_{\text{validation}}$: -19307.53
$\mathcal{L}(\mathcal{D}^{\text{test}})$: -9603.66

(c) Iteration 2.
$\mathcal{S}^k_{\text{CV}}$: -63761.92
$\mathcal{S}_{\text{validation}}$: -15864.52
$\mathcal{L}(\mathcal{D}^{\text{test}})$: -7907.58

(d) Iteration 3.
$\mathcal{S}^k_{\text{CV}}$: -58534.58
$\mathcal{S}_{\text{validation}}$: -14577.28
$\mathcal{L}(\mathcal{D}^{\text{test}})$: -7239.26

(e) Iteration 4.
$\mathcal{S}^k_{\text{CV}}$: -57619.25
$\mathcal{S}_{\text{validation}}$: -14342.90
$\mathcal{L}(\mathcal{D}^{\text{test}})$: -7137.57

(f) Iteration 5.
$\mathcal{S}^k_{\text{CV}}$: -57598.67
$\mathcal{S}_{\text{validation}}$: -14323.82
$\mathcal{L}(\mathcal{D}^{\text{test}})$: -7135.64

(g) Iteration 6.
$\mathcal{S}^k_{\text{CV}}$: -57597.02
$\mathcal{S}_{\text{validation}}$: -14320.80
$\mathcal{L}(\mathcal{D}^{\text{test}})$: -7134.90

(h) Final model.
$\mathcal{S}^k_{\text{CV}}$: -57597.02
$\mathcal{S}_{\text{validation}}$: -14320.80
$\mathcal{L}(\mathcal{D}^{\text{test}})$: -7134.90

Figure 6.3: Learning progress for the HC algorithm with 10,000 training instances from the start model to the final model. An arc addition is shown with a red arc. The change from CKDE node type to LG node type is shown with striped nodes. At each iteration, the training score, $\mathcal{S}^k_{\text{CV}}$, the validation score, $\mathcal{S}_{\text{validation}}$, and the test log-likelihood, $\mathcal{L}(\mathcal{D}^{\text{test}})$ (we omit the $\mathcal{G}, \boldsymbol{\theta}$ arguments of the $\mathcal{L}$ function), are shown.

To illustrate the learning progress, Figure 6.3 shows how the model changes at each iteration of the HC algorithm trained with 10,000 training instances. We can see that the algorithm first added all the arcs of the structure, and then, changed node types. It is important to note that Algorithm 6.1 allows interleaving arc operators with node type change operators. In this specific execution, the arcs have been added first because the CKDE CPD is good enough estimating a Gaussian distribution with 10,000 instances. Thus, in the first iterations the delta scores of the arc addition operators are higher. However, in the last iterations there are no more arc changes that improve the score, so the node types are changed because they provide a refinement over the CKDE CPD.

6.3.2 Data Sampled from Gaussian Networks

Considering that GBNs constitute a special case of SPBNs, in this section, we test the SPBN learning in the case when the training data follow a multivariate Gaussian distribution. We selected four GBNs from the `bnlearn`'s [Scutari, 2010] Bayesian network repository:

True model	Nodes	Arcs
ECOLI70	46	70
MAGIC-NIAB	44	66
MAGIC-IRRI	64	102
ARTH150	107	150

Table 6.2: Properties of the tested GBNs.

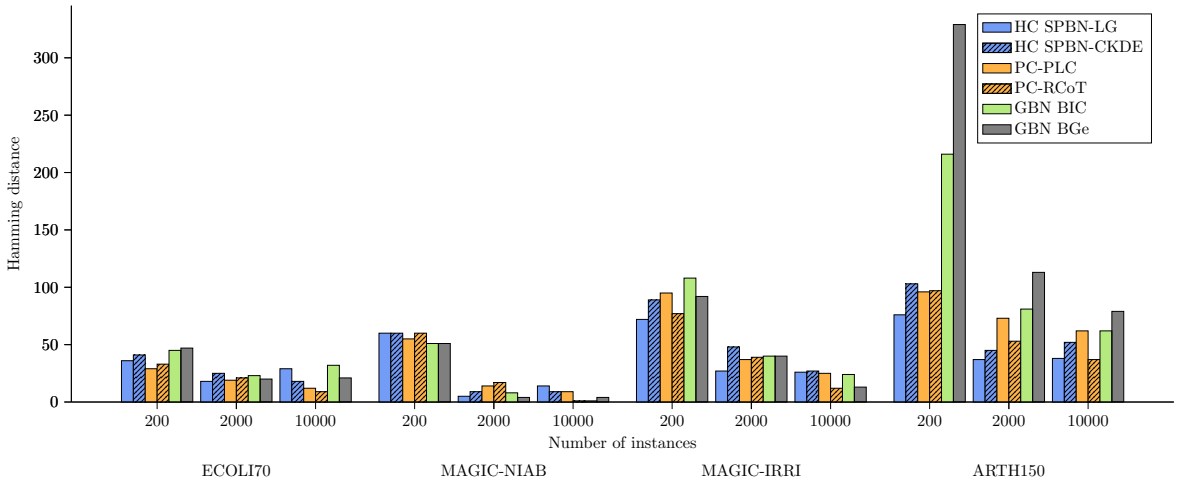


Figure 6.4: HMD of the trained models to the ground truth model.

ECOLI70, MAGIC-NIAB, MAGIC-IRRI, and ARTH150. We describe the properties of each Bayesian network in Table 6.2. For each GBN, we sampled three training datasets with the different number of instances: 200, 2,000, and 10,000. Similarly as in the previous section, we sampled a test dataset of 1,000 instances to compare the log-likelihood of the trained models. The GBN models were learned using the HC algorithm with the BIC and BGe scores, and also the PC-PLC and PC-RCoT algorithms. The SPBN models were learned using the same configurations as in the previous section. In this case, the patience $\lambda = 5$ performed a little better, so we omit the results for $\lambda = 0$ here.

The HMD and SHD measures of each trained model are represented in Figure 6.4 and Figure 6.5, respectively. All the models learned with PC have the same graph, so they are represented as PC-PLC and PC-RCoT in the figures. The SPBN-CKDE models have a worse structural accuracy than SPBN-LG. This is meaningful because in SPBN-LG the starting graph has a correct node type for all the nodes, so the algorithm only needs to optimize the arcs of the graph. Moreover, we can see that the HC algorithm for SPBNs is competitive with the PC algorithm in terms of structural accuracy. This is remarkable because the PC algorithm is known to be a better algorithm than HC in reducing the SHD [Scutari et al., 2019]. For the PC algorithm, there are not important differences between PC-PLC and PC-RCoT. In addition, BGe score shows a specially poor structural accuracy in ARTH150.

We show the THMD value between the SPBN models and the true model in Figure 6.6. In

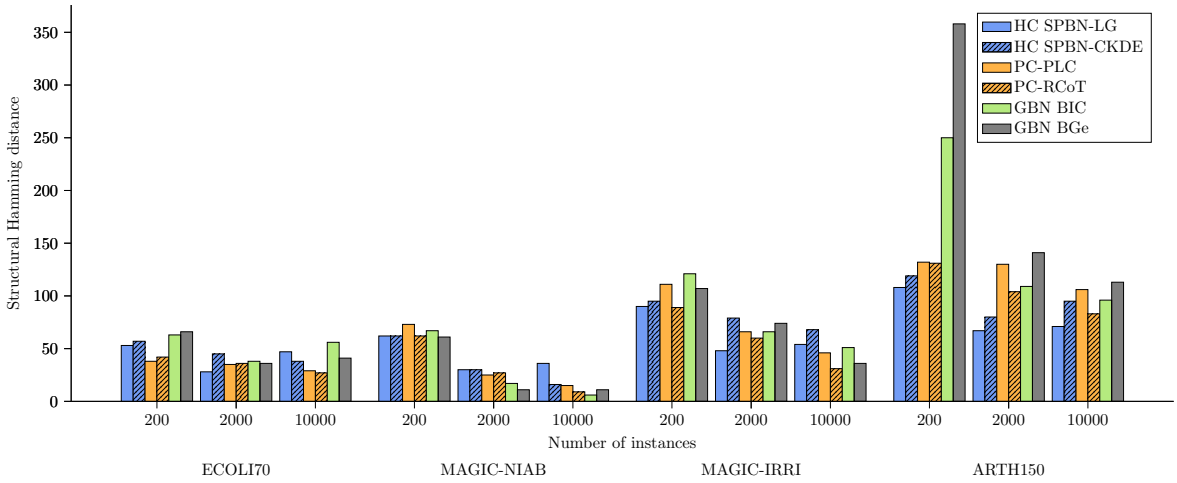


Figure 6.5: SHD of the trained models to the ground truth model.

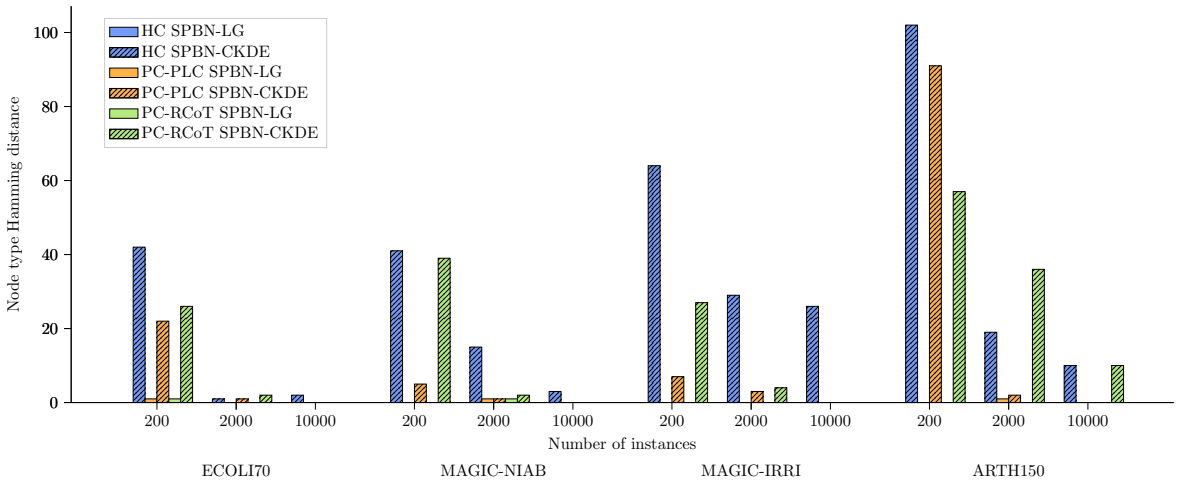


Figure 6.6: THMD of SPBNs to the ground truth model.

this experimental framework, THMD was equal to the number of CKDE nodes, as all nodes in the true model were of the LG type. It is clear that SPBN-LG outperformed SPBN-CKDE in finding the best node types. This is reasonable because the starting point of SPBN-LG is optimal in the node types, while the SPBN-CKDE is the worst model possible in the search space. However, there is a clear trend towards a THMD reduction when a larger sample size is available. For small sample sizes, it is possible that a normality test such as Shapiro-Wilks (as suggested in Section 6.2.2.3) could reduce the THMD, so we leave that analysis as future work.

The log-likelihood of the test dataset in the trained models is shown in Table 6.3. We observe rather similar results for all models and datasets when using the same learning algorithm. The HC algorithm tends to have a higher log-likelihood than the PC algorithm, which is coherent because the HC search process is guided by the improvement of the log-

GBN		ECOLI70			MAGIC-NIAB	
Instances	200	2,000	10,000	200	2,000	10,000
True model		−41522.34			−48469.84	
GBN BIC	−42426.52	−41610.22	−41529.44	−49682.02	−48537.96	**−48476.6**
GBN BGe	−42372.28	−41592.76	**−41526.69**	**−49587.08**	**−48536.56**	−48479.21
HC SPBN-LG	**−42258.08**	**−41580.96**	−41528.95	−49712.7	−48577.6	−48497.42
HC SPBN-CKDE	−45033.49	−41638.3	−41529.25	−50365.1	−48642.04	−48486.65
PC-PLC GBN	−44759.66	−44286.85	−43274.39	−49637.17	−48572.77	−48488.73
PC-RCoT GBN	−47721.66	−43604.97	−43013.12	−49655.38	−48678.99	−48513.3
PC-PLC SPBN-LG	−44818.33	−44286.85	−43274.39	−49637.17	−48577.39	−48488.73
PC-RCoT SPBN-LG	−47780.34	−43604.97	−43013.12	−49655.38	−48683.61	−48513.3
PC-PLC SPBN-CKDE	−45406.38	−44294.19	−43274.39	−49778.77	−48577.39	−48488.73
PC-RCoT SPBN-CKDE	−48381.11	−43618.0	−43013.12	−50333.59	−48684.85	−48513.3

GBN		MAGIC-IRRI			ARTH150	
Instances	200	2,000	10,000	200	2,000	10,000
True model		−76193.11			−36471.74	
GBN BIC	−78312.61	**−76322.27**	**−76209.87**	−41745.83	**−36709.18**	−36495.68
GBN BGe	−77986.66	−76353.57	−76213.31	−43537.06	−36755.5	−36500.39
HC SPBN-LG	**−77638.45**	−76377.09	−76218.89	**−38841.11**	−36709.64	**−36484.31**
HC SPBN-CKDE	−79465.69	−76576.7	−76279.79	−43112.32	−36837.14	−36510.63
PC-PLC GBN	−78319.14	−76844.8	−76598.0	−42207.35	−39526.29	−38491.47
PC-RCoT GBN	−78340.23	−76653.77	−76356.27	−45827.92	−41233.34	−40556.53
PC-PLC SPBN-LG	−78319.14	−76844.8	−76598.0	−42207.35	−39532.54	−38491.47
PC-RCoT SPBN-LG	−78340.23	−76653.77	−76356.27	−45827.92	−41233.34	−40556.53
PC-PLC SPBN-CKDE	−78517.12	−76857.15	−76598.0	−44519.13	−39539.88	−38491.47
PC-RCoT SPBN-CKDE	−78751.04	−76672.95	−76356.27	−46582.94	−41370.77	−40564.74

Table 6.3: Log-likelihood of the test dataset in the trained models using the data sampled from GBNs. We also show the log-likelihood of the test dataset in the ground truth model for reference (the best result for each dataset is highlighted boldface).

likelihood. To detect statistically significant differences between all algorithms, we performed a Friedman test with $\alpha = 0.05$ and a Bergmann-Hommel post-hoc procedure to detect the pairwise significant differences [García and Herrera, 2008]. We illustrate the obtained results graphically in Figure 6.7 using a critical difference diagram [Demšar, 2006] that represents the mean rank of each algorithm. The horizontal black lines connect the groups of algorithms that do not have a significant difference. The models learned with the HC algorithm have a statistical significant difference with the models learned with PC. However, the differences are not statistically significant between learning GBNs or SPBNs using the same algorithm. Therefore, we concluded that SPBN learning was as suitable as GBN learning for the training data sampled from a GBN.

6.3.3 UCI Repository Data

In this section, we present the results of testing the SPBN model learning based on the real data extracted from the UCI repository. In this experimental framework, we did not have the information about the structure of an underlying Bayesian network that produced the data. Table 6.4 presents the number of instances (N) and variables (n) in each dataset. Some of these datasets were designed for classification so that they included a discrete class variable.

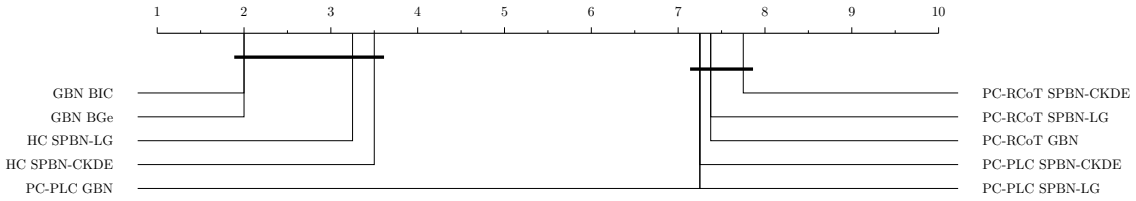


Figure 6.7: Critical difference diagram for the mean rank of each algorithm trained using the data sampled from GBNs.

Every class/discrete variable was removed from each dataset. The information shown in Table 6.4 reflects the number of variables after the removal procedure.

We compared the different types of continuous Bayesian networks: KDEBNs, GBNs, and SPBNs. We did not include discrete Bayesian networks because they model a probability mass function, as opposed to the continuous Bayesian networks that model PDF. As we did not know the structure of the underlying Bayesian network (if any) that produced the data, we tested the density estimation capabilities of each model. For this purpose, we applied a 10-fold cross-validation approach to estimate the log-likelihood of the unseen data in each model. Accordingly, ten models were trained using different training folds, and the log-likelihood was estimated in the unseen instances of the test fold. The estimation of the expected log-likelihood in the unseen data was derived as the mean test log-likelihood for every fold. For HC, the KDEBNs were learned using a procedure similar to the SPBN learning procedure described in Algorithm 6.1 with a cross-validation score and a validation dataset to detect convergence. However, the Type-Change operator is not valid to learn the structure of a KDEBN model (i.e., its node type is fixed in advance). We tested the same learning configurations for GBNs and SPBNs as in the previous sections. The selection of $\lambda = 5$ often returned better models than $\lambda = 0$ (for both KDEBNs and SPBNs), so we present only the results with $\lambda = 5$ in this section.

As in the previous section, we performed a Friedman test with $\alpha = 0.05$ and a Bergmann-Hommel post-hoc procedure. The critical difference diagram is represented in Figure 6.8. We concluded that SPBNs and KDEBNs perform better than GBNs because their differences in the expected log-likelihood were statistically significant. This was reasonable because, in general, the real data do not follow the multivariate Gaussian distribution. Also, the models trained with HC are the top ranked, except GBN BIC and GBN BGe. This is meaningful because the optimization criterion of HC is the cross-validated log-likelihood unlike the conditional independence tests in PC. The PLC test tends to have a better mean rank than RCoT, even though there is no statistically significant difference. This suggests that there are some linear relationships between the variables in the real data, so the PLC test is competitive with respect to RCoT. In addition, since the GBNs show a significant lower log-likelihood than the more flexible models, the real data probably also contain nonlinear

Dataset	N	n	Dataset	N	n
Balance	625	4	QSAR fish toxicity	908	7
Block	5473	10	Sensor	5456	24
Breast Cancer	683	9	Sonar	208	60
Breast Tissue	106	9	Spambase	4601	57
CPU	209	8	Vehicle	846	18
Cardiotocography	2126	19	Vowel	990	10
Ecoli	336	5	Waveform	5,000	21
Glass	214	9	Waveform-Noise	5,000	40
Ionosphere	351	33	Wdbc	569	30
Iris	150	4	Wine	178	13
Liver	345	6	WineQuality-Red	1599	12
Magic Gamma	19020	10	WineQuality-White	4898	12
Parkinsons	195	21	Wpbc	194	33
QSAR Aquatic	546	9	Yeast	1484	8

Table 6.4: Datasets from the UCI repository.

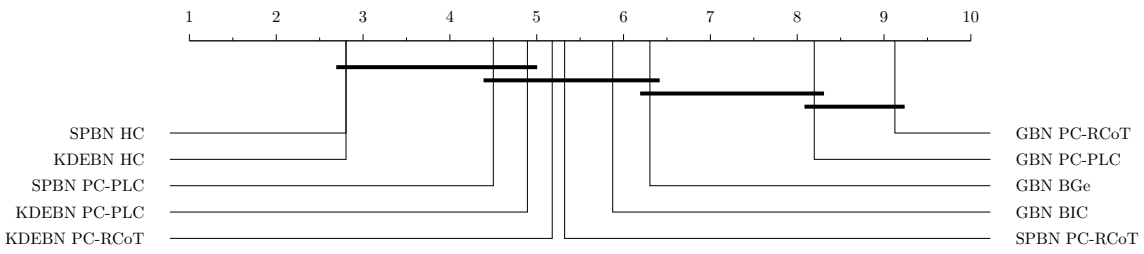


Figure 6.8: Critical difference diagram for the mean rank of each algorithm in the UCI datasets.

relationships. This emphasizes the importance of modeling explicitly a combination of linear and nonlinear relationships as SPBNs do.

The KDEBN and SPBN models obtain a fairly similar mean rank for all learning configurations, so there is not statistically significant difference. In Figure 6.9, we represent the ratio of CKDE nodes in the SPBN models for all the learning algorithms. The three different algorithms return a similar number of CKDE nodes. We can see that the proportion of CKDE nodes is quite high, and for about half of the datasets, every node type was CKDE; and therefore, according to Proposition 6.2, the final SPBN was equivalent to a KDEBN network. This justifies why there was no statistically significant difference between the SPBN and KDEBN models: most datasets were better represented by KDEBN models or the models with many CKDE CPDs. Noticeably, Waveform and Waveform-Noise datasets have the lowest proportion of CKDE nodes. This is explained by the fact that these datasets include variables with Gaussian noise, which the SPBN model was able to detect correctly.

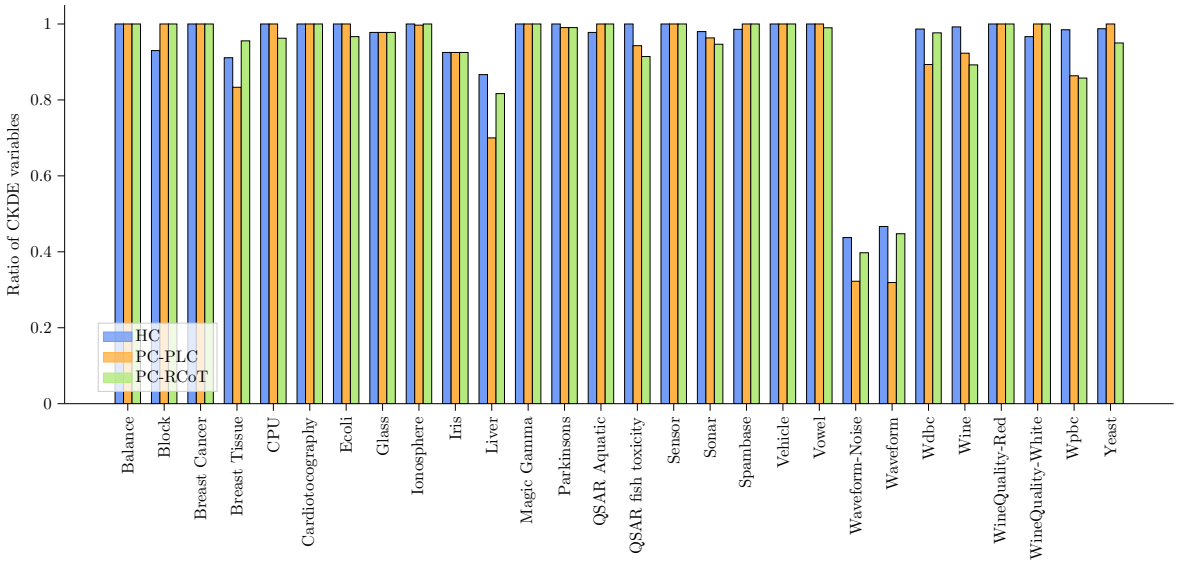


Figure 6.9: Ratio of CKDE nodes on different datasets learned with HC (blue), PC-PLC (orange) and PC-RCoT (green) algorithms.

6.3.4 Monitoring Bearing Degradation

In this section, we use SPBNs to monitor the degradation of rolling bearings. Rolling bearings are one of the most commonly used elements in industrial machines. Usually, these bearings suffer from degradation and can be a cause of machine breakdowns. For this reason, monitoring the state of bearing degradation can be a useful technique in machine maintenance. We will use the data provided by PRONOSTIA [Nectoux et al., 2012], which is an experimentation platform that degrades the bearings in a few hours. The data is captured with two accelerometers in the horizontal and vertical axes, to detect the bearing vibration. As is common in bearing diagnostics, we will analyze the data in the frequency domain, focusing on some frequencies of interest and their harmonics: ball pass frequency outer (BPFO) race, ball pass frequency inner (BPFI) race, fundamental train frequency (FTF) and ball spin frequency (BSF) [Randall and Antoni, 2011].

The PRONOSTIA dataset provides data with three different load conditions, but in this section, we will only use the first load condition. The training dataset contains two different bearings that were run to failure. To construct a model of the normal behaviour of a bearing, we segmented the data into three different state conditions: good state, average state and bad state. We detected these condition states using a hidden Markov model (HMM) assuming Gaussian emissions [Rabiner and Juang, 1986]. Typically, the good state is at the start of the data and exhibits low amplitudes for the frequencies studied. The average and bad states are usually located at the middle and the end of the data, and have average and high amplitudes respectively. Figure 6.10 shows the segmentation found for a training dataset into good, average and bad state. From this segmentation, we can learn three different SPBNs to model the good, average and bad state using the two learning training datasets.

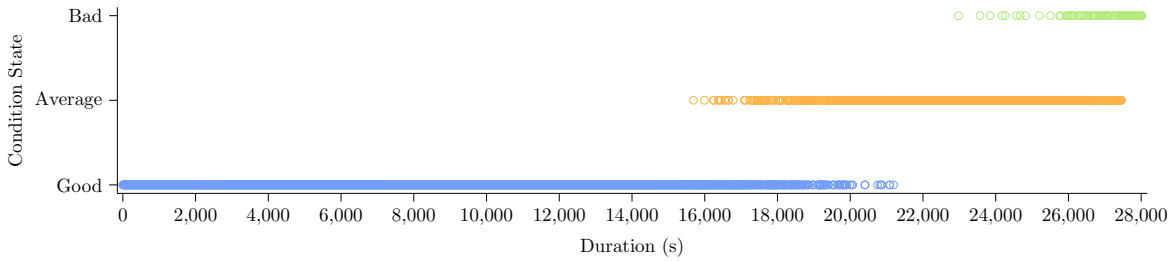


Figure 6.10: Segmentation of a bearing dataset (Bearing1_1) into good, average and bad state instances.

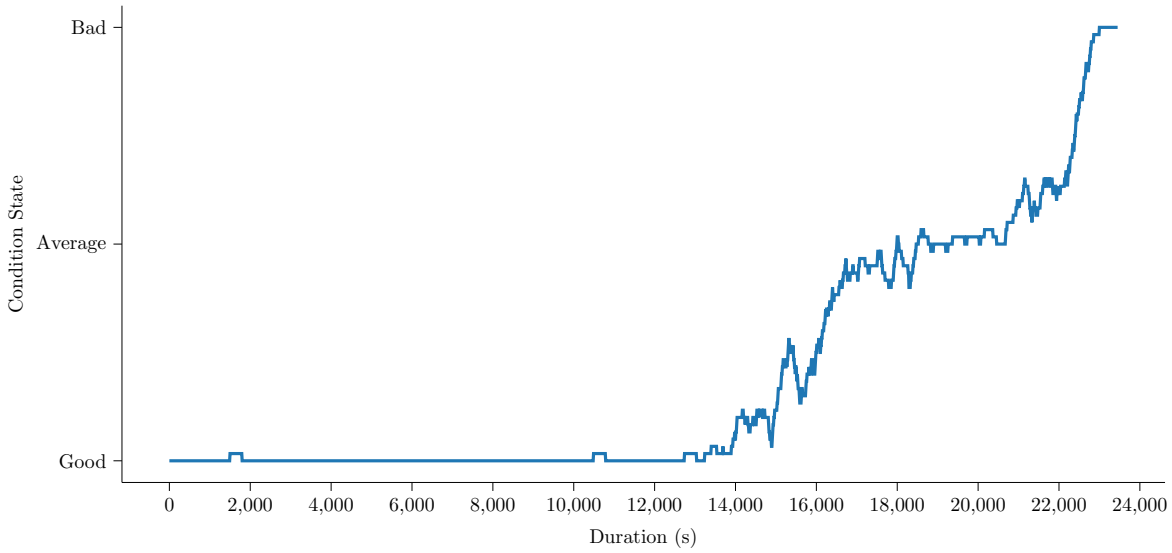


Figure 6.11: Estimated degradation process of a bearing dataset (Bearing1_3).

Learning three models of the bearing state can help us to track the degradation process of the bearing. For each instance of a test dataset, we can detect which SPBN model provides the larger log-likelihood. Figure 6.11 shows a gradual degradation process for a given bearing, by selecting the best model on each instance and applying a moving average to smooth the final result. Other bearings show different degradation patterns, e.g., abrupt degradation.

One of the main features of Bayesian networks is their interpretability because the contribution of each variable to the global log-likelihood can be analyzed. Figure 6.12 shows a decrease in the log-likelihood of a test bearing dataset according to the good state model. Also, we included the local log-likelihood contribution of each frequency and their harmonics. We can see that this decrease in the log-likelihood is mainly explained by abnormal BPFI amplitudes. This information suggests that a defect has ocurred in the inner race of the bearing.

This work has a clear temporal component. Therefore, an interesting alternative would be the use of dynamic Bayesian networks. We leave as a future research line the creation of

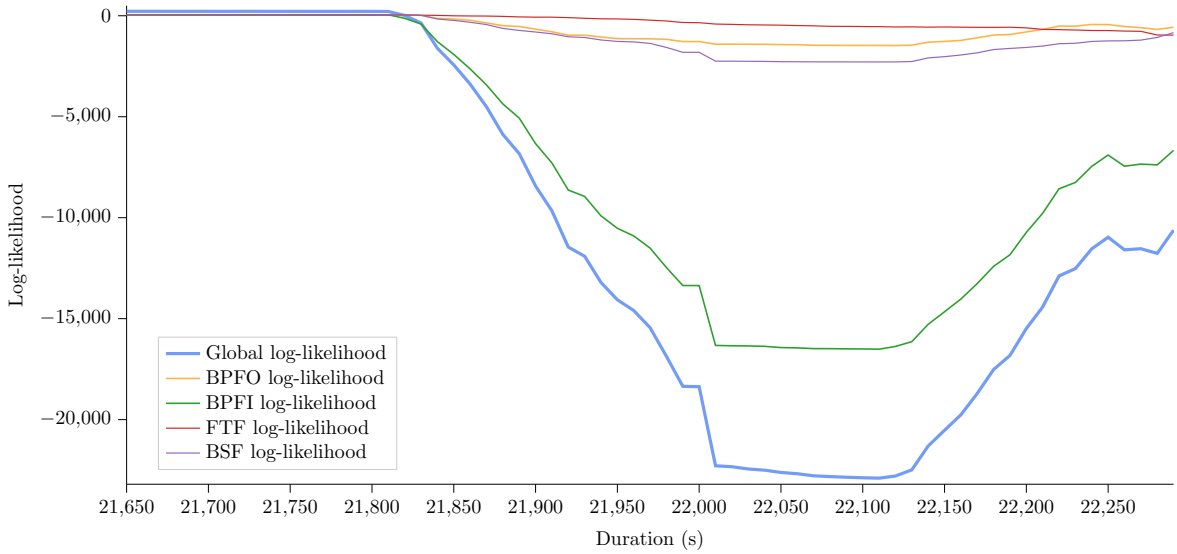


Figure 6.12: Global log-likelihood and local log-likelihood for the BPFO, BPFI, FTF and BSF frequencies and their harmonics of a bearing dataset (Bearing1_7) according to the good state model.

dynamic semiparametric Bayesian networks.

6.3.5 Execution Times

In this section, we show how the learning time of each learning procedure compares in practice. We created three synthetic models with different number of variables: a small model with five variables (the model in Figure 6.2), a medium-size model with 10 variables and a large model with 20 variables (not shown). We defined half of the nodes with a LG relationship and the other half using nonlinear relationships. For each model we sampled datasets with different number of instances: 200, 500, 2000, 4000 and 10000 instances. Then, we measured the execution time of all learning algorithms by repeating the learning process several times (with different sets of indices $\mathcal{I}$) and calculating the average value. For HC, we set $\lambda = 0$.

Figure 6.13 presents the execution time for all the learning algorithms in logarithm scale. The HC and PC algorithm times are shown with solid and dashed lines respectively. The PC-PLC Graph and PC-RCoT Graph show the time to learn just the graph of the model. This is the final model for the GBN and KDEBN models. To finish the learning process of a SPBN, an HC is executed that selects the best node types (PC-HC-NodeType). From these results we can see that the most performant methods are the HC with the BIC and BGe scores, and the PC-PLC algorithm. All of these learning procedures make parametric assumptions. Recall also that in BGe and the PLC test, some information can be cached at the start of the algorithm so the asymptotic time complexity only depends on the number of variables during the HC and PC iterations. Furthermore, we can check that the PC-HC-NodeType algorithm is usually much faster than the HC SPBN algorithm that also has to search for the best set of arcs. In all runs, HC SPBN-LG was faster than HC KDEBN. This is because the initial

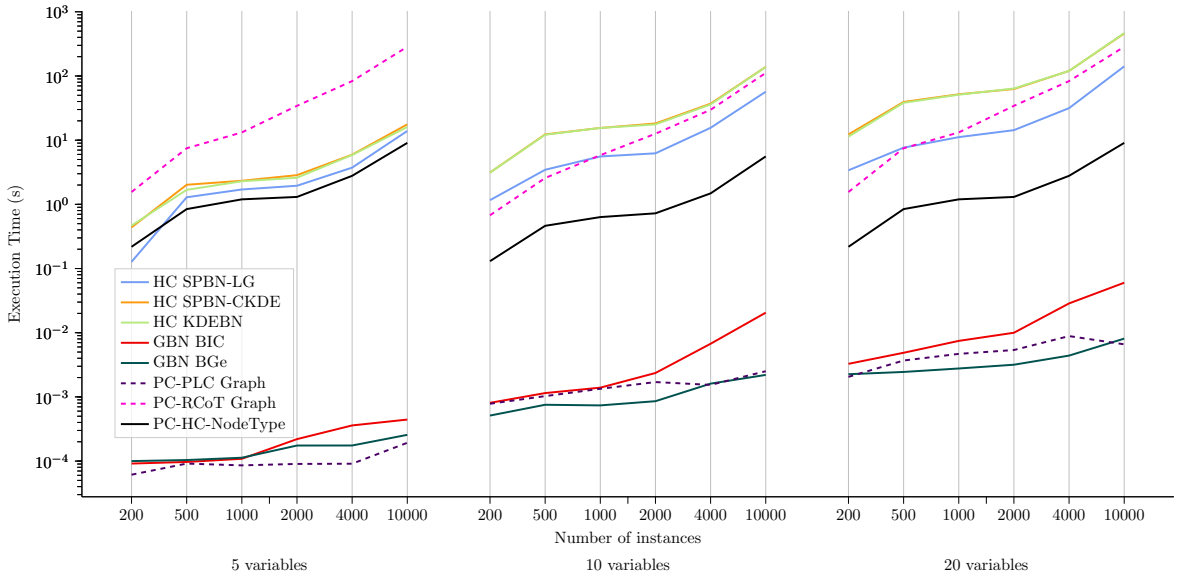


Figure 6.13: Execution times of all the learning procedures with different number of training instances and variables.

delta score cache for SPBN-LG takes less time. We discussed the difference in the asymptotic complexity of the cross-validated score (Equation (6.6)) between the LG and CKDE CPDs in Section 6.2.3. This can be easily verified by seeing that the HC SPBN-CKDE and HC KDEBN models take almost the same time in all runs.

6.4 Conclusion and Future Work

In the present chapter we introduced a new class of continuous Bayesian networks called semiparametric Bayesian networks that could be applied to model continuous data using both parametric and nonparametric estimation models. The class of SPBNs includes every possible GBN and every possible KDEBN. GBNs are fully parametric models, and KDEBNs correspond to fully nonparametric ones. In between these two extreme cases, an SPBN framework could be used to build a network in which some parts were parametric while other parts were nonparametric. Therefore, this approach allowed automatically adopting the advantage of parametric assumptions when appropriate, while also providing the flexibility of nonparametric models when necessary.

We proposed learning SPBNs based on the HC and PC algorithms. We note that other state-of-the-art learning algorithms could also be considered. Notably, as the proposal introduced in this chapter is general, other score and search algorithms can be implemented using the Type-Change operator (Section 6.2.2.2).

The results of the conducted experiments indicated that SPBNs could be implemented finding a suitable combination of parametric and nonparametric components. If the considered data followed the Gaussian distribution, the corresponding learned SPBN tends to be a

GBN model. However, if the data clearly did not belong to the Gaussian distribution, the proposed learning algorithm would produce more flexible SPBNs that combine the advantages of GBNs and KDEBNs.

There are multiple research lines that can be further investigated in the future. A better bandwidth selection for CKDE could improve the density estimation results. Discrete variables can be included to develop hybrid SPBNs (Chapter 7). In addition, introducing a tractable inference algorithm to perform queries in SPBNs would be of great interest. Moreover, temporal data could be better analyzed using dynamic semiparametric Bayesian networks (Chapter 10). To conclude, we also plan to train SPBN classifiers in the future.

Chapter 7

Hybrid Semiparametric Bayesian Networks

7.1 Introduction

Bayesian networks can be used to jointly model uncertain domains with discrete and continuous random variables. In the present chapter, we propose a new class of hybrid Bayesian networks, called hybrid semiparametric Bayesian networks, which extend the support of SPBNs (Chapter 6) to model categorical data as well. Thus, continuous variables can also have discrete variables as parents, while allowing the CPD of continuous variables to be modeled using parametric or nonparametric estimation models. In this way, this type of Bayesian networks can model hybrid probability distributions, i.e., probability distributions that combine discrete and continuous random variables, which are the most common data types in real applications. In addition, we include a learning algorithm that automatically detects which parts of the Bayesian network are best represented with a parametric or a nonparametric estimation model. This information can also be read in the graph, so some qualitative information can be extracted about the type of relation between the variables than in standard Bayesian networks.

The contributions of the chapter are as follows: (1) definition of a new class of Bayesian networks that models hybrid data and combines parametric and nonparametric estimation models; (2) adaptation of standard learning techniques to learn the structure and the parameters of hybrid semiparametric Bayesian networks; (3) a procedure to sample new data from this new class of Bayesian networks; and (4) an intuitive connection between hybrid semiparametric Bayesian networks and the AKDE models.

The chapter is organized as follows. In Section 7.2, the hybrid semiparametric Bayesian network class is described, along with a learning algorithm for the structure and parameters of the network. Section 7.3 provides experimental results obtained by testing hybrid semiparametric Bayesian networks in synthetic and real world data. Section 7.4 concludes the chapter and provides future work proposals.

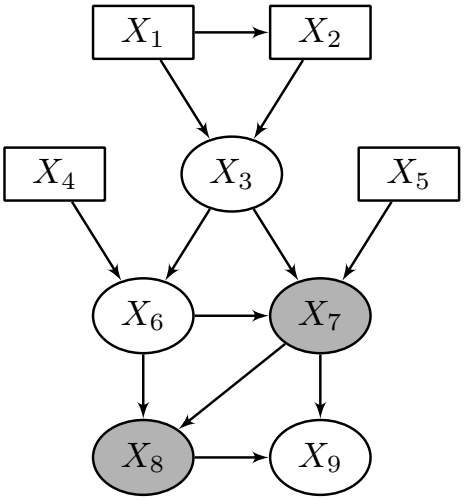


Figure 7.1: Example representing the structure of an HSPBN. Discrete variable nodes are represented with rectangles and continuous variable nodes with ellipses. Parametric CPD nodes are represented with white nodes and nonparametric CPDs nodes are represented with gray shaded nodes.

7.2 Hybrid Semiparametric Bayesian Networks

This section proposes the class of hybrid semiparametric Bayesian networks (HSPBNs). First, we describe their representation in Section 7.2.1, which describes the CPDs of HSPBNs. In Section 7.2.2, we propose a learning process of HSPBNs. Then, we present a procedure to sample new data from an HSPBN in Section 7.2.3. Finally, Section 7.2.4 discusses the relation between HSPBNs and the adaptive KDE models (Equation (3.57) and Equation (3.58)).

7.2.1 Representation

An HSPBN is a class of Bayesian network that can model discrete and continuous variables. The discrete variables are conditionally distributed as a categorical distribution. This distribution is usually represented by a CPT. Like CLGBNs, the graph is restricted so that a discrete variable cannot have a continuous variable as parent. The CPD of the continuous variables can be parametric or nonparametric, as in SPBNs. The parametric CPDs make the assumptions that, for each possible configuration of the discrete parents, the continuous variables are conditionally distributed as a normal distribution and have a linear relationship with their continuous parents. When this parametric assumption is not met, the HSPBN model can have more flexibility by using a nonparametric CPD, which does not make any parametric assumption.

The HSPBN model includes the CLGBNs as a special case when all the continuous variables CPDs are parametric. However, the HSPBNs have the possibility of more flexible models when nonparametric CPDs are used for some variables.

Figure 7.1 presents an example of HSPBN. Discrete nodes can only have other discrete nodes as parents. For the continuous nodes, any combination of parent node types is allowed.

7.2.1.1 Parametric Conditional Probability Distribution

The parametric CPDs in HSPBNs are CLG CPDs, which are also used by CLGBNs (Section 4.2.1). The CLG CPD (see Definition 4.9) is composed of an LG CPD (see Definition 4.8) for each discrete configuration of its evidence.

Note that if all the CPDs of a Bayesian network are LG, as in GBNs, the marginal and conditional distributions of each random variable in the Bayesian network are normal. Also, the joint distribution is multivariate normal.

If all the CPDs of a Bayesian network are CLGs, as in CLGBNs, the conditional distribution of each random variable given the evidence ($\mathbf{Y}$ and $\mathbf{Z}$) is normal, but its marginal distribution $f(x_i)$ may be non-normal. Indeed, the marginal distribution of each variable can be represented using a normal mixture where each component of the mixture is constructed from each discrete configuration of $\mathbf{Z}$. Thus, the CLG CPDs allow to relax the normality assumptions with respect the LG CPDs, so only the conditional distribution given the discrete evidence needs to be normal (which may not be the case for the marginal distribution).

Note that the LG CPD is a special case of the CLG CPD in which there is no discrete evidence $\mathbf{Z}$.

Proposition 7.1. An HSPBN in which all the nodes are CLG CPDs is equivalent to a CLGBN with the same arcs and parameter values.

Proof. The proof is straightforward, as, by definition, a CLGBN is a Bayesian network in which all continuous CPDs are CLG CPDs. □

Based on Proposition 7.1, we can easily deduce that every possible CLGBN is contained in the class of HSPBNs.

7.2.1.2 Nonparametric Conditional Probability Distribution

The nonparametric CPDs of an HSPBN are based on the CKDE CPD described in Definition 6.1. The CKDE is defined as the ratio of two joint distributions estimated with KDE models.

The CKDE CPD does not assume the marginal or conditional distribution of its variable. However, the CKDE CPD only supports continuous evidence. The hybrid conditional kernel density estimation (HCKDE) supports also discrete evidence by defining a CKDE for each discrete evidence configuration, inspired by the CLG CPD approach.

Definition 7.1. (HCKDE CPD). Let X_i be a random variable following a HCKDE conditional distribution, $\mathbf{Y} = \mathbf{X}_{\mathrm{Pa}_{\mathbf{Y}}(i)}$ and $\mathbf{Z} = \mathbf{X}_{\mathrm{Pa}_{\mathbf{Z}}(i)}$; then, the conditional distribution of X_i given $\mathbf{X}_{\mathrm{Pa}(i)}$ (with discrete and continuous variables) is defined as:

$$\hat{f}_{\mathrm{HCKDE}}(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)}) = \hat{f}_{\mathrm{CKDE},\mathbf{z}}(x_i \mid \mathbf{y}) = \frac{\sum_{j=1:\mathbf{z}^j=\mathbf{z}}^{N} K_{\mathbf{H}(\mathbf{z})}\left(\begin{bmatrix} x_i \\ \mathbf{y} \end{bmatrix} - \begin{bmatrix} x_i^j \\ \mathbf{y}^j \end{bmatrix}\right)}{\sum_{j=1:\mathbf{z}^j=\mathbf{z}}^{N} K_{\mathbf{H}_{-i}(\mathbf{z})}\left(\mathbf{y} - \mathbf{y}^j\right)}, \tag{7.1}$$

where $\hat{f}_{\text{CKDE},\mathbf{z}}$ is a CKDE constructed with the instances with the discrete evidence $\mathbf{z}$, $\mathbf{H}(\mathbf{z})$ and $\mathbf{H}_{-i}(\mathbf{z})$ are bandwidth matrices that depend on the discrete evidence configuration $\mathbf{z}$.

An HCKDE CPD does not require assumptions about the marginal or conditional distribution of X_i. Note that this is a difference with respect to CLG, which assumes a conditional Gaussian distribution given its parents. Note that the CKDE CPD is a special case of the HCKDE CPD in which there is no discrete evidence $\mathbf{Z}$.

7.2.2 Learning

A Bayesian network can be constructed by taking advantage of knowledge from experts of the domain or automatically from data. In this work, we focus our attention on learning automatically from data because usually there are large amounts of data that can be obtained cheaply.

There are two parts on the learning process of a Bayesian network: parameter learning and structure learning. The parameter learning estimates the parameters of the set of CPDs, $\boldsymbol{\theta}$, for a given structure. The structure learning estimates the graph, $\mathcal{G}$, of the Bayesian network. For most Bayesian networks, the structure learning involves learning the arcs in the graph. In addition, HSPBNs contain extra information specifying the type of CPD for each continuous node.

Usually, the learning process involves first performing structure learning to find the best Bayesian network structure. Then, parameter learning is applied on the best found structure.

7.2.2.1 Parameter Learning

The parameter learning requires knowing the type of CPD and the parents of each node to estimate the parameters of each CPD, $P(x_i \mid \mathbf{x}_{\text{Pa}(i)})$. In this section, we describe the parameter learning process for each CPD type.

For the categorical CPD, the MLE may be inconvenient when $N[x_i, \mathbf{x}_{\text{Pa}(i)}] = 0$ if the ground truth probability $P(x_i \mid \mathbf{x}_{\text{Pa}(i)})$ is not 0. In that case, the probability of an instance with $X_i = x_i$ and $\mathbf{X}_{\text{Pa}(i)} = \mathbf{x}_{\text{Pa}(i)}$ is equal to 0 and the log-likelihood is not defined. Moreover, the MLE is not defined when $N[\mathbf{x}_{\text{Pa}(i)}] = 0$. This problem can be addressed using a Bayesian prior for the categorical CPD. In this chapter, we use an uniform Dirichlet prior (Equation (4.14)), where $\alpha > 0$ is the equivalent sample size of the prior. In the limiting case $\alpha = 0$ the Dirichlet prior is not defined, but when α is close to 0 the estimate is close to the MLE. In this chapter, we use $\alpha = 1$.

The CLG CPDs parameters are estimated with the MLE as described in Section 4.3.3. Therefore, the parameters of a CLG can be found applying ordinary least squares for each discrete configuration $\mathbf{z} \in \boldsymbol{\Omega}_{\text{Pa}_\mathbf{Z}(i)}$ on the subset of data $\mathcal{D}_{\downarrow \mathbf{z}}$.

The HCKDE CPD for X_i given $\mathbf{X}_{\text{Pa}(i)}$ needs to estimate a CKDE model, $\hat{f}_{\text{CKDE},\mathbf{z}}$, for each discrete parent configuration $\mathbf{z} \in \boldsymbol{\Omega}_{\text{Pa}_\mathbf{Z}(i)}$. The parameter learning process of a CKDE is described in Section 6.2.2.1. As in Chapter 6, we will use a multivariate Gaussian kernel.

Algorithm 7.1 summarizes the learning process of an HCKDE CPD. For every discrete configuration of the parent variables $\mathbf{z}$, a new CKDE must be learned, which also requires learning two KDE models. The HCKDE CPD is the combination of all the trained CKDE CPDs.

Algorithm 7.1 HCKDE parameter learning

Require: Training data $\mathcal{D}$, variable X_i, evidence variables $\mathbf{X}_{\mathrm{Pa}(i)}$

1: **for** $\mathbf{z}$ in $\mathbf{\Omega}_{\mathrm{Pa}_{\mathbf{Z}}(i)}$ **do**
2: Estimate bandwidth $\hat{\mathbf{H}}$ from $\mathcal{D}_{\{i\}\cup\mathrm{Pa}_{\mathbf{Y}}(i),\downarrow\mathbf{z}}$
3: $\hat{\mathbf{H}}_{-i} \leftarrow \hat{\mathbf{C}}$
4: $\hat{f}_{\mathrm{KDE},\mathbf{z}}\left(x_i, \mathbf{x}_{\mathrm{Pa}_{\mathbf{Y}}(i)}\right) \leftarrow$ KDE constructed with $\hat{\mathbf{H}}$ and $\mathcal{D}_{\{i\}\cup\mathrm{Pa}_{\mathbf{Y}}(i),\downarrow\mathbf{z}}$
5: $\hat{f}_{\mathrm{KDE},\mathbf{z}}\left(\mathbf{x}_{\mathrm{Pa}_{\mathbf{Y}}(i)}\right) \leftarrow$ KDE constructed with $\hat{\mathbf{H}}_{-i}$ and $\mathcal{D}_{\mathrm{Pa}_{\mathbf{Y}}(i),\downarrow\mathbf{z}}$
6: $\hat{f}_{\mathrm{CKDE},\mathbf{z}}\left(x_i \mid \mathbf{x}_{\mathrm{Pa}_{\mathbf{Y}}(i)}\right) \leftarrow$ CKDE constructed with $\hat{f}_{\mathrm{KDE},\mathbf{z}}\left(x_i, \mathbf{x}_{\mathrm{Pa}_{\mathbf{Y}}(i)}\right)$ and $\hat{f}_{\mathrm{KDE},\mathbf{z}}\left(\mathbf{x}_{\mathrm{Pa}_{\mathbf{Y}}(i)}\right)$
7: **end for**
8: **return** $\hat{f}_{\mathrm{HCKDE}}\left(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)}\right)$ created with all the $\hat{f}_{\mathrm{CKDE},\mathbf{z}}\left(x_i \mid \mathbf{x}_{\mathrm{Pa}_{\mathbf{Y}}(i)}\right)$

7.2.2.2 Structure Learning

The structure learning of an HSPBN is based on the learning process described in Section 6.2.2.2. For SPBNs learning, a Type-Change(i) operator is included in the learning process. The Type-Change operator can change the type of a single node in the graph. For the HSPBN framework, the CLG CPDs can be change to HCKDE CPDs, while the HCKDE CPDs are transformed into CLG CPDs.

Note that the cross-validated score defined in Equation (6.6) is valid for HSPBNs without applying any change. Similarly as for SPBNs, the likelihood function is computed by Equation (4.7), and the contribution of each node depends on its type, following Definition 4.9 for CLG nodes and Definition 7.1 for HCKDE nodes.

In this chapter, the HC algorithm described in Algorithm 6.1 is used to learn the structure of the HSPBN. A small modification is included to forbid the operators (add arc or reverse arcs) that produces arcs from a continuous variable to a discrete variable.

7.2.3 Sampling from Nonparametric Conditional Probability Distributions

Bayesian networks are known to be generative models, which model a probability distribution $P(\mathbf{x})$. One of the advantages of a generative model is the ability to sample new data from the model. In the case of Bayesian networks, there are also some inference algorithms that are based on sampling, such as likelihood weighting and Markov chain Monte Carlo techniques [Darwiche, 2009]. In this section, we detail how new data can be sampled from the CKDE and HCKDE CPDs.

Let X_i be an HCKDE node, and $\hat{f}_{\mathrm{HCKDE}}(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)})$ be its conditional distribution given its parents. By definition, Equation (7.1), its conditional distribution is equal to

$\hat{f}_{\text{CKDE},\mathbf{x}_{\text{Pa}_{\mathbf{Z}}(i)}}(x_i \mid \mathbf{x}_{\text{Pa}_{\mathbf{Y}}(i)})$. Thus, sampling from an HCKDE requires sampling from the corresponding CKDE CPD.

Assuming K is the normal multivariate kernel, the CKDE CPD can be expressed in the following way:

$$\begin{aligned}\hat{f}_{\text{CKDE}}(x_i \mid \mathbf{y}) &= \frac{\sum_{j=1}^{N} \mathcal{N}\left(\begin{bmatrix} x_i \\ \mathbf{y} \end{bmatrix}; \begin{bmatrix} x_i^j \\ \mathbf{y}^j \end{bmatrix}, \mathbf{H}\right)}{\sum_{j=1}^{N} \mathcal{N}\left(\mathbf{y}; \mathbf{y}^j, \mathbf{H}_{-i}\right)} \\ &= \frac{\sum_{j=1}^{N} \mathcal{N}\left(\mathbf{y}; \mathbf{y}^j; \mathbf{H}_{-i}\right) \mathcal{N}(x_i; \mu_i^{\text{cond, j}}, h_i^{\text{cond}})}{\sum_{j=1}^{N} \mathcal{N}\left(\mathbf{y}; \mathbf{y}^j, \mathbf{H}_{-i}\right)} \\ &= \sum_{j=1}^{N} w_j \mathcal{N}\left(x_i; \mu_i^{\text{cond, j}}, h_i^{\text{cond}}\right),\end{aligned} \tag{7.2}$$

with

$$w_j = \frac{\mathcal{N}\left(\mathbf{y}; \mathbf{y}^j; \mathbf{H}_{-i}\right)}{\sum_{k=1}^{N} \mathcal{N}\left(\mathbf{y}; \mathbf{y}^k, \mathbf{H}_{-i}\right)}, \tag{7.3}$$

where $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the normal PDF with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, evaluated at $\mathbf{x}$. The second equality holds because the joint distribution $f(x_i, \mathbf{y})$ is expressed as $f(x_i \mid \mathbf{y})f(\mathbf{y})$. For the normal distribution, the marginal, $f(\mathbf{y})$, and the conditional, $f(x_i \mid \mathbf{y})$, distributions are easy to find. The parameters $\mu_i^{\text{cond, j}}$ and $h_i^{\text{cond, j}}$ are:

$$\begin{aligned}\mu_i^{\text{cond, j}} &= x_i^j + \mathbf{b}^T \mathbf{C}^{-1}(\mathbf{y} - \mathbf{y}^j) \\ h_i^{\text{cond}} &= a - \mathbf{b}^T \mathbf{C}^{-1} \mathbf{b},\end{aligned} \tag{7.4}$$

where a, $\mathbf{b}$ and $\mathbf{C}$ are the bandwidth matrix blocks defined in Equation (6.3).

Sampling from a CKDE is easier from the expression in Equation (7.2). Algorithm 7.2 details the sampling procedure from a CKDE. First, the weights $\mathbf{w} = (w_1, \ldots, w_N)$ are calculated (line 1). Then, a sample is generated from a categorical distribution with the weights $\mathbf{w}$ as parameters (line 2). Finally, the $\mu_i^{\text{cond, j}}$ and h_i^{cond} parameters are calculated (line 3), and a new sample it is obtained sampling a normal with mean $\mu_i^{\text{cond, j}}$ and variance h_i^{cond} (line 4).

7.2.4 Relation with Adaptive KDE

In the HCKDE CPD (Equation (7.1)) the bandwidth matrix depends on the discrete configuration $\mathbf{z}$. This bandwidth matrix expression is similar to the bandwidth matrices used in adaptive KDEs (Equation (3.57) and Equation (3.58)). Furthermore, we can consider an

Algorithm 7.2 Sampling new data from a CKDE CPD

Require: CKDE CPD $\hat{f}(x_i \mid \mathbf{y})$, evidence instantation $\mathbf{y}$
1: $\mathbf{w} \leftarrow$ Compute vector of w_j (Equation (7.3))
2: $j \sim$ Categorical($\mathbf{w}$) // Sample from a categorical distribution with parameters $\mathbf{w}$
3: $\mu_i^{\text{cond, j}}, h_i^{\text{cond}} \leftarrow$ Compute conditional mean and variance (Equation (7.4))
4: $x \sim \mathcal{N}(\mu_i^{\text{cond, j}}, h_i^{\text{cond}})$ // Sample normal with mean $\mu_i^{\text{cond, j}}$ and variance h_i^{cond}
5: **return** x

HCKDE CPD to be a type of adaptive KDE. The main diference with the adaptive KDEs defined in the state of the art is that in an HCKDE the bandwidth matrix depends on discrete variables instead of continuous variables. In addition, since the HCKDE CPD is part of a Bayesian network, it can automatically learn which discrete variables are most useful for splitting the data and apply a different amount of smoothing at each split. When the Bayesian network does not select any discrete variable as a parent of the HCKDE, the resulting CPD is a CKDE, which is not adaptive. Thus, the HSPBN learning algorithm automatically chooses between using standard KDEs or adaptive KDEs.

7.3 Experiments

In this section, we experimentally check the capabilities of HSPBNs to obtain a good fit to the data. We will perform two types of experiments: experiments with synthetic data, where we know the data distribution and the expected HSPBN model, and real world data from the UCI repository, where the underlying model that generated the data is unknown.

The experiments were conducted using our `PyBNesian` package (Chapter 8), and the source code is available at https://github.com/davenza/HSPBN-Experiments.git.

7.3.1 Synthetic Data

In this section we compare the results of CLGBN models and HSPBN models learned with the procedure described in Algorithm 6.1. To test the ability of the HSPBNs to capture conditional linear relationships (with a CLG CPD) and also conditional nonlinear relationships

(with an HCKDE CPD), we will generate the data from the following model:

$$
\begin{aligned}
P(a) &= \text{Categorical}(0.75, 0.25) \\
P(b \mid a) &= \begin{cases} \text{Categorical}(0.33, 0.33, 0.34), \text{ if } a = a_1 \\ \text{Categorical}(0, 0.8, 0.2), \text{ if } a = a_2 \end{cases} \\
P(c) &= \text{Categorical}(0.25, 0.25, 0.25, 0.25) \\
f(d|a,b) &= \begin{cases} \mathcal{N}_D\left(\mu_D = -3, \sigma_D^2 = 1\right), \text{ if } a = a_1 \text{ and } b = b_1 \\ \mathcal{N}_D\left(\mu_D = -1.5, \sigma_D^2 = 0.7\right), \text{ if } a = a_1 \text{ and } b = b_2 \\ \mathcal{N}_D\left(\mu_D = 0, \sigma_D^2 = 0.5\right), \text{ if } a = a_1 \text{ and } b = b_3 \\ \mathcal{N}_D\left(\mu_D = 1, \sigma_D^2 = 1.25\right), \text{ if } a = a_2 \text{ and } b = b_1 \\ \mathcal{N}_D\left(\mu_D = 2, \sigma_D^2 = 1.5\right), \text{ if } a = a_2 \text{ and } b = b_2 \\ \mathcal{N}_D\left(\mu_D = 3, \sigma_D^2 = 2\right), \text{ if } a = a_2 \text{ and } b = b_3 \end{cases} . \\
P(e) &= \text{Categorical}(0.5, 0.5) \\
f(g \mid c,d) &= \begin{cases} \mathcal{N}_G\left(\mu_G = 0, \sigma_G^2 = 1\right), \text{ if } c = c_1 \\ -3 + 2.5 \cdot d + \epsilon_G, \text{ where } \epsilon_G \sim \mathcal{N}_G\left(\mu_G = 0, \sigma_G^2 = 0.5\right), \text{ if } c = c_2 \\ 2 - 1.25 \cdot d + \epsilon_G, \text{ where } \epsilon_G \sim \mathcal{N}_G\left(\mu_G = 0, \sigma_G^2 = 2\right), \text{ if } c = c_3 \\ 5 \cdot d + \epsilon_G, \text{ where } \epsilon_G \sim \mathcal{N}_G\left(\mu_G = 0, \sigma_G^2 = 0.25\right), \text{ if } c = c_4 \end{cases} \\
f(h|d,e) &= \begin{cases} 0.5 \cdot \mathcal{N}(\mu_{H_1} = -d, \sigma_{H_1}^2 = 1) + 0.5 \cdot \mathcal{N}(\mu_{H_2} = d, \sigma_{H_2}^2 = 1), \text{ if } e = e_1 \\ 0.3 \cdot \mathcal{N}(\mu_{H_2} = -2 + 0.5 \cdot d, \sigma_{H_2}^2 = 0.5) + 0.4 \cdot \mathcal{N}(\mu_{H_2} = 0, \sigma_{H_2}^2 = 1) + \\ \quad 0.3 \cdot \mathcal{N}(\mu_{H_3} = 5 - 1.5 \cdot d, \sigma_{H_3}^2 = 2), \text{ if } e = e_2 \end{cases} \\
f(i|g,h) &= g \cdot h + \epsilon_I, \text{ where } \epsilon_I \sim \mathcal{N}_I\left(\mu_I = 0, \sigma_I^2 = 0.5\right)
\end{aligned}
\tag{7.5}
$$

Variables D and G have a linear relationship with their evidence variables, while the relationship of variables H and I with their parents is nonlinear. The expected HSPBN model that represents the set of conditional independences and the linearity/nonlinearity of the relationships is shown in Figure 7.2.

To compare the performance of the models, we sampled datasets with different number of instances: 200, 2,000 and 10,000. For each dataset, we learned the following models: CLGBNs with the BIC score, CLGBNs with the cross-validated score in Equation (6.6) using $\lambda = 0$ and $\lambda = 5$, HSPBNs where the initial node type for all nodes is parametric (HSPBN-CLG), i.e., the CPDs for the continuous nodes of the starting model are CLGs, and HSPBNs where the initial node type for all nodes is nonparametric (HSPBN-HCKDE), i.e., the CPDs for the continuous nodes of the starting model are HCKDEs. In all cases, the starting graph ($\mathcal{G}_0$ in Algorithm 6.1) had no arcs. During the structure learning, the scores in Equation (6.6) and Equation (6.7) of the HCKDE CPDs are calculated estimating the bandwidth matrix using the normal reference rule because is the fastest criterion available. Then, when the final

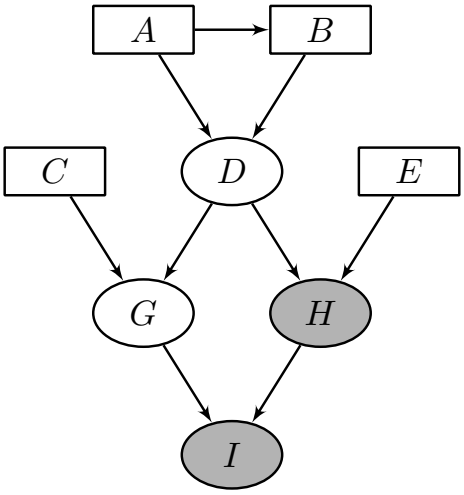


Figure 7.2: Structure of the ground truth HSPBN model. Discrete variable nodes are represented with rectangles and continuous variable nodes with ellipses. Parametric CPD nodes are represented with white nodes and nonparametric CPDs nodes are represented with gray shaded nodes.

structure is learned, we tried the normal reference rule, the UCV criterion and the plug-in estimation method to estimate the bandwidth matrices (Section 3.3.3).

To evaluate the performance of the models, we sampled another test dataset of 1,000 instances that was not previously seen by the learned models, i.e., a new dataset different to $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{val}}$. This test dataset can estimate the expected performance of the models on the new and unseen data by calculating the log-likelihood of the test dataset. In addition, we also compare the structural accuracy of the graphs (how good the learned graph compares to the graph in Figure 7.2) using the HMD, SHD and THMD criteria described in Section 6.3. For all the structural accuracy criteria, the lower the value, the better. To improve the reliability of the results, we repeated this process 50 times. Thus, we sampled 50 training datasets for each number of instances and 50 test datasets.

The results of this comparison are shown in Table 7.1. The log-likelihood, HMD, SHD and THMD values are the mean value of the 50 repetitions of the experiments. Table 7.1 shows the results when the bandwidth matrices are estimated using the normal reference rule. The table shows that HSPBNs have better log-likelihood results as well as structural accuracy than CLGBNs, even though there are only two nonparametric nodes in the ground truth model. As expected, the log-likelihood improves with the increase of the training dataset instances. This is most noticeable in the case of the HSPBNs. For small datasets, the difference between CLGBNs and HSPBNs is small, but for larger datasets, the difference is so significant that an HSPBN trained with 2,000 instances can be as good or even better than a CLGBN trained with 10,000 instances. Also, the HSPBNs usually return better structural accuracy than the CLGBNs. In the case of the THMD, we can see that the accuracy improves significantly when the number of instances increases. For the HSPBNs trained with 10,000 instances, the algorithm found the best node type for each node in all the simulations.

Model		Log-likelihood	HMD	SHD	THMD
Max possible value			32	32	4
Ground truth		−9299.56	0	0	0
	CLGBN BIC	−14469.37	4.88	5.22	-
	CLGBN $\lambda = 0$	−13368.14	5.06	5.36	-
	CLGBN $\lambda = 5$	−13270.05	4.58	5.16	-
200 instances	HSPBN-CLG $\lambda = 0$	−13132.65	3.94	4.40	1.04
	HSPBN-CLG $\lambda = 5$	**−12902.56**	3.04	3.70	**0.54**
	HSPBN-HCKDE $\lambda = 0$	−13316.62	4.06	4.58	1.48
	HSPBN-HCKDE $\lambda = 5$	−12960.27	**2.80**	**3.48**	0.98
	CLGBN BIC	−12549.98	9.50	10.04	-
	CLGBN $\lambda = 0$	−12019.92	7.16	7.90	-
	CLGBN $\lambda = 5$	−11999.32	7.30	8.14	-
2000 instances	HSPBN-CLG $\lambda = 0$	−12070.95	6.32	6.70	0.32
	HSPBN-CLG $\lambda = 5$	−11847.31	6.68	7.46	**0.26**
	HSPBN-HCKDE $\lambda = 0$	−11291.73	4.22	4.94	0.82
	HSPBN-HCKDE $\lambda = 5$	**−11192.60**	**4.10**	**4.90**	0.52
	CLGBN BIC	−11816.46	6.86	8.24	-
	CLGBN $\lambda = 0$	−11810.82	6.44	7.00	-
	CLGBN $\lambda = 5$	−11810.04	6.36	6.92	-
10000 instances	HSPBN-CLG $\lambda = 0$	**−10714.92**	**3.18**	**3.68**	**0.00**
	HSPBN-CLG $\lambda = 5$	**−10714.92**	**3.18**	**3.68**	**0.00**
	HSPBN-HCKDE $\lambda = 0$	−10718.43	5.88	7.24	**0.00**
	HSPBN-HCKDE $\lambda = 5$	−10717.15	5.02	5.98	**0.00**

Table 7.1: Results of learning from synthetic data sampled from the model in Equation (7.5). HMD stands for the Hamming distance, SHD denotes the structural Hamming distance, and THMD corresponds to the node type Hamming distance computed between the learned model and the ground truth model.

Dataset	N	n	c	d
Abalone	4177	9	8	1
Adult	45222	14	5	9
Australian Statlog	690	15	6	9
Cover Type	11340	55	10	45
Credit Approval	653	16	6	10
German Statlog	1000	21	7	14
KDD Cup 1999	10000	29	23	6
Liver Disorders	341	7	6	1
Thyroid-Hypothyroid	2000	25	6	19
Thyroid-Sick	1947	30	7	23

Table 7.2: Datasets from the UCI repository.

We also tested the UCV and the plug-in methods to estimate the bandwidth matrices. We found experimentally that these two bandwidth estimation methods are especially sensitive to the existence of outliers in the data. Thus, this caused bad results for the small datasets (especially for 200 instances) because some instances that are sampled with low probabilities can be seen as outliers. Although we do not include these results in the manuscript, we highlight that the plug-in estimation returned better results than UCV. In addition, for the large dataset of 10,000 instances, the plug-in estimation returned a log-likelihood of -10454.28, -10454.28, -10457.81 and -10456.53 for the HSPBN-CLG with $\lambda = 0$, HSPBN-CLG with $\lambda = 5$, the HSPBN-HCKDE with $\lambda = 0$ and the HSPBN-HCKDE with $\lambda = 5$, respectively. These results are even better than the results obtained by the normal reference rule in Table 7.1. With these results, we can conclude that the UCV and plug-in bandwidth estimation methods are only useful if there is a large amount of data.

7.3.2 UCI Repository Data

In this section we test the ability of the HSPBNs to fit to real world data analyzing datasets from the UCI repository [Dua and Graff, 2017]. The characteristics of all the UCI datasets are shown in Table 7.2. For the KDD Cup 1999 we selected a subset of the data because it is a very large dataset (almost 5 million instances) and the experiments would have been too time consuming. Thus, we selected the first 10,000 instances of the dataset. For all the datasets, we removed the constant columns (as they are not interesting to be analyzed with a Bayesian network). The number of columns indicated in Table 7.2 are the result obtained after this preprocessing step.

For the UCI datasets, we do not know the data distribution or the model that generated the data, as this datasets are usually extracted from the real world. For this reason, we cannot estimate the log-likelihood using a new test dataset and we cannot calculate the structural accuracy. To estimate the goodness of our model, we perform a 10-fold cross-validation. On each fold, we learn a model on the training subset and we estimate the goodness of the model calculating the log-likelihood of the test subset on the learned model. We tested the same

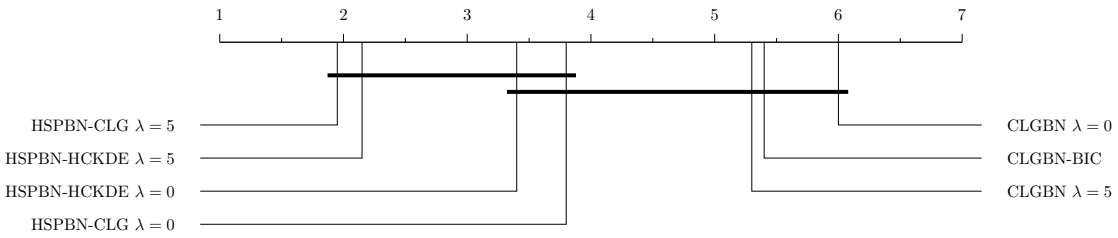


Figure 7.3: Critical difference diagram for the mean rank of each algorithm in the UCI datasets.

configurations of models as in the synthetic experiments. In this case, we found that the UCV and the plug-in bandwidth estimation methods are again sensitive to outliers. For this reason, in some datasets they returned very poor results, while in other datasets they offered competitive results. Therefore, we omit the presentation of results with these bandwidth selection techniques and we focus our attention on the results using the normal reference rule.

We calculated the mean log-likelihood of the test subset in the 10-fold cross-validation. Then, to find statistically significant differences between all the algorithms, we performed a Friedman test with $\alpha = 0.05$ and a Bergman-Hommel post-hoc procedure to detect the pairwise significant differences [García and Herrera, 2008]. The results are shown in Figure 7.3 using a critical difference diagram [Demšar, 2006]. The critical difference diagram shows the mean rank of each model over all the datasets. The black horizontal bars indicate the groups of models whose mean rank difference is not statistically significant. Therefore, we can conclude that there is no statistically significant difference between the HSPBNs trained with $\lambda = 5$ and the HSPBNs trained with $\lambda = 0$. Also, there is no statistically significant difference between the HSPBNs trained with $\lambda = 0$ and the CLGBNs. However, the statistical test shows that there is a statistically significant mean rank difference between the HSPBNs trained with $\lambda = 5$ and the CLGBN models. For all the models trained with a validated score (Equation (6.6) and Equation (6.7)), the obtained mean rank is better for $\lambda = 5$ than for $\lambda = 0$. This is reasonable because the algorithm takes more time to optimize the model trying to escape from local optima. These results demonstrate the usefulness and applicability of the HSPBN models to real world data. This result is not surprising because real world data may contain non-Gaussian and nonlinear relationships between the variables. The HSPBN models can adapt better to this kind of relationships, obtaining a better fit to the data.

7.4 Conclusion and Future Work

This chapter presented a new class of hybrid Bayesian networks called hybrid semiparametric Bayesian networks as an extension of the SPBN model (Chapter 6). To the best of our knowledge this is the first type of semiparametric Bayesian network with the ability to model

hybrid data containing both discrete and continuous data. In addition, for the continuous data, the type of relationships between variables are explicitly represented in the graph. If there is a CLG relationship, it can be defined using a parametric model (a CLG CPD). If the conditional distribution is non-Gaussian or there are nonlinear relationships, usually this can be better represented using a nonparametric model (an HCKDE CPD).

We described a learning procedure that estimates the parameters and the structure of HSPBN. This procedure is based on other standard techniques in the state of the art. Following this work, we would like to adapt other standard types of algorithms to learn HSPBNs in the future, such as constraint-based learning (e.g., the PC algorithm [Spirtes et al., 2000]). In addition, we introduced a procedure to sample new data from an HSPBN and showed its relation with the adaptive KDE models.

The experimental work showed that the HSPBN model can improve the results of the CLGBN models. This is not surprising because the class of CLGBNs is included in the class of HSPBNs.

In the future, we would like to work on semiparametric Bayesian network classifiers. Also, other types of CPDs can be incorporated for the discrete variables, so the arcs between continuous variables and discrete variables are allowed. Finally, other types of nonparametric models can be investigated to create HSPBNs. An interesting instance is the ratio density estimation [Sugiyama et al., 2012], which may improve the results even further.

Chapter 8

PyBNesian: a Python package for Bayesian networks

8.1 Introduction

In this chapter, we introduce our Python package called `PyBNesian`, which implements Bayesian networks, including the semiparametric Bayesian networks (Chapter 6) and hybrid semiparametric Bayesian networks (Chapter 7). In addition, the package provides an implementation for many different types of Bayesian networks in the state of the art, so discrete, continuous and hybrid data are supported.

Moreover, the package is designed such that it can be fully extensible, so new types of Bayesian networks, CPDs, learning scores, learning operators, or conditional independence tests can be easily implemented. These new components can be effortlessly integrated with the standard implementation of many algorithms such as HC and PC provided by the package. This capability can speed up the research on Bayesian networks, minimizing the amount of code necessary to test new ideas. This also facilitates the sharing of the source code and its audit by peer reviewers, which often improves the quality of the research [Shamir et al., 2013; Eglen et al., 2017].

The package includes implementations for other special categories, such as conditional Bayesian networks and dynamic Bayesian networks. Also, it provides a tailored implementation of different types of graphs (directed, undirected or partially directed) to simplify the implementation of many algorithms related to Bayesian networks, e.g., finding the equivalence class of a DAG, returning a DAG from an equivalence class, topological sorting, path finding or fast access to the roots and leaves of a graph, etc.

Special care was also taken to obtain the best possible performance. For KDEs, an implementation with OpenCL is included, so GPU acceleration can be obtained as it is common in other machine learning libraries nowadays [Abadi et al., 2015; Paszke et al., 2019].

Finally, the package provides easy serialization (writing and reading) of all the relevant

types of objects: Bayesian networks, CPDs, graphs, etc.

The chapter is organized as follows. Section 8.2 details all the functionalities implemented in the package. It includes implementations of conditional Bayesian networks and dynamic Bayesian networks. Section 8.3 provides some details about the implementation. Section 8.4 reviews some other packages implementations in the state-of-the-art. Section 8.5 compares the execution times of `PyBNesian` with other implementations in the state of the art. Finally, Section 8.6 concludes the chapter and provides some interesting future work.

8.2 Functionalities

In this section, we describe the list of functionalities implemented in `PyBNesian`.

8.2.1 Bayesian Network Categories

`PyBNesian` implements different categories of Bayesian networks. Three different categories of Bayesian networks are implemented:

- Bayesian networks
- Conditional Bayesian networks
- Dynamic Bayesian networks

Bayesian networks are described in Chapter 4. Conditional and dynamic Bayesian networks are defined below.

Definition 8.1. (Conditional Bayesian network). A conditional Bayesian network is a tuple $\mathcal{B} = (\mathcal{G}, \boldsymbol{\theta})$, where $\mathcal{G} = (V, A)$ is a DAG with a set of nodes $V = V_N \cup V_I = \{1, \dots, n\}$ with V_N and V_I disjoint, and a set of arcs $A = A_N \cup A_I$, with $A_N \subseteq V_N \times V_N$ and $A_I \subseteq V_I \times V_N$. The set of nodes V indexes the vector of random variables, so $\mathbf{X}_V = \mathbf{X}$. The nodes in V_N and V_I are called the nodes and the interface nodes of the conditional Bayesian network, respectively. Also, the arcs in A_N and A_I are called the arcs and the interface arcs of the conditional Bayesian network, respectively. The set of parameters $\boldsymbol{\theta} = \{P\left(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)}\right), i \in V_N\}$ defines a CPD for each node of the conditional Bayesian network. Note that the interface nodes of the conditional Bayesian networks cannot have parents, while the nodes can have a node or interface node as a parent. A conditional Bayesian network models a conditional distribution $P(\mathbf{x}_{V_N} \mid \mathbf{x}_{V_I})$ factorizing the distribution according to:

$$P(\mathbf{x}_{V_N} \mid \mathbf{x}_{V_I}) = \prod_{i \in V_N} P(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)}). \tag{8.1}$$

The conditional Bayesian network is especially suited for use cases where a set of interface variables $\mathbf{X}_{V_I}$ is always observed and there is no uncertainty, but they can be useful for modeling the uncertainty on the set of nodes $\mathbf{X}_{V_N}$. These conditional Bayesian networks are a special case of Definition 5.17 in Koller and Friedman [2009].

The dynamic Bayesian networks are suitable to model temporal data generated from discrete-time stochastic processes. Therefore, the data are extracted at equally spaced intervals. A dynamic Bayesian network models the probability of a time trajectory $P\left(\mathbf{x}^{(1:T)}\right)$, where T is the number of temporal slices in the data. This probability can be decomposed evaluating the probability of $\mathbf{X}^{(t)}$ given the past $\mathbf{X}^{(1:t-1)}$

$$P\left(\mathbf{x}^{(1:T)}\right) = P\left(\mathbf{x}^{(1)}\right) \prod_{t=2}^{T} P\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(1:t-1)}\right). \tag{8.2}$$

Note that the first temporal slice cannot depend on the past.

Modeling the transition probability $P\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(1:t-1)}\right)$ for every $t = 2, \ldots, T$ can be difficult, especially for large temporal sequences. The dynamic Bayesian network usually makes two assumptions that simplify the definition of this transition probability: the Markovian property and stationarity assumptions. A temporal process satisfies the Markovian property of order k if:

$$P\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(1:t-1)}\right) = P\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-k:t-1)}\right),$$

for every t. Note that this is the same as saying that $\mathbf{X}^{(t)}$ is conditionally independent of $\mathbf{X}^{(1:t-k-1)}$ given $\mathbf{X}^{(t-k:t-1)}$. Therefore, the estimation of $\mathbf{X}^{(t)}$ only depends on the last k temporal slices. The Markovian property is useful because it can reduce the complexity of the transition probability representation. The stationarity assumption is satisfied if the transition probability $P\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-k:t-1)}\right)$ is the same for every temporal slice t. The stationarity assumption allows to reutilize the same transition probability $P\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-k:t-1)}\right)$ for every temporal slice t, so it can be defined just once.

Definition 8.2. (Dynamic Bayesian network). A dynamic Bayesian network of Markovian order k is a pair $(\mathcal{B}_k, \mathcal{B}_{\to k})$ where:

- $\mathcal{B}_k$ is a Bayesian network or dynamic Bayesian network of Markovian order smaller than k that estimates the initial probability $P(\mathbf{x}^{(1:k)})$. This is usually called the initial Bayesian network.

- $\mathcal{B}_{\to k}$ is a Bayesian network that estimates the transition probability $P\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-k:t-1)}\right)$. This is usually called the transition Bayesian network.

These Bayesian networks forbid all arcs from one time slice to a time slice in the past.

Note that the initial Bayesian network of order k can also be modeled using another dynamic Bayesian network of order less than k. However, our implementation uses a normal Bayesian network.

`PyBNesian` implements dynamic Bayesian networks using conditional Bayesian networks (Definition 8.1) to represent $\mathcal{B}_{\to k}$. Therefore, the set of nodes and interface nodes of these conditional Bayesian networks are $\mathbf{X}^{(t)}$ and $\mathbf{X}^{(t-k:t-1)}$, respectively. Thus, the only arcs

Model	Intra-time-slice arcs	Inter-time-slice arcs
k-TBN	$k \cdot \frac{n(n-1)}{2}$	$\frac{k(k-1)}{2} \cdot n^2$
Simplified k-TBN	0	$\frac{k(k-1)}{2} \cdot n^2$
k-TCBN	$\frac{n(n-1)}{2}$	$(k-1) \cdot n^2$

Table 8.1: Number of possible arcs for different types of transition Bayesian networks.

allowed in $\mathcal{B}_{\to k}$ are inter-time-slice arcs from nodes $\mathbf{X}^{(t-k:t-1)}$ to nodes $\mathbf{X}^{(t)}$, and intra-time-slice arcs between the nodes in $\mathbf{X}^{(t)}$. We will denote this type of transition Bayesian networks for the Markovian order $k-1$ as k-transition conditional Bayesian networks (k-TCBNs).

Other works in the literature propose more general transition Bayesian networks (k-TBN, for a Markovian order of $k-1$), in which all inter-time-slice arcs from the past to the future and all intra-time-slice arcs between the nodes of the same time slice are allowed. Alternatively, the simplified k-TBN forbids the intra-time-slice arcs [Trabelsi, 2013]. Figure 8.1 illustrates the different arcs allowed in k-TBNs, simplified k-TBNs and k-TCBNs. Inter-time-slice arcs and intra-time-slice arcs are represented with color blue and red, respectively.

Table 8.1 shows the number of possible intra-time-slice and inter-time-slice arcs in the different transition Bayesian networks. From this table, we can deduce that the number of possible arcs in k-TBNs and simplified k-TBNs is of the order $O(k^2)$, while the number of arcs in k-TCBNs is of the order $O(k)$. This shows that the learning complexity with standard learning algorithms scales better with k-TCBNs than with k-TBNs or simplified k-TBNs.

Finally, we note that a dynamic Bayesian network with $(k+1)$-TCBNs obtains the same results as $(k+1)$-TBNs to evaluate the likelihood of a temporal series:

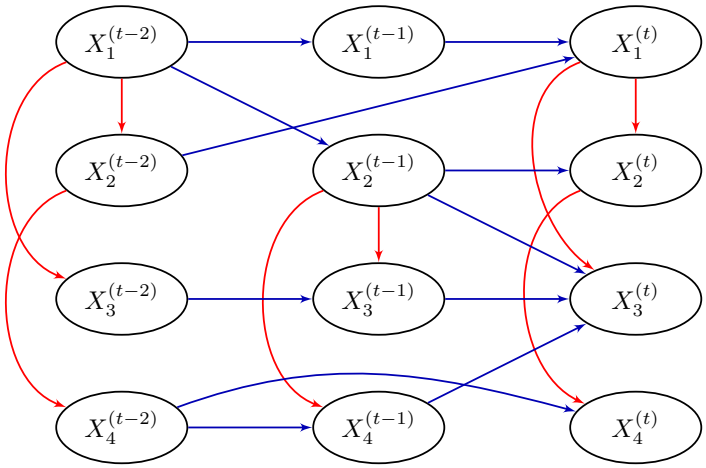
$$P(\mathbf{x}^{(1:T)}) = P\left(\mathbf{x}^{(1:k)}\right) \prod_{t=k+1}^{T} P\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-k:t-1)}\right), \tag{8.3}$$

if the arcs pointing to the variables $\mathbf{X}^{(t)}$ are the same in the k-TCBN and k-TBN models and $\mathcal{B}_k$ is the same in both models. This is because in the evaluation of the transition network $\mathcal{B}_{\to k}$, the variables $\mathbf{X}^{(t-k:t-1)}$ are always observed.

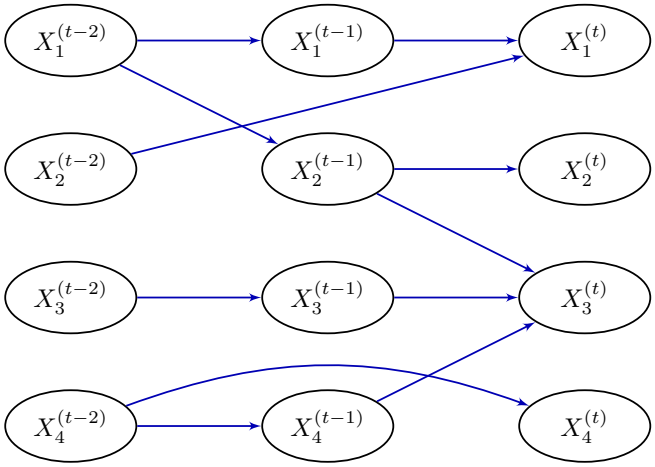
8.2.2 Bayesian Network Types

A Bayesian network type is defined by the CPDs allowed by the network for each node and its arc constraints. Along with the different categories, `PyBNesian` implements different types of Bayesian networks:

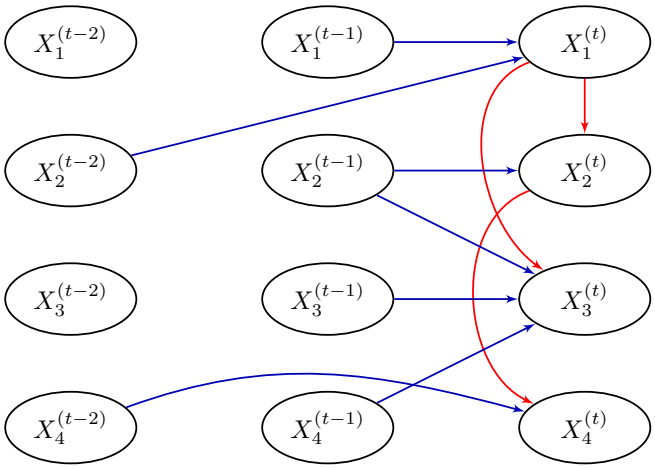
- Discrete Bayesian networks
- GBNs



(a) General 3-TBN



(b) Simplified 3-TBN



(c) 3-TCBN

Figure 8.1: Example representing the structure of a (a) 3-TBN, a (b) simplified 3-TBN and a (c) 3-TCBN. Inter-time-slice arcs and intra-time-slice arcs are represented with color blue and red, respectively.

- CLGBNs
- KDEBNs
- SPBNs / HSPBNs

Each Bayesian network type can be used combined with each Bayesian network category described in Section 8.2.1, so `PyBNesian` implements 15 different Bayesian network models.

To implement these types of Bayesian networks, the following types of CPDs are also implemented:

- Categorical
- LG
- CLG
- CKDE
- HCKDE

Moreover, `PyBNesian` allows implementing new CPD and Bayesian network types. Once a new Bayesian network type is defined, it can be used automatically with the three different categories in Section 8.2.1. This is also described in Section 8.2.8.

8.2.3 Graph Support

To implement the Bayesian network categories described in Section 8.2.1, two categories of graphs are implemented:

- Normal graphs
- Conditional graphs

Conditional graphs partition nodes into normal nodes and interface nodes, and also forbid arcs pointing to an interface node, as also described in Definition 8.1 for conditional Bayesian networks. Normal graphs do not partition the nodes and do not impose any arc constraints.

For each graph category, four different types of graphs are implemented:

- Undirected graphs
- Directed graphs
- DAGs
- PDAGs

Therefore, `PyBNesian` implements 8 different graphs.

For conditional graph models containing undirected edges (undirected and PDAG), the undirected edges between two interface nodes are forbidden, while undirected edges between an interface node and a normal node are allowed. This is justified because the undirected edges between two interface nodes cannot be converted into a directed arc that creates a valid conditional DAG. However, undirected edges between an interface node and a normal node always have a valid direction in which they can generate a valid conditional DAG: directing the edge always from the interface node to the normal node. This distinction is important because some learning algorithms like PC return a PDAG as the representation of the equivalence class. To construct a Bayesian network, then the PDAG is converted into a DAG of the same equivalence class.

The implementation of these graphs is tailored to facilitate the development of probabilistic graphical models, and Bayesian networks in particular. Therefore, the graphs implement basic graph operations: adding and removing nodes, adding and removing arcs, reversing arcs, adding and removing undirected edges, checking if an arc or undirected edge exists, converting an undirected edge into a directed arc and vice versa, etc. Also, some functionalities more relevant for the Bayesian network researcher are implemented: accessing the parents of a node, accessing the children of a node, accessing the neighbors (nodes connected by an undirected edge) of a node, fast access to the roots and leaves of the graph, checking if there is an undirected or directed path between two nodes, obtaining a topological ordering, converting a DAG to a PDAG that represents its equivalence class [Chickering, 1995, 2002], converting an equivalence class PDAG to a DAG [Dor and Tarsi, 1992]. Moreover, when there is no valid conversion of an equivalence class PDAG to a DAG, an original approximate algorithm was implemented that generates a DAG creating a topological ordering that tries to preserve a similar node ordering as the PDAG.

Finally, there exist conversion functions between the graph categories, so a normal graph can be easily converted into a conditional graph and vice versa.

8.2.4 Parameter Learning

`PyBNesian` implements parameter learning for all CPDs described in Section 8.2.2. The parametric CPDs (categorical, LG and CLG) are learned using MLE. The CKDE and HCKDE require estimating the bandwidth matrix. `PyBNesian` implements three bandwidth selection methods (see Section 3.3.3):

- Normal reference rule
- Scott's rule
- UCV criterion

The functionality of `PyBNesian` can be extended implementing new bandwidth selection methods.

Algorithm	BNs	CBNs	DBNs
HC	✓	✓	✗
PC stable	✓	✓	✗
MMPC	✓	✓	✗
MMHC	✓	✓	✗
DMMHC	✗	✗	✓

Table 8.2: Category of the Bayesian networks that can be learned by each learning structure algorithm. Bayesian networks (BNs), conditional Bayesian networks (CBNs) and dynamic Bayesian networks (DBNs).

8.2.5 Structure Learning

`PyBNesian` implements the following structure learning algorithms:

- HC
- PC stable [Colombo and Maathuis, 2014]
- MMPC [Tsamardinos et al., 2003b]
- MMHC [Tsamardinos et al., 2006]
- Dynamic max-min hill-climbing (DMMHC) [Trabelsi et al., 2013]

Table 8.2 shows the category of the Bayesian networks that can be learned for each learning algorithm.

The following subsections illustrate the different components that can be used to modify the behavior of the structure learning algorithms.

8.2.5.1 Learning Operators

The score and search learning algorithms (Section 4.4.1) require defining a set of operators that are used in the optimization process. `PyBNesian` implements the following operators:

- Arc addition
- Arc removal
- Arc reversal
- Type-Change operator (Section 6.2.2.2)

Table 8.3 summarizes the operators that can be used to learn each Bayesian network type. In addition, new learning operators can be created by the user (Section 8.2.8).

Operator	Discrete	GBNs	CLGBNs	KDEBNs	SPBNs / HSPBNs
Arc addition	✓	✓	✓	✓	✓
Arc removal	✓	✓	✓	✓	✓
Arc reversal	✓	✓	✓	✓	✓
TYPE-CHANGE	✗	✗	✗	✗	✓

Table 8.3: List of operators that can be used learning from data by each Bayesian network type.

Score	Discrete	GBNs	CLGBNs	KDEBNs	SPBNs / HSPBNs
BIC	✓	✓	✓	✗	✗
BGe	✗	✓	✗	✗	✗
BDe	✓	✗	✗	✗	✗
$\mathcal{S}^k_{\text{CV}}$	✓	✓	✓	✓	✓
$\mathcal{S}_{\text{validation}}$	✓	✓	✓	✓	✓
$\mathcal{S}^k_{\text{CV}} + \mathcal{S}_{\text{validation}}$	✓	✓	✓	✓	✓

Table 8.4: Compatibility of each function score with each Bayesian network type.

8.2.5.2 Learning Scores

A score function is needed for the score and search learning algorithms (Section 4.4.1). `PyBNesian` implements the following score functions:

- BIC score [Schwarz, 1978]
- BGe score [Kuipers et al., 2014]
- BDe score [Heckerman et al., 1995]
- Cross-validation score (Equation (6.6))
- A holdout score (Equation (6.7))
- A score that combines the cross-validation score and the holdout score as described in Section 6.2.2.2 ($\mathcal{S}^k_{\text{CV}} + \mathcal{S}_{\text{validation}}$)

Table 8.4 describes the compatibility of the above score functions to learn different types of Bayesian networks.

Moreover, new score functions can be easily added by the user (Section 8.2.8).

8.2.5.3 Conditional Independence Tests

The constraint-based learning algorithms are driven by a conditional independence test that identifies the conditional independences in the data. `PyBNesian` implements the following conditional independence tests:

- χ^2 test. Only for categorical data.

- PLC test [Fisher, 1915, 1921]. Only for continuous data.
- CMIknn [Runge, 2018]. A nonparametric estimator of the mutual information based on k-nearest neighbors. This implemenation uses k-d trees [Bentley, 1975]. Only for continuous data.
- RCoT [Strobl et al., 2019]. A nonparametric independence test based on random Fourier features. Only for continuous data.
- A likelihood ratio test based on estimating the mutual information assuming Gaussianity of the continuous data. Valid for hybrid data.

Also, new conditional independence tests can be easily implemented that can work with constraint-based algorithms.

8.2.6 Nonparametric Models

The CKDE (Definition 6.1) and HCKDE (Definition 7.1) CPDs are defined as the ratio of two KDE models. For this reason, `PyBNesian` implements general and product KDE models with Gaussian kernels. This implementation supports univariate and multivariate data. The KDE model evaluation (Equation (3.25)) is an embarrassingly parallel problem [Herlihy and Shavit, 2008] because each kernel evaluation can be executed independently. For this reason, `PyBNesian` implements this problem with OpenCL 1.2, which enables GPU acceleration as it is common in other machine learning libraries nowadays [Abadi et al., 2015; Paszke et al., 2019]. In addition, other functionalities such as sampling data from a CKDE or HCKDE CPD (Section 7.2.3), or the UCV bandwidth selection criterion are also implemented using GPU acceleration. To improve the precision of the results, the *log-sum-exp* trick is used when it can be applied.

Finally, `PyBNesian` also implements product KDE models (Equation (3.27)) for Gaussian kernels with GPU acceleration, as well as bandwidth estimation for product KDEs.

8.2.7 Utilities

`PyBNesian` has its own implementation of the k-fold cross-validation and holdout validation. These two techniques are used to implement the $\mathcal{S}^k_{\text{CV}}$ and $\mathcal{S}_{\text{validation}}$ function scores. However, they are also accessible for the user, so they can be used to implement other components, since both cross-validation and holdout are very common utilities in the machine learning community.

On each Bayesian network model, many interesting functionalities have been implemented:

- Accessing the graph of the Bayesian network, and the most important functionalities of the graph
- Accessing the CPDs of the Bayesian network

- Manually adding CPDs to the Bayesian network
- Learning the parameters of all CPDs
- Computing the log-likelihood (Equation (4.7)) of each instance of a dataset $\mathcal{D}$, or the sum of the log-likelihoods for all instances
- Sampling a new dataset $\mathcal{D}'$ from the Bayesian network

Finally, the most relevant components of the package can be saved and loaded from a file. This includes the following components:

- Bayesian networks, which may contain only the graph or also include its CPDs
- Graphs
- CPDs
- KDE and product KDE models

This process is performed using `pickle` objects, which is a very well-known serialization module in Python.

8.2.8 PyBNesian Functionalities Extension

One of the main characteristics of `PyBNesian` is the possibility of extending its functionality in a modular way. Currently, `PyBNesian` allows creating new components for:

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

Therefore, when conducting new research it is only necessary to implement the new components, which can interoperate with the state-of-the-art algorithms already implemented in `PyBNesian`.

The documentation of the package explains how these extensions can be performed in practice: https://pybnesian.readthedocs.io/en/latest/extending.html.

8.3 Implementation

The package is implemented almost entirely in C++ for the best possible performance. The package `pybind11` [Jakob et al., 2017] is used to enable fast interoperability between Python and C++. In addition, data transfers between C++ and Python are performed using `Apache Arrow`, which almost completely eliminates the overhead of data copy operations.

However, package extensions (Section 8.2.8) can be implemented in Python. This allows for a faster development process of research and interoperation with the large list of machine learning projects in Python.

8.4 Related Software

Other software libraries implement Bayesian networks. In this section we review some of the most common ones.

`bnlearn` [Scutari, 2010] is probably the most mature project in the Bayesian network community, and implements a comprehensive list of techniques for learning Bayesian networks. It implements most of the Bayesian networks in the state-of-the-art such as discrete Bayesian networks, GBNs and CLGBNs. Also, it implements many different score functions and conditional independence tests. This includes some interesting features, such as support for ordinal data. The `bnlearn` package is available for the R language, and it is partially developed in R and C, to improve the performance. However, it may be complex to add extensions to the package.

`pcalg` [Kalisch et al., 2012; Hauser and Bühlmann, 2012] is an R package that mainly implements constraint-based learning algorithms and causal inference. It is implemented in R with some parts in C++ to improve the performance.

`pgmpy` [Ankur Ankan and Abinash Panda, 2015] is a Python package of probabilistic graphical models. It includes Bayesian networks, but with full support only for discrete Bayesian networks. It is implemented completely in Python.

`pomegranate` [Schreiber, 2018] is a Python package of probabilitic graphical models, that includes Bayesian networks. However, it only implements discrete Bayesian networks. It is mainly implemented in Cython, to improve performance.

`LGNpy` [Ostwal, 2020] is a Python package that implements GBNs. It is completely implemented in Python.

8.5 Execution Times

In Section 6.3.5 we provided an analysis of the execution time of different learning algorithms implemented in `PyBNesian`. Therefore, in this section we compare the performance of `PyBNesian` and `bnlearn`, the most mature and efficient state-of-the-art implementation. The comparison is performed using the last version of both libraries: 0.3.4 for `PyBNesian` and 4.7 for `bnlearn`. To compare the performance of both libraries, a large discrete Bayesian network

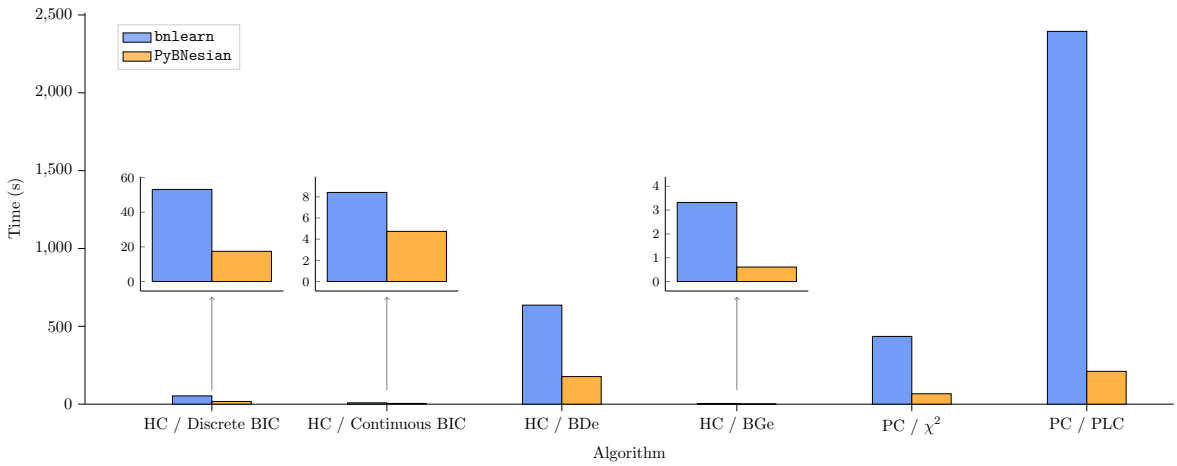


Figure 8.2: Mean execution times of different structure learning algorithms for `bnlearn` (blue) and `PyBNesian` (orange). The comparison is performed on an Ubuntu 16.04 machine with 32GB of RAM and a CPU Intel 6700K (4 GHz).

(DIABETES) and a large continuous Bayesian network (ARTH150) were selected from the `bnlearn`'s [Scutari, 2010] Bayesian network repository. Table 8.5 describes the properties of both Bayesian networks.

Bayesian network	Nodes	Arcs
DIABETES	413	602
ARTH150	107	150

Table 8.5: Properties of the Bayesian networks used for comparison.

We independently sampled 20 different datasets of 10,000 instances from each Bayesian network. Then, we measured the time required to learn a Bayesian network model from these datasets with the two algorithms used in this thesis: HC and PC. For this, we used the score functions implemented in both libraries: BIC (discrete and continuous), BDe, BGe. For the PC algorithm, we used the following independence tests that are implemented in both libraries: χ^2 test (for discrete data) and PLC test (for continuous data). The execution times for both libraries are shown in Figure 8.2. These results show that `PyBNesian` offers a competitive implementation of Bayesian networks compared with an state-of-the-art package. The largest relative difference between the two libraries ocurred for the PC algorithm with the PLC test (`PyBNesian` was approximately 11.35 times faster). In contrast, the smallest relative difference was for the HC with BIC on continuous data (`PyBNesian` was approximately 1.78 times faster).

8.6 Conclusion and Future Work

In this chapter we introduced a Python package called `PyBNesian`, which provides a framework to facilitate research in Bayesian networks. `PyBNesian` implements three different categories of Bayesian networks: Bayesian networks, conditional Bayesian networks and dynamic Bayesian networks. Moreover, it includes many learning algorithms for Bayesian networks in the state of the art. In addition, the `PyBNesian` package allows new components to be developed with a modular approach. In this way, Bayesian network researchers and practitioners only need to implement the components which are outside of state of the art. Finally, since performance can be an issue when running experiments, the runtimes of `PyBNesian` have been shown to be competitive with the state of the art. Furthermore, `PyBNesian` can take advantage of GPU acceleration to evaluate KDE models, thus accelerating the SPBN models.

As future work, we would like to add more structure learning algorithms. Moreover, more parameter learning techniques can be implemented, and it should be easy for the user to create new learning parameter approaches. Lastly, the inclusion of an inference engine would be an important addition.

Part IV

CONTRIBUTIONS TO ANOMALY DETECTION

Chapter 9

KDE-Anomaly Movement Detection

9.1 Introduction

Recently, a full range of industries has made efforts to produce better products more reliably using modern technologies. In the near future, they are expected to undergo major transformations in this regard to improve productivity. This transformation has been termed "Industry 4.0" [Drath and Horch, 2014] and is closely associated with the industrial internet of things (IIoT), both of which encourage the fusion of advanced sensors using machine learning techniques. One of the most useful tasks performed by machine learning in industry is anomaly detection [Clifton et al., 2008; Rao et al., 2009; Yin and Zhu, 2015; Zhang et al., 2011; Tobon-Mejia et al., 2012; Mori and Yu, 2013; Gupta and Ray, 2007; Sarkar et al., 2014]. Anomaly detection [Chandola et al., 2009] is the task of identifying anomalous patterns in data. Usually, anomalous patterns are related to a fault in the system under study. In various industries, anomalies should ideally be detected instantaneously to allow for immediate corrective actions.

In this chapter, we detect anomalies within a novel laser-surface heat-treatment process recorded using a high-speed thermal camera at 1,000 frames per second (fps). The laser heat-treatment process improves the mechanical surface properties of cylindrical steel workpieces (see Section 9.2 for further details). In the premise of our study, we have limited knowledge about the possible anomalies that may occur. In fact, only a single anomaly has been recorded out of thousands of completed processes. Thus, our model describes the normal behavior of a system. Then, it computes an anomaly score for each workpiece. To take advantage of the detected anomalies, the classification should be completed in-process. That is, a workpiece must be classified into normal or anomalous within a short period (∼5 s) after the end of the process. For this reason, we need a model that provides a fast classification. This is a challenging objective because of the numerous video frames required to view a rapidly changing dynamic process.

We train our model using real data provided by a company in the automotive sector. The application of anomaly detection procedures has a significant impact on the quality and cost-effectiveness of production in this industry. If a fault is not detected, it can lead to a loss of reputation of a company, high repair costs, and sometimes, human losses. Methods of anomaly detection based on the destruction of a sample of the entire production have two problems: the cost of the pieces destroyed and the non-exhaustiveness of the analysis. Our approach (Section 9.5) offers a promising capability to detect subtle differences in system behavior to enable the analysis of each working piece. Moreover, our approach is compared with other algorithms to validate its effectiveness.

The chapter is organized as follows. Section 9.2 describes the laser heat-treamtent process. Section 9.3 presents some previous works related to anomaly detection on laser processes. Section 9.4 proposes a new approach to this problem using a novel methodology that employs multiple KDE models (Section 3.3), to characterize the movement of the system's laser along the workpiece surface. Section 9.5 introduces the experimental results obtained using this methodology. Section 9.6 analyzes the sensitivity of our methodology to changes in its parameter values. Finally, Section 9.7 concludes the chapter and presents some future work.

9.2 Laser-surface Heat-treatment Process

In this section, we describe the laser-surface heat-treatment (laser-heating) process, which is often applied to cylindrical workpieces made of steel to improve their surface mechanical properties. The laser-heating process uses a laser beam that heats the surface of a cylindrical workpiece as it rotates around its axis (see Figure 9.1). The laser power source is 8 kW. Thus, it is only safe to operate in a controlled environment. While applying the laser-heating process, the laser beam heats a region (i.e., the laser spot), which is always much smaller than the workpiece surface. Furthermore, high-power (> 6 kW) laser sources have laser spots with Gaussian or top-hat energy distributions. For these reasons, a static laser spot would not provide the energy distribution required to treat the entire surface of the workpiece. Thus, to provide the correct energy deposition and distribution over the surface, the process moves the laser spot following a known trajectory, as represented in Figure 9.2(a). The trajectory pattern is executed at a frequency of 100 Hz to guarantee the desired energy magnitudes. However, the expected pattern is modified whenever the laser beam is required, to avoid an obstacle (see Figure 9.3). To control the pattern, the laser process uses a scanner system with galvanometric mirrors to accurately position the laser spot over the surface. The scanner is a closed-loop control system that sends a specific pattern to the galvanometric mirrors and receives feedback from the encoders where the laser-spot positioning is adjusted. The need to use a moving laser spot is justified because the average workpiece weight is around 12 kg. Thus, it would not be possible to move any mechanical/physical part with the necessary frequency. The laser spot has no such restrictions. All details of the laser heating process can be found in [Gabilondo et al., 2015].

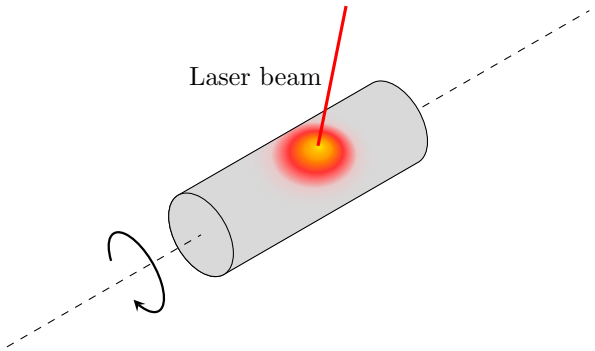
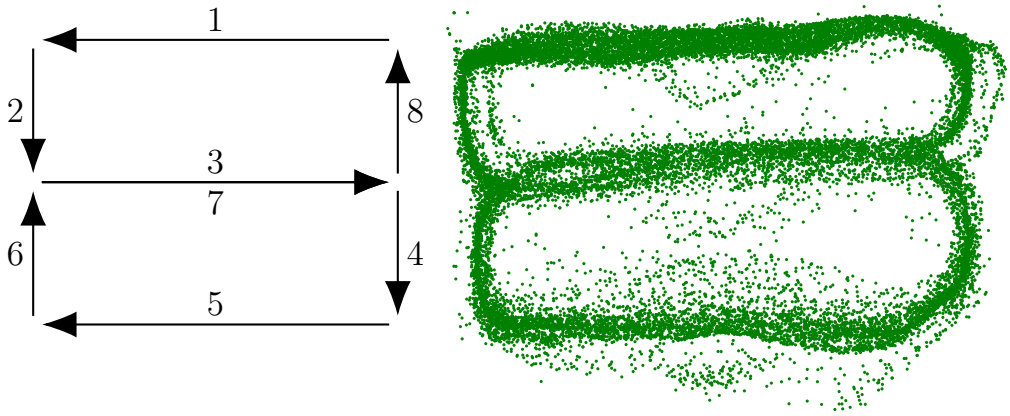


Figure 9.1: Laser-surface heat-treatment process.



(a) (b)

Figure 9.2: Laser spot pattern. (a) Expected. (b) Computed laser spot positions.

Anomalies in the laser-heating process can originate from different components: the scanner system, the workpiece positioning system, and the laser source. This chapter focuses on anomalies generated by the scanner system. As such, the control system can produce anomalies in which positioning data are lost because of communication failures between the electronics and the mirrors, as well as incorrect movement/pattern creation. Additionally, anomalies can be generated through mechanical degradation in the scanner (e.g., changes in the rigidity of the physical linkages between galvanometers and mirrors or galvanometers seizures). All these disturbances strongly affect the scanner behavior and its positioning precision.

Given the novelty of the laser-heating process, prior to this study, there were no automatic procedures for detecting anomalies in this particular laser-treatment process. However, after reviewing the anomaly detection procedures for other laser-treatment processes (Section 9.3.1), we found that the most common technique is temperature reading using pyrometers and thermal cameras. Thus, the laser-heating process is recorded with a high-speed thermal camera (Tachyon μCore) at a resolution of 32×32 pixels and a frame rate of $1,000$

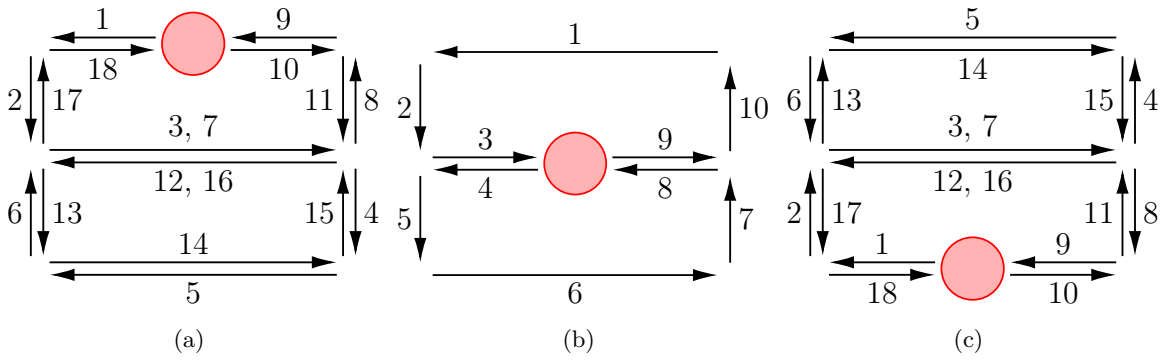


Figure 9.3: Modified patterns when the laser beam avoids an obstacle at different positions. (a) Top. (b) Middle. (c) Bottom.

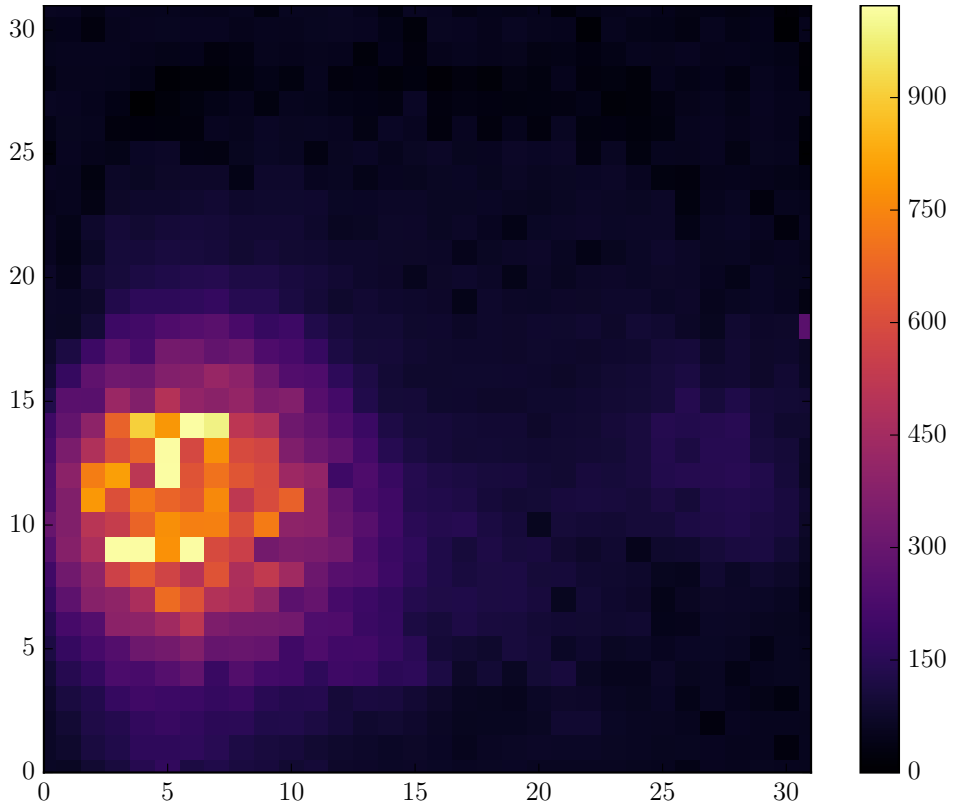


Figure 9.4: Video frame recorded during the laser-surface heat-treatment process.

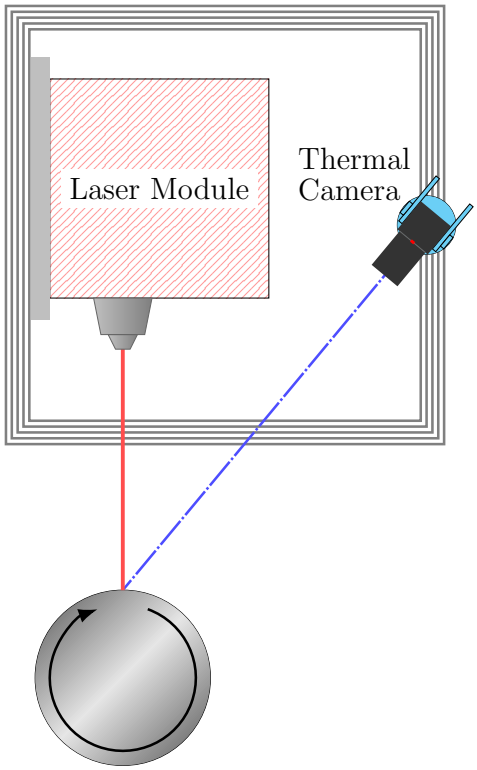


Figure 9.5: Diagram of the physical arrangement of the laser-heating process components.

fps. Each pixel can take a value ranging from 0 to 1,023, proportional to the surface temperature of the workpiece. Figure 9.4 shows a sample frame from one of these videos. A schematic indicating the physical arrangement of the laser beam (red line) and the thermal camera is shown in Figure 9.5.

9.3 Related Work

Before introducing the proposed algorithm, we review other related work of anomaly detection in the laser-treatment processes.

9.3.1 Laser-treatment Anomaly Detection

Our proposal is based on the tracking of the laser movement. The most closely related studies apply to anomaly detection during a laser-welding process using a laser-tracking technique. A model based on the Sage–Husa adaptive Kalman filters using an embedded Elman neural network was proposed in [Gao et al., 2012] to minimize the tracking error of the weld-seam position. The laser movement traced a straight line, and therefore, the tracking was 1D, as opposed to the complex pattern addressed in our study. Another study [Jäger et al., 2008a] aimed to find anomalies by analyzing the trajectories of the sputters produced during the laser-welding process. These sputters were visualized in a video as hot objects moving across

the field of view. A Kalman filter was used for leveraging a state transition matrix that assumed a constant accelerated motion.

Moreover, there are other approaches unrelated to tracking for video analysis, such as HMMs [Jäger et al., 2008b], where the likelihood ratio of a model trained with non-anomalous sequences to a model extended with anomalous sequences is used as the classification criterion. Because the number of pixels can be quite large, dimensionality reduction techniques, such as principal component analysis, have been proposed [Jäger and Hamprecht, 2009; Linares et al., 2015] as a prior step to classification. In [Linares et al., 2015], the same camera as that used in this research was employed, but at a higher frame rate, 10,000 fps, for the classification of defects in a laser-welding process.

Our proposal combines the tracking of laser movements while training a statistical model that computes the likelihood of the test data to detect anomalies. As opposed to our research, none of the previous related studies had any classification time requirement.

9.4 KDE-Anomaly Movement Detection (AMD)

We detect the anomalies in the laser heating process by identifying unusual laser-spot movements. For this reason and because the algorithm relies on a combination of KDE models, we call our approach "KDE–AMD." If the laser-spot movement deviates significantly from the expected pattern shape or the expected laser speed, the workpiece can be considered anomalous. Figure 9.6 presents the flowchart of the proposed methodology.

A model of the process evolution is learned using the training data, which capture the normal behavior of the process. In our case, the model considers the spatio-temporal characteristics of the laser-heating process by learning the expected movements of the laser in different spatial regions and temporal moments. Thus, KDE-AMD can be considered an ensemble of KDE models taking advantage of the temporal/spatial locality of the laser movements to train a faster and more accurate model. This is because only the most similar (in terms of time and position) training laser movements are considered to classify each movement. The divergence between the outcomes of the model learned using the training and test data is used as the discriminant to detect an anomaly. The details of each procedure are presented in the remainder of this section.

9.4.1 Extraction of the Laser-Spot Positions

Our preprocessing step entails the extraction of the laser-spot positions from the recorded videos. In Section 9.2, we mentioned that the laser spot follows a known pattern. The closer the surface region is to the laser beam, the larger is the expected temperature increase. Thus, we can detect the laser-spot positions by computing the regions having higher temperatures.

Let $\mathbf{O} = (\mathbf{O}_1, \ldots, \mathbf{O}_N)$ denote an original video from the laser-heating process, where $\mathbf{O}_i$ represents the i-th frame.

First, we create a video showing the positive temperature variation between each pair of

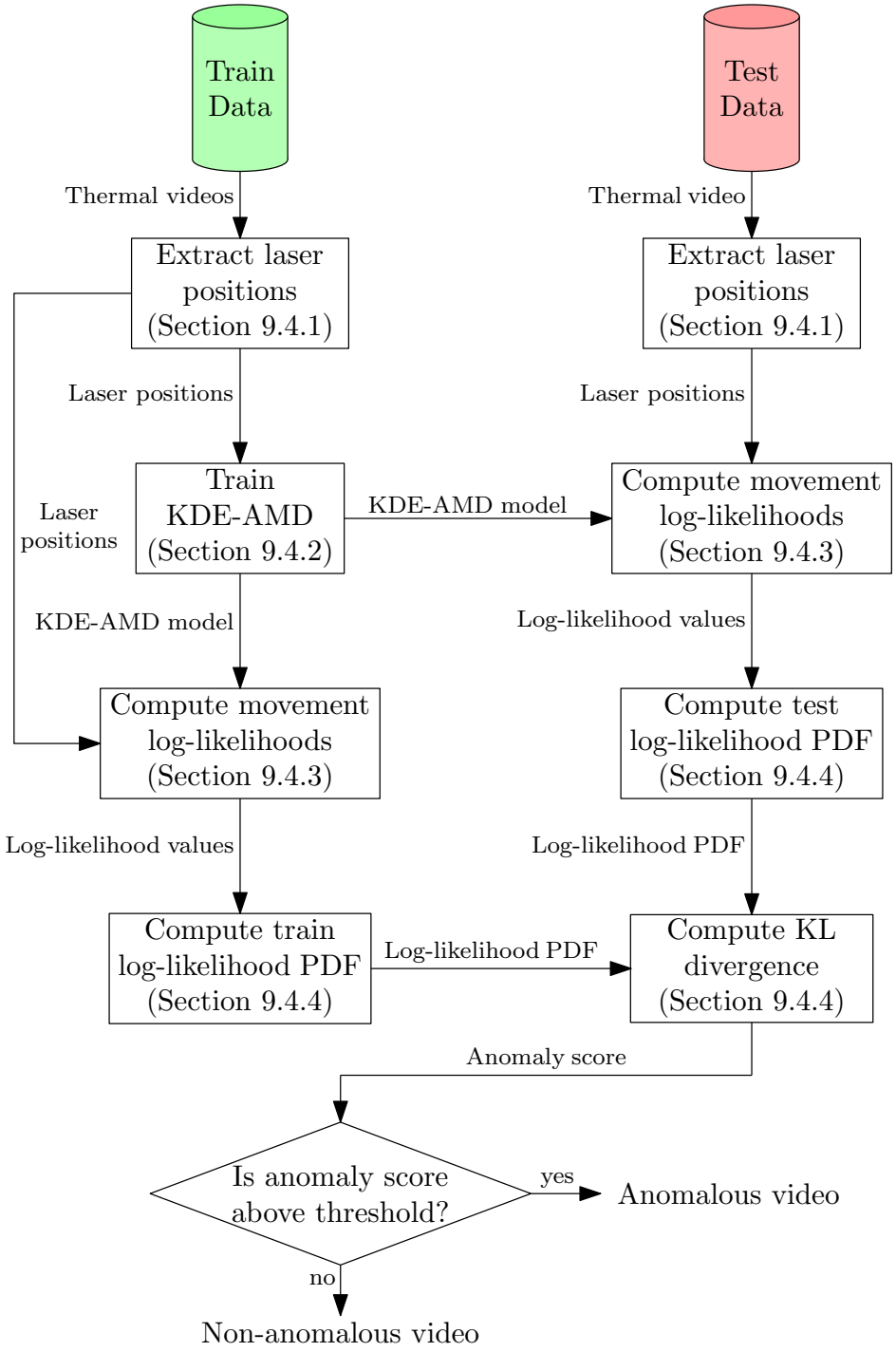


Figure 9.6: KDE–AMD preprocessing, training, and classification flowchart.

two consecutive frames. This video is called the *difference* video:

$$\mathbf{F} = (\mathbf{F}_1, \ldots, \mathbf{F}_{N-1}), \text{ such that } f^i_{j,k} = \begin{cases} o^{i+1}_{j,k} - o^i_{j,k}, & \text{for } o^{i+1}_{j,k} - o^i_{j,k} > 0 \\ 0, & \text{for } o^{i+1}_{j,k} - o^i_{j,k} \leq 0 \end{cases} \tag{9.1}$$

$o^i_{j,k}$ is the pixel intensity value in the j-th row and k-th column of $\mathbf{O}_i$, and $f^i_{j,k}$ denotes the difference in pixel intensity value in the j-th row and k-th column between frames $\mathbf{O}_{i+1}$ and $\mathbf{O}_i$. Thus, the frames $\mathbf{F}_i$ produce the *difference* videos. Note that the pixels having negative variation are set as zero in the *difference* video, which contains one less frame than does the original video. Using $\mathbf{F}_i$, the laser-spot position for that frame can be computed as a weighted mean, as follows:

$$\begin{aligned} \mathrm{s}^{\mathrm{row}}_i &= \frac{1}{\sum\limits_{f^i_{j,k} \in \mathbf{F}_i} f^i_{j,k}} \sum_{f^i_{j,k} \in \mathbf{F}_i} j \cdot f^i_{j,k}, \\ \mathrm{s}^{\mathrm{col}}_i &= \frac{1}{\sum\limits_{f^i_{j,k} \in \mathbf{F}_i} f^i_{j,k}} \sum_{f^i_{j,k} \in \mathbf{F}_i} k \cdot f^i_{j,k}. \end{aligned} \tag{9.2}$$

$\mathrm{s}^{\mathrm{row}}_i$ and $\mathrm{s}^{\mathrm{col}}_i$ denote the row and column, respectively, of the laser-spot position in frame i. With this computation, the calculated laser-spot position will be near the pixels having higher temperature values. A similar approach was used in Gao et al. [2012]. Because $\mathrm{s}^{\mathrm{row}}_i$ and $\mathrm{s}^{\mathrm{col}}_i$ are computed with a weighted mean, they are continuous values. This implies that the center of the laser spot does not necessarily lie in a pixel coordinate; it can be found between pixels.

9.4.2 Training

KDE–AMD must be trained using data representing the normal processes. For numerous anomaly detection applications, only a few instances of anomalous data are often available. Therefore, our methodology builds a model of the normal system behavior instead of an anomalously driven classifier. The anomalous processes are expected to show a different behavior than the trained model. In our use case, the laser-spot positions described in Section 9.4.1 are used to train the KDE–AMD model.

Let $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_{N-1})$ denote the calculated laser-spot positions, where $\mathbf{s}_i \in [0, 31] \times [0, 31]$. $\mathbf{s}_i$ is the laser-spot position for frame i, whose value comprises the computed row and column, $(\mathrm{s}^{\mathrm{row}}_i, \mathrm{s}^{\mathrm{col}}_i)$, in accordance with Equation (9.2).

We define a movement $\mathbf{M} = (\mathbf{s}_O, \mathbf{s}_D)$ as the transition between two consecutive frames. Thus, $\mathbf{s}_O$ is the original point of the movement and $\mathbf{s}_D$ is the destination point, such that $D = O+1$. A video has a set of $N-2$ movements, because there are $N-1$ laser-spot positions, and each of the two consecutive laser spot positions are grouped in a single movement. The KDE–AMD model can be trained using the movements of multiple videos. Therefore, from

the training data, we have a set

$$\mathbf{E} = \{\mathbf{M}_1, \ldots, \mathbf{M}_L\}, \tag{9.3}$$

containing L different movements that are obtained from different videos. Because all videos may have different lengths, L may not be equal to $(N-2) \cdot \#\text{videos}$. Note that, in the set $\mathbf{E}$, the destination position of the movement, $\mathbf{M}_i$, can differ from the origin position of the movement, $\mathbf{M}_{i+1}$, because $\mathbf{M}_i$ and $\mathbf{M}_{i+1}$ can be obtained from different videos.

The KDE–AMD algorithm assumes that the destination points of two movements should be close if their origin points are close. We model the expected destination point of a movement using bivariate KDE models to model the two axes of the laser-spot position. Multiple KDE models are generated to consider the spatio-temporal characteristics of each movement. The KDE models are generated at the time of training to expedite the classification phase. First, we distribute the set of movements, $\mathbf{E}$, into W non-overlapping temporal windows. The number of windows, W, is determined from the number of obstacles that we need to avoid (see Figure 9.3), i.e., $W = 1 + 2 \cdot \#\text{obstacles}$. For example, if there is a single obstacle, then there would be three different temporal frames: pre-obstacle, obstacle, and post-obstacle. The start/end frame for each temporal moment is known in advance, because the laser-heating process is controlled by a CAD/CAM system that ascertains the position of the obstacles. The aim of the temporal division is to learn different KDE models when the expected pattern is different (see Figure 9.3), because both the recorded thermal footprint and computed laser positions will be quite different. The number of obstacles can be different for each type of workpiece. Details about the number of obstacles for our use case are given in Section 9.5. This type of temporal division can be avoided if the laser-heating process were to not exhibit different patterns at different temporal moments.

We then divide each temporal window into R rows and C columns to consider the spatial information. The result of these divisions is a 3D matrix of $W \times R \times C$ regions. R and C are parameters of the algorithm that determine the number of regions, r, in the 3D matrix. Uniform partitioning is used to attach the same importance to all spatial regions. As mentioned, the laser-spot positions are continuous values, and can be found between pixels. Therefore, any integer value can be used for R and C, because we can create subpixel size partitions. For example, if we select $R = C = 64$, the spatial regions would be of size 0.5×0.5 pixels.

For each region, $r = (t, j, k)$ in the t-th temporal window, and the j-th row and k-th column of the 3D matrix, we find a set of movements whose point of origin is inside region r.

$$\mathbf{E}_r = \{\mathbf{M}_i = (\mathbf{s}_O^i, \mathbf{s}_D^i) | \mathbf{s}_O^i \in r\},\ i = 1, \ldots, L, \tag{9.4}$$

where $\mathbf{s}_O^i$ denotes the point of origin of the i-th movement, $\mathbf{M}_i$. Note that in the set $\mathbf{E}_r$, not all movements are necessarily contiguous. That is, the destination position of $\mathbf{M}_i$ is not the original position of $\mathbf{M}_{i+1}$. This is because the set $\mathbf{E}_r$ only contains movements having an origin position in region r. Usually, the laser-spot position changes its region between consecutive frames. Thus, contiguity is lost. Additionally, the movements can originate

Parameter	Description	Value
γ	Minimum number of positions to generate a KDE model	50
R	Number of rows in the 3D matrix	35
C	Number of columns in the 3D matrix	35

Table 9.1: Parameters of our methodology

from different videos. Thus, there is no contiguity for movements from different videos. A bivariate KDE model is assigned to region r, trained with the set of points of destination of the movements in $\mathbf{E}_r$ as

$$\mathbf{D}_r = \{\mathbf{s}_D^i | \mathbf{M}_i = (\mathbf{s}_O^i, \mathbf{s}_D^i) \in \mathbf{E}_r\},\ i = 1, \ldots, L. \tag{9.5}$$

When $\mathbf{D}_r = \emptyset$, r will not have an assigned KDE model, because there is no information on the possible point of destination of the laser spot. In fact, we can try to generate more informative KDE models by ignoring the KDE models trained with fewer than γ movements. If we use a larger γ, we expect more regions without an assigned KDE model.

As an example of the application of the methodology, we show the training process for a given region, r, in Figure 9.7. All origin positions inside region r are represented by blue squares in Figure 9.7(a), and their respective movement destination positions are depicted as red triangles in Figure 9.7(b). The red triangles are the training set for the KDE model assigned to region r. As illustrated in Figure 9.7(c), the PDF of the KDE model shows higher densities in a small region where the destination positions are located. Therefore, the KDE model for region r can be considered an estimation of the destination position PDF for the movements with an origin position inside region r. For a test movement, $\mathbf{M}_t = (\mathbf{s}_O^t, \mathbf{s}_D^t)$, where $\mathbf{s}_O^t \in r$, the PDF for the destination position, $\mathbf{s}_D^t$, is

$$\hat{f}_{\text{KDE}}(\mathbf{s}_D^t) = \frac{1}{|\mathbf{D}_r| 2\pi |\mathbf{H}|^{1/2}} \sum_{\mathbf{t} \in \mathbf{D}_r} \exp\left(-\frac{1}{2}(\mathbf{s}_D^t - \mathbf{t})^T \mathbf{H}^{-1} (\mathbf{s}_D^t - \mathbf{t})\right), \tag{9.6}$$

where $\mathbf{t}$ denotes the training positions in set $\mathbf{D}_r$, $\mathbf{H}$ is the bandwidth matrix obtained using Scott's rule (Section 3.3.3.1), $|\mathbf{D}_r|$ is the cardinality of the set $\mathbf{D}_r$, and $|\mathbf{H}|$ is the determinant of the matrix $\mathbf{H}$.

The training procedure of KDE–AMD is summarized in Algorithm 9.1. In lines 3–5, the temporal and spatial indices in the 3D matrix are computed for each movement. To compute them, we need to know the limits of the temporal moments and spatial regions. As indicated, the temporal window limits are known in advance, and the spatial limits are known when R and C are fixed. When the loop in lines 2–7 is completed, each region in the 3D matrix, KDE–AMD$[t, j, k]$, is equal to $\mathbf{D}_r$. In line 9, we check that there are enough points in a given region to sufficiently train accurate KDE models.

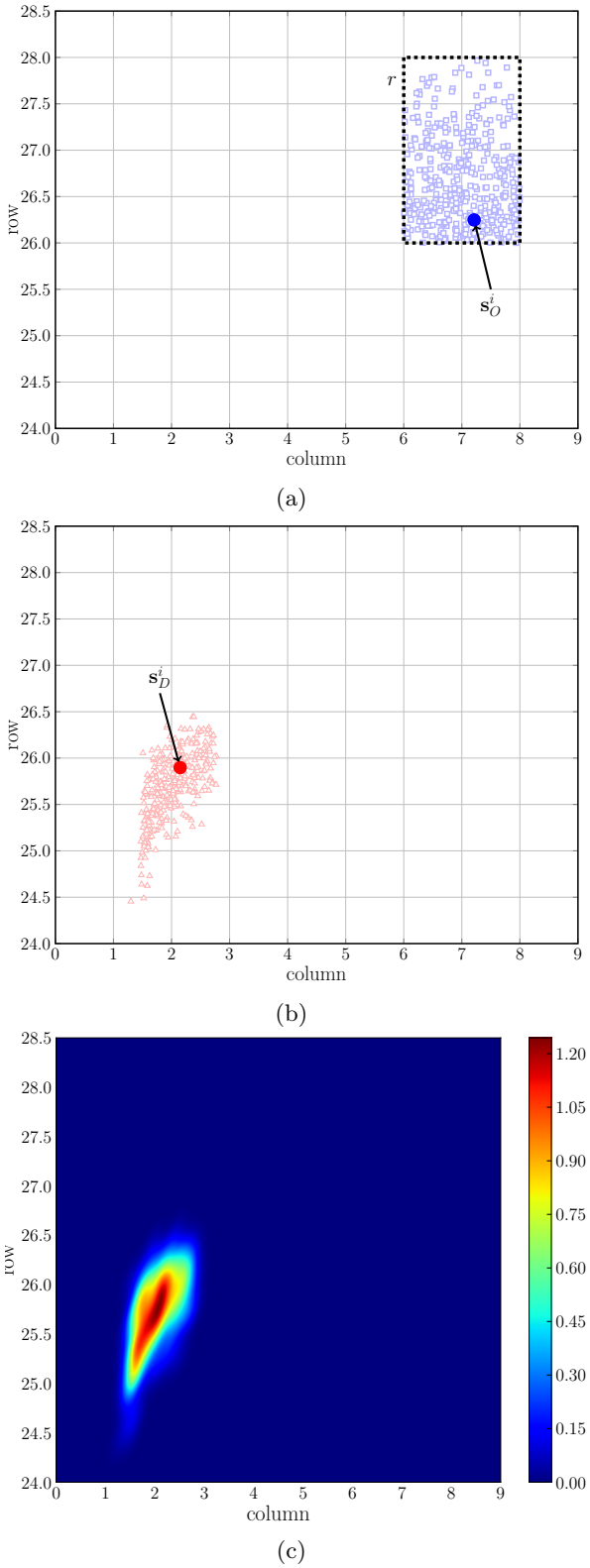


Figure 9.7: Example application of the methodology. (a) Points with origin in region r. (b) Their corresponding destinations. (c) PDF $\hat{f}_{\text{KDE}}$ in Equation (9.6) assigned to region r.

Algorithm 9.1 KDE-AMD training procedure

Require: Set of movements $\mathbf{E}$ (see Equation (9.3)), number of windows W, number of rows R, number of columns C, minimum number of positions γ

Ensure: Trained KDE-AMD model

Initialisation :
KDE–AMD $\leftarrow$ Create a 3D matrix of $W \times R \times C$ regions.

Computes $\mathbf{D}_r$ *(see Equation (9.5)), for all regions.*
for each movement $(\mathbf{s}_O, \mathbf{s}_D)$ **in E do**
    $t \leftarrow$ Temporal index of the frame O in the video.
    $j \leftarrow$ Row index computed using $\mathrm{s}_O^{\mathrm{row}}$.
    $k \leftarrow$ Column index computed using $\mathrm{s}_O^{\mathrm{col}}$.
    KDE-AMD$[t, j, k] \leftarrow$ KDE-AMD$[t, j, k] \cup \mathbf{s}_D$.
end for

Train the bivariate KDE models.
for each region r in KDE-AMD **do**
    if npositions$(r) \geq \gamma$ **then**
        Train a KDE model with the positions included in r and save it in r.
    else
        Discard the positions saved in r.
    end if
end for
return KDE-AMD

9.4.3 Movement Likelihood

To classify a test video, the likelihood of each test movement is first computed at the time of classification. The likelihood of each movement can be obtained by evaluating the destination position in the correct KDE model of the trained 3D matrix.

First, we must extract the laser-spot positions of the test video, as described in Section 9.4.1. For each movement, $\mathbf{M}_i$, the appropriate KDE model is selected using position $\mathbf{s}_O^i$. Two types of information are needed to access the correct KDE model: temporal (to select the correct temporal window) and spatial (row and column). After the suitable KDE model is found, the likelihood of the movement is obtained by evaluating $\mathbf{s}_D^i$ in the KDE model with Equation (9.6). A Laplace smoothing is applied to the evaluated result to deal with the 0 likelihood values. As noted in Section 9.4.2, some regions will not have any assigned KDE model. In that case, we can assign a minimum likelihood value to the movement (not 0, because of the Laplace smoothing).

Figure 9.7 also shows an example of the computation of the likelihood of a test movement. Because $\mathbf{s}_O^i$ (in Figure 9.7(a)) is inside region r, we use the KDE model depicted in Figure 9.7(c), trained with the points in Figure 9.7(b). In the example, the result of evaluating position $\mathbf{s}_D^i$ (in Figure 9.7(b)) in the KDE model is a likelihood, $\hat{f}_{\text{KDE}}$, of 1.2 for the movement.

We note that each test movement is evaluated using only the destination positions in $\mathbf{D}_r$. Therefore, KDE–AMD can be fast, because it does not use all training movements. Additionally, the classification procedure can be easily parallelized, because the evaluation of different KDE models can be computed in parallel without further dependencies on each movement. Additionally, we do not need the complete video to begin evaluating the KDE models. For instance, the evaluation of the first movement only requires the first two frames. The next movement can be classified with one more frame. Thus, some movements can be classified before the laser-heating process is completed. These characteristics can reduce the classification time even more, so that the algorithm can achieve in-process classification.

Table 9.1 summarizes all parameters defined in the methodology and their values when applied to real data in the experiments (Section 9.5).

9.4.4 Anomaly Score

We compute an anomaly score using all of the obtained movement probabilities, as discussed in the previous section, to classify a given test workpiece. An anomaly score provides an assessment of an abnormality for a given workpiece. When the anomaly score is above a predefined threshold, the workpiece is classified as anomalous. Usually, the threshold is found by studying the receiver operating characteristic (ROC) curves [Fawcett, 2006], selecting the threshold that provides a convenient trade-off between the true- and false-positive rate. This decision can be motivated by the cost of a misclassified anomalous/non-anomalous workpiece. The likelihood of movement $\mathbf{M}_i$ is denoted as p_i. For each likelihood value, its negative log-likelihood, $l_i = -\log p_i$, is calculated.

The set of l_i values characterizes a laser-heating video and is used to compute an anomaly score, which is based on the dissimilarity between the training and test log-likelihood values. A convenient method for comparing the training and test videos is to estimate the PDF of the log-likelihood values for the training and test data. Analyzing the PDFs allows a comparison of the log-likelihood information, even when there are more available training log-likelihood values than test ones. The PDFs of the log-likelihood values can be estimated using a univariate KDE model, $\hat{f}_{\text{KDE}}(l)$, over a set of log-likelihood values.

After the PDFs are computed, the anomaly score is calculated using the Kullback–Leibler divergence [Kullback and Leibler, 1951] between the training and test PDFs. This divergence is lower-bounded by zero. Thus, a non-anomalous workpiece is expected to have a near-zero value. The Kullback–Leibler divergence is neither upper-bounded nor normalized. Thus, the anomalous workpieces can have arbitrarily large anomaly scores.

The methodology is described in terms of our use case (i.e., a laser-heating process) for easiness. However, the methodology can be adapted to other problems by creating multiple regions over each variable of interest. In our use case, the variables of interest are the axes of the laser-spot position. For other use cases, the dimensionality of the generated matrix can be customized, depending on the number of variables and if the data exhibit different behavior at several temporal moments.

9.5 Experiments

The KDE–AMD algorithm is compared with other methods, including deep neural networks [Goodfellow et al., 2016], Kalman filters [Kalman, 1960], D-Markov machines [Rao et al., 2009], and a global KDE algorithm, to demonstrate its effectiveness. Real (Section 9.5.1) and simulated (Section 9.5.2) data are used to test the algorithms.

The global KDE algorithm trains a single bivariate KDE model with a complete set of positions of the set of normal videos. Therefore, there is no spatial division in the complete set of positions as in KDE–AMD. Another single bivariate KDE model is generated for the test positions. Then, the Kullback–Leibler divergence was computed between both KDE models. Similar to the presented approach, temporal windows can be created for the global KDE model. In this case, we computed W different Kullback–Leibler divergences. The classification criterion was based on the weighted (by the length of each temporal window) average of the Kullback–Leibler divergences.

The D-Markov machine builds a Markov chain of order D, where the possible states are represented by different symbols. First, a symbolization (partition) process is applied to the data, where a symbol is assigned to each partition of the input space. Next, a transition matrix is computed such that the probability of the next symbol depends only on the previous D symbols. The classification criterion is based on a state probability vector calculated from the transition matrix. In Rao et al. [2009], further details are provided on the use of D-Markov machines.

The Kalman-filter algorithm is based on the research reported in Jäger et al. [2008a] and

is described in Section 9.3. In our study, we cannot obtain good estimates of acceleration because the process has no jerk (i.e., da/dt) control, allowing high magnitudes of acceleration change only limited by the physics of the mechanical design. As a result, the sampling rate of only 10 frames per cycle is too low to reliably reconstruct a signal with high jerk. Thus, the state vector of our Kalman filter comprises only the laser-spot position and speed.

The deep neural network is trained to predict the next laser-spot position from the current one. The network comprises seven hidden layers with 8, 128, 256, 512, 256, 128, and 8 neurons, with rectified linear-unit activation functions. Thus, the network has 331,058 parameters. This is the best structure found after trying different numbers of hidden layers and neurons per layer. The deep neural network is trained using Keras [Chollet, 2015]. The anomaly score is computed as the sum of the Euclidean distance between the predicted and actual laser-spot positions. For this reason, smaller anomaly scores are expected for non-anomalous videos.

For each algorithm, the tests are conducted multiple times to select the best values for their parameters. In the KDE–AMD and D-Markov algorithms, different numbers of spatial divisions (R rows $\times$ C columns) are tested: 16×16, 20×20, 25×25, 30×30, 35×35, and 40×40. The best results (reported here) are obtained using 35×35 divisions for KDE–AMD. For the D-Markov machines, various symbolization techniques are used: equal width, equal frequency, and a variant of equal frequency in which the areas within the limits of the space without any position are not considered for the purpose of symbolization. Slightly better results are obtained for a 40×40 equal-width division, where D is set to 1 because a higher D is highly computationally expensive.

The D-Markov machine and KDE–AMD algorithms are tested using leave-one-out cross-validation. The Kalman filter does not need training data to build a model. The global KDE cannot be evaluated using the leave-one-out cross-validation method because of the associated computational burden. Therefore, we train the global KDE model with four videos and test it with the remaining 4,219. The deep neural network is validated using two-fold cross-validation to reduce the high computational cost of the training phase.

The code for all algorithms can be found in the repository `https://github.com/davenza/KDE-AMD`. Also, the data are available in Atienza et al. [2019] to reproduce all results in the experiments.

9.5.1 Real Data

To test the presented methodology, 4,223 videos of real laser-surface heat-treatment processes are available, which are recorded on four different workstations and contain nine different workpiece types. Each workpiece type has a different size between 12,000 and 14,500 frames and several obstacles between 0 and 2. Altogether, there are 36 batches of videos, because there is a batch type for each workstation and workpiece type. The mean length in frames, its standard deviation, and the number of obstacles, for each batch type, are described in Table 9.2.

For this reason, each video is classified using only videos of the same type, applying leave-one-out cross-validation. Out of the 4,223 videos, only a single video is classified as

Type	Length (frames)	Obs.	Type	Length (frames)	Obs.
1	14541.50 ± 22.41	0	19	14492.94 ± 43.95	0
2	14495.28 ± 37.47	2	20	14473.76 ± 36.67	2
3	14157.77 ± 33.04	0	21	14137.31 ± 33.90	0
4	14529.95 ± 22.05	2	22	14483.90 ± 37.51	2
5	14097.07 ± 36.20	0	23	14135.84 ± 39.85	0
6	12310.78 ± 28.60	1	24	12333.90 ± 24.51	1
7	12316.06 ± 34.07	1	25	12338.62 ± 22.92	1
8	12296.55 ± 19.19	1	26	12334.88 ± 26.64	1
9	12071.47 ± 21.71	1	27	12115.34 ± 24.29	1
10	14633.50 ± 92.56	0	28	14665.61 ± 18.99	0
11	14624.44 ± 61.77	2	29	14652.72 ± 19.91	2
12	14274.75 ± 67.28	0	30	14314.01 ± 25.06	0
13	14644.38 ± 70.09	2	31	14659.39 ± 19.67	2
14	14265.64 ± 65.26	0	32	14309.83 ± 22.49	0
15	12458.49 ± 65.27	1	33	12510.37 ± 16.12	1
16	12457.25 ± 55.25	1	34	12516.75 ± 18.32	1
17	12454.47 ± 43.15	1	35	12520.57 ± 22.16	1
18	12230.65 ± 55.81	1	36	12296.82 ± 20.28	1

Table 9.2: Batch-type length (number of frames) and number of obstacles (Obs.)

anomalous by human experts. Each final workpiece on this production line was inspected by non-destructive testing procedures. In addition, a sample of them (1 in 1000) was inspected by destructive testing procedures. These testing procedures confirmed that only a single video was anomalous. In the anomalous video, the laser moves with a slightly different speed than in the remaining videos, changing the heat distribution and final result of the workpiece. However, the pattern shape in the anomalous video is the same as that in Figure 9.2(a). Table 9.3 lists the minimum, mean, maximum, and standard deviation values for the non-anomalous anomaly scores. For the anomalous video, only one anomaly score is reported. The anomaly scores are computed differently for each algorithm, and therefore, not comparable with each other. However, we analyze the behavior of the anomaly score for each algorithm when classifying anomalous and non-anomalous workpieces. Anomaly scores should be higher for anomalous workpieces than those for non-anomalous workpieces. Note that the D-Markov does not detect the anomaly, because the anomaly score for the anomalous video is lower than the maximum anomaly score for the non-anomalous videos. However, the KDE–AMD, global KDE, Kalman filter, and deep neural network detect the real anomaly with an anomaly score approximately 6.24, 11.63, 10.71, and 1.54 times higher than the maximum anomaly score value in the non-anomalous videos, respectively. Moreover, the standard deviations are quite small, showing that the anomaly scores are quite stable and are very close to zero in the non-anomalous videos. Note that the judgment of human experts is used only for evaluation purposes. Using human experts to inspect each workpiece would be infeasible, given the numerous videos generated. In fact, the human experts found an anomaly video driven by the anomaly scores provided by the KDE–AMD algorithm. In a real-world scenario, the system

Anomaly score		Anomalous video	Non-anomalous videos
KDE-AMD	Min.	1.775×10^{-1}	5.552×10^{-5}
	Mean.	1.775×10^{-1}	8.676×10^{-4}
	Max.	1.775×10^{-1}	2.845×10^{-2}
	Std.	0	1.186×10^{-3}
Global KDE	Min.	1.102	2.960×10^{-2}
	Mean.	1.102	3.727×10^{-2}
	Max.	1.102	9.478×10^{-2}
	Std.	0	4.175×10^{-3}
D-Markov machine	Min.	1.832	5.810×10^{-2}
	Mean.	1.832	1.036
	Max.	1.832	2.064
	Std.	0	9.280×10^{-1}
Kalman filter	Min.	7.915×10^{-2}	3.718×10^{-3}
	Mean.	7.915×10^{-2}	5.271×10^{-3}
	Max.	7.915×10^{-2}	7.392×10^{-3}
	Std.	0	8.319×10^{-4}
Deep neural network	Min.	1558.703	133.241
	Mean.	1558.703	358.351
	Max.	1558.703	1014.479
	Std.	0	162.292

Table 9.3: Anomaly score values in the anomalous and non-anomalous videos in the real data.

can work completely autonomously, because a threshold value between 2.845×10^{-2} and 1.775×10^{-1} will correctly classify all videos. Thus, we consider the KDE–AMD performance on par with the global KDE and Kalman-filter algorithms, even when the anomaly score ratio of the anomalous to non-anomalous videos (6.24) is slightly smaller than that of the other algorithms. Furthermore, the KDE–AMD anomaly score ratio is higher than the deep neural network anomaly score ratio. Further experimentation (computing the anomaly score values of more anomalous videos) is necessary to fine-tune the threshold value and provide a more accurate comparison of the models.

9.5.2 Simulated Data

Owing to the high reliability of the laser-heating process, there is only a single anomalous video. Consequently, the evaluation can be strengthened using simulated anomalous data by modifying the normal datasets to generate them. The real-data scenario in Section 9.5.1 showed that KDE–AMD can detect unusual movement patterns. However, more subtle deviations from the baseline can occur with a gradual degradation of the laser positioning system. To simulate this behavior, Gaussian noise is introduced at each laser-spot position in both dimensions independently. Such anomaly is simulated because a gradual degradation of the laser positioning system is expected to modify the laser-spot positions very slightly. This is important because the laser positioning system requires constant maintenance, and appropriate corrective actions can be taken only when the anomaly score deviation from the baseline

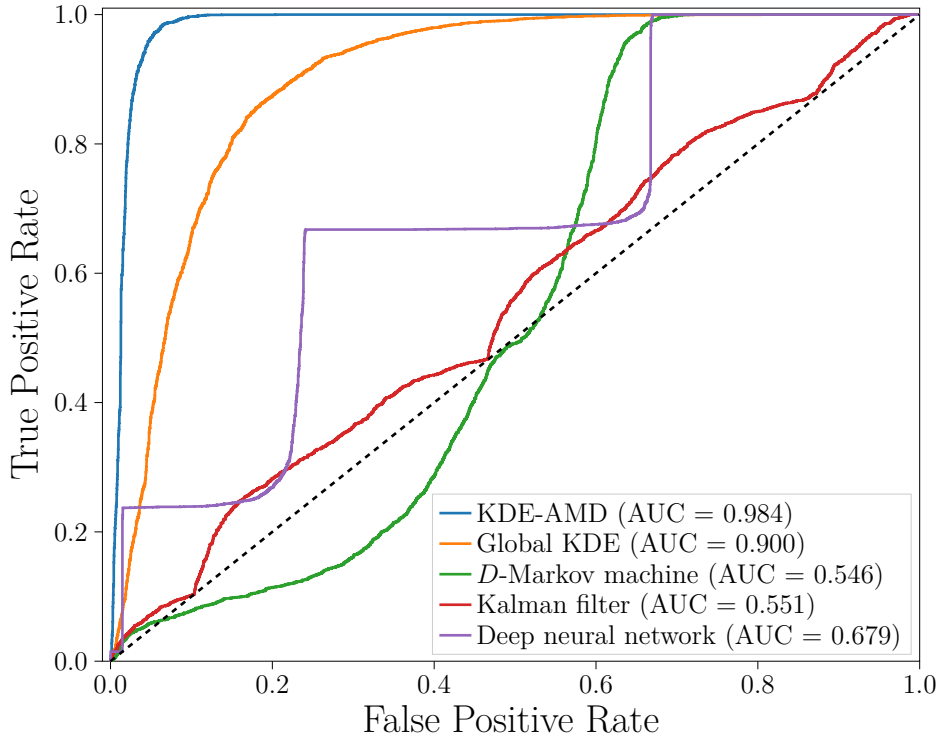


Figure 9.8: Comparison results of the algorithms with $\sigma^2 = 0.02$.

is sufficiently large (predictive maintenance).

A total of 4,222 datasets are available for testing, because an anomalous dataset can be created for each normal dataset. In this analysis, we omit the known real anomaly to generate subtle variations relative to the normal data. A Gaussian noise of mean 0 and variance 0.02 is added to each laser position dimension in the normal datasets to produce anomalous datasets. The methods are validated using ROC curves and the area under the curve (AUC) [Bradley, 1997]. Figure 9.8 shows the results obtained from the four algorithms. The AUC is rounded to three decimals in the results reported here. The ROC curves show the true- and false-positive rates of the algorithm for different threshold values for the anomaly score. The best threshold should be selected while accounting for the importance of reducing the false-positive rate or increasing the true-positive rate. This decision can be made by measuring the cost of errors (false positive vs. false negative).

The D-Markov and Kalman filter have an AUC close to 0.5. Their performance is almost equivalent to that of a random classifier (a classifier that selects a class randomly). The deep neural network algorithm performs worse than the KDE–AMD algorithm but clearly better than a random classifier. However, the global KDE and KDE–AMD algorithms have a reasonably good performance with an AUC greater than 0.9. In particular, KDE–AMD is extremely close to the perfect classification (achieved when AUC = 1). This simulation demonstrates the good classification capabilities of the KDE–AMD algorithm for the different anomaly types.

Algorithm	Time (s)
KDE-AMD	3.072
Global KDE	4.137
D-Markov machine	0.147
Kalman filter	0.634
Deep neural network	0.142

Table 9.4: Classification times for different algorithms.

9.5.3 Classification Times

As noted, we require an in-process classification. Given the different characteristics of each batch type (Table 9.2), the classification times can be different for each. Nevertheless, the analysis of the classification times for the same batch type can give us some insights about those for each algorithm. Table 9.4 lists the average classification time taken for a single workpiece for batch type 1. The deep neural network and D-Markov machine are the fastest models and perform in-process classification. Note that for the D-Markov machine having different symbolization techniques and number of divisions, the classification times are rather different. For example, with a symbolization of equal frequency and 35×35 divisions, the average classification time is equal to 3.014 s. The Kalman filter can also conduct the in-process classification.

As mentioned, the global KDE model is only trained with a complete set of positions of the four videos. The classification times would have been impractical if we had used the leave-one-out cross-validation. Therefore, it cannot be implemented in a real system trained with 4,222 normal videos.

The KDE–AMD algorithm is slightly slower than the Kalman filter and D-Markov machines. However, the classification times are sufficiently good to achieve in-process classification. The classification time for the KDE–AMD algorithm is measured without implementing the work parallelization, so that the system can be made faster without changing the algorithm. Furthermore, approximately 0.5 s are required to compute the Kullback–Leibler divergence, regardless of the selected parameter values for the KDE–AMD algorithm. A faster evaluation of the Kullback–Leibler divergence can be obtained by computing it with less precision.

9.6 Parameter Sensitivity Analysis

The proposed KDE–AMD algorithm requires three parameters to train the model, as outlined in Table 9.1. In this section, we show how the selection of these parameters affects the classification performance and classification times of the proposed algorithm.

We always set parameters R and C as equal, because the size of both axes is the same in the input data. This is a simplification of the parameter search space because it is clearly possible for non-equal R and C values to have better results. As noted, we test parameters R and C using the configurations 16×16, 20×20, 25×25, 30×30, 35×35, and 40×40, fixing the

$R \times C$	AUC score	Time (s)
16×16	0.979	22.416
20×20	0.977	13.048
25×25	0.969	9.307
30×30	0.975	4.328
35×35	0.979	3.098
40×40	0.974	2.771

Table 9.5: Results for different R and C with $\gamma = 5$.

γ	AUC score	Time (s)
5	0.979	3.098
10	0.980	3.098
15	0.978	3.070
20	0.981	3.110
30	0.982	3.090
40	0.975	3.069
50	0.984	3.072
200	0.983	3.026

Table 9.6: Results for different γ with $R = C = 35$.

parameter γ to 5. We choose $\gamma = 5$ because a higher value can remove too much data from the analysis. On the contrary, with a lower value, the algorithm would estimate densities with less than five instances, which is very less data. The AUC scores for all configurations are shown in Table 9.5, where the differences in the classification performance are marginal. This result demonstrates that the KDE–AMD algorithm is quite robust to the parameter values selected for R and C.

The classification times are also affected by the selection of parameters R and C. It is remarkable that classification times are reduced when the number of regions increases. This is explained by the fact that when there are more regions, the KDE model of each region contains fewer laser-spot positions. Therefore, KDE takes less time to evaluate each region. Additionally, because there are fewer laser-spot positions per region, it is more likely that a KDE model would not be assigned to some regions, because the number of laser-spot positions is below γ. This also reduces the classification time because the non-assigned regions are easily evaluated by assigning the minimum likelihood.

For the γ parameter, values 5, 10, 15, 20, 30, 40, 50, and 200 are tested by fixing R and C to 35, because this is the best configuration in Table 9.5. The results are shown in Table 9.6. Again, the differences in the classification performance are marginal. Thus, the KDE–AMD algorithm is also quite robust with respect to the γ parameter. Note that increasing the γ parameter can improve the classification performance, because the KDE models of these regions having few instances are probably produced by noise in the data. An increase in the γ parameter implies that noise-generated KDE models are not included in the KDE–AMD model. Thus, the classification performance increases. In our use case, the difference is not

important, probably because the data are not noisy enough. However, we recommend testing different values of γ for other use cases if there is noise in the data. Increasing the γ value should decrease the classification time because there are fewer regions with KDE models assigned. However, the classification times do not change significantly in the results shown in Table 9.6, because the changes in the γ parameter are not large enough to notice the difference.

9.7 Conclusion and Future Work

Anomaly detection is a key step for ensuring the production of high-quality products in industry. This chapter presents a novel methodology (i.e., KDE–AMD) for an efficient detection of anomalies in a laser surface-heating process. This process applies energy to the surface of a workpiece in a controlled manner following a known pattern containing numerous changes in direction (eight changes per cycle). It is applied at a high frequency (100 Hz and 10 fps). Pattern analysis using a high-speed thermal camera with a frame rate of 1,000 fps is complex, because the classification times must be short, even when processing numerous frames per video. The first step of the anomaly detection methodology is to detect the laser-spot positions in each frame. Then, a matrix of KDE models is built to accurately represent the changes in the position between two consecutive frames. This matrix is an ensemble of KDE models taking advantage of the temporal/spatial locality of each laser-spot movement. Using this matrix of KDE models, we compute the likelihood for each movement. An anomaly score can then be computed using the Kullback–Leibler divergence between the distribution of the training and test movement probabilities. KDE–AMD is fast enough to achieve in-process classification, and the classification results are at least as good as the other state-of-the-art algorithms.

In the future, we will improve the performance and classification times of the methodology. An alternative variant of the algorithm can entail using nearest neighbors. To this end, for a given movement $\mathbf{M} = (\mathbf{s}_O, \mathbf{s}_D)$, we would search for other movements whose points of origin are the nearest neighbors of $\mathbf{s}_O$, instead of distributing all movements in different regions during training. This variant can be computationally expensive. Thus, a fast approximation to the nearest neighbors, such as k-d trees [Bentley, 1975], is required. The parallelization and streaming-like optimizations can improve the speed of our approach. If we obtain more anomalous videos in the future, a fine-grained threshold can be computed to detect an anomaly. For example, the application of the extreme value theory [Balkema and de Haan, 1974; Pickands, 1975] would be an interesting future research direction, because it would use a small sample of real anomalies to obtain another sound anomaly score.

Chapter 10

Dynamic Semiparametric Bayesian Networks for Anomaly Detection

10.1 Introduction

In the previous chapter, we presented a methodology to detect anomalies in a laser heat-treatment process. This anomaly detector was based on detecting anomalous movements of the laser spot. Proceeding in this way, the set of images (32×32 pixels) of the camera was transformed into a set of positions (coordinates x and y). This transformation greatly reduces the dimensionality of the data to bivariate data. Therefore, a nonparametric model (KDE) could be used without worrying about the curse of dimensionality (Section 3.3.1). Then, the KDE-AMD model was proposed to take into account the temporal characteristics of the data.

However, this transformation can also mean a loss of information. In particular, information about the temperatures applied to the surface of the workpiece is lost. Then, even if the laser spot moved correctly, the laser may have not applied the correct amount of energy to the workpiece. In this chapter, we propose using SPBNs (Chapter 6) to analyze the raw image data. This approach is based on estimating the probability distribution of the image data, so anomalies are detected as low-probability workpieces. Using a SPBNs allows us to deal with the curse of dimensionality of nonparametric models as we are dealing with high-dimensional data. Note that analyzing the raw image data could also detect some anomalous movement patterns because these affect the probability distribution of the image data too.

To take into account the temporal component of the data, we propose using a dynamic semiparametric Bayesian network. The dynamic Bayesian networks (Definition 8.2) were described in Section 8.2.1.

The chapter is organized as follows. Section 10.2 describes our proposed approach to dynamic anomaly detection. Section 10.3 presents some related work on anomaly detection on laser heat-treatment processes with dynamic Bayesian networks. Section 10.4 illustrates the experiments performed to check the validity of the method. Finally, Section 10.5 concludes

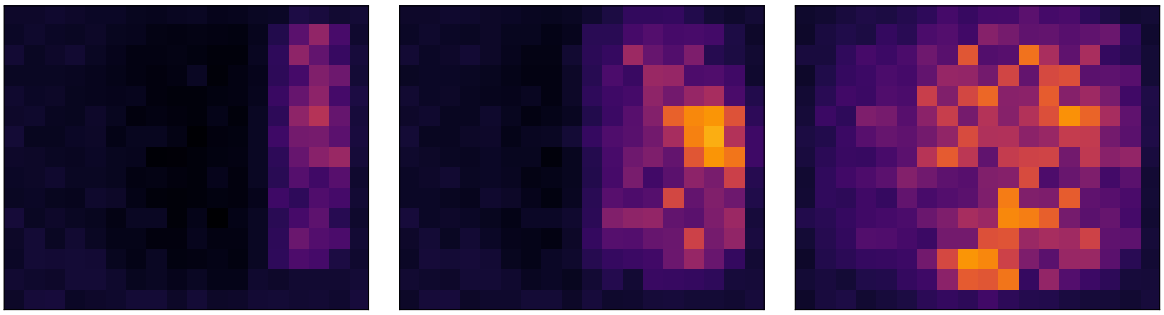


Figure 10.1: Evolution of the recorded images while the laser heats the workpiece at the beginning.

the chapter and offers some interesting future work directions.

10.2 Dynamic Semiparametric Bayesian networks

The dynamic semiparametric Bayesian networks (DPSBNs) are dynamic Bayesian networks where the initial ($\mathcal{B}_k$) and transition ($\mathcal{B}_{\to k}$) Bayesian networks are SPBNs. In this chapter, we will use k-transition conditional Bayesian networks (Section 8.2.1) as the transition Bayesian networks.

To solve the anomaly detection problem in the laser heating process, we will estimate the probability distribution of the images. Therefore, each pixel of the image will correspond to a node in a Bayesian network. As we described in Section 9.2, the laser heating was recorded using a 32×32 pixels camera. However, given the perspective of the camera, only a subset of these pixels is relevant. This is because the laser spot can be seen moving only in a smaller range. Therefore, a common area for all videos is extracted, in which the laser spot can be seen moving. For example, for the Type 16 workpieces (Table 9.2), which we will use in the experiments, the laser spot is moving in a region of 15×18 pixels. We will call these videos with restricted range, the region of interest (ROI) videos.

As we discussed in Section 9.2 some workpieces contain obstacles that force the laser to change the movement pattern. This was solved for KDE-AMDs using different temporal windows. We use a similar strategy in this chapter and we train a different DSPBN for each temporal window. Note that the laser heating process is a nonstationary (Section 8.2.1) process. This breaks the usual stationarity assumption of dynamic Bayesian networks. The use of a DPSBN for each temporal window alleviates the nonstationarity of each temporal window. Of course, this not solves the problem completely, as inside a temporal window the stationarity assumption is also broken. This is specially marked at the beginning of the videos where the workpiece is cold and gradually exhibits more temperature. This is illustrated in Figure 10.1, where the laser beam is heating the workplace and the exhibited pattern in the images is clearly different. The three images in Figure 10.1 are taken from the same temporal window.

Although the assumption of stationarity is broken, we are confident in the ability of the

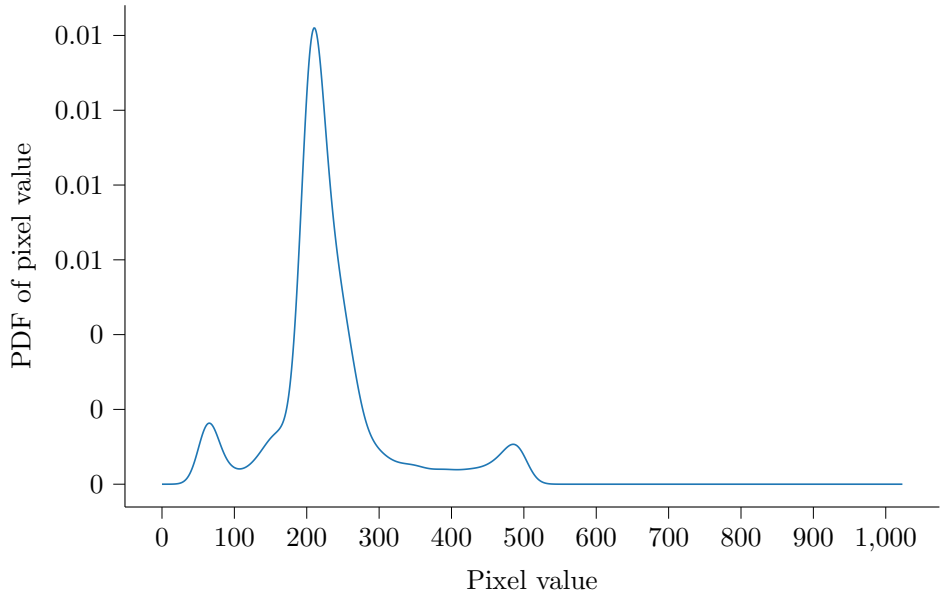


Figure 10.2: Estimate of the marginal PDF of the pixel values for one of the pixels affected by the laser spot.

nonparametric estimation models to capture the mixture of different probability distributions caused by the nonstationary process within the same temporal window. Also, note that the probability distribution of the pixels is naturally multimodal. This can be easily understood as the laser is moving in a cycle at 100 Hz, while the camera records the laser heating process at 1000Hz. This means that each cycle of the laser is recorded in 10 images. In the images where the laser spot is affecting a pixel, its value greatly increases. When the laser spot is far away from a pixel, then its value decreases. This technique was used to track the laser spot in Section 9.4.1. This also means that the marginal probability distribution of the pixels affected by the laser spot cannot be unimodal. We illustrate this in Figure 10.2 where the marginal PDF for the values of one of these pixels is estimated. The estimate shows a trimodal (clearly, not unimodal) marginal distribution. This effect discourages the use of GBNs as the (unimodal) normality assumption is not satisfied. For this reason, in this chapter, we will use DSPBNs.

10.2.1 Learning DSPBNs

The DPSBNs with k-TCBNs can be learned using an adaptation of the DMMHC of Trabelsi et al. [2013] that also allows an arbitrary k Markovian order. The DMMHC algorithm executes the MMHC algorithm [Tsamardinos et al., 2006] to compute the initial and transition Bayesian networks (Algorithm 10.1). However, the execution of the MMHC algorithm for DMMHC has some subtle differences with respect to the static MMHC version. In the initial Bayesian network $\mathcal{B}_k$, no arcs from the future to the past are allowed, i.e., arcs of the form $V^{(t_1)} \rightarrow W^{(t_2)}$ with $t_1 > t_2$. For the transition Bayesian network, an adapted version of MMHC (CMMHC, short for conditional MMHC) for conditional Bayesian networks

is executed (Algorithm 10.2). This version requires defining the set of normal nodes and interface nodes. Note that the CMMHC version contains a modified form of MMPC and HC. The modified form of MMPC (lines 1-14) differs from the state-of-the-art version because for interface nodes, the candidate parent children nodes do not contain other interface nodes (line 5 of Algorithm 10.2). This is because no arc can exist between two interface nodes. Furthermore, the HC version of MMHC forbids those operations that generate parents for the interface nodes.

Algorithm 10.1 DMMHC algorithm

Require: A set of random variables $\mathbf{X}$, Markovian order k
Ensure: Initial ($\mathcal{B}_k$) and transition ($\mathcal{B}_{\rightarrow k}$) Bayesian networks
1: $\mathcal{B}_k \leftarrow \text{MMHC}(\mathbf{X}^{(1:k)})$
2: $\mathcal{B}_{\rightarrow k} \leftarrow \text{CMMHC}(\mathbf{X}^{(t)}, \mathbf{X}^{(t-k:t-1)})$
3: **return** $\mathcal{B}_k$ and $\mathcal{B}_{\rightarrow k}$

Algorithm 10.2 CMMHC algorithm for conditional Bayesian networks

Require: A set of normal nodes $\mathbf{X}$, a set of interface nodes $\mathbf{Y}$
Ensure: The best found conditional Bayesian network $\mathcal{B}$
// CPC of normal nodes
1: **for** $G \in \mathbf{X}$ **do**
2: $\quad \text{CPC}_G \leftarrow \overline{\text{MMPC}}(G, \mathbf{X} \cup \mathbf{Y} \setminus G)$
3: **end for**
// CPC of interface nodes
4: **for** $H \in \mathbf{Y}$ **do**
5: $\quad \text{CPC}_H \leftarrow \overline{\text{MMPC}}(H, \mathbf{X})$
6: **end for**
// Apply symmetric correction
7: **for** $G \in \mathbf{X}$ **do**
8: $\quad$ **for** $H \in \text{CPC}_G$ **do**
9: $\quad\quad$ **if** $G \notin \text{CPC}_H$ **then**
10: $\quad\quad\quad \text{CPC}_G \leftarrow \text{CPC}_G \setminus \{H\}$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end for**
14: $\mathcal{G} \leftarrow$ Run an HC algorithm (Algorithm 6.1), where:
the arc addition $X \rightarrow G$ is only allowed if $X \in \text{CPC}_G$ and G is not an interface node,
the arc reversal of $X \rightarrow G$ is only allowed if X is not an interface node
15: $\mathcal{B} \leftarrow$ Train the parameters of graph $\mathcal{G}$ to construct a Bayesian network
16: **return** $\mathcal{B}$

10.2.2 Anomaly Score

The DSPBN must be learned with non-anomalous data. Then, following the common assumption for statistical anomaly detection techniques that a low probability event is an anomaly

(Section 5.3), we can define our anomaly score. The likelihood of some temporal data (Equation (8.3)) defines how probable is the time trajectory of the random variables. Thus, an appropriate anomaly score of a sequence of data is the negative mean of the log-likelihood (NMLL):

$$\text{NMLL}(\mathcal{D}) = -\frac{1}{T}\left(\log f(\mathbf{x}^{(1:k)}) + \sum_{t=k+1}^{T} \log f(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-k:t-1)})\right). \tag{10.1}$$

It is important to compute the mean instead of the sum because not all the video sequences are of the same length.

Also, recall that we learn a different DPSBN for each temporal windov. Then, the anomaly score of a workpiece must be computed with a weighted average of the NMLL score for each temporal window taking into account its length:

$$\text{AS}(\mathcal{D}) = \sum_{w=1}^{W} \frac{|\mathcal{W}_w|}{T} \text{NMLL}(\mathcal{W}_w), \tag{10.2}$$

where $\mathcal{W}_w$ is the subset of data of $\mathcal{D}$ which corresponds to temporal window w.

10.3 Related Work

An anomaly detector for the same heat-treatment process recorded with the same camera is presented in Ogbechie et al. [2017] and Larrañaga et al. [2018]. However, both works divided the image into nine clusters and then, they extracted four summary features with the pixels on each cluster (median, standard deviation, maximum and minimum of all the pixel values in a cluster). Then, a discrete dynamic Bayesian network is constructed by discretizing all the generated features. Note also, that these works use a different set of videos containing only 32 videos on which the laser spot occupies almost all the image area as opposed to the data in this chapter where we use the ROI videos.

10.4 Experiments

In this section we experimentally check the capability of the DPSBNs to detect anomalies in the laser heating process. We will use a subset of the videos in Chapter 9. In particular, we will use the videos of Type 16. This decision is motivated because this is the workpiece type that contained the real anomaly. Thus, in this section we can check if the DPSBNs can detect the anomalous video.

We will train the DPSBNs using a single training video. We will then compute the anomaly score for each of the other videos. From these anomaly scores, we can deduce if there exists a threshold that classifies the workpieces correctly using a methodology similar to that of Chapter 9. To increase the robustness of the experimentation, this process is repeated 5 times with 5 different training videos chosen randomly (ensuring that the anomalous video

Training workpiece	Temporal window	Intra arcs	Inter arcs	CKDE nodes
Training workpiece 1	Pre-obstacle	160	214	270 (100%)
	Obstacle	52	63	10 (3.70%)
	Post-obstacle	82	96	0 (0.00%)
Training workpiece 2	Pre-obstacle	157	221	270 (100%)
	Obstacle	34	16	1 (0.37%)
	Post-obstacle	113	124	16 (5.93%)
Training workpiece 3	Pre-obstacle	160	212	270 (100%)
	Obstacle	96	213	256 (94.81%)
	Post-obstacle	115	122	55 (20.37%)
Training workpiece 4	Pre-obstacle	167	238	270 (100%)
	Obstacle	55	75	21 (7.77%)
	Post-obstacle	120	129	76 (28.15%)
Training workpiece 5	Pre-obstacle	173	227	270 (100%)
	Obstacle	52	96	32 (11.85%)
	Post-obstacle	95	123	6 (2.22%)

Table 10.1: Characteristics of the trained transition Bayesian networks. The table shows the intra-time-slice arcs (Intra arcs), the inter-time-slice arcs (Inter arcs) and the number of CKDE nodes and the percentage with respect the total in parenthesis.

is not selected for training).

We executed the DMMHC algorithm of Section 10.2.1 with Markovian order 1. Further experimentation is needed to check if increasing the Markovian order improves the performance of the DPSBNs. In particular, given the characteristics of the data (a laser cycle is recorded in 10 frames), a Markovian order of 10 is a reasonable setting, and we would like to test it in the future.

The ROI of the videos in Type 16 has a size of 15×18. Therefore, the number of nodes for the initial Bayesian networks is 270, and the transition Bayesian networks contain 270 interface nodes and 270 normal nodes. The temporal windows are generated as in Chapter 9 using the information about the obstacles.

The main characteristics of the learned transition Bayesian networks are shown in Table 10.1. We do not include the information of the initial Bayesian networks, as they are used to obtain the log-likelihood of only 3 frames (the first of each temporal window), so they are less relevant. We can see that the number of arcs and CKDE nodes is very similar between different workpieces, with a few exceptions (such as the obstacle network for training workpiece 3). The pre-obstacle DPSBNs assign a CKDE for all the nodes of the network. We believe this is caused because the pre-obstacle segment is the longest one. Then, to alleviate the effects of the nonstationary process, it is better to use CKDE nodes, since they can learn different patterns using a multimodal distribution. The unique exception is the obstacle segment of the traning workpiece 3, where the number of CKDE nodes is quite high.

The anomaly scores calculated with this setting for the five training videos are summarized in Table 10.2. We can see that all the DPSBNs, except the one trained with training workpiece 4, can classify easily the real anomaly because its anomaly score is approximately

Anomaly score		Anomalous video	Non-anomalous videos
Training workpiece 1	Min.	1.327×10^5	5.679×10^2
	Mean.	1.327×10^5	6.711×10^3
	Max.	1.327×10^5	3.659×10^4
	Std.	0	6.971×10^3
Training workpiece 2	Min.	1.206×10^5	6.034×10^2
	Mean.	1.206×10^5	5.770×10^3
	Max.	1.206×10^5	2.682×10^4
	Std.	0	4.894×10^3
Training workpiece 3	Min.	2.377×10^5	4.927×10^2
	Mean.	2.377×10^5	8.586×10^3
	Max.	2.377×10^5	5.007×10^4
	Std.	0	9.219×10^3
Training workpiece 4	Min.	6.701×10^3	5.568×10^2
	Mean.	6.701×10^3	1.150×10^4
	Max.	6.701×10^3	3.021×10^4
	Std.	0	9.993×10^3
Training workpiece 5	Min.	1.120×10^5	5.449×10^2
	Mean.	1.120×10^5	7.752×10^3
	Max.	1.120×10^5	3.950×10^4
	Std.	0	8.141×10^3

Table 10.2: Statistics of anomaly score values in the anomalous and non-anomalous videos.

one order of magnitude higher than the non-anomalous videos. With respect to training workpiece 4, it is important to note that this workpiece was processed immediately after the anomalous workpiece. We believe that this had an impact on the classification results. However, understanding how the anomalous workpiece affected the video of the next workpiece requires further investigation.

In addition, we show a graphical representation of the anomaly scores for the five training videos in Figure 10.3. Each workpiece is assigned with an index that reflects the temporal ordering in which the laser heating process was performed. The green vertical bar illustrates the index of the training workpiece, and the anomalous workpiece is marked using a red marker. From these figures, we can see that the anomalous workpiece has a much larger value than the non-anomalous videos for the four DPSBNs which demonstrated good results in Table 10.2. Also, we can see that the workpieces processed before the anomalous video show a somewhat high anomaly score. The reason for these high anomaly scores is unknown, but it is an interesting line of future research because it could be related in some way to the anomalous workpiece. In that case, we could try to predict future anomalies before they occur by tracking the anomaly score of successive workpieces. In addition, there are other workpieces that exhibit a higher anomaly score for all the trained models, e.g., workpiece index 35. Finally, we can observe that the workpieces which are farther away from the training instance obtain a slightly higher anomaly score. We believe that this trend is caused by a concept drift in the data so the stationarity assumption is also broken between different

Training workpiece	Time (s)
Training workpiece 1	141.90
Training workpiece 2	143.23
Training workpiece 3	156.15
Training workpiece 4	151.75
Training workpiece 5	151.90

Table 10.3: Classification time of a test video for the five trained DPSBNs.

workpieces. This concept drift might also be related to the bad classification results of training workpiece 4. This idea is reinforced by the fact that workpieces farther apart in time are classified with too high anomaly scores for this particular trained DPSBN.

10.4.1 Classification Times

Evaluating the anomaly score of a video requires the calculation of the NMLL score for the three segments of the video. Table 10.3 shows the mean time required to calculate the anomaly score of a test video. The evaluation time for DSPBNs is highly related to the number of CKDE nodes and the number of instances contained in the CKDE models. Since the longest segment of data contained only CKDE nodes, the evaluation times increase greatly.

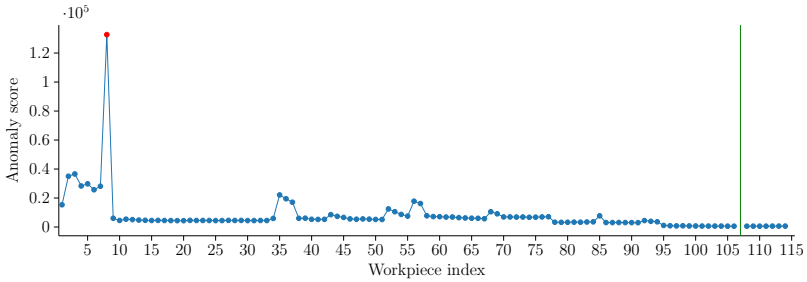
These classification times do not allow in-process classification. For this reason, an important line of future research is reducing the computational cost, particularly in the case where N (T for temporal data) is too high.

10.5 Conclusion and Future Work

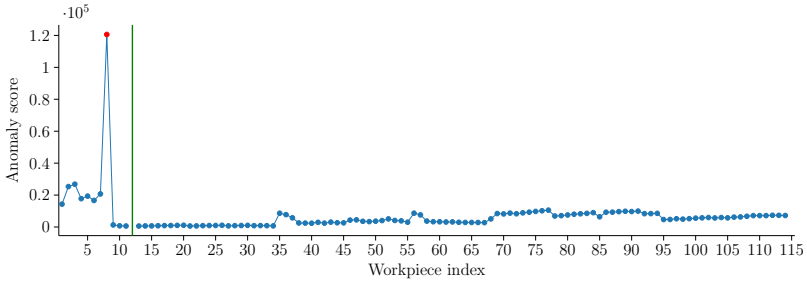
In this chapter we proposed performing anomaly detection on a laser heat-treatment recorded with thermal cameras using DSPBNs. This proposal consists of modeling the probability distribution of the video pixel values, as well as their temporal evolution taking advantage of the dynamic Bayesian network framework. To train our DPSBNs, we used a variant of DMMHC that can train k-TCBNs (Section 8.2.1). The anomaly score was defined using the log-likelihood decomposition performed with dynamic Bayesian networks.

Although the analyzed process does not satisfy the assumptions usually imposed by the dynamic Bayesian networks, i.e., being a stationary process, the proposed approach obtained positive results for 4 out of the 5 learning datasets. Our experiments also showed that the CKDE nodes appear to be an important component in dealing with the nonstationary process effects. In addition, we found a concept drift in the data that needs to be further investigated. This concept drift could be related to the worse results found on a specific training dataset.

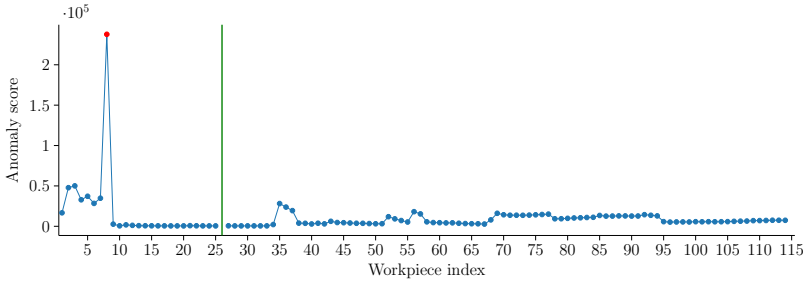
There are many lines of research to be explored in the future. First, it would be interesting to investigate better modeling of nonstationary processes using semiparametric Bayesian networks. The experimentation could also be improved testing other configurations such as a higher Markov order. A value of 10 is an interesting setting for the Markovian order knowing the dynamics of the laser. In addition, an important research line is to develop techniques



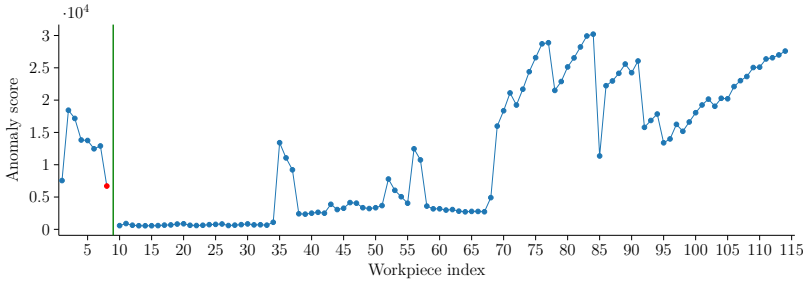
(a) Training workpiece 1



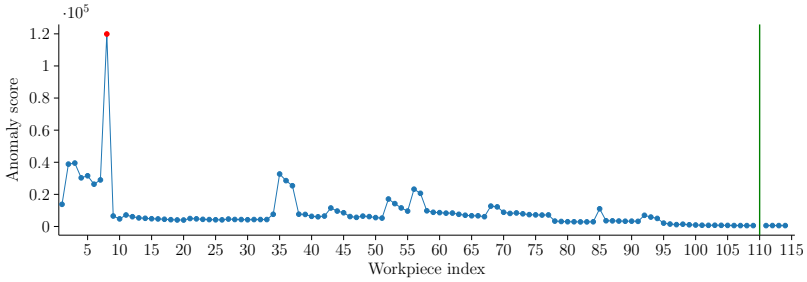
(b) Training workpiece 2



(c) Training workpiece 3



(d) Training workpiece 4



(e) Training workpiece 5

Figure 10.3: Anomaly scores for all the workpieces when the DSPBNs are trained with the workpiece marked by the green vertical line. The anomalous workpiece is shown with a red marker.

that reduce the computational cost of evaluating CKDE nodes. In previous chapters, we suggested using random Fourier features [Rahimi and Recht, 2007]. Another alternative is the research of instance subset selection techniques for KDEs. Finally, an important addition would be the combination of the laser movement (Chapter 9) and imaging (this chapter) anomaly detectors into an integrated system.

Part V

CONCLUSIONS

Chapter 11

Conclusions and Future Work

This chapter summarizes the most important contributions of this thesis. This is followed by a list of publications and submissions derived during its development. The software implemented during the research process is then listed. We conclude describing some interesting lines of research for the future.

11.1 Summary of Contributions

The contributions of this thesis have been divided into two parts:

- Part III describes our contributions to Bayesian networks. Chapter 6 presents the representation and two learning procedures of a new class of Bayesian networks called semiparametric Bayesian networks. SPBNs combine parametric and nonparametric CPDs, so the parametric assumptions can be considered only when they are satisfied. For this purpose, the form of these CPDs is described and a new score based on k-fold cross-validation is developed. The learning algorithm is able to automatically detect the best type of CPD (parametric or nonparametric) for each node. This class of Bayesian networks generalizes two common Bayesian networks in the state of the art: GBNs and KDEBNs. Therefore, if the parametric assumptions are satisfied by all nodes, the learning procedure is shown to obtain a GBN. Conversely, if the parametric assumptions are not satisfied by any node, the learning procedure will return a KDEBN. In between these two extremes, the SPBN is capable to model a plethora of models that can represent distributions in which only a subset of variables satisfy the parametric assumptions. By using the Bayesian network framework, we also attack one of the main problems of KDEs for high-dimensional data: the curse of dimensionality. In this way, the distribution can be represented with as few small-dimensional KDEs as possible.

  However, the Bayesian networks proposed in Chapter 6 are only defined to model continuous data. Chapter 7 extends this support to hybrid data by defining hybrid semiparametric Bayesian networks. This can be addressed by imposing constraints on

the graph similar to those of CLGBNs. Then, a nonparametric CPD that accepts continuous and discrete evidence is proposed. HSPBNs are shown to generalize CLGBNs, so if the parametric assumptions are satisfied for all nodes the learning algorithm may return a model equivalent to a CLGBN. Since Bayesian networks are generative models, a procedure to sample new data from nonparametric CPDs is proposed. This procedure is valid both for SPBNs and HSPBNs. In addition, we show that the hybrid nonparametric CPDs exhibit a behavior analogous to adaptive KDEs.

To support the development of the previously proposed models, we implement the open source `PyBNesian` package (Chapter 8). The package implements three different categories of Bayesian network models: Bayesian networks, conditional Bayesian networks and dynamic Bayesian networks. The transition network of the dynamic Bayesian networks are conditional Bayesian networks, since the learning complexity of this type of structures has been shown to be lower than that of state-of-the-art transition networks. Also, `PyBNesian` supports different types of Bayesian networks (defined with different CPDs) in the state of the art, parameter and structure learning, and nonparametric models. In addition, it contains several graph types to facilitate the implementation of probabilistic graphical models. Furthermore, the package is almost completely extensible, so new components can be easily created and these new components can interoperate with the state-of-the-art implemented algorithms. Finally, we show that the performance of `PyBNesian` is competitive with the current implementations of Bayesian networks.

- Part IV illustrates our contributions to anomaly detection for industrial data using nonparametric models and Bayesian networks. Chapter 9 presents the development of an anomaly detection methodology for a laser heat-treatment process. To perform this anomaly detection, first the laser spot position is extracted from a high-speed thermal camera. Then, the anomalous processes can be found detecting unusual laser spot movements. Since this is not a high-dimensional problem, only KDE models are used because the curse of dimensionality is not relevant. To improve the detection of unusual movements, a new model called KDE-Anomaly Movement Detection (KDE-AMD) is proposed. This model takes into account the spatio-temporal information in the movement of the laser to detect an anomaly score for each laser movement. Then, an anomaly score for each workpiece can be calculated, so the anomalous ones can be conveniently treated. The experimental results shows the validity of the proposed approach and its capabilities to classify in-process.

  Chapter 10 proposes using dynamic semiparametric Bayesian networks to create an anomaly detector system based on the images rather than the laser spot positions of the laser heat-treatment process also discussed in Chapter 9. For this purpose, the probability distribution of the pixel values and their evolution over time is modeled using the dynamic Bayesian network framework. The DPSBNs are learned using a variant of DMMHC. Then, the anomaly score is derived for the log-likelihood decomposition

of dynamic Bayesian networks. The experimental results shows the validity of the proposed approach and open many research lines of interest.

11.2 List of Publications

During the development of this thesis, the following publications and submissions were produced:

JCR articles:

- D. Atienza, C. Bielza, J. Diaz-Rozo, and P. Larrañaga. Efficient anomaly detection in a laser-surface heat-treatment process via laser-spot tracking. *IEEE/ASME Transactions on Mechatronics*, 26(1):405–415, 2021a, (impact factor: 5.673, Q1)

- D. Atienza, C. Bielza, and P. Larrañaga. Semiparametric Bayesian networks. *Information Sciences*, 2021b. Accepted, https://arxiv.org/abs/2109.03008

- D. Atienza, P. Larrañaga, and C. Bielza. Hybrid semiparametric Bayesian networks. *TEST*, 2021c. Submitted

Books:

- P. Larrañaga, D. Atienza, J. Diaz-Rozo, A. Ogbechie, C. Puerto-Santana, and C. Bielza. *Industrial Applications of Machine Learning*. CRC Press, 2018

Proceedings:

- D. Atienza, C. Bielza, J. Diaz, and P. Larrañaga. Anomaly detection with a spatio-temporal tracking of the laser spot. In *Proceedings of the 8th European Starting AI Researcher Symposium*, volume 284, pages 137–142, 2016

11.3 Software

The following software packages were developed to support the research proposals of this thesis:

- `PyBNesian`: An extensible Python package that implements many different Bayesian network models and nonparametric models. It is in the official Python repository (PyPi) since April 2021.

- `kde-ocl`: A simple Python package that implements KDE models with a similar interface to the implementation of the `scipy` [Virtanen et al., 2020] package but supporting GPU acceleration.

11.4 Future Work

Each contribution chapter included some possible future work or open research lines. In this section, we summarize the most interesting future work.

Chapter 6 and Chapter 7 define SPBNs and HSPBNs, respectively. For both of them, bandwidth matrix of the nonparametric CPDs plays an important role in the performance of the density estimation. It would be interesting to create bandwidth section techniques similar to those in Section 3.3.3 tailored to estimate conditional distributions. Also, it would be interesting to test new types of nonparametric CPDs inspired by the progress in density ratio estimation [Sugiyama et al., 2012]. For the HSPBNs, an extension could be proposed that allows a discrete variable to have continuous parents. Moreover, the development of a tractable inference algorithm for SPBNs and HSPBNs would be a very interesting addition. In this line, the work on nonparametric belief propagation [Sudderth et al., 2010] or variational inference [Winn and Bishop, 2005] can be helpful to accomplish tractable inference. Finally, the definition of general SPBN/HSPBN classifiers is an estimulating research line. This would extend the work of John and Langley [1995] and Pérez et al. [2009] to more general Bayesian network structures. Moroever, the definition of bandwidth selection techniques that improve the discriminative capabilities of the classifier is a research line that would complement that work.

Chapter 8 presents a Python package that implements Bayesian networks. In the future, the package could implement more learning algorithms (both for parameters and structure), and an inference engine. Also, the extension capabilities of the package can be enhanced by including support for parameter learning extensions. Moreover, we would like to submit a paper to a JCR journal presenting the package.

Chapter 9 introduces the KDE-AMD model, which generates a grid of KDE models to take into account the spatio-temporal characteristics of the data. Alternatively, this grid can be removed and could be replaced with a k-nearest neighbor implementation that finds the best possible destination points for each origin point. This must be performed for every movement, so to speed up this process a k-d tree [Bentley, 1975] can be used. Finally, if more laser heating process are found to be anomalous, a better anomaly score criterion could be found based on that information. An interesting line of research is the extreme value theory [Balkema and de Haan, 1974; Pickands, 1975] to model the probability of extreme anomaly scores.

Chapter 10 opens many research lines for the anomaly detector based on DPSBNs. From the perspective of the laser heat-treatment process it would be interesting to analyze the source of the concept drift in the data. Another important research line is to try to predict future anomalies using the anomaly scores from successive videos. From a machine learning perspective, research on modeling nonstationary processes would be important. Also, testing different Markovian orders could improve the performance of the DPSBNs. Reducing the computational cost of evaluating CKDE nodes would also be of interest, especially to learn high-dimensional Bayesian networks with many instances. Finally, an important contribution

would be the integration of the anomaly detectors proposed in Chapter 9 and Chapter 10.

Bibliography

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

D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

A. O. Adewumi and A. A. Akinyelu. A survey of machine-learning and nature-inspired based credit card fraud detection techniques. *International Journal of System Assurance Engineering and Management*, 8(2):937–953, 2017.

I. A. Ahmad and A. R. Mugdadi. Weighted Hellinger distance as an error criterion for bandwidth selection in kernel estimation. *Journal of Nonparametric Statistics*, 18(2):215–226, 2006.

M. Ahmed, A. Naser Mahmood, and J. Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016.

C. F. Aliferis, A. Statnikov, I. Tsamardinos, S. Mani, and X. D. Koutsoukos. Local causal and Markov blanket induction for causal discovery and feature selection for classification. Part I: Algorithms and empirical evaluation. *Journal of Machine Learning Research*, 11 (7):171–234, 2010.

Ankur Ankan and Abinash Panda. pgmpy: Probabilistic Graphical Models using Python. In *Proceedings of the 14th Python in Science Conference*, pages 6–11, 2015.

D. Atienza, C. Bielza, J. Diaz, and P. Larrañaga. Anomaly detection with a spatio-temporal tracking of the laser spot. In *Proceedings of the 8th European Starting AI Researcher Symposium*, volume 284, pages 137–142, 2016.

D. Atienza, C. Bielza, J. Diaz-Rozo, and P. Larrañaga. Anomaly detection with laser heat treatment thermal videos, 2019. *IEEE Dataport*. [Online]. Available: http://dx.doi.org/10.21227/7zbf-se57.

D. Atienza, C. Bielza, J. Diaz-Rozo, and P. Larrañaga. Efficient anomaly detection in a laser-surface heat-treatment process via laser-spot tracking. *IEEE/ASME Transactions on Mechatronics*, 26(1):405–415, 2021a.

D. Atienza, C. Bielza, and P. Larrañaga. Semiparametric Bayesian networks. *Information Sciences*, 2021b. Accepted, https://arxiv.org/abs/2109.03008.

D. Atienza, P. Larrañaga, and C. Bielza. Hybrid semiparametric Bayesian networks. *TEST*, 2021c. Submitted.

A. A. Balkema and L. de Haan. Residual life time at great age. *The Annals of Probability*, 2(5):792–804, 1974.

P. Baruah and R. B. Chinnam. HMMs for diagnostics and prognostics in machining processes. *International Journal of Production Research*, 43(6):1275–1293, 2005.

J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

C. Bielza and P. Larrañaga. Discrete Bayesian network classifiers: A survey. *ACM Computing Surveys*, 47(1):Article 5, 2014.

C. Bielza and P. Larrañaga. *Data-Driven Computational Neuroscience: Machine Learning and Statistical Models*. Cambridge University Press, 2020.

S. Boukabour and A. Masmoudi. Semiparametric Bayesian networks for continuous data. *Communications in Statistics - Theory and Methods*, pages 1–23, 2020.

A. Boukerche, L. Zheng, and O. Alfandi. Outlier detection: Methods, models, and classification. *ACM Computing Surveys*, 53(3):Article 55, 2020.

A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

L. Breiman, W. Meisel, and E. Purcell. Variable kernel estimates of multivariate densities. *Technometrics*, 19(2):135–144, 1977.

R. Castelo and A. Siebes. Priors on network structures. Biasing the search for Bayesian networks. *International Journal of Approximate Reasoning*, 24(1):39–57, 2000.

J. E. Chacón and T. Duong. Multivariate plug-in bandwidth selection with unconstrained pilot bandwidth matrices. *TEST*, 19(2):375–398, 2010.

J. E. Chacón and T. Duong. Efficient recursive algorithms for functionals based on higher order derivatives of the multivariate Gaussian density. *Statistics and Computing*, 25(5): 959–974, 2015.

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

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):Article 15, 2009.

G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

D. M. Chickering. A transformational characterization of equivalent Bayesian network structures. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 87–98, 1995.

D. M. Chickering. Learning equivalence classes of Bayesian-network structures. *Journal of Machine Learning Research*, 2:445–498, 2002.

F. Chollet. Keras. https://keras.io, 2015.

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

D. A. Clifton, L. A. Clifton, P. R. Bannister, and L. Tarassenko. Automated novelty detection in industrial systems. In *Advances of Computational Intelligence in Industrial Systems*, pages 269–296. 2008.

D. Codetta-Raiteri and L. Portinale. Dynamic Bayesian networks for fault detection, identification, and recovery in autonomous spacecraft. *IEEE Transactions on Systems, Man, and Cybernetics*, 45(1):13–24, 2015.

D. Colombo and M. H. Maathuis. Order-independent constraint-based causal structure learning. *Journal of Machine Learning Research*, 15:3921–3962, 2014.

G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

S. Dasgupta. Learning polytrees. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 134–141, 1999.

L. M. de Campos. A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. *Journal of Machine Learning Research*, 7(77): 2149–2187, 2006.

L. M. de Campos, J. M. Fernández-Luna, J. A. Gámez, and J. M. Puerta. Ant colony optimization for learning Bayesian networks. *International Journal of Approximate Reasoning*, 31(3):291–311, 2002.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

M. J. Desforges, P. J. Jacob, and J. E. Cooper. Applications of probability density estimation to the detection of abnormal conditions in engineering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 212(8):687–703, 1998.

L. Devroye. The equivalence of weak, strong and complete convergence in L_1 for kernel density estimates. *The Annals of Statistics*, 11(3):896–904, 1983.

L. Devroye and C. S. Penrod. The consistency of automatic kernel density estimates. *The Annals of Statistics*, 12(4):1231–1249, 1984.

D. Dor and M. Tarsi. A simple algorithm to construct a consistent extension of a partially oriented graph. Technical Report R-185, UCLA, 1992.

R. Drath and A. Horch. Industrie 4.0: Hit or hype? *IEEE Industrial Electronics Magazine*, 8(2):56–58, 2014.

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

S. J. Eglen, B. Marwick, Y. O. Halchenko, M. Hanke, S. Sufi, P. Gleeson, R. A. Silver, A. P. Davison, L. Lanyon, M. Abrams, T. Wachtler, D. J. Willshaw, C. Pouzat, and J.-B. Poline. Toward standard practices for sharing computer code and programs in neuroscience. *Nature neuroscience*, 20(6):770–773, 2017.

V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability & Its Applications*, 14(1):153–158, 1969.

E. Eskin. Anomaly detection over noisy data using learned probability distributions. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 255–262, 2000.

T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

T. Fernando, H. Gammulle, S. Denman, S. Sridharan, and C. Fookes. Deep learning for medical anomaly detection – A survey. *ACM Computing Surveys*, 54(7):Article 141, 2021.

R. A. Fisher. Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika*, 10(4):507–521, 1915.

R. A. Fisher. On the probable error of a coefficient of correlation deduced from a small sample. *Metron*, (1):3–32, 1921.

J. Fox. *Applied Regression Analysis, Linear Models, and Related Methods*. SAGE Publications, 1997.

N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, pages 252–262, 1996.

N. Friedman and I. Nachman. Gaussian process networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 211–219, 2000.

A. Gabilondo, J. Domínguez, C. Soriano, and J. L. Ocaña. Method and system for laser hardening of a surface of a workpiece, 2015. US20150211083A1 patent.

X. Gao, D. You, and S. Katayama. Seam tracking monitoring based on adaptive Kalman filter embedded Elman neural network during high-power fiber laser welding. *IEEE Transactions on Industrial Electronics*, 59(11):4315–4325, 2012.

S. García and F. Herrera. An extension on "Statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694, 2008.

M. Gasse, A. Aussem, and H. Elghazel. An experimental comparison of hybrid algorithms for Bayesian network structure learning. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 58–73, 2012.

D. Geiger and D. Heckerman. Learning Gaussian networks. In *Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence*, pages 235–243, 1994.

D. Geiger and D. Heckerman. Parameter priors for directed acyclic graphical models and the characterization of several probability distributions. *The Annals of Statistics*, 30(5): 1412–1440, 2002.

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

F. Glover and M. Laguna. *Tabu Search*. John Wiley & Sons, 1993.

M. Goldstein and A. Dengel. Histogram-based outlier score (HBOS): A fast unsupervised anomaly detection algorithm. In *Proceedings of 35th German Conference on Artificial Intelligence*, pages 59–63, 2012.

R. Gonzalez, B. Huang, and E. Lau. Process monitoring using kernel density estimation and Bayesian networking with an industrial case study. *ISA Transactions*, 58:330–347, 2015.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, volume 2, pages 2672–2680, 2014.

F. E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11 (1):1–21, 1969.

S. Gupta and A. Ray. Real-time fatigue life estimation in mechanical structures. *Measurement Science and Technology*, 18(7):1947–1957, 2007.

P. Hall. On Kullback-Leibler loss and density estimation. *The Annals of Statistics*, 15(4): 1491–1519, 1987.

P. Hall and J. Marron. Estimation of integrated squared density derivatives. *Statistics & Probability Letters*, 6(2):109–115, 1987.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009.

A. Hauser and P. Bühlmann. Characterization and greedy learning of interventional Markov equivalence classes of directed acyclic graphs. *Journal of Machine Learning Research*, 13: 2409–2464, 2012.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann Publishers, 2008.

V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

R. Hofmann and V. Tresp. Discovering structure in continuous variables using Bayesian networks. In *Proceedings of Advances in Neural Information Processing Systems 8*, pages 500–506, 1995.

J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* The MIT Press, 1992.

W. Hu, J. Gao, B. Li, O. Wu, J. Du, and S. Maybank. Anomaly detection using local kernel density estimation and context-based regression. *IEEE Transactions on Knowledge and Data Engineering*, 32(2):218–233, 2020.

K. Ickstadt, B. Bornkamp, M. Grzegorczyk, J. Wieczorek, M. Rahuman Sheriff, H. E. Grecco, and E. Zamir. Nonparametric Bayesian networks. In *Bayesian Statistics 9*, pages 1–40, 2012.

I. Inza, P. Larrañaga, and B. Sierra. Feature subset selection by Bayesian networks: A comparison with genetic and sequential algorithms. *International Journal of Approximate Reasoning*, 27(2):143–164, 2001.

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

G. H. John and P. Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.

M. C. Jones and R. F. Kappenman. On a class of kernel density estimate bandwidth selectors. *Scandinavian Journal of Statistics*, 19(4):337–349, 1992.

M. C. Jones, J. S. Marron, and S. J. Sheather. A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association*, 91(433):401–407, 1996.

M. Jäger and F. A. Hamprecht. Principal component imagery for the quality monitoring of dynamic laser welding processes. *IEEE Transactions on Industrial Electronics*, 56(4): 1307–1313, 2009.

M. Jäger, S. Humbert, and F. A. Hamprecht. Sputter tracking for the automatic monitoring of industrial laser-welding processes. *IEEE Transactions on Industrial Electronics*, 55(5): 2177–2184, 2008a.

M. Jäger, C. Knoll, and F. A. Hamprecht. Weakly supervised learning of a classifier for unusual event detection. *IEEE Transactions on Image Processing*, 17(9):1700–1708, 2008b.

M. Kalisch, M. Mächler, D. Colombo, M. H. Maathuis, and P. Bühlmann. Causal inference using graphical models with the R package pcalg. *Journal of Statistical Software*, 47(11): 1–26, 2012.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao. Packetscore: A statistics-based packet filtering scheme against distributed denial-of-service attacks. *IEEE Transactions on Dependable and Secure Computing*, 3(2):141–155, 2006.

A. Kind, M. P. Stoecklin, and X. Dimitropoulos. Histogram-based traffic anomaly detection. *IEEE Transactions on Network and Service Management*, 6(2):110–121, 2009.

S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. In *Readings in Computer Vision*, pages 606–615. 1987.

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

S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31(3):249–268, 2007.

J. Kuipers, G. Moffa, and D. Heckerman. Addendum on the scoring of Gaussian directed acyclic graphical models. *The Annals of Statistics*, 42(4):1689–1691, 2014.

S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

H. Langseth, T. D. Nielsen, R. Rumí, and A. Salmerón. Mixtures of truncated basis functions. *International Journal of Approximate Reasoning*, 53(2):212–227, 2012.

P. Larrañaga, H. Karshenas, C. Bielza, and R. Santana. A review on evolutionary algorithms in Bayesian network learning and inference tasks. *Information Sciences*, 233:109–125, 2013.

P. Larrañaga, D. Atienza, J. Diaz-Rozo, A. Ogbechie, C. Puerto-Santana, and C. Bielza. *Industrial Applications of Machine Learning*. CRC Press, 2018.

S. L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, 17(1):31–57, 1989.

U. Lerner, E. Segal, and D. Koller. Exact inference in networks with discrete children of continuous parents. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 319–328, 2001.

J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu. Feature selection: A data perspective. *ACM Computing Surveys*, 50(6):Article 94, 2017.

R. Linares, G. Vergara, R. Gutiérrez, C. Fernández, V. Villamayor, L. Gómez, M. González-Camino, A. Baldasano, G. Castro, R. Arias, Y. Lapido, J. Rodríguez, and P. Romero. Laser beam welding quality monitoring system based in high-speed (10 kHz) uncooled MWIR imaging sensors. In *Proceedings of SPIE, Thermosense: Thermal Infrared Applications XXXVII*, volume 9485, pages 289–295, 2015.

D. O. Loftsgaarden and C. P. Quesenberry. A nonparametric estimate of a multivariate density function. *The Annals of Mathematical Statistics*, 36(3):1049–1051, 1965.

D. Margaritis. *Learning Bayesian Network Model Structure from Data*. PhD thesis, School of Computer Science, Carnegie Mellon University, 2003.

D. Margaritis and S. Thrun. Bayesian network induction via local neighborhoods. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pages 505–511, 1999.

M. Markou and S. Singh. Novelty detection: A review - Part 1: statistical approaches. *Signal Processing*, 83(12):2481–2497, 2003a.

M. Markou and S. Singh. Novelty detection: A review - Part 2: neural network based approaches. *Signal Processing*, 83(12):2499–2521, 2003b.

J. S. Marron and A. B. Tsybakov. Visual error criteria for qualitative smoothing. *Journal of the American Statistical Association*, 90(430):499–507, 1995.

L. Martí, N. Sanchez-Pi, J. M. Molina, and A. C. B. Garcia. Anomaly detection based on sensor data in petroleum industry applications. *Sensors*, 15(2):2774–2797, 2015.

S. Mascaro, A. E. Nicholso, and K. B. Korb. Anomaly detection in vessel tracks using Bayesian networks. *International Journal of Approximate Reasoning*, 55(1):84–98, 2014.

K. Masmoudi and A. Masmoudi. A new class of continuous Bayesian networks. *International Journal of Approximate Reasoning*, 109:125–138, 2019.

C. Meek. Causal inference and causal explanation with background knowledge. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 403–410, 1995.

S. Moral, R. Rumí, and A. Salmerón. Mixtures of truncated exponentials in hybrid Bayesian networks. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 156–167, 2001.

J. Mori and J. Yu. Dynamic Bayesian network based networked process monitoring for fault propagation identification and root cause diagnosis of complex dynamic processes. *IFAC Proceedings Volumes*, 46(32):678–683, 2013.

N. Moustafa, J. Hu, and J. Slay. A holistic review of network anomaly detection systems: A comprehensive survey. *Journal of Network and Computer Applications*, 128:33–55, 2019.

E. A. Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1): 141–142, 1964.

E. A. Nadaraya. On non-parametric estimates of density functions and regression curves. *Theory of Probability & Its Applications*, 10(1):186–190, 1965.

P. Nectoux, R. Gouriveau, K. Medjaher, E. Ramasso, B. Chebel-Morello, N. Zerhouni, and C. Varnie. PRONOSTIA: An experimental platform for bearings accelerated life test. In *IEEE Conference on Prognostics and Health Management*, pages 1–8, 2012.

J. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.

A. Ogbechie, J. Díaz-Rozo, P. Larrañaga, and C. Bielza. Dynamic Bayesian network-based anomaly detection for in-process visual inspection of laser surface heat treatment. In *Machine Learning for Cyber Physical Systems*, pages 17–24, 2017.

P. Ostwal. Lgnpy: v1.0.0, 2020. URL https://zenodo.org/record/3902122.

G. Pang, C. Shen, L. Cao, and A. V. D. Hengel. Deep learning for anomaly detection: A review. *ACM Computing Surveys*, 54(2):Article 38, 2021.

E. Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

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

K. Pearson. Contributions to the mathematical theory of evolution.–ii. Skew variation in homogeneous material. *Philosophical Transactions of the Royal Society of London A*, 186: 343–414, 1895.

A. Pérez, P. Larrañaga, and I. Inza. Bayesian classifiers based on kernel density estimation: Flexible classifiers. *International Journal of Approximate Reasoning*, 50(2):341–362, 2009.

D. T. Pham and G. A. Ruz. Unsupervised training of Bayesian networks for data clustering. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 465 (2109):2927–2948, 2009.

J. Pickands. Statistical inference using extreme order statistics. *The Annals of Statistics*, 3 (1):119–131, 1975.

M. A. F. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

L. Prechelt. Early stopping — but when? In *Neural Networks: Tricks of the Trade*, pages 53–67, 2012.

L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE Acoustics, Speech and Signal Processing Magazine*, 3(1):4–16, 1986.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pages 1177–1184, 2007.

M. M. Rahman and Z. Govindarajulu. A modification of the test of Shapiro and Wilk for normality. *Journal of Applied Statistics*, 24(2):219–236, 1997.

R. B. Randall and J. Antoni. Rolling element bearing diagnostics – A tutorial. *Mechanical Systems and Signal Processing*, 25(2):485–520, 2011.

C. Rao, A. Ray, S. Sarkar, and M. Yasar. Review and comparative evaluation of symbolic dynamic filtering for detection of anomaly patterns. *Signal, Image and Video Processing*, 3(2):101–114, 2009.

A. Ray. Symbolic dynamic analysis of complex systems for anomaly detection. *Signal Processing*, 84(7):1115–1130, 2004.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

R. W. Robinson. Counting unlabeled acyclic digraphs. In *Combinatorial Mathematics V*, pages 28–43, 1977.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.

M. Rudemo. Empirical choice of histograms and kernel density estimators. *Scandinavian Journal of Statistics*, 9(2):65–78, 1982.

L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich, and K.-R. Müller. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 109(5):756–795, 2021.

J. Runge. Conditional independence testing based on a nearest-neighbor estimator of conditional mutual information. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, volume 84, pages 938–947, 2018.

J. V. Ryzin. On strong consistency of density estimates. *The Annals of Mathematical Statistics*, 40(5):1765–1772, 1969.

S. R. Sain, K. A. Baggerly, and D. W. Scott. Cross-validation of multivariate densities. *Journal of the American Statistical Association*, 89(427):807–817, 1994.

S. Sarkar, S. Sarkar, N. Virani, A. Ray, and M. Yasar. Sensor fusion for fault detection and classification in distributed physical processes. *Frontiers in Robotics and AI*, 1(16):1–9, 2014.

J. Schreiber. Pomegranate: Fast and flexible probabilistic modeling in Python. *Journal of Machine Learning Research*, 18(164):1–6, 2018.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

D. W. Scott. On optimal and data-based histograms. *Biometrika*, 66(3):605–610, 1979.

D. W. Scott. Frequency polygons: Theory and application. *Journal of the American Statistical Association*, 80(390):348–354, 1985.

D. W. Scott. A note on choice of bivariate histogram bin shape. *Journal of Official Statistics*, 4(1):47–51, 1988.

D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley, second edition, 2015.

D. W. Scott and G. R. Terrell. Biased and unbiased cross-validation in density estimation. *Journal of the American Statistical Association*, 82(400):1131–1146, 1987.

M. Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010.

M. Scutari, C. E. Graafland, and J. M. Gutiérrez. Who learns better Bayesian network structures: Accuracy and speed of structure learning algorithms. *International Journal of Approximate Reasoning*, 115:235–253, 2019.

R. D. Shachter and C. R. Kenley. Gaussian influence diagrams. *Management Science*, 35(5): 527–550, 1989.

L. Shamir, J. F. Wallin, A. Allen, B. Berriman, P. Teuben, R. J. Nemiroff, J. Mink, R. J. Hanisch, and K. DuPrie. Practices in source code sharing in astrophysics. *Astronomy and Computing*, 1:54–58, 2013.

P. P. Shenoy and J. C. West. Inference in hybrid Bayesian networks using mixtures of polynomials. *International Journal of Approximate Reasoning*, 52(5):641–657, 2011.

B. W. Silverman. Weak and strong uniform consistency of the kernel estimate of a density and its derivatives. *The Annals of Statistics*, 6(1):177–184, 1978.

B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall/CRC, 1986.

A. Smiti. A critical overview of outlier detection methods. *Computer Science Review*, 38: 100306, 2020.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. The MIT Press, 2000.

J. E. Stone, D. Gohara, and G. Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science Engineering*, 12(3):66–73, 2010.

E. V. Strobl, K. Zhang, and S. Visweswaran. Approximate kernel-based conditional independence tests for fast non-parametric causal discovery. *Journal of Causal Inference*, 7(1): 1–24, 2019.

H. A. Sturges. The choice of a class interval. *Journal of the American Statistical Association*, 21(153):65–66, 1926.

E. B. Sudderth, A. T. Ihler, M. Isard, W. T. Freeman, and A. S. Willsky. Nonparametric belief propagation. *Communications of the ACM*, 53(10):95–103, 2010.

M. Sugiyama, T. Suzuki, and T. Kanamori. *Density Ratio Estimation in Machine Learning*. Cambridge University Press, 2012.

B. Thiesson, C. Meek, D. M. Chickering, and D. Heckerman. Learning mixtures of DAG models. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 504–513, 1998.

D. A. Tobon-Mejia, K. Medjaher, and N. Zerhouni. CNC machine tool's wear diagnostic and prognostic by using dynamic Bayesian networks. *Mechanical Systems and Signal Processing*, 28:167–182, 2012.

G. Trabelsi. *New Structure Learning Algorithms and Evaluation Methods for Large Dynamic Bayesian Networks*. PhD thesis, Université de Nantes, 2013.

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

I. Tsamardinos, C. F. Aliferis, and A. Statnikov. Algorithms for large scale Markov blanket discovery. In *Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference*, pages 376–381, 2003a.

I. Tsamardinos, C. F. Aliferis, and A. Statnikov. Time and sample efficient discovery of Markov blankets and direct causal relations. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 673–678, 2003b.

I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

A. B. Tsybakov. *Introduction to Nonparametric Estimation*. Springer, 2008.

F. van Wyk, Y. Wang, A. Khojandi, and N. Masoud. Real-time sensor anomaly detection and identification in automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1264–1276, 2020.

P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

M. P. Wand. Error analysis for general multivariate kernel estimators. *Journal of Nonparametric Statistics*, 2(1):1–15, 1992.

M. P. Wand and M. C. Jones. Comparison of smoothing parameterizations in bivariate kernel density estimation. *Journal of the American Statistical Association*, 88(422):520–528, 1993.

M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman and Hall/CRC, 1994.

G. S. Watson. Smooth regression analysis. *Sankhya: The Indian Journal of Statistics, Series A (1961-2002)*, 26(4):359–372, 1964.

D. Wied and R. Weißbach. Consistency of the kernel density estimator: A survey. *Statistical Papers*, 53(1):1–21, 2012.

J. Winn and C. M. Bishop. Variational message passing. *Journal of Machine Learning Research*, 6(23):661–694, 2005.

D. Xu and Y. Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

N. Ye and Q. Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International*, 17(2):105–112, 2001.

D. Yeung and C. Chow. Parzen-window network intrusion detectors. In *Proceedings of the 16th International Conference on Pattern Recognition, 2002*, volume 4, pages 385–388, 2002.

S. Yin and X. Zhu. Intelligent particle filter and its application to fault detection of nonlinear system. *IEEE Transactions on Industrial Electronics*, 62(6):3852–3861, 2015.

B. Zhang, C. Sconyers, C. Byington, R. Patrick, M. E. Orchard, and G. Vachtsevanos. A probabilistic fault detection approach: Application to bearing fault detection. *IEEE Transactions on Industrial Electronics*, 58(5):2011–2018, 2011.

L. Zhang, J. Lin, and R. Karim. Adaptive kernel density-based anomaly detection for nonlinear systems. *Knowledge-Based Systems*, 139:50–63, 2018.

C. Zhou and R. C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–674, 2017.

S. Zhou, J. Zhang, and S. Wang. Fault diagnosis in industrial processes using principal component analysis and hidden Markov model. In *Proceedings of the 2004 American Control Conference*, volume 6, pages 5680–5685, 2004.