Online Density Estimates: A Probabilistic Condensed Representation of Data for Knowledge Discovery

A thesis presented for the degree of

Doktor der Naturwissenschaften

at the Department of Physics, Mathematics, and Computer Science at the Johannes Gutenberg University in Mainz

Michael Geilke

born in Kiel

Mainz, November 22, 2017

Mündliche Prüfung: 22.11.2017

Acknowledgements

The acknowledgements have been removed from the electronic version.

Abstract

The *Internet of Things* (IoT) and the data that is generated from its sensors are making new demands on data mining methods. These demands stem from the desire to benefit from the knowledge contained in this data and the increasing number of devices that are equipped with these sensors. According to companies like Intel or HP, the number of sensors worldwide is likely to reach more than one trillion by 2022. All of them will produce streams of measurements and leveraging knowledge from these streams requires infrastructure to analyze them in real-time. From a data mining perspective, this involves challenging tasks such as cleaning the data, handling large amounts of data, and preserving their privacy, to name a few.

The state of the art in data mining already addressed some of these challenges, but the proposed methods are typically designed for a specific task (e.g., predicting a certain variable or finding frequent patterns) and perform this task while scanning the data stream. However, at the time of collecting the data, it is often not known what kind of analysis needs to be performed or there are several – possibly even dependent – analysis tasks. This means that whenever storing the original data is either not feasible due to the sheer volume or impossible due to privacy concerns, the user has to wait for more data to initiate another analysis task, which impedes the use of conventional data mining algorithms. Therefore, we present a framework in this thesis, called *MiDEO* (Mining Density Estimates inferred Online), which decouples the process of collecting the data from the actual analysis. It uses density estimates to maintain a compact representation of the data stream and provides inference capabilities to perform queries on them. The queries can be combined to complex data mining tasks and allow to adapt the estimates to the current needs of the user or the algorithm. Compared to current methods that typically focus on one task at a time, this enables a more interactive analysis of the data stream, where the task selection is part of the analysis.

In the course of designing such a framework, we develop several methods to improve the state of the art. This includes online density estimators for conditional joint densities with mixed types of variables, an online density estimator for high-dimensional data, algorithms to perform pattern mining on online density estimates, an online density estimator that is able to represent recurrences in the data stream, and algorithms that enforce well-known privacy-preserving properties to protect the entities described by the data. To show the effectiveness of these methods, we prove some of their theoretical properties and perform an extensive set of experiments.

Publications

Parts of this thesis have been published on international data mining and machine learning conferences and journals:

- Michael Geilke, Andreas Karwath, Eibe Frank, and Stefan Kramer. Online Estimation of Discrete, Continuous, and Conditional Joint Densities using Classifier Chains. In: *Data Mining and Knowledge Discovery*, pages 1-43, Springer 2017.
  doi:10.1007/s10618-017-0546-6

- Michael Geilke and Stefan Kramer. Privacy-Preserving Pattern Mining on Online Density Estimates In: *Proceedings of the International Conference on Big Knowledge (ICBK 2017)*, pages 25-32, IEEE 2017.
  doi:10.1109/ICBK.2017.32

- Michael Geilke, Andreas Karwath, and Stefan Kramer. Online Density Estimation of Heterogeneous Data Streams in Higher Dimensions. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD 2016)*, pages 65-80, Springer 2016.
  doi:10.1007/978-3-319-46128-1_5

- Michael Geilke, Andreas Karwath, and Stefan Kramer. Modeling Recurrent Distributions in Streams using Possible Worlds. In: *Proceedings of the International Conference on Data Science and Advanced Analytics 2015 (DSAA 2015)*, pages 1-9, IEEE 2015.
  doi:10.1109/DSAA.2015.7344814

- Michael Geilke, Andreas Karwath, and Stefan Kramer. A Probabilistic Condensed Representation of Data for Stream Mining. In: *Proceedings of the International Conference on Data Science and Advanced Analytics 2014 (DSAA 2014)*, pages 297-303, IEEE 2014.
  doi:10.1109/DSAA.2014.7058088

- Michael Geilke, Andreas Karwath, Eibe Frank, and Stefan Kramer. Online Estimation of Discrete Densities. In: *Proceedings of the 13th IEEE International Conference on Data Mining (ICDM 2013)*, pp. 191-200, IEEE 2013.
  doi:10.1109/ICDM.2013.91

Contents

Chapter 1

Introduction

The *Internet of Things* (IoT) and the data that is generated from its sensors are making new demands on data mining methods. These demands stem from the desire to benefit from the knowledge contained in this data and the increasing number of devices that are equipped with these sensors.[1] According to companies like Intel or HP, the number of sensors is likely to reach more than one trillion by 2022. All of them will produce streams of measurements[2] and leveraging knowledge from these streams requires infrastructure to analyze them in real-time.[3] From a data mining perspective, this involves challenging tasks such as cleaning the data, handling large amounts of data, and preserving their privacy, to name a few (Chen et al., 2015; Galushka et al., 2006).

The state of the art in data mining already addressed some of these challenges, but the proposed methods are typically designed for a specific task (e.g., predicting a certain variable or finding frequent patterns) and perform this task while scanning the data stream (see Figure 1.1 for an illustration). However, at the time of collecting the data, it is often not known what kind of analysis needs to be performed or there are several – possibly even dependent – analysis tasks. This means that whenever storing the original data is either not feasible due to the sheer volume or impossible due to privacy concerns, the user has to wait for more data to initiate another analysis task, which impedes the use of conventional data mining algorithms. Therefore, we present a framework in this thesis, called *MiDEO* (Mining Density Estimates inferred Online), which decouples the process of collecting the data from the actual analysis (see Figure 1.2 for an illustration). It uses density estimates to maintain a compact representation of the data stream and provides inference capabilities to perform queries on them. The queries can be combined to complex data mining tasks and allow to adapt the estimates to the current needs of the user or the algorithm. Compared to current methods that typically focus on one task at a time, this enables a more interactive analysis of the data stream, where the task selection is part of the analysis.

Besides being able to process and represent large volumes of data, MiDEO also contributes to privacy preservation to a certain degree, since it provides a statistical representation that generally "disregards" information associated with individual per-

[1] `http://dataconomy.com/how-sensors-will-shape-big-data-and-the-changing-economy/`
[2] `http://www.tsensorssummit.org/Resources/Why%20TSensors%20Roadmap.pdf`
[3] `https://www.wired.com/insights/2014/11/the-internet-of-things-bigger/`

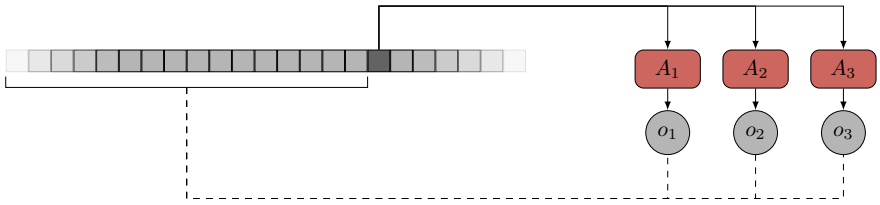


Figure 1.1: In traditional data mining, one would pick an algorithm for each data mining task (in this case, A_1, A_2, A_3). The gray boxes are the instances of the data stream from left to right, i.e., instances to the left are older. Output o_i is the result of processing all observed instances observed by algorithm A_i, $1 \leq i \leq 3$. The instance that is marked dark gray is currently processed by each of the algorithms.

sons. As this level of privacy preservation is limited and cannot be guaranteed, we also propose algorithms that enforce well-known privacy-preserving properties.

1.1 Motivation and Potential Applications

The methods proposed in this thesis are motivated by real-world applications from various domains generating large volumes of data – often in real-time. In the following, these applications will serve as examples to point out potential problems, offer a motivation for certain aspects of the presented methods, and guide future work. We briefly present these applications in this section.

1.1.1 Smart Homes

Smart homes are equipped with many sensors measuring various parameters of the house (e.g., temperature or humidity). By learning from past measurements, data mining algorithms have the possibility to identify patterns in the data, to use them to distinguish between typical and abnormal behavior, and to suggest appropriate actions to the user. For example, whereas an increase of the humidity in the basement can be explained by a tumble dryer that has been started some time ago, such an increase in the bed room could be due to water entering the room through an open window. The former situation is probably quite normal for a household and requires no action, but the latter situation needs attention from the user. A density estimate of the sensor measurements that provides facilities to analyze the data in hindsight can be useful to develop such applications.

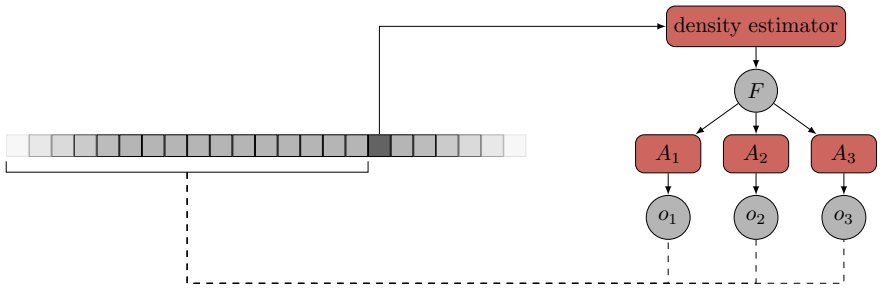


Figure 1.2: MiDEO performs data mining in two steps: given a data stream, it first maintains an estimate of the joint density of the data stream and then performs data mining tasks on this estimate. In this illustration, *density estimator* is an online density estimator that produces a density estimate F, which is potentially modified to meet certain privacy-preserving properties and extended by inference capabilities. The latter allows to restrict the estimate horizontally or vertically, i.e., selecting only a subset of the data or removing random variables. The inference operation can be combined to complex data mining task (in this case, A_1, A_2, A_3).

1.1.2 Google Flow

Together with the U.S. Department of Transportation, Google's Sidewalk Labs is working on a platform, called *Flow*, that aims at analyzing transit information in real-time.[4] More precisely, they want to assist people in finding the best routes by aggregating and analyzing data from several sources, i.e., smartphones, Waze[5], municipal data, and, in the long term, traffic sensors. Developing such a platform on a broader scale is a challenging endeavor that either requires very strong hardware or a focus on a selection of the data to keep it small enough. With a framework such as MiDEO, the data could be stored in a more compact way without loosing valuable information about historical data. For example, if a certain event only occurs at Christmas eves falling on a Saturday, data needs to be aggregated over several years to find such a pattern. Without a compact representation, one would have to store tremendous volumes of data.

1.1.3 Patient Data

Although the MiDEO framework is specifically designed for representing large volumes of data in a compact way, it can also make sense to use it for small datasets. If the knowledge contained in a dataset needs to be shared, but the data itself has to remain private, online density estimates can provide an effective way of sharing this knowledge without sharing the data. For example, a hospital wants to participate in a large study

[4] `http://www.citylab.com/commute/2016/03/google-sidewalk-labs-flow-smart-cities/474267/`

[5] `https://www.waze.com/`

to get further insights into a certain disease. For reasons of medical confidentiality, it cannot share the data itself but is allowed to build a density estimate, to modify it to meet certain privacy concerns (possibly enforced by law), and to send the modified version to the group conducting the study. This way, sensitive information about individual entities could be easily protected, while everyone can still benefit from the extracted knowledge. For legal reasons and to gain the trust of the data holder, however, one needs a framework with theoretical guarantees – something which a framework such as MiDEO could offer.

1.2 Contributions

We are contributing to the data mining community in several ways. The methods proposed in this thesis not only enhance the state of the art for existing problems but also enable novel use cases that have not been considered before.

1.2.1 Online Density Estimation

One of the most important contributions – and also the cornerstone of many methods proposed in this thesis – are the online density estimators using classifier chains. They use the product rule to split a joint density defined over several variables into several conditional densities with only one target variable. Each of these conditional densities is then estimated by online classifiers acting as univariate density estimators. This way, the estimate can be updated on an instance-by-instance basis and also enables the application of a broad range of fast and efficient classifiers. Which classifiers are selected depends on the type of the variables. For densities with discrete target variables, we use Hoeffding trees to estimate the probability mass of each variable value. For continuous target variables, we extend a method designed for conditional density estimation (Frank and Bouckaert, 2009) to the online setting. In the end, we obtain a framework for online density estimators, called *EDO* (Estimating Densities Online), that is able to handle (conditional) joint densities with discrete variables, continuous variables, and mixtures thereof. In particular, we propose three variants of this scheme including a single random classifier chain (CCs), an ensembles of classifier chains (ECCs), and an ensemble of weighted classifier chains (EWCCs). Although classifier chains and ensembles of classifier chains have already been used in the multi-label setting, it is the first time that they are applied to the world of unsupervised online learning.

The effectiveness of EDO is demonstrated in two ways: We prove that EDO estimates are consistent as long as the univariate density estimators are and illustrate the strengths and limitations of the methods with an extensive set of experiments. The latter includes a comparison with 12 Bayesian structure learner and the state-of-the-art kernel density estimator on a broad range of synthetic and real-world datasets. It shows that EDO is performing equally well or better on most datasets, while providing additional benefits such as native support for inference capabilities.

However, we will also show that EDO is not applicable to every kind of data stream. To produce accurate estimates, it makes a few assumptions that limits is applicability: (1) There is no support for data streams that are not independently and identically distributed (IID). (2) As long as the density has only a few variables, EDO provides an accurate description of the data. But as soon as the dimensionality increases, the number of classifiers and their size grows quickly – making this approach unsuitable for data of high dimensionality. (3) The stream needs to be generated from a single stationary distribution. Although EDO estimates are able to deal with data distribution drift in principle, they are only designed to capture the current data distribution of the stream. This way, the history of the data stream is lost, which makes a historical analysis of the data impossible.

Limitation (1) is analyzed only briefly (see Section 4.6.3), as a more detailed analysis is beyond the scope of this thesis. To address Limitation (2), we propose another density estimator, called *RED* (Representative-based online Estimation of Densities), that builds on EDO and estimates the joint density of heterogeneous data streams with many variables. For this purpose, it projects the original data stream into a lower dimensional vector space and uses a set of representatives to provide an estimate. Due to the structure of the estimates, it enables the density estimation of higher-dimensional data and approaches the true density with increasing dimensionality of the vector space. Finally, to support the historical analysis of the data (Limitation (3)), we design a density estimator, called *REC* (RECurrent densities estimated online) that captures the possibly recurrent data distributions of the stream using one EDO estimate per data distribution.

1.2.2 Probabilistic Condensed Representation

With the introduction of *probabilistic condensed representations of data*, we propose a novel way of performing data mining. Traditional data mining algorithms operate on the data itself, which poses problems in stream settings, where the amount of data is often too large to be kept in memory. Established data stream solutions try to avoid memory limitations by pursuing window-based approaches. However, they implicitly assume that collecting the data and performing the data mining task is either happening simultaneously or with a small temporal delay. Hence, the user has to know in advance, i.e., before collecting the data, whether she wants to perform a certain data mining task – something which is unrealistic in interactive environments where data streams may need to be analyzed after days or even weeks. For example, after performing pattern mining, she discovers that the patterns contained in a particular subset of the data would be interesting. A window-based solutions would require to initiate another pattern mining process and to wait for new data instances to arrive. To overcome these limitations, we propose the MiDEO framework (Mining Density Estimates inferred Online), in which stream mining is no longer performed on original data in any form (e.g., windows, random samples, ...) but on online density estimates providing an estimate of the joint density of the data stream. These density estimates

are extended by inference capabilities, so that users are able to preprocess the data beforehand (e.g., selecting a subset of the data or specifying already known information, querying, ...). Algorithms, on the other hand, can exploit these inference capabilities to form complex data mining operations, thereby embracing two important principles of data mining and knowledge discovery in databases: the use of probabilistic methods to model the data and at the same time the value of human-readable patterns and models to gain a deeper understanding of the data. This leads to a number of advantages compared to traditional data mining approaches:

1. MiDEO can analyze data of much greater volume than could ever fit into main memory. As the density estimates can be updated continuously and the sizes of the estimates are a fraction of the size of the represented data, potential memory limitations do not play a major role anymore.

2. Along the same lines, the speed of the stream and the speed of analysis need not be coupled anymore. While the online density estimator constantly updates the density estimate, this density can be accessed by the user according to their own needs and analysis requirements.

3. A probabilistic condensed representation of the data can be useful, if the data mining task is not known at the time of collecting the data or several tasks need to be performed – possibly even depending on each other. As density estimates are "universal representations" of data, they can be used to derive any information needed, be it patterns, rules, outliers, clusters, or the like. To make this possible, inference tasks need to be supported for this specific type of density estimate.

4. To protect the privacy of the entities described by the data, it could be required that algorithms must not operate on the original data but on some condensed representation of the data. This could be relevant, if contracts between companies prohibit or restrict the access to data but allow to perform certain evaluation tasks. Density estimates are suitable for this task, because they keep all the relevant statistical properties of the data, in principle without revealing information about individual entities.

5. The probabilistic condensed representation of data enables a clear separation of concerns, such that each step in the knowledge discovery process could be performed by a specialist without disclosing information that is irrelevant for the task at hand. For example, a company that does not have any expertise in the area of data mining could use an online density estimator to estimate the joint density of its data stream, modifies it to meet certain privacy concerns, and sends the resulting condensed representation of data to a data scientist, who can use her expertise to provide an in-depth analysis of the data.

The clear separation of concerns and the privacy-preservation are two highlights of this framework. However, the framework only provides a certain level of privacy-preservation, because also statistical information could disclose sensitive information.

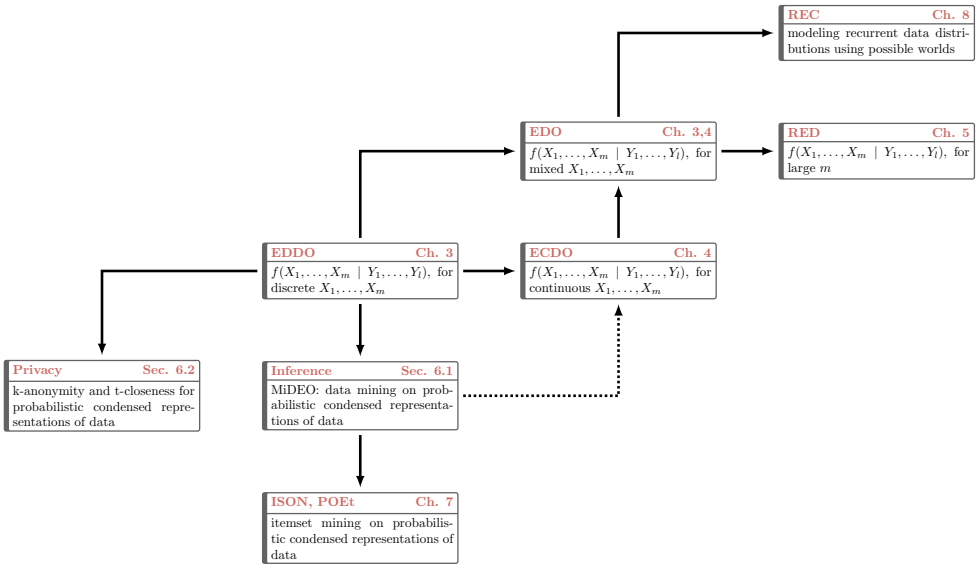


Figure 1.3: This diagram summarizes the methods proposed in this thesis and their relationships. An arrow indicates that the source is used for the target. An arrow with a dotted line indicates that there are some comments regarding the target.

To alleviate this problem, we propose algorithms to achieve k-anonymity and t-closeness for online density estimates, which means that small groups of entities are protected that can be easily identified based on characteristic properties.

To demonstrate that the inference capabilities of the MiDEO framework are in fact sufficient to form complex data mining task, we present an itemset mining algorithm that discovers frequent itemsets without having access to the original data. It provably yields all frequent itemsets and performs well on synthetic and real-world data, if appropriate univariate base estimators are used and the density estimate is an accurate description of the data stream. Moreover, it offers new possibilities of defining the interestingness of itemsets, which we illustrate by introducing a new type of itemsets, so-called *characteristic itemsets*.

1.3 Outline

In the remainder of this thesis, we first introduce relevant concepts and define its scope (see Chapter 2). In particular, we explain the difference between learning from data streams and learning from batch data, formally define online density estimation, and present method from the area of density estimation, pattern mining, and privacy-preserving data mining. Based on these concepts, we then discuss the main topics of this thesis in two parts (see Figure 1.3 for a visualization of the dependencies between the chapters).

In Part I, we basically introduce two online density estimators: EDO and RED. Both aim at estimating joint densities $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$ defined over random variables $X_1, \ldots, X_m, Y_1, \ldots, Y_l$. The general framework of EDO is described in Chapter 3 together with the case of conditional joint densities $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$ where all variables are discrete, i.e., all $X \in \{X_1, \ldots, X_m\}$ are discrete. This is extended in Chapter 4 to conditional joint densities with discrete and/or continuous variables. Since EDO is limited to densities of lower dimensionality, we also propose RED, which is designed for data streams of higher dimensionality, i.e., conditional joint densities $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$ where $m + l$ is large (see Chapter 5).[6]

In Part II, we show that the density estimates not only serve as a compact representation of the data stream, but that they can be extended by inference capabilities to perform complex data mining tasks. For this purpose, we introduce the MiDEO framework and the notion of a probabilistic condensed representation of data (see Chapter 6). We do this by first describing the inference capabilities that an online density estimator has to support to be called a probabilistic condensed representation of data, and then define algorithms that modify density estimates to make sure that they preserve the privacy of entities described by the data. To demonstrate the suitability of MiDEO, we illustrate how the inference capabilities can be combined to more complex data mining tasks and use itemset mining as an example (see Chapter 7).

Part II is concluded with Chapter 8, where we propose a probabilistic condensed representation of data that is able to model recurrent data distributions and provides capabilities to perform complex queries for the historical analysis of the data stream. Chapter 9 concludes the thesis with a summary, a discussion, and thoughts on future work.

[6] In data mining, one would usually consider variables in the hundreds or thousands as *many variables*. For density estimation, however, even 50 binary variables is already considered high-dimensional, as there are 2^{50} value combinations. For each of these combinations, the estimator has to assign a density value, which makes the estimation a challenging task.

Chapter 2

Background

The methods proposed in this thesis are related to many different areas of data mining and machine learning. In this chapter, we introduce concepts and algorithms from these areas and review the corresponding literature. Whereas more relevant methods are described in detail, others are described only briefly to define the scope of the thesis. The main topics are:

1. Mining on data streams: We introduce data stream mining and discuss the current state of the art in concept drift detection.
2. Density estimation: We propose a formal definition of the problem, explain the difference between various types of density estimators, describe Bayesian networks as an example for estimating discrete joint densities, and describe kernel density estimators as example for estimating continuous densities. Additionally, we give an overview of the current state of the art.
3. Pattern mining: We introduce the problem of itemset mining, present methods for the batch and stream setting, and discuss condensed representations.
4. Privacy-preserving data mining: We discuss the current state of the art of privacy-preserving data mining.

We assume that the reader is familiar with the basic notions from computer science (in particular, trees, graphs, and logics) and mathematics (in particular, probability theory and statistics). For reasons of readability, we use some notation to simplify formal statements and algorithmic procedures. The basic notation is defined in Table 2.1. Further notation is defined during the course of this chapter.

2.1 Data Mining on Data Streams

Data mining is a step in the *knowledge discovery process* and is concerned with extracting patterns and models from data. Which patterns are extracted depends on the preprocessing of the data (i.e., data cleaning and data selection) and the so-called *interestingness measure*, which is used to evaluate patterns with respect to the relevant knowledge. The data itself comes in various forms (e.g., transactional data, text data, graph data, network data, spatial data, or time-series data) and is preprocessed to

Table 2.1: The table summarizes some of the notation used in this thesis. The *context* specifies in which context the notation is used.

Context	Notation	Meaning
estimates	$\hat{\theta}$	an estimate for the quantity θ
	$\hat{f}(X_1, \ldots, X_m)$	an estimate for the joint density $f(X_1, \ldots, X_m)$
	$\hat{f}[Z_1, \ldots, Z_{m'}]$	$\hat{f}$ restricted to $\{Z_1, \ldots, Z_{m'}\} \subseteq \{X_1, \ldots, X_m\}$
lists	$A := [a_1, \ldots, a_k]$	a list of elements
	$B := [b_1, \ldots, b_l]$	a list of elements
	$A[i] := a_i$	i-th element of A
	$A[i:j] := [a_i, \ldots, a_j]$	a sublist of A, $i \leq j$
	$A \circ B = [a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_l]$	the concatenation of A and B
trees	$node \in T$	a node of tree T
	$node.isRoot()$	true, iff *node* is a root
	$node.isLeaf()$	true, iff *node* is a leaf
	$node.paths()$	all paths starting at *node* and ending at a node on a lower level
	$node.children()$	all direct successors of *node*
	$node.branches()$	edges pointing to $node.children()$
instances	$\vec{x} = (x_1, \ldots, x_m)$	an instance
	$\vec{x}[X_i]$	x_i
	$\vec{x}[Z_1, \ldots, Z_{m'}]$	$(\vec{x}[Z_1], \ldots, \vec{x}[Z_{m'}])$
	$g \to 0$	$g_N \overset{N \to \infty}{\longrightarrow} 0$, where g_N is a function receiving the instances $\vec{x}_1, \ldots, \vec{x}_N$

meet the requirements of the given task and the selected data mining algorithm (Han and Kamber, 2000).

In this thesis, we will solely focus on *database data* and, in particular, on data residing in a single relation. More precisely, we assume that there is a fixed set of *attributes*, denoted as $\mathcal{X}$, which we also call *variables*. Each attribute $X \in \mathcal{X}$ is associated with a set of possible outcomes, denoted as $values(X)$. If $a, b \in \mathbb{R}$ and $[a, b] \subseteq values(X)$, we call the attribute *continuous*. Otherwise, i.e., if it can only take a fixed and finite set of values, we call it *discrete*. An assignment of attribute values is called an *instance*, if $X_1 = v_1, \ldots, X_m = v_m$ with $X_1, \ldots, X_m \in \mathcal{X}$, such that $v_i \in values(X_i)$ for $1 \leq i \leq m$, which we often denote by its vector notation, i.e., $\vec{x}_i$. Although many algorithms presented in this thesis are also able to deal with instances where $\{X_1, \ldots, X_m\} \subset \mathcal{X}$, we will focus on $\{X_1, \ldots, X_m\} = \mathcal{X}$.

Example 1. Let $\mathcal{X} = \{userid, time, age, gender, heartrate, activity\}$ be a set of attributes measured by a smart watch. In this case, *gender* with $values(gender) = \{female, male\}$ or *activity* with $values(activity) = \{running, sleeping, \ldots\}$ are discrete attributes, whereas *time* in seconds since 1970-01-01 is continuous – allowing fractions of a second. A possible instance is $\vec{x} = (34, 1503589798.5248756, 0{-}25, female, running)$, where $userid = 34$, $time = 1503589798.5248756$, $age = 0{-}25$, $gender = female$, and $activity = running$.

In most data mining settings, an algorithm obtains a set of instances and is supposed to either directly produce some output such as a set of patterns or learn a model from them that is applied to future instances. How the instances are provided depends on the learning scheme, of which *batch learning* and *stream learning* are the prevailing ones. In batch learning, the collection of all data instances are usually referred to as a *dataset* and is either available at once and or can be scanned multiple times. In stream learning, data instances are arranged in a potentially infinite sequence, which is often referred to as a *data stream*. Each instance can only be processed once and is either provided on an instance-by-instance basis or batch-wise. Besides the way instances are provided to the data mining algorithms, batch learning and stream learning also differ in terms of the process generating the instances. Whereas most batch algorithms assume a stationary probability distribution, stream algorithms also take possible drifts into account, i.e., they assume that segments of the stream are potentially generated from different probability distributions (Kifer et al., 2004). These changes are often referred to as concept drifts and occur abruptly, i.e., from one instance to another, or gradually, i.e., there is sequence of instance where each instance can belong to the previous or the next probability distribution.

Example 2. A problem from the area of machine learning that received a lot of attention in both settings is *classification*: Let $X_1, \ldots, X_m$ be a set of attributes and let Y be a discrete attribute that has to be predicted based on $X_1, \ldots, X_m$, i.e., $Y = f(X_1, \ldots, X_m)$, where f is some function. Given an evaluation measure L, the different classification settings can be described as follows:

Table 2.2: The classification into types of concept drifts as proposed by Gao et al. (2007). *no* indicates no change, *yes* indicates a change.

Type of change	$f(X_1, \ldots, X_m)$	$f(Y \mid X_1, \ldots, X_m)$
no change	no	no
feature change	yes	no
conditional change	no	yes
dual change	yes	yes

Batch: There are two datasets S_{train} and S_{test}, which are both generated from the same probability distribution. Based on S_{train}, the goal is to learn a function $\hat{f}$, such that L is minimized on S_{train} and S_{test}.

Stream: There is a stream of instances $S := \vec{x}_1, \vec{x}_2, \ldots$ A learning algorithm A receives segments $S_i := \vec{x}_j, \ldots, \vec{x}_{j+k}$, $j, k \geq 1$, and is supposed to produce an estimate $\hat{f}_i$ after each segment, such that L is minimized for $\hat{f}_i$ on S_{i+1}. In case S consists of segments having differing probability distributions, L is not necessarily monotonically decreasing but should be as small as possible overall.

In the literature, there are mainly two directions how stream classification is approached: either by (1) training a model that is able to adapt to changes in the data stream (e.g., Hulten et al. (2001)) or by (2) training separate models for each segment of the stream (e.g., Kifer et al. (2004); Gonçalves Jr and de Barros (2013)).

2.1.1 Concept Drifts

Since we do not develop methods in this thesis to detect concept drifts but focus on existing methods, we only explain the different types of concept drifts and review the state of the art in concept drift detection.

Gao et al. (2007) classified concept drifts into several categories, each of which describing the nature of the change. They defined these changes in terms of the densities $f(X_1, \ldots, X_m, Y)$, $f(X_1, \ldots, X_m)$, and $f(Y \mid X_1, \ldots, X_m)$ and classified drifts into *no change*, *feature change*, *conditional change*, and *dual change* (see Table 2.2).

Developing methods that either adapt their model when a concept drift occurs or create a new model for every concept already received some attention in the community. For example, Gama et al. (2004) exploited the relationship between the underlying data distribution of the stream and the error rate of the learning algorithm to detect drifts and to learn a new model every time a drift occurs. A similar approach was pursued by Harel et al. (2014), who resampled from the current data instances to obtain statements via the errors of the underlying algorithms. Another example is the work by Bach and Maloof (2010), who performed concept-drift detection based on the distribution of the classifiers' predictions. They employed Bayesian model comparisons but considered

the conditional probability of the prediction given the feature vector instead of the corresponding joint distribution. This is supposed to capture the classifier's ability to predict the values of the class attribute instead of generating its values, which is supposed to improve the detection of change points. A rather different direction was taken by Demsar et al. (2014). They used the explanation methodology to obtain a stream of explanations, on which they then ran a Page Hinkley test to detect concept drifts. A more general approach was pursued by Kifer et al. (2004), who made an attempt at formalizing the detection and quantification of change. They proposed a general framework to capture changes in the distribution of the stream and evaluated several test statistics with respect to their performance. Dries and Rückert (2009) also based the detection of concept drifts on the distribution of the underlying stream and proposed several tests such as the *CNF density estimation test*.

Recently, there was also a substantial increase of work in the direction of recurrent concepts. For example, Gama and Kosina (2013) proposed a framework for handling recurrent concepts in a data stream. In order to decide whether to reuse previous models, they train meta-learners. Meta-learners were also used by Gomes et al. (2011) to select models based on context information for an ensemble classifier. Other approaches were pursued

- by Sakthithasan and Pears (2014), who used the discrete Fourier transform to store previously seen concepts in a compact way and to match the concepts without the need for user-defined thresholds,
- by Lazarescu (2005), who presented a multi-resolution learning approach to track concept drift and recurrent concepts, or
- by Gonçalves Jr and de Barros (2013), who employ statistical tests to detect recurrent concepts based on the distribution of the data.

In general, most of the approaches that deal with recurrent concepts employ some kind of model repository to store previously seen models. Whenever a concept drift is detected, the model repository is checked for a model fitting the emerging concept. If one is found, the model is reused, otherwise a new model is initialized.

Besides concept drifts and recurrent concepts, there are also several more specific topics that were studied. For example, the detection of novel and recurrent classes (Masud et al., 2011), one-class classifiers in data streams (Krawczyk and Wozniak, 2013; Liu et al., 2014), or mining concept-drifting data streams with skewed distributions (Gao et al., 2007).

2.2 Density Estimation

Density estimation on data streams is one of the main problems addressed in this thesis. In this section, we provide a thorough introduction to the problem and present possible solutions. This includes a discussion of the methods listed in Table 2.3.

Table 2.3: An overview of related methods is summarized in this table. For each method, we provide information on the setting (i.e., whether it is an offline or online estimator) and what kind of densities can be estimated. By *Multivariate*, we mean $f(X_1, \ldots, X_m)$, by *Conditional*, we mean $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$, and by *Variables*, we mean whether the random variables are discrete and / or continuous.

Reference	Setting	Multivariate	Conditional	Variables
Ram and Gray (2011)	offline	yes	no	mixed
Davies and Moore (2002)		yes	yes	continuous
Wang and Wang (2015)		yes	no	
Kim and Scott (2012)		yes	no	
Elgammal et al. (2003)		yes	no	
Peherstorfer et al. (2014)		yes	no	
Liu et al. (2007)		yes	no	
Holmes et al. (2012)		no	yes	
Frank and Bouckaert (2009)		no	yes	
Buchwald et al. (2010)		no	yes	
Raykar and Duraiswami (2006)		no	no	
Sheather and Jones (1991)		no	no	
Kristan and Leonardis (2010)	online	yes	no	continuous
Kristan et al. (2011)		yes	no	
Wu et al. (2014)		yes	no	
Lambert et al. (1999)		yes	no	
Zhou et al. (2003)		yes	no	

2.2.1 Formal Definition

Strictly speaking, probability distributions of continuous variables are defined by a density function, whereas probability distributions of discrete variables are defined in terms of probability masses. However, for reasons of readability, we use the term density to refer to probability masses, densities, and mixtures thereof. Then the *joint density* over random variables $X_1, \ldots, X_m$ is a function f, such that

$$Pr(X_1 \in [a_1, b_1], \ldots, X_m \in [a_m, b_m]) := \int_{a_1}^{b_1} \cdots \int_{a_m}^{b_m} f(x_1, \ldots, x_m) \, \mathrm{d}x_1 \cdots \mathrm{d}x_m$$

and f is a non-negative Lebesgue-integrable function. To model probability masses of discrete variables, we rewrite them using the Dirac delta function δ and obtain $f(X) = \sum_{i=1}^{|values(X)|} Pr(X = v_i) \cdot \delta(X - v_i)$ in the univariate case (Chakraborty, 2008). In case the density is conditioned on some random variables $Y_1, \ldots, Y_l$, we write $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$. We assume that the density is estimated from a stream of data and that the stream is generated from a set of densities. More formally:

Definition 1. Let $\mathcal{F} := \{f^i(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l) \mid 1 \leq i \leq k \in \mathbb{N}\}$ be a set of joint densities. A *data stream of* $f^i \in \mathcal{F}$, denoted as $stream(f^i)$, is a possibly infinite sequence of instances $\vec{x_1}, \vec{x_2}, \ldots, \vec{x}_N, \vec{x}_{N+1}, \vec{x}_{N+2}$ that are drawn according to the probability distribution induced by f^i, and $stream(f^i)[1:N]$ is the subsequence $\vec{x_1}, \vec{x_2}, \ldots, \vec{x}_N$. A *data stream over* $\mathcal{F}$, denoted as $stream(\mathcal{F})$, is a possibly infinite sequence of instances $stream(\mathcal{F}) := stream(f^{j_1})[1:N_{j_1}] \circ stream(f^{j_2})[1:N_{j_2}] \circ \ldots$, where $f^{j_i} \in \mathcal{F}$ and $j_i, N_{j_i} \in \mathbb{N}$.

The process of finding an estimate for the current density of a data stream is called *online density estimation*:

Definition 2. Let $\mathcal{F} := \{f^i(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l) \mid 1 \leq i \leq k \in \mathbb{N}\}$ be a set of joint densities and let $stream(\mathcal{F})$ be a data stream over $\mathcal{F}$. An algorithm is called *online density estimator*, if

1. it receives this sequence instance by instance,
2. it has a limited amount of memory Mem, and,
3. after receiving an instance $\vec{x}_i$, it produces a density estimate $\hat{f}_i$.

The process of estimating the current density of a data stream by an online density estimator is called *online density estimation*.

Although Definition 2 describes the general behavior of an online density estimator and the given constraints that have to be fulfilled, it makes no statement of how well the estimate represents the true density f. Without such a statement, however, the user has no guarantees what is actually represented. Therefore, similarly to the definition of other estimators known from statistics, we will define additional properties for online density estimates that provide this feedback. The most relevant property in the context of this paper is the *consistency*, which states whether $\hat{f}$ truthfully represents f in the limit. We express it by measuring the expected loss of $\hat{f}$ given f, i.e., $\mathrm{KL}_{\mathrm{cond}}(f, \hat{f}) = \sum_{\vec{y}} \hat{f}(\vec{y}) \cdot \sum_{\vec{x}} \hat{f}(\vec{x} \mid \vec{y}) \cdot \ln \frac{\hat{f}(\vec{x}|\vec{y})}{f(\vec{x}|\vec{y})} = \mathbb{E}_f(\log(f(\vec{x} \mid \vec{y})) - \log(\hat{f}(\vec{x} \mid \vec{y}))) = \mathbb{E}_f(\log(f) - \log(\hat{f}))$, which is also known as the conditional relative entropy or the conditional Kullback-Leibler divergence.

Definition 3. An online density estimator is said to be *consistent*, if *for all* $f \in \mathcal{F}$: $\mathrm{KL}_{\mathrm{cond}}(f, \hat{f}_N) \overset{N\to\infty}{\longrightarrow} 0$ on the stream of instances $stream(f^{h_1})[1:N_{h_1}] \circ stream(f^{h_2})[1:N_{h_2}] \circ \ldots$ with $h_i \in \{j_i \in \mathbb{N} \mid f^{j_i} = f\}$, and $\sum N_{h_i} \to \infty$.

Remark 1. Notice that this definition also takes data streams with recurrent densities into account. If drifts in the data distribution are detected properly, one can provide a consistent density estimate by maintaining an estimate for each $f \in \mathcal{F}$ and estimating the corresponding transition probabilities for each pair $(f_i, f_j) \in \mathcal{F} \times \mathcal{F}$. The challenge lies in splitting the data stream into segments and correctly associating these segments

with the underlying densities. Each time the data distribution changes, the appropriate estimate becomes active and receives the forthcoming instances until another drift occurs.

Example 3. Let $\mathcal{F} := \{f_1, f_2, f_3, f_4\}$ be a set of densities and $\vec{x}_{i,1}, \vec{x}_{i,2}, \ldots$ be a stream of instances over $\mathcal{F}$ that is generated from the f_i and divided into 5 segments. Then

$$\underbrace{\vec{x}_{1,1}, \ldots, \vec{x}_{1,N_1}}_{f_1}, \underbrace{\vec{x}_{2,1}, \ldots, \vec{x}_{2,N_2}}_{f_2}, \underbrace{\vec{x}_{3,1}, \ldots, \vec{x}_{3,N_3}}_{f_3}, \underbrace{\vec{x}_{2,N_2+1}, \ldots, \vec{x}_{2,N_2+N_4}}_{f_2}, \underbrace{\vec{x}_{3,N_3+1}, \ldots}_{f_3}.$$

The consistency property demands that, *for all* $f \in \mathcal{F}$, the estimator approaches the true density, if $\sum_{h_i \in \{j_i \in \mathbb{N} | f^{j_i} = f\}} N_{h_i} \to \infty$. For f_1, f_2, f_4, this condition does not apply, because there are only finitely many instances, i.e., N_1, $N_2 + N_4$, and 0. Hence, the density estimator can only be consistent with respect to the densities $\mathcal{F}' = \{f_3\} \subset \mathcal{F}$. For f_3, $\{j_i \in \mathbb{N} \mid f^{j_i} = f_3\} = \{3, 5\}$ and $N_3 + N_5 \to \infty$, which means that the estimator has to approach the true density on $\vec{x}_{3,1}, \ldots, \vec{x}_{3,N_3}, \vec{x}_{3,N_3+1}, \ldots$ for increasing numbers of instances.

Remark 2. Property (2) of Definition 2 ensures that the density estimator does not simply store all the instances. Property (3) of Definition 2 requires that there is always a density estimate for the given sequence of instances, and the consistency demands that the densities from $\mathcal{F}$ are approached with increasing numbers of instances.

Remark 3. For reasons of readability, we only modeled streams with abrupt drifts, but one can easily extend the definition for gradual drifts. In the latter case, there would be transition phases between two stream segments:

$$stream(f^{j_i})[1 : N_{j_i}] \circ transition(f^{j_i}, f^{j_{i+1}}, T_i) \circ stream(f^{j_{i+1}})[1 : N_{j_{i+1}}],$$

where $transition(f^{j_i}, f^{j_{i+1}}, T_i) := \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{T_i}$ with $\vec{x}_i \in values(X_1) \times \ldots \times values(X_m)$ and $1 \leq i \leq T_i$. In Part I, we will focus on data streams with no concept drifts, and, in Part II, on data streams with recurrent data distributions.

Remark 4. If Definition 2 is modified as follows: (1) $k = 1$, (2) $N \in \mathbb{N}$, (3) the first property is removed, (4) the second property is modified to: Mem is a limited amount of memory that is large enough to store the model, some temporary variables, and N instances, (5) the third property is removed, and (5) the algorithm only produces an estimate $\hat{f}_N$, then we call the corresponding process *offline density estimation*. In the literature, it is typically known as density estimation, because online density estimation emerged only several decades later.

2.2.2 Types of Density Estimators

Density estimation can be categorized into three main types: parametric, non-parametric, and semi-parametric density estimation (Archambeau and Verleysen, 2003).

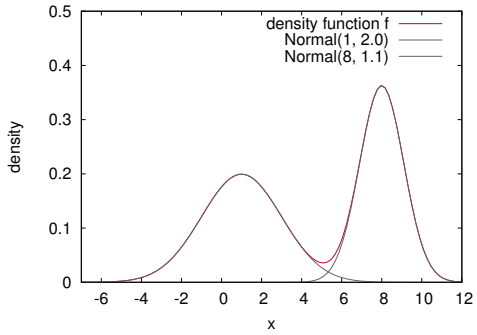


Figure 2.1: The plot shows a density function that can be represented as a mixture of two Gaussians: $\mathcal{N}(1, 2)$ and $\mathcal{N}(8, 1.1)$. A single Gaussian would not be sufficient to provide a consistent estimate.

Parametric Density Estimation

Parametric density estimation assumes that the density is coming from a fixed family of densities $\mathcal{F} = \{f(X_1, \ldots, X_m \mid \Theta)\}$ with parameters Θ. The density estimator aims at finding the best possible parameters for the given data. One example is a family of univariate Gaussians such as

$$\mathcal{F} = \left\{ \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \mid \mu, \sigma \in \mathbb{R}, \sigma^2 \geq 0 \right\}.$$

If N data instances are available, the maximum likelihood estimates $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$ and $\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{\mu})^2$ would explain the data best. Although this type of density estimation is simple and easy to understand, it has a rather large bias. It only works reliably, if the data is normally distributed. As soon as the data is not normally distributed, the density cannot be estimated consistently and can only be an imprecise representation of the data (see Figure 2.1 for an example).

Non-Parametric Density Estimation

Whereas parametric density estimation makes strong assumptions about the density to be estimated, *non-parametric estimation* makes only weak assumptions and is in most cases driven by the data. Most prominent are *kernel density estimators* (Silverman and Jones, 1989; Rosenblatt, 1956; Parzen, 1962), which are still used in some of the state-of-art methods. Unlike histogram estimators, which group the data into bins that are independent from data instances, kernel density estimators define windows around data instances and assign a function to this window. Parzen (1962) formulated the density estimate in terms of a function K, also called *kernel function* or *Parzen window*

(Bishop, 2006; Izenman, 1991), and a smoothing parameter h, also called *bandwidth*:

$$\hat{f}(x) = \frac{1}{N \cdot h} \sum_{i=1}^{N} \mathrm{K}\left(\frac{x - x_i}{h}\right). \tag{2.1}$$

A very simple example for a kernel function is K_1 with $\mathrm{K}_1(y) := \frac{1}{2}$ for $|y| \leq 1$ and $\mathrm{K}_1(y) := 0$ otherwise (Parzen, 1962). The corresponding kernel density estimate suffers from artificial discontinuities (Bishop, 2006), something which is usually not desired in the context of continuous density functions.

To ensure that the estimate exhibits certain desirable properties of a density estimate (e.g., unbiasedness, consistency), one can choose an appropriate kernel function (Rosenblatt, 1956; Parzen, 1962; Izenman, 1991), because it directly influences the properties of the overall estimate (Rosenblatt, 1956). In the past, many possible kernel functions have been considered, and Gaussians belong to the most popular ones. Compared to K_1, which equally spreads its contribution to the final density value over the whole window, a Gaussian kernel has a stronger contribution around its mean and decreases its contribution symmetrically with increasing distance. This yields a strongly consistent estimator, if the smoothing parameter h is a function in N, denoted as h_N, and for $N \to \infty$, $h_N \to 0$, $N \cdot h_N \to \infty$. The choice of the bandwidth can have a large impact on the rate of convergence of the overall density estimate, which is one of the main reasons that it has received a lot of attention in the past (see Raykar and Duraiswami (2006) or Sheather and Jones (1991) for an example and Figure 2.2 for an illustration).

Semi-Parametric Density Estimation

Since the density to be estimated is in many cases unknown, non-parametric models seem to be the only choice to obtain reliable and precise estimates. Due to their weak assumptions, however, non-parametric models have a few disadvantages of their own. For example, kernel density estimates are typically very memory-intensive, which makes them less suitable for large numbers of data instances. At the same time, they cannot deal with high-dimensional data spaces having sparse data regions, because they have too few examples to provide reliable estimates. To mitigate these drawbacks and as a compromise between parametric and non-parametric models, *semi-parametric density estimators* are available, which consist of a parametric and a non-parametric component (Archambeau and Verleysen, 2003; Powell, 1986). One example are finite Gaussian mixture models, which are a mixture of k Gaussians (McLachlan and Peel, 2004):

$$\hat{f}(x) = \sum_{i=1}^{k} w_i \cdot \mathcal{N}(\mu_i, \sigma_i), \tag{2.2}$$

where $\sum_{i=1}^{k} w_i = 1$. By fixing the number of parametric components to some predefined k, one can model more complex densities than a purely parametric density estimator, while not requiring as much memory as a non-parametric density estimator.

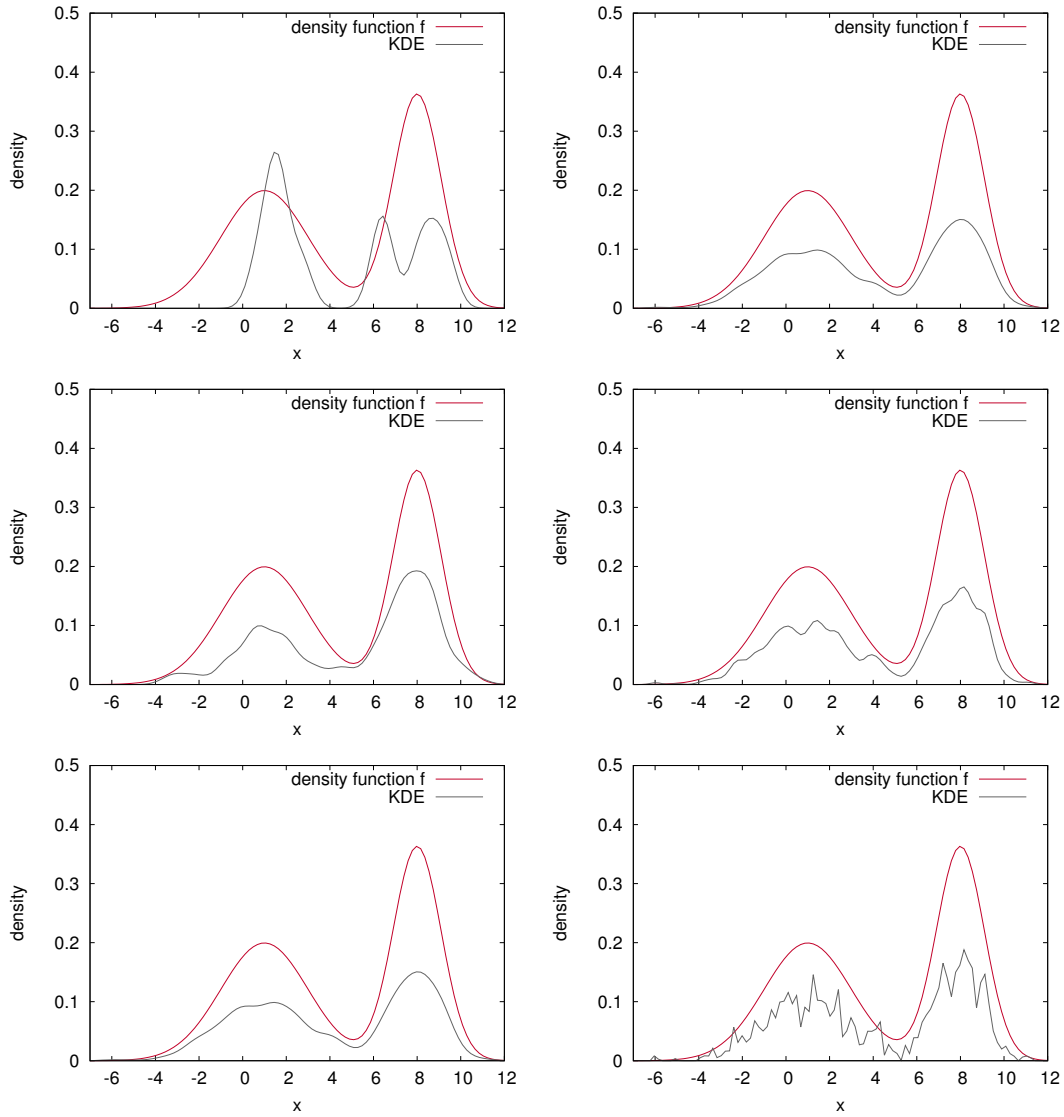
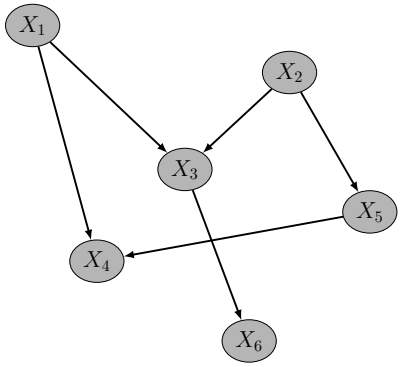


Figure 2.2: These plots illustrate how the number of training instances and the smoothing parameter influence the overall estimate. The column on the left shows the estimate when trained on 10, 100, and 1000 instances. On the right are the estimates when the smoothing parameter is varied: 0.5, 0.25, and 0.05.



X_1	X_2	$x_{3,1}$	$x_{3,2}$
$x_{1,1}$	$x_{2,1}$	0.85	0.15
$x_{1,2}$	$x_{2,1}$	0.32	0.68
$x_{1,3}$	$x_{2,1}$	0.13	0.87
$x_{1,4}$	$x_{2,1}$	0.27	0.73
$x_{1,1}$	$x_{2,2}$	0.56	0.44
$x_{1,2}$	$x_{2,2}$	0.71	0.29
$x_{1,3}$	$x_{2,2}$	0.04	0.96
$x_{1,4}$	$x_{2,2}$	0.25	0.75
$x_{1,1}$	$x_{2,3}$	0.27	0.73
$x_{1,2}$	$x_{2,3}$	0.06	0.94
$x_{1,3}$	$x_{2,3}$	0.46	0.54
$x_{1,4}$	$x_{2,3}$	0.50	0.50

Figure 2.3: The structure of a Bayesian network on the left and one of the conditional probability table (CPT) on the right (in this case for the variable X_3).

A popular choice for finding $\hat{f}$ is the *Expectation Maximization* (EM) algorithm (McLachlan and Peel, 2004). It randomly initializes $\hat{\mu}_i, \hat{\sigma}_i$, $1 \leq i \leq k$, and then alternates between the so-called *E-step* and *M-step*. In the E-step, it computes, for each data instance x_j, a probability distribution $dist_j$, which basically provides the probabilities that x_j belongs to component $\mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i)$, $1 \leq i \leq k$. Based on $dist_j$, the parameters $\hat{\mu}_i$ and $\hat{\sigma}_i$ are then updated in the M-step, e.g., $\hat{\mu}_k$ at time $t+1$ is computed as $\hat{\mu}_k^{t+1} = \frac{1}{\sum_{j=1}^{N} dist_j[k]} \sum_{j=1}^{N} dist_j[k] \cdot x_j$. This is continued until no significant changes are observed anymore.

2.2.3 Bayesian Networks

One way to represent conditional joint densities $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$ with discrete variables are Bayesian networks. They can be categorized as a non-parametric density estimator and consist of two components. The qualitative component is the *Directed Acyclic Graph* (DAG), which consists of one node per random variable $X \in \{X_1, \ldots, X_m\}$ and models interdependencies between these variables by directed arcs. Additionally, it assumes the Markov property, which means that, given its parents, each node is conditionally independent from its non-descendants (Russell and Norvig, 2010). The second, quantitative component is a set of so-called *Conditional Probability Tables* (CPTs) and a one-to-one correspondence between the nodes and the CPTs.

An example of Bayesian network is given in Figure 2.3. On the left is the structure of the network. The nodes are represented by circles and the interdependencies are represented by arcs, where the target of the arc is dependent on its source. Each of the nodes has a CPT, but, for reasons of readability, we only provided the CPT of X_3. The first two columns are the values of the random variables X_1 and X_2. Here, X_1 takes the

values $\{x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}\}$ and X_2 takes the values $\{x_{2,1}, x_{2,2}, x_{2,3}\}$. The CPT lists all possible combinations using the first two columns, i.e., $values(X_1) \times values(X_2)$, and denotes the conditional probabilities $f(X_3 = x_{3,1} \mid X_1, X_2)$ and $f(X_3 = x_{3,2} \mid X_1, X_2)$ in the last two columns.

These two components of a Bayesian network represent a joint probability distribution with discrete variables. The DAG provides the factorization of the joint density into conditional densities, and the CPTs provide the corresponding probabilities. In general, a joint density $f(X_1, \ldots, X_m)$ can be written as $\prod_{i=1}^{m} f_i(X_i \mid parents(X_i))$, where the f_i are given by the CPT of the X_i, $1 \leq i \leq m$. On the other hand, for every discrete probability distribution, there is a Bayesian network that represents its density (Bishop, 2006), which make Bayesian networks a powerful tool to estimate joint densities with discrete variables. (When replacing the CPTs by other representations such as decision trees (Friedman and Goldszmidt, 1996; Su and Zhang, 2006), Bayesian networks can also be generalized to joint densities with continuous variables.)

Although Bayesian networks are a powerful tool to represent joint densities, it is not that easy to find a Bayesian network from a given set of data instances. Typically, one divides the learning in two phases: learning the structure of the network and learning the CPTs, which are either executed sequentially or in an alternating manner. Structure learning can be further categorized into two main directions: *constraint-based* and *score-based* algorithms (Scutari, 2010). Whereas constraint-based algorithms try to learn the network structure by performing independence tests between variables and adding or removing arcs, score-based algorithms try to maximize a scoring function. This scoring function is defined on the set of all Bayesian networks with the given random variables and is used as part of heuristic search algorithm to find the best matching network structure. Based on a suitable network structure, one can then select an algorithm to learn the CPTs of the network. This can be done by estimating the probabilities in the CPTs using maximum likelihood estimation (Cheng et al., 2002; Mitchell, 1997).

2.2.4 Multivariate Density with Discrete and/or Continuous Variables

Besides Bayesian networks, which estimate multivariate densities defined over discrete variables, there are many methods designed for a mixture of discrete and continuous variables or multiple continuous variables. One direction are tree-based density estimators, which includes recent work based on decision trees (see Section 2.3 for an introduction to decision trees), introducing so-called density estimation trees (Ram and Gray, 2011). The tree structure of those trees serves as a mechanism to partition the feature space into regions with similar density. For each of these regions, which are m-dimensional bounding boxes, the authors measure the number of instances falling into this bounding box and the volume. Based on these two quantities, they then propose a density estimator, which is basically a sum over all regions. A similar approach was pursued by Davies and Moore (2002) as part of a conditional density estimator, with which they used to model the CPTs in a Bayesian network. Here, a tree repre-

sents a conditional density, where the leaves may contain non-constant densities. In contrast to the approach by Ram and Gray, the leaves did not represent the density values in terms of volumes and instance counts but by functions or probability masses.

Work towards the estimation of conditional densities has been pursued among others by Holmes et al. (2012), Frank and Bouckaert (2009), and Buchwald et al. (2010). The approach by Frank and Bouckaert is the corner stone of the online density estimator for continuous densities proposed in this thesis. In order to obtain a conditional density estimate of a given variable X, they discretize X and employ a classifier providing class probability estimates for the discretization bins. Subsequently, these probability estimates are used as weights to construct a density estimate (e.g., using a kernel density estimator or mixture models). This approach has recently been used by Rau et al. (2015) to estimate redshift density functions.

Multivariate density estimation over continuous variables has been tackled using many different techniques. For example, Vapnik and Mukherjee (1999) employed support vector machines (SVM). They approached the problem by solving the integral over the density function using SVMs. More frequent approaches for multivariate densities are mixture models (Wang and Wang, 2015) and kernel density estimates (Hwang et al., 1994; Scott and Sain, 2004). Both assume that the density is generated from several random processes. As in the univariate case, the kernel bandwidth can have a large impact on the quality of the overall density estimate, which is one of the main reasons that it has received a lot of attention in the past, and the problem of bandwidth selection is even more challenging in the multivariate case (Wang and Wang, 2015).

Kernel density estimation is also the predominant direction of research on online variants of density estimation developed so far. For example, Kristan and Leonardis (2010), Kristan et al. (2011), and Kim and Scott (2012) proposed a method yielding results that are comparable to corresponding batch approaches. Their online kernel density estimator, called oKDE, constantly re-estimates the bandwidth of the kernels and compresses the kernels if necessary. Lambert et al. (1999) suggested a density estimator employing multipole expansions to achieve fast or even constant update time of the density estimate. Efficient density estimation was also the aim of other kernel based density estimators, e.g., the ones proposed by Zhou et al. (2003) and Elgammal et al. (2003).

Datasets with many instances were considered by Peherstorfer et al. (2014). They proposed to use a sparse grid to describe the density estimate, where basis functions are not centered on the instances but at grid points. Partitioning the space of the data instances was also the main focus of the RS-Forest approach by Wu et al. (2014). They used a forest of trees, where each tree partitioned the space based on one of the data stream variables. Datasets with many attributes have been tackled among others by Liu et al. (2007). They exploited certain sparsity assumptions to represent multivariate densities of high-dimensional data.

2.3 Tools for Density Estimation

The online density estimators proposed in this thesis build upon several well-known algorithms from data mining and machine learning. Most importantly, they will use Hoeffding trees to model conditional densities, i.e., $f(X \mid Y_1, \ldots, Y_l)$, and classifier chains to model joint densities, i.e., $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$. We briefly introduce the idea of decision tree induction, show how Hoeffding trees transfer this idea to the data stream setting, and discuss how (probabilistic) classifier chains can be used to deal with multiple target variables.[1]

2.3.1 Decision Trees

In machine learning, decision trees can be considered as one of the first classification methods. In their simplest form, they consist of a tree structure in which the leaf nodes hold predictions for the class attribute Y and the inner nodes partition the data instances according to remaining attributes. Each inner node is associated with an attribute and its outgoing edges are associated with the values of this attribute. To classify an instance $\vec{y}$, one starts at the root of the tree and then follows the nodes by always picking the edge that matches the attribute value of $\vec{y}$ (with respect to the associated attribute). When a leaf is reached, the corresponding prediction for Y is returned.

The structure of a decision tree is considered to be interpretable by human experts and is one of the reasons for their popularity. Despite this simple structure, decision tree are able to represent many functions – including complete families such as Boolean functions. However, finding the smallest tree that minimizes the error on the training instances is an NP-hard problem (Hancock et al., 1996), so that most algorithms approach decision tree learning by performing a greedy search. Popular examples are ID3 (Quinlan, 1986, 1987), C4.5 (Quinlan, 1993), and CART (Breiman et al., 1984). All of them are typically initialized with a single node (the root) and are constructed by splitting the data instances as a function of the available attributes. When a certain stopping criterion is fulfilled (e.g., all instances have the same value for attribute X), the current node is turned into a leaf node and is associated with a prediction for X. The general framework of top-down decision tree learning is given in Algorithm 1. To improve readability, we used $S_{Y=v}$ to denote $\{s \in S \mid s \text{ has value } v \text{ at attribute } Y\}$. The splitting criterion and stopping criterion depend on the concrete decision tree learner and are therefore provided as parameters. Possible choices for these parameters will be discussed in the following.

[1] Please notice that the notation in this section differs from the corresponding notation in the classification literature. In the density estimation setting, we write $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$ for a joint density with variables X_i and Y_j, $1 \leq i \leq m$, $1 \leq j \leq l$. In the classification setting, however, one typically distinguishes between features and class attributes and denotes the features by $X_1, X_2, \ldots$ and the class attributes by Y respectively $Y_1, Y_2, \ldots$. To be consistent with the notation in the remainder of this thesis, we therefore write $f(X \mid Y_1, \ldots, Y_l)$ and $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$.

Algorithm 1: Decision tree learner

Input: dataset S, discrete attributes $Y_1, \ldots, Y_l$, class attribute X, splitting criterion G, stopping criterion C

Output: a node of the decision tree

```
1  root ← createNode()
2  if C(S, Y_1, ..., Y_l, X) then
3  |  root.prediction ← argmax_{v∈values(X)} |S_{X=v}|
4  |  return root
5  else
6  |  root.attribute ← argmax_{Y∈{Y_1,...,Y_l}} G(S | Y)
7  |  for v ∈ values(Y) do
8  |  |  if |S_{Y=v}| = 0 then
9  |  |  |  node ← createNode()
10 |  |  |  node.prediction ← argmax_{v∈values(X)} |S_{X=v}|
11 |  |  else
12 |  |  |  node ← DecisionTreeLearner(S_{Y=v}, {Y_1, ..., Y_l} \ {Y}, X, G, S)
13 |  |  end
14 |  |  root.addEdge(node, label = v)
15 |  end
16 end
```

Splitting Criterion

One of the central tasks of top-down decision tree learning is splitting nodes to grow the tree, i.e., extending nodes to subtrees. Most algorithms associate a single attribute X with such a node and partition the data instances according to the values of X. If X is discrete, one edge per value is created. If X is continuous, its range is usually split into two intervals $(-\infty; c]$ and $(c; \infty)$, where c is determined by sorting the values of X, considering all points $c_{a,b} := \frac{b-a}{2}$ for which the class of a differs from that of b, and taking the $c_{a,b}$ that maximizes a given heuristic criterion G (Mitchell, 1997). The challenge is to find a G that improves the accuracy of the decision tree while keeping the tree small. Popular choices are impurity-based measures such as $InfoGain$, $GainRatio$, and $GiniGain$.

Loosely speaking, the information gain is the expected reduction in entropy that results from splitting a node according to an attribute X. To compute the information gain, one computes the entropy of the given node and subtracts the entropies of the resulting subtrees. Hence, the higher the information gain, the lower the impurity of the class attribute.

Definition 4. (*Entropy* (Mitchell, 1997)) Let S be instances with attributes $X_1, \ldots, X_l, Y$ and let Y be the class attribute. The entropy H is defined as follows:

$$H(S \mid X) = -\sum_{v \in values(X)} \frac{|S_{X=v}|}{|S|} \cdot \log \frac{|S_{X=v}|}{|S|} \tag{2.3}$$

Definition 5. (*InfoGain* (Mitchell, 1997)) Let S be instances with attributes $Y_1, \ldots, Y_l, X$ and let X be the class attribute. The information gain is defined as follows:

$$InfoGain(S, Y \mid X) = H(S \mid X) - \sum_{v \in values(Y)} \frac{|S_{Y=v}|}{|S|} \cdot H(S_{Y=v} \mid X) \tag{2.4}$$

One of the major disadvantages of the information gain is its biasedness towards attributes with many values. The more values an attribute has, the easier it is to separate the instances into groups with equal class values. To counteract this behavior, normalized impurity measures have been proposed. One example of such a measure is the gain ratio, which has been used in the context of C4.5 and normalizes the information gain by the entropy of the split attribute:

Definition 6. (*GainRatio* (Quinlan, 1993; Rokach and Maimon, 2005)) Let S be instances with attributes $Y_1, \ldots, Y_m, X$ and let X be the class attribute. The gain ratio is defined as follows:

$$GainRatio(S, Y \mid X) = \frac{InfoGain(S, Y \mid X)}{H(S \mid Y)} \tag{2.5}$$

Another impurity-based measure is GiniGain, which builds upon a well-known tool from statistics: the Gini index. It assigns a probability mass to each value of a potential

split attribute and evaluates the divergence of the probability masses. Similarly to the information gain but with the Gini index instead of the entropy, Gini gain computes the average reduction of the Gini index that results from splitting a node according to an attribute Y.

Definition 7. (*GiniIndex* (Breiman et al., 1984; Rokach and Maimon, 2005)) Let S be instances, let Y be some attribute, and let X be the class attribute. The Gini index is defined as follows:

$$Gini(S \mid X) = 1 - \sum_{v \in values(X)} \left(\frac{|S_{X=v}|}{|S|} \right)^2 \tag{2.6}$$

Definition 8. (*GiniGain* (Breiman et al., 1984; Rokach and Maimon, 2005)) Let S be instances and let X be some attribute. The Gini gain is defined as follows:

$$GiniGain(S, Y \mid X) = Gini(S \mid X) - \sum_{v \in values(Y)} \frac{|S_{Y=v}|}{|S|} \cdot Gini(S_{Y=v} \mid X) \tag{2.7}$$

Stopping criterion

The splitting criterion is responsible for growing the tree. If no stopping criterion is imposed, each branch would grow until every attribute has been selected once. This, however, results in unnecessarily large trees, of which many splits do not provide any benefit. Therefore, most algorithms use additional stopping criteria to limit the tree growth such as (Rokach and Maimon, 2005)

1. all instances that are filtered to the given leaf have the same class,
2. the maximal depth of the tree is reached, or
3. the number of instances that is filtered to the given leaf is below a certain threshold.

Pruning

The stopping criterion C can have a large impact on the generalization error. If C stops the growing process too early, the decision tree tends to underfit. If C stops the growing process too late, the tree tends to overfit. As finding the perfect balance is a challenging task, most decision tree learners approach the problem by first letting the tree overfit and then pruning the tree if necessary, i.e., turning subtrees into leaf nodes (Rokach and Maimon, 2005). Common choices are reduced error pruning and cost-complexity pruning, which both use a separate validation set to measure the generalization error. Reduced error pruning iteratively tests whether replacing a subtree by a leaf does not decrease the accuracy of the tree. If there are several such nodes, the one with largest increase is chosen. Cost-complexity uses the training set in addition to the validation

set and performs pruning in two stages. In the first stage, trees $T_0, T_1, \ldots, T_k$ are built, where T_{i+1} is build from T_i by removing subtrees that leads to the lowest increase in training error per pruned leaf. In the second stage, the T_i with lowest generalization error is chosen.

2.3.2 Hoeffding Trees

Hoeffding trees are an extension of decision trees that bring tree classifiers to the world of data streams. The primary difference is the sequential training, which is enabled by making split decisions on a subset of the data. Whereas a decision tree evaluates the heuristic measure G on the full dataset, Hoeffding trees restrict this computation to the instances that have been observed so far. Since storing these instances is in conflict with the data stream setting, the Hoeffding tree only keeps counts of the number of variable values and computes the heuristic measure based on the counts, denoted by $\hat{G}$. This allows to compute $\hat{G}$ with a limited amount of memory, but it also introduces a certain error ϵ. In order to correct for this error and to make decisions holding with high confidence, the Hoeffding bound provides an estimate for ϵ: Let Y_a and Y_b be the random variables that have the highest and second highest values with respect to $\hat{G}$. A split is performed, if

$$\hat{G}(Y_a) - \hat{G}(Y_b) > \sqrt{\frac{R^2 \cdot \ln(1/\delta)}{2 \cdot N}} =: \epsilon,$$

where R is the range of the heuristic measure, $1-\delta$ is the probability the decision is correct, and N is the number of instances. Although this approach potentially introduces an error with every split decision, the tree produced by the Hoeffding tree algorithm is asymptotically very similar to the tree produced by a batch learner (Domingos and Hulten, 2000), which can be controlled by the confidence level δ.

This basic idea of learning a decision tree from a data stream has been refined in several ways and led to two extensions: the *VFDT* (Very Fast Decision Tree learner) system and *CVFDT* (Concept-adapting Very Fast Decision Tree learner). VFDT does not introduce any new concepts and is rather a collection of minor improvements, e.g., (1) to decrease the running time by not recomputing $\hat{G}$ after every instance, (2) to decrease the memory usage by deactivating the least promising leaves in case not enough memory is available, or (3) to perform splits after fewer instances by avoiding *ties* (i.e., if two attributes Y_a and Y_b are equally good, one of them is picked for the split instead of waiting until $\hat{G}(Y_a) - \hat{G}(Y_b)$ is large enough) (Domingos and Hulten, 2000).

CVFDT is a further of extension of the VFDT system that is able to handle concept drifts in the data stream (Hulten et al., 2001). It keeps statistics for every node and periodically scans the Hoeffding tree to validate previous split decisions. If a decision turns out to be wrong or does not fit the current concept anymore (e.g., because of a concept drift), the algorithm grows an alternative subtree for the corresponding node and replaces the subtree as soon as it is more accurate than the current one.

2.3.3 Classifier Chains

With Hoeffding trees, we will be able to represent univariate densities. To extend these estimates to joint densities defined over several variables, we will combine them using classifier chains. Classifier chains (Read et al., 2011) and probabilistic classifier chains (Dembczynski et al., 2010) have been proposed in the context of multi-label classification, where classification is no longer restricted to a single class attribute but to a set of so-called *labels*. Whereas a dataset typically has a set of attributes $\mathcal{Y} := \{Y_1, \ldots, Y_l\}$ and a class attribute $X \notin \mathcal{Y}$, a multi-label datasets has a set of attributes $\mathcal{Y}$ and a set of labels $\mathcal{X} := \{X_1, \ldots, X_m\}$, $\mathcal{X} \cap \mathcal{Y} = \emptyset$. Each label can take two values to indicate whether it is present for a given instance: 1 (*present*) or 0 (*absent*). The overall goal is to learn a function $Y_1 \times \ldots \times Y_l \to X_1 \times \ldots \times X_m$ that maps an attribute configuration to a set of labels.

Example 4. A typical example of a multi-label classification problem is labeling movies with genres. In many cases, it is not possible to assign a single genre to a movie, because it not only belongs, e.g., to science fiction but also to comedy. In the multi-label setting, one would have a label for each genre and would set the labels $\{scifi, comedy\}$ to 1 and all the other labels to 0.

A simple approach to multi-label classification is called *binary relevance*. It trains single-class classifiers $\hat{f}_{X_i}$ for each $X_i \in \mathcal{X}$ and predicts the label set of an instance $\vec{y} = (Y_1, v_1), \ldots, (Y_m, v_l)$ by combining the predictions of all classifiers, i.e., the labels are set to $(X_1, \hat{f}_{X_1}(\vec{y})), \ldots, (X_l, \hat{f}_{X_m}(\vec{y}))$. However, one can only expect a good classification performance, if all labels are completely independent of each other, because binary relevance disregards potential interdependencies between variables due to the construction of the classifiers. To take possible interdependencies into account, many algorithms have been developed and classifier chains as proposed by Read et al. (2011) are a very popular one. Similarly to the online density estimator proposed in this thesis, they use the product rule to split the task into several subtasks, i.e.,

$$f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l) = f_1(Y_1, \ldots, Y_l) \cdot \prod_{i=2}^{m} f_i(X_i \mid Y_1, \ldots, Y_l, X_1, \ldots, X_{i-1}),$$

and then use classifiers such as decision trees to estimate the f_i. To train the classifiers, one selects a loss function and minimizes the prediction error based on the ground truth. To obtain a prediction, one simply iterates over the classifiers from $\hat{f}_1$ to $\hat{f}_m$, predicts the label for each $\hat{f}_i$, and then uses the input of $\hat{f}_i$ for the next classifier $\hat{f}_{i+1}$, $1 \leq i \leq m-1$.

Later, classifier chains have been extended by a theoretical framework, which led to probabilistic classifier chains (Dembczynski et al., 2010). The main difference is that the $\hat{f}_i$ return class probabilities and that a prediction is determined by considering all possible combinations with their probabilities.

Table 2.4: Some examples of shopping cart contents. The transaction are subsequently turned into a database.

Transaction ID	Items
⋮	⋮
20160714-0057	bananas, strawberries, blueberries
20160714-0058	water, chewing gums, salad
20160714-0059	gouda, butter, bread, dried tomatoes
20160714-0060	toast, butter, chocolate creme, tissues, milk
20160714-0061	cereals, milk, blueberries, almonds
20160714-0062	pineapple, bananas, strawberries, milk, water, cereals
20160714-0063	water, apple juice, bread, olives

2.4 Pattern Mining

Density estimates can be considered as a compact representation of data, which is particularly helpful in stream settings, where memory is often a limiting factor. But if the original data is not available, one has to make sure that the density estimate itself provides facilities to perform typical analysis tasks. To show that this requirement is fulfilled by the online density estimates proposed in this thesis, we demonstrate their effectiveness at the example of pattern mining. It is one of the main data mining tasks (Calders et al., 2004) and refers, amongst others, to the mining of itemsets, association rules, correlations, subsequences, and substructures (Han and Kamber, 2000; Han et al., 2007). As the focus of this thesis lies on itemset mining, we will only introduce itemsets and association rules in this chapter.

To introduce the problem of pattern mining, the analysis of shopping cart contents is a typical example: Imagine a supermarket that has a large database and stores the purchases of its customers. Although it does not know the identity of the customers, it keeps track of all the items that have been bought together, i.e., the items contained in a single shopping cart (see Table 2.4 for an example). By analyzing the contents of the shopping carts, the supermarket tries to identify those items that are usually bought together (e.g., customers buying bananas are also buying strawberries) and uses this knowledge to place these items nearby (to increase the likelihood that customers buy both products) or far apart (to make sure that customers see many other products on their way).

In traditional pattern mining, knowledge is usually presented in form of itemsets and association rules, which both consist of so-called *items*. The set of all possible items is denoted by $\mathcal{I}$ and is, in the example given above, the set {bananas, strawberries, blueberries, water, bread, olives, chewing gums, salad, gouda, butter, dried tomatos, toast, chocolate creme, tissues, milk, cereals, blueberries, almonds, pineapple, apple juice}. An itemset is a set $I \subseteq \mathcal{I}$ and an association rule is a statement of the form

$I \Rightarrow J$ with $I, J \subseteq \mathcal{I}$ and $I, J \neq \emptyset$. The latter means that customers who are buying the products in I are also buying the products in J. These rules are often constructed from an itemset by splitting it into two disjoint subsets.

To mine patterns of this kind, algorithms are usually provided with a database S. It is basically a sequence of so-called *transactions*, which consist of a unique identifier *tid* to reference the transaction and a set $I \subseteq \mathcal{I}$ to define which items are contained. I is often represented by a Boolean vector in which the element at index i specifies whether item $\mathcal{I}[i]$ is contained in the transaction. To simplify notation, we write $t.tid$ to get the *tid* and $t.items$ to get the items of the transaction.

To distinguish relevant from non-relevant patterns, they are typically evaluated with a measure of interestingness. For a database S and an itemset I, the support is such a measure, $\frac{|\{t.tid | I \subseteq t.items \text{ for } t \in S\}|}{|S|}$, and is denoted by $freq(S)$. If it exceeds a user-defined threshold $0 \in [0, 1]$, the itemset is considered frequent and therefore interesting.[2] For association rules, the support is usually insufficient to measure its interestingness, because the relationship between the left-hand side and right-hand side is neglected. To overcome this issue, additional measures have been proposed such as the confidence or the lift. The confidence of a rule $I \Rightarrow J$ is defined as $conf(I \Rightarrow J) = \frac{freq(I \cup J)}{freq(I)}$ and basically estimates the conditional probability of J given I (Agrawal et al., 1993; Agrawal and Srikant, 1994). But since the *confidence* does not take the support of J into account, completely unrelated I and J can sometimes lead to large confidence values. To correct this, Brin et al. (1997) proposed the so-called *lift*, which is defined as $\frac{freq(A \cup B)}{freq(A) \cdot freq(B)}$.

2.4.1 Frequent Itemset Mining

Although the concept of frequent itemsets is simple, finding all frequent itemsets efficiently, i.e., $\mathcal{F}_S = \{I \subseteq \vec{x} \mid \vec{x} \in S \wedge freq(I) \geq \theta\}$, is a challenging task that has attracted many researchers. After the topic has been introduced by Agrawal et al. (1993), a lot of research went into finding good data structures, optimizing the usage of main memory, and pruning the search space as early as possible. In the following, we will discuss the main ideas behind the most famous itemset mining algorithms.

The Apriori Algorithm

Apriori is an improved version of the very first itemset mining algorithm, called *AIS* (Agrawal et al., 1993). It searches the space of all frequent itemsets in a level-wise manner, first considering itemsets of length 1, then itemsets of length 2, and so on. As the search space is exponential in the number of items, Apriori tries to prune the search space as early as possible by applying the *anti-monotonicity* property to frequent itemsets:

[2] Please notice that some algorithms use the support count instead of the support (i.e., $|\{t.tid \mid I \subseteq t.items \text{ for } t \in S\}|$), which simplifies updates.

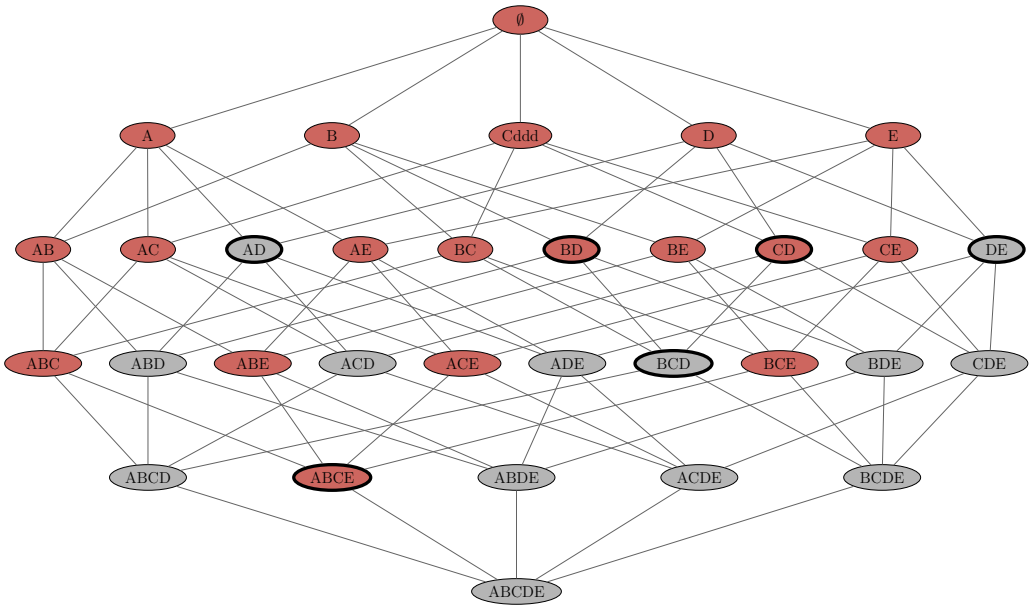


Figure 2.4: An example for a lattice. The itemsets defined over the items $\{A, B, C, D, E\}$. Frequent itemsets are illustrated by a red circle and infrequent itemsets by a gray circle. Border elements (see Section 2.4.2) are marked with thicker border, i.e., AD, DE, BCD for the negative border and BD, CD, $ABCE$ for the positive border.

Lemma 1. *Let $\theta \in [0, 1]$ and $\mathcal{I}$ be a set of items. If $I, J \subseteq 2^{\mathcal{I}}$ with $I \subseteq J$ and $I.support < \theta$, then $J.support < \theta$.*

Hence, any superset of an infrequent itemset is also infrequent and does not need to be considered on higher levels. The algorithmic implications of this property are best illustrated with an example. In Figure 2.4, all itemsets that can be generated from the items $\{A, B, C, D, E\}$ are represented in form of a so-called *lattice*. At the top is the empty set and at the bottom is the itemset containing all five items. After generating all itemsets of length 2, it turns out that AD and DE are both infrequent. By applying the anti-monotonicity property, it follows that every itemset that is longer and below these two itemsets is infrequent, i.e., $ABD, ACD, ADE, BDE, CDE, ABCD, ABDE, ACDE, BCDE, ABCDE$.

Apriori exploits this property when joining itemsets (see Algorithm 2). On level one, it first enumerates all frequent itemsets of length 1 (lines 1,2, and 10). Then, for levels $i > 1$, it induces an ordering on the items $\mathcal{I}$ and creates a new itemset candidate by joining two itemsets of length $i-1$ (lines 12-13). If the resulting itemset is infrequent, it can be immediately pruned, according to the anti-monotonicity property (lines 19-21).

Besides this pruning strategy, two further details influence the running time positively. First, Apriori does not join arbitrary itemsets of length $i - 1$ but takes those

Algorithm 2: Apriori

Input: items $\mathcal{I}$, transaction database S, support threshold θ
Output: frequent itemsets $\mathcal{F}(S)$

```
1  k ← 1
2  C_k ← {{X_i} | i ∈ I}
   // level-wise itemset generation
3  for (tid, I) ∈ S do
4  |  for J ∈ C_k do
5  |  |  if J ⊆ I then
6  |  |  |  J.support = J.support + 1
7  |  |  end
8  |  end
9  end
10 F_k := {J ∈ C_k | J.support ≥ θ · |S|}
11 while |F_k| > 0 do
12 |  for I_1, I_2 ∈ F_k do
13 |  |  if I_1[1 : k − 1] = I_2[1 : k − 1] and I_1[k] < I_2[k] then
14 |  |  |  for i ≤ k do
15 |  |  |  |  J[i] ← I_1[i]
16 |  |  |  end
17 |  |  |  J[k + 1] ← I_2[k]
18 |  |  end
19 |  |  if ∃I ⊆ J : |I| = k and I ∉ F_k then
20 |  |  |  C_{k+1} ← C_{k+1} ∪ J
21 |  |  end
22 |  end
23 |  k ← k + 1
24 |  F_k ← {J ∈ C_k | J.support ≥ θ · |S|}
25 end
26 return F_1 ∪ ... ∪ F_k
```

that have a common prefix of length $i-2$ (line 13). This ensures that each itemset is generated at most once. Second, the **for loops** in lines 3 and 4 first iterate over the transactions in the database and then over the itemsets. This limits the number of database scans and avoids unnecessary switching between the main memory and a hard disk.

After Apriori has been published, several optimizations have been proposed that positively influence the memory consumption (e.g., data structures such as hash-trees and tries), speed up the computation of supports (e.g., AprioriTid, AprioriHybrid, MaxMiner, and Apriori-LB), reduce the number of generated candidates (e.g., DHP), and reduce database scans (e.g., DIC and sampling) (Goethals, 2003).

FP-Growth

Despite the optimizations already applied by Apriori, the algorithm often generates tremendous amounts of itemsets and performs up to $k+1$ complete database scans, where k is the longest frequent itemset. These shortcomings motivated Han et al. (2004) to design an algorithm, called FP-Growth, that only requires two database scans and does not generate any itemset candidates.[3]

The corner stone of this algorithm is a data structure called FP-Tree, which is basically a prefix trie providing a lossless compression of all transactions. Each node *node* of the trie is associated with an item and a support count, such that a path $node_1(I_1, c_1), edge_1, \ldots, edge_l, node_l(I_l, c_l)$ represents the itemset $\{I_1, \ldots, I_l\}$ with support counts $c_1, \ldots, c_l$ for the prefixes. In order to access nodes more quickly, it also provides a table storing the support counts of the items and a list linking all nodes associated with a certain item.

To generate the FP-Tree, two database scans are required. The first scan determines the support counts of the items and orders them according to these counts. The second scan inserts the items of the transactions with respect to that ordering, which means that it identifies the longest common prefix and adds nodes as required. Ordering the items is actually not necessary, but it supposed to yield a smaller prefix trie.

Itemset mining is directly performed on the FP-Tree and does not require any further database scans. FP-Growth performs a depth-first search and creates new FP-Tree for every recursion step of the search. At each step, it has to compute the support counts of itemset candidates, which is performed by starting at the item table, following the links to the nodes associated with that item, and following the paths of these nodes to the root.

Another depth-first search algorithm is Eclat (Zaki et al., 1997), which is in many respects similar to FP-Growth. The main difference is that Eclat computes the support counts by directly intersecting sets of transactions, which are stored in a vertical database format (i.e., for each itemset, it stores a list of transaction ids containing this itemset).

[3] Please notice that this claim has been challenged later (e.g., by Goethals (2003)).

2.4.2 Condensed Representations

When applying itemset mining to real-world domains, users are often confronted with large volumes of frequent itemsets, easily outnumbering the data items themselves. Scanning these volumes is in many cases neither possible nor desired, so that a lot of research focused on finding so-called *condensed representation*s. Such a representation should consist of substantially fewer itemsets from which all frequent itemsets can be derived. First steps in this direction have been undertaken by Mannila and Toivonen (1997), who proposed *positive* and *negative borders*:

Definition 9. Let S be a database and θ a support threshold. The *positive border* contains the most specific itemsets that are still frequent, i.e., $BD_\theta^+(S) = \{I \subseteq \mathcal{I} \mid \forall J' \subseteq I \subset J'' : J' \in \mathcal{F}_\theta(S) \wedge J'' \notin \mathcal{F}_\theta(S)\}$, and the *negative border* contains the most general itemsets that are not frequent, i.e., $BD_\theta^-(S) = \{I \subseteq \mathcal{I} \mid \forall J' \subseteq I : J' \in \mathcal{F}_\theta(S) \wedge I \notin \mathcal{F}_\theta(S)\}$.

Example 5. In Figure 2.4, the negative and positive borders are marked with thicker borders. Compared to the number of frequent itemsets (18), the number of border elements is substantially smaller (6).

Although all frequent itemsets can be derived from the borders, the frequencies of non-border elements are lost. To solve this issue, other types of condensed representations have been proposed (Calders et al., 2004; Bykowski and Rigotti, 2001). One example are *closed sets*:

Definition 10. Let S be a database. An itemset I is *closed*, iff $\nexists J \supset I$: $freq(J) = freq(I)$.

In other words, for every distinct support value, there is a closed itemset. The set of all closed itemsets is sufficient to derive all other frequent itemsets together with their frequencies. Hence, one could consider the itemsets having the same support as equivalence classes and the closed itemsets of these classes as representatives (Bonchi and Lucchese, 2006). Whereas closed itemsets are the maximal itemsets of the equivalence classes, another condensed representation consists of the minimal itemsets of these classes, the *free itemsets* (Bastide et al., 2000; Boulicaut et al., 2003; Calders et al., 2004):

Definition 11. Let S be a database. I is free, iff $\nexists J \subset I$, $X \in I$: $X \notin J$ and $freq(J) = freq(J \cup \{X\})$.

Remark 5. Free itemsets are typically defined in two steps, using so-called δ-*strong rules*. To facilitate the comparison with other types of itemsets, we combined these two definitions into one.

Since the free itemsets alone are not sufficient to derive all frequent itemsets, they need to be combined with elements from the negative border to constitute a condensed

representation (Calders et al., 2004). Combined with the closed itemsets, the closed and free itemsets characterize the equivalence classes by specifying their most specific and most general elements.

Unfortunately, it can still happen that there are too many equivalence classes and, therefore, too many itemsets are presented to the user. Therefore, researchers relaxed the definitions by introducing a parameter δ to control the size of the equivalence classes. The resulting condensed representations have fewer itemsets but still allow to derive most of the frequent itemsets. The corresponding generalization are *δ-tolerant closed itemsets* (Cheng et al., 2006), and *δ-free itemsets* (Boulicaut and Bykowski, 2000; Boulicaut et al., 2003):

Definition 12. Let S be a database and δ be a real number between 0 and 1. An itemset I is *δ-closed*, iff $\nexists J \supset I$: $|J| = |I| + 1$ and $freq(J) \geq (1 - \delta) \cdot freq(I)$.

Definition 13. Let δ be a natural number. I is δ-free, iff $\nexists J \subset I$, $X \in I$: $X \notin J$ and $freq(J) - freq(J \cup \{X\}) \leq \delta$.

By increasing δ, one basically increases the size of the equivalence classes while decreasing the number of classes, which results in fewer itemsets. This, however, comes at the cost of no longer having exact support values for the itemsets. Only if δ equals 0, exact support values can be computed. For $\delta > 0$, the support can only be computed with an error depending on δ (Boulicaut et al., 2003; Cheng et al., 2006). Hence, queries posed on the condensed representation do not yield exact results but approximations that are imprecise to a certain controllable degree.

Deriving the support of other itemsets – exactly or approximately – has also been the main focus of other condensed representations such as *disjunction-free itemsets* (Bykowski and Rigotti, 2001), *disjunction-bordered condensation* (Bykowski and Rigotti, 2003), and *non-derivable itemsets* (Calders and Goethals, 2002). But all of them used itemsets to derive other itemsets, which is different from the approach we are pursuing in this thesis.

2.4.3 Data Streams

With the increasing importance of data streams, the pattern mining community also addressed the problem of finding frequent itemsets from data streams (an extensive survey has been provided by Cheng et al. (2008)). Most approaches either introduce some kind of decay rate to diminish the influence of old transactions or use a window. The window-based approaches can be classified into *time-based* or *count-based* and into *landmark-window-based* or *sliding-window-based*:

time-based window Let W be a window containing the instances $\vec{x}_{start}, \vec{x}_{start+1}, \ldots, \vec{x}_{end}$. W is *time-based*, if it is partitioned into consecutive segments $W_1, W_2, \ldots, W_M$, such that the elapsed time between the first and last instance in W_i is equal for $1 \leq i \leq M$.

count-based window Let W be a window containing the instances $\vec{x}_{start}, \vec{x}_{start+1}, \ldots, \vec{x}_{end}$. W is *count-based*, if it is partitioned into consecutive segments $W_1, W_2, \ldots, W_M$ of equal length, i.e., $W_i = x_{\lfloor \frac{end-start}{M} \rfloor \cdot (i-1)+1}, \ldots, x_{\lfloor \frac{end-start}{M} \rfloor \cdot (i-1)+(end-start)}$.

landmark window Let $S = \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N$ be a stream of instances. W is a *landmark window*, if $W = \vec{x}_1, \ldots, \vec{x}_{|W|}$.

sliding window Let $S = \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N$ be a stream of instances. W is a *sliding window*, if $W = \vec{x}_i, \ldots, \vec{x}_{i+|W|-1}$, for $1 \leq i \leq N - |W| + 1$.

Since window-based approaches are only provided with the instances in the window and previous instances cannot be stored arbitrarily, the frequency of an itemset is usually redefined with respect to the window, i.e., $freq(I) := \frac{\#\text{transactions in } W \text{ containing } I}{|W|}$. As a result, approaches using a sliding window can only capture the currently frequent itemsets, whereas approaches using a landmark window capture the frequency with respect to the current stream prefix, i.e., all available data instances. Due to the constantly growing data stream, however, the latter is usually infeasible in practice, so that many approaches only work with approximated frequencies. Therefore, stream mining algorithm often consider a relaxed minimum frequency θ_ϵ, which lies between 0 and the actual minimum frequency. This way, the algorithms temporarily store certain infrequent itemsets that potentially turn out to be frequent after receiving more instances. Popular algorithms either use data structures maintaining frequent and possibly-frequent itemsets or apply probabilistic bounds. For example, the *lossy counting algorithm* by Manku and Motwani (2002) maintains a set of tuples $\mathcal{T}$ where each tuple stores an itemset I, the frequency $\widehat{freq}(I)$ of I after inserting it into $\mathcal{T}$, and an upper bound $err(I)$ for the frequency of I before it has been inserted into $\mathcal{T}$. Based on the current batch W_i, I is then either added to $\mathcal{T}$ or deleted. Another example is the *DSM-FI* algorithm by Li et al. (2008), which is based on a prefix-based in-memory data structure, called *Item-suffix Frequent Itemset forest (IsFI-forest)*. The data structure maintains the frequent itemsets with their approximated frequencies by using header tables and prefix trees. An example for an algorithm using probabilistic bounds is *FDPM* by Yu et al. (2004). It uses Chernoff bounds to get an interval for the frequency of an itemset that holds with high probability.

A popular method using a sliding window is *Moment* (Chi et al., 2006), which maintains the set of currently closed frequent itemsets. Similarly to DSM-FI, it also uses an in-memory prefix-tree-based data structure, but, in contrast to DSM-FI, it only creates a single prefix-tree and uses four types of nodes to decide whether an itemset is infrequent, a potential candidate of becoming a frequent closed itemset (unpromising or promising), or a frequent closed itemset: *Infrequent Gateways Nodes (IGN)*, *Unpromising Gateway Nodes (UGN)*, *Intermediate Nodes (IN)*, and *Closed Nodes (CN)*. To define these node types, the authors consider the frequency of the itemset, its parent, and its siblings.

2.5 Privacy-Preserving Data Mining

Privacy preservation in data mining is a balancing act between maintaining data utility and data privacy (Pinkas, 2002; Chen and Liu, 2011). Putting the emphasis on one of these extremes usually means giving up the other. However, finding the perfect balance is a challenging task and evaluating the effectiveness of a privacy-preserving method is even more difficult – as history has proven many times (Kargupta et al., 2003; Machanavajjhala et al., 2007; Soria-Comas and Domingo-Ferrer, 2013). Methods that were thought to be privacy-preserving later turned out to disclose sensitive information partially or completely. The main challenge for any privacy-preserving method is to provide just enough information to perform well on the data mining task but not enough to give away any sensitive information. So, in the end, it is not a question of whether information is disclosed, but which information and how to measure and control it. To categorize privacy-preserving data mining methods, Verykios et al. (2004) proposed five aspects, which they called *dimensions*:

1. *Data distribution* specifies, if and how data is distributed. Possible subcategories are no distribution, a horizontal distribution, and a vertical distribution. For a dataset with instances $\vec{x}_1, \dots \vec{x}_N$ and attributes $X_1, \dots, X_m$, a horizontal distribution affects $X_1, \dots, X_m$ and a vertical distribution affects $\vec{x}_1, \dots, \vec{x}_N$.

2. *Data modification* specifies the scheme of modifying data. This includes data perturbation techniques (e.g., adding noise or performing geometric transformations), aggregating values, swapping values among data instances, and obfuscating values.

3. *Data mining algorithm* is the task that needs to be performed, e.g., classification, clustering, or pattern mining.

4. *Data or rule hiding* specifies whether data or rules should be hidden.

5. *Privacy Preservation* defines the concrete technique for modifying the data. These techniques are further categorized into heuristic-based, cryptography-based, and reconstruction-based approaches.

In the following, we will first present some general data modification techniques (Section 2.5.1), describe how these techniques are used to perform a specific data mining task (Section 2.5.2), and discuss how to perform data mining on horizontally and vertically distributed data (Section 2.5.3).

2.5.1 Data Modification

Data modification aims at hiding sensitive information by modifying the data accordingly. It typically happens before applying a data mining algorithm. In this subsection, we focus on the three main directions: *randomization*, *linear-transformation-based approaches*, and *aggregation and merging*.

Randomization

Randomization aims at modifying data, so that only statistics can be disclosed (Aggarwal and Yu, 2008). It originates from a survey technique proposed by Warner (1965) (Adam and Wortmann, 1989), where sensitive information needs to be acquired from its participants. It assumes that the participants have to provide some binary information and are reluctant to share it. To increase the cooperation, Warner suggests to disguise the response of a participant P using a known probability distribution f. At the end of the survey, the interviewer only obtains a probability distribution f' over the responses, from which statistical properties (e.g., expectation or variance) of the true probability distribution can be determined. This way, P can provide an honest response without disclosing any sensitive information.

Subsequently, this idea inspired many others to more general settings. For example, given a numeric attribute X and instantiations $x_1, \ldots, x_N$ of X, Traub et al. (1984) proposed to perturb the data with independent samples from a probability distribution f with zero mean and some fixed variance σ^2. In particular, they draw N independent samples $e_1, \ldots, e_N$ from f and add them to the instantiations, i.e., $x_i + e_i$ for $1 \leq i \leq N$. Since the mean of f is zero and σ^2 can be controlled, statisticians can determine statistical properties, with an error only depending on the sample size. However, this way of adding noise has two main disadvantages: (1) Knowing f is one of the key points to reconstruct the original data distribution, but it is also its major weakness (Kargupta et al., 2003). Since all data points are modified by the same f, it is possible to apply a filtering technique that mostly reduces the noise induced by f. Hence, with a sufficient number of perturbed data values, the actual data values can be reconstructed quite accurately. (2) Randomization may cause large biases for conditional means (Matloff, 1986). For example, let X be a numeric attribute and X' be the perturbed version of X. If the distribution for disturbing the data is symmetric and the distribution of the data is strictly decreasing, then the conditional expectation $\mathbb{E}(X \mid X' = x_1)$ is strictly smaller than x_1 (Adam and Wortmann, 1989).

To prevent that the original data can be reconstructed using f, Dowd et al. (2006) proposed an alternative perturbation method based on *random substitution*s. Instead of using a single distribution for each attribute X, one distribution per data value is generated. In the end, the user obtains the perturbed data $\tilde{x_1}, \ldots, \tilde{x_N}$ and a matrix $[M_{i,j} = Pr(v_j \rightarrow v_i)]_{N \times N}$, with M_{i_j} being the probability that $v_j \in values(X)$ is replaced by $v_i \in values(X)$. By combining the perturbed data and M, one can reconstruct the original data distribution but not the original data.

A related but different way of introducing noise to the data is multiplicative noise (Liu et al., 2006). In its simplest form, the data values are perturbed by computing $X_i \cdot e_i$ for $1 \leq i \leq N$, where e_i is drawn from a truncated Gaussian distribution with mean 1 and a small variance σ^2 (Muralidhar et al., 1995).

Linear-Transformation-Based Approaches

The additive and multiplicative perturbation techniques presented above do not preserve the distances between data values (Liu et al., 2006), which impedes data mining tasks such as clustering. Therefore, some researchers considered linear transformations as an alternative. For example, Oliveira and Zaïane (2010) and Rajalaxmi and Natarajan (2008) used a family of geometric transformations to modify the data. As basic building blocks, they used translation, scaling, and rotation, all of which can be expressed by matrices. Unfortunately, these geometric transformation are not protected against various attacks (Chen and Liu, 2011). For example, if some prior knowledge is available (e.g., max and min values of each column), an independent component analysis could be used to reconstruct the original data. Other kinds of attacks also exploit some additional knowledge and try to infer data from the distances of the perturbed data (Liu et al., 2006; Chen and Liu, 2011).

Instead of applying geometric transformations, Xu et al. (2006) reduced the dimensionality of the data matrix by applying singular value decomposition and obtained a new matrix, which is still representative of the original data. As further improvement, they also set entries close to zero to zero, so that data mining algorithm can focus on the important features and it becomes more difficult to estimate the original data matrix.

Following a similar direction, Liu et al. (2006) reduced the dimensionality of the data by applying a random-projection-based multiplicative perturbation. It is based on the Johnson-Lindenstrauss Lemma (Johnson and Lindenstrauss, 1984) and allows to preserve the distance between data points in Euclidean space.

Aggregation and Merging

Even if data is perturbed by randomization techniques, it is still possible to combine the data with publicly available information to reveal the identity of outliers (Aggarwal, 2007). For example, a perturbed database on politicians may easily disclose information on current female heads of government. To protect these entities, Sweeney (2002) introduced the notion of *k-anonymity*. It demands that, for a set of attributes $\mathcal{Q}$ that could be used to identify entities, there are at least k instances for every possible attribute-value combination over $\mathcal{Q}$. The attributes in $\mathcal{Q}$ are called *quasi-identifiers* and are typically something like age, gender, or occupation. Sweeney suggested two strategies to ensure k-anonymity: removing attributes from the data and grouping attribute values (e.g., age ranges instead of individual ages).

Although k-anonymity is a first step towards privacy-preserving data mining, it still gives ways to different types of attacks that disclose confidential data, e.g., background knowledge can be exploited to reduce the number of entries by combining several attributes (Machanavajjhala et al., 2007). Moreover, k-anonymity has the additional disadvantages (1) that the sensitive attribute is known and (2) that the data utility for high-dimensional datasets can be substantially reduced (Aggarwal, 2005), since large

unrelated regions are possibly merged, if data is sparse.

To address some of these problems, several extensions have been developed. One of the strongest properties is the *t-closeness* model (Li et al., 2007), which demands that, for a certain attribute-value combination, the distribution of a sensitive attribute does not deviate more than a threshold t from the corresponding global distribution. Although this property results in strong privacy guarantees, it can easily reduce the utility of the data substantially due to the necessary changes, thereby making it unsuitable for real-world applications (Soria-Comas and Domingo-Ferrer, 2013). As an alternative, a more general framework has been proposed, which allows to trade data privacy for data utility and give the user control over this trade-off. It is based on the notion of ϵ-differential privacy and basically demands that the presence of a data record should only have a limited effect on queries posed on this data. There are several settings (e.g., non-interactive and interactive), but one commonly assumes that a so-called *curator* or *sanitizer* is placed between the database and the user. Whenever the user poses a query, the curator performs the query and modifies the response to ensure ϵ-differential privacy. More precisely, let K be the function that performs queries and modifies the result. If there are two datasets S_1 and S_2, such that S_1 can be transformed into S_2 by removing or adding a single row, then ϵ-differential privacy demands that the probability that K takes a certain value from $range(\mathrm{K})$ is controlled by ϵ. To comply with this property, K could add noise to the result of a query, e.g., using a Laplace distribution.

Definition 14. (Dwork, 2006) A randomized function K gives ϵ-differential privacy, if for all datasets S_1 and S_2 differing in at most one element, and all $S \subseteq range(\mathrm{K})$:

$$Pr[\mathrm{K}(S_1) \in S] \leq e^{\epsilon} \cdot Pr[\mathrm{K}(S_2) \in S]. \tag{2.8}$$

Since ϵ-differential privacy is sometimes too strict, a relaxation has been proposed (Dwork et al., 2006), which adds an additive term δ on the right-hand side of Equation 2.8.

2.5.2 Data Mining

Privacy preservation comes at the cost of data utility, thereby affecting data mining methods that aim at discovering patterns from the data. In this subsection, we present methods for classification, clustering, and association rule mining that either work on a perturbed version of the data or work on the original data but modify their results to ensure privacy preservation.

Classification

Although researchers have been already exploring data privacy for decades, the first data mining algorithm was only proposed in 2000 in the context of decision tree learning (Agrawal and Srikant, 2000). The authors assumed that sensitive attributes have

been modified by two data perturbation methods: *value-class membership* and *value distortion*, where the former means grouping attribute values into intervals and the latter means adding a value drawn according to some known probability distribution f_p. Since the original data distribution f is no longer available, they reconstruct the original distribution using their knowledge of f_p: Bayes' rule yields the posterior probability of each data value x given its perturbed version $\tilde{x}$. By averaging over all data values, one obtains a formula containing the densities f and f_p, of which only f is unknown. To approximate f, Agrawal and Srikant start with a uniform distribution and iteratively update f until a statistical test considers two consecutive updates as insignificant.

As already pointed out in Section 2.5.1, knowing f_p is one of the key points to reconstruct the original data distribution, but it is also its major weakness (Kargupta et al., 2003). To address this issue, Dowd et al. (2006) proposed an alternative perturbation method based on *random substitution*s (see Section 2.5.1). Using the perturbed data and the matrix, the original data distribution can be reconstructed.

However, reconstructing the original data distribution is not always necessary, as data mining can also be performed on perturbed data. In case of classification, Liu et al. (2009) proposed to perform decision tree learning directly on the perturbed data, which is in line with the initial idea of data perturbation proposed by Warner (1965). Liu et al. basically combined the ideas of Warner (1965) and Traub et al. (1984) and applied them to Naive Bayes classification (Liu et al., 2008) and decision tree learning (Liu et al., 2009). Another way to perform data mining on perturbed data has been proposed by Aggarwal and Yu (2004). They created anonymized data from so-called *condensed groups*, which have at least k records and provide statistical information about their records. Using these groups, a dataset can be generated that has similar properties as the original one but does not disclose any information about individual records.

Clustering

Oliveira and Zaïane (2010) were the first who performed clustering on perturbed data with numerical attributes. They considered a family of geometric data transformation that modify the data without distorting the distances between the objects. The basic building blocks they used were translation, scaling, and rotation, which can be expressed by applying matrices to the data. Rajalaxmi and Natarajan (2008) pursued a similar direction but extended the method in two aspects: they selected the geometric transformation randomly and they took nominal attributes into account by flattening the attribute to a binary vector and representing this vector as a numeric vector.

Instead of transforming data geometrically, clustering can also be performed on a lower dimensional version of the data. Liu et al. (2006) and Xu et al. (2006) proposed dimensionality reduction methods to obtain a perturbed matrix that still allows to perform clustering or other data mining tasks.

Association Rules

Privacy-preserved association rule mining can be classified into two main directions: association rule mining on perturbed data and association rule hiding (i.e., removing association rules from the result to ensure privacy-preservation). Most of the work, however, is more concerned with association rule hiding, because even if the data was perfectly perturbed, data instances could still be recovered by running a mining algorithm with different settings. One of the few exceptions is the work by Evfimievski et al. (2002), who extended a randomization approach (Warner, 1965; Agrawal and Srikant, 2000) to mining association rules from perturbed data, and a method proposed by Li and Liu (2009). The objective of association rule hiding is to return only non-sensitive association rules to the user, which is either accomplished by sanitizing the transactions of a database or by sanitizing the rules returned by an algorithm. The latter usually means deleting association rules until no sensitive rules can be inferred (Oliveira et al., 2004), whereas the former requires to change the support or the confidence of a rule (Verykios et al., 2004).

Atallah et al. (1999) proved that transforming a database S into a database S' such that the rules R_s are hidden and as many of the rules $R \setminus R_s$ as possible are still visible in S' is NP-hard. Therefore, many researchers pursued heuristic approaches. Atallah et al. (1999) and Oliveira et al. (2004) suggested a method that removed sensitive association rules based on the structure of the itemset graph. It continues until no sensitive rule is returned and none of the rules can be used to infer sensitive rules. Verykios et al. (2004) and Saygin et al. (2001) proposed to modify the transactions of the database to change the support and confidence of a rule. More specifically, Verykios et al. (2004) proposed to add and remove items, and Saygin et al. (2001) proposed to replace items by unknowns. Jain et al. (2011), on the other hand, suggested to directly change the left-hand and the right-hand side. In the context of frequent itemsets mining, Sun and Yu (2007) used the border to monitor the effect of changing transactions. For this purpose, they compute the impact of each modification and aim at selecting modifications with minimal impact to the border. Gkoulalas-Divanis and Verykios (2009) also pursued a border-based approach, but they formulate the hiding of rules as a constraint satisfaction problem, which they solved by binary integer programming.

A consequence of these heuristic procedures is a number of side effects that come with the sanitization process (Agrawal and Srikant, 2000; Jain et al., 2011). With every change made to lower the support or confidence of a sensitive rule, there is the risk that non-sensitive rules are hidden and that non-existent rules are introduced (Agrawal and Srikant, 2000). Wu et al. (2007) tried to limit these side effects but were not able to eliminate them completely. To avoid side effects, Guo et al. (2006) applied the idea of inverse frequent itemset mining to the FP-tree of the database. Inverse frequent itemset mining follows a reconstruction-based approach and basically boils down to finding a dataset that has the same support for a given set of frequent itemsets and a lower support for the remaining itemsets (Xu et al., 2014).

2.5.3 Distributed Data Mining

The methods presented in the previous subsections aim at data residing in a single place. In some cases, however, data is distributed over multiple parties, and each of these parties wishes to benefit from collaborations with other parties without providing any of its valuable data. Therefore, there is a growing interest in developing algorithms that operate on distributed data – potentially involving multiple parties. The methods are mainly categorized into horizontally distributed data (i.e., distributing the attributes) and vertically distributed data (i.e., distributing the data instances).

Following this trend, many data mining tasks are now extended to the multi-party setting, often employing techniques from the area of multi-party computation. This includes work on classification (Lindell and Pinkas, 2002; Emekçi et al., 2007; Du and Zhan, 2002; Wang et al., 2006; Vaidya et al., 2008), pattern mining (Vaidya and Clifton, 2002; Kantarcioglu and Clifton, 2004; Tassa, 2014), and clustering (Vaidya and Clifton, 2003; Jha et al., 2005; Lin et al., 2005; De and Tripathy, 2013). Since our work mostly deals with Hoeffding trees, we focus our literature review on decision trees.

Horizontally-Distributed Data

Lindell and Pinkas (2002) were the first to design a decision tree learner for data that is horizontally distributed over two parties. The main idea is to adapt the ID3 algorithm by computing the information gain in a distributed manner. Let S be a database, X be a class attribute, and Y be an attribute. $H(S \mid X)$

$$
\begin{aligned}
&= \sum_{v_1 \in V_Y} \frac{|S_{Y=v_1}|}{|S|} \cdot H(S_{Y=v_1}) \\
&= \frac{1}{|S|} \sum_{v_1 \in V_Y} |S_{Y=v_1}| \cdot \sum_{v_2 \in V_X} -\frac{|S_{Y=v_1, X=v_2}|}{|S_{Y=v_1}|} \cdot \log\left(\frac{|S_{Y=v_1, X=v_2}|}{|S_{Y=v_1}|}\right) \\
&= \sum_{\substack{v_1 \in V_Y \\ v_2 \in V_X}} \frac{|S_{Y=v_1, X=v_2}|}{|S|} \cdot \log(|S_{Y=v_1, X=v_2})|) + \sum_{v_1 \in V_Y} \frac{|S_{Y=v_1}|}{|S|} \cdot \log(|S_{Y=v_1}|).
\end{aligned}
$$

Here, $V_Y = values(Y)$, $V_x = values(X)$, $S_{Y=v_1}$ are those instances in S where attribute Y has value v_1, and $S_{Y=v_1, X=v_2}$ are those instances in S where attribute Y has value v_1 and attribute X has value v_2. Reformulating the entropy this way allows each party to compute $S_{Y=v_1, X=v_2}$ and $S_{Y=v_1}$ on their own, without disclosing the actual data values. To also ensure that nothing can be learned about a party's data when computing the information gain, Lindell and Pinkas additionally redesigned the main computation steps of ID3 to be privacy-preserving, namely (1) *finding the best split attribute* and (2) *finding the majority class of a leaf*. To do so, they used tools from the area of multi-party computation, such as computing random shares and Yao's two-party protocol (Yao, 1986), where the latter builds on the 1-out-2 oblivious transfer protocol (Even et al., 1985).

Unfortunately, extending the protocol by Lindell and Pinkas to several parties would involve a large communication overhead (Pinkas, 2002). Therefore, Emekçi et al. (2007) pursued a different approach to constructing a decision tree. They used Shamir's secret sharing (Shamir, 1979), which divides a secret to be shared (assumed to be a number) into pieces and recombines them later using polynomial interpolation.

Vertically-Distributed Data

Inspired by the privacy-preserving ID3 algorithm for horizontally-distributed data (Lindell and Pinkas, 2002), several researchers aimed at designing decision tree learners for vertically-distributed data. For example, Du and Zhan (2002) proposed an algorithm that constructs a decision tree over two parties A and B. Both parties do not want to disclose any of their data except the attribute names and the class labels. To achieve this, two aspects of decision tree learning are modified: computing the information gain and partitioning the data after a split. Since computing the information gain involves subset sizes of the data and the data is not directly available, Du and Zhan expressed the computation of the information gain as scalar product of vectors that can be computed by each party separately. For every split, A (B) computes a vector V_A (V_B) of length N, where the i-th entry is true, iff $\vec{x}_i$ matches the attributes values of A that occurred on the path from the root of the tree to the current split node. Additionally, one of the parties computed for each class label c_j a vector V_{c_j}, where the i-th entry of this vector is true, iff $\vec{x}_i$ has label c_j. Two negative aspects of this procedure are the assumptions that every party needs to know the class attribute and the tree is publicly available in the end.

A more general setting of vertically-distributed data has been considered by Wang et al. (2006) and by Vaidya et al. (2008). Both allowed multiple parties and arbitrary partitionings of the attributes. But instead of creating a publicly available decision tree from the data, they distributed the tree over all parties and only made its basic structure publicly available. The structure mapped each node to a party without disclosing the underlying split attribute. Whereas Wang et al. (2006) uses the transaction identifiers to propagate splits to other parties, Vaidya et al. (2008) use so-called constraint sets, which allow each party to keep track of the instances that belong to a node.

Part I

Density Estimation

Chapter 3

Discrete Densities

Online density estimators are the corner stones of our work. In Chapter 3 (discrete case) and 4 (continuous case), we propose estimators using classifier chains to represent joint densities. They build on the product rule and express a joint density $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$ as a product of conditional densities, such that

$$f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l) = f_1(X_1 \mid Y_1, \ldots, Y_l) \cdot \prod_{i=2}^{m} f_i(X_i \mid Y_1, \ldots, Y_l, X_1, \ldots, X_{i-1})$$

and each of the factors can be estimated by a classifier providing class probability estimates (see Section 3.2). Although several types of classifiers would be suitable for this purpose, we will solely focus on Hoeffding trees (Domingos and Hulten, 2000), since they are able to deal with data streams, facilitate the application of inference operations (see Chapter 6), and are suitable for the goal of online density estimation, i.e., minimization of the KL-divergence between the true density and the estimate. The latter may be surprising at first, since Hoeffding trees use heuristic measures such as the information gain or Gini index to find a well-performing structure. However, the heuristic measure only affects the size of the tree and how quickly the estimator provides good estimates, whereas minimizing the KL-divergence is ensured by the law of large numbers (see Section 3.5).

Because a single density estimator may not provide a reliable estimate, we also apply ensemble techniques (see Section 3.3) and weight the ensemble members according to their current performance using the log-likelihood (see Section 3.4). As in the case of Hoeffding trees, this will not affect the consistency of the estimator but only the number of instances that are required to provide good estimates as early as possible. The KL-divergence is still minimized, since every ensemble member minimizes the KL-divergence (see Section 3.5).

3.1 Problem Statement

Let $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$ be a joint density with discrete variables $X_1, \ldots X_m$. In this chapter, we address online density estimation from data streams $\mathit{stream}(f)$, where the $\vec{x}_i \in \mathit{stream}(f)$ are drawn independently and identically distributed (IID) from f. Moreover, we assume that $\mathit{stream}(f)$ is generated by a single density, i.e., there are no data distribution drifts in the stream.

3.2 Classifier Chains

As a first step, we provide an algorithm that employs a single classifier chain to determine a density estimate. Let $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$ be a density with discrete random variables. Then we can apply the product rule and obtain the following equality:

$$f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l) = f_1(X_1 \mid Y_1, \ldots, Y_l) \cdot \prod_{i=2}^{m} f_i(X_i \mid Y_1, \ldots, Y_l, X_1, \ldots, X_{i-1}). \tag{3.1}$$

In other words, in order to model the density f, it is sufficient to model the densities $f_1(X_1 \mid Y_1, \ldots, Y_l)$ and $f_i(X_i \mid Y_1, \ldots, Y_l, X_1, \ldots, X_{i-1})$, $i \in \{2, \ldots, m\}$. For each f_i, $1 \leq i \leq m$, we employ classifiers that return class probability estimates, and, in particular, Hoeffding trees (Domingos and Hulten, 2000). In case the density changes over time, one can employ classifiers that are able to deal with concept drift such as the Concept-adapting Very Fast Decision Tree learner (Hulten et al., 2001).

Based on the classifier chain implied by Equation 3.1, the proposed algorithm initializes the base classifiers for $f_1, \ldots, f_m$. When the algorithm receives an example $(\vec{y}, \vec{x})$ from an instance stream, it produces m instances, where instance i contains the features $Y_1, \ldots, Y_l, X_1, \ldots, X_i$. The instance $(\vec{y}, x_1)$ is forwarded to the classifier for f_1 and the instance $(\vec{y}, x_1, \ldots, x_i)$ is forwarded to the classifier for f_i, $1 \leq i \leq m$. Subsequently, each classifier processes its example and updates its current estimate.

3.3 Ensembles of Classifier Chains

The product on the right-hand side of Equation 3.1 is only one way to represent the discrete joint density – there are many other possibilities. Let $\pi : \{1, \ldots, m\} \to \{1, \ldots, m\}$ be a bijective mapping. Then

$$f_1(X_{\pi(1)} \mid Y_1, \ldots, Y_l) \cdot \prod_{i=2}^{m} f_i(X_{\pi(i)} \mid Y_1, \ldots, Y_l, X_{\pi(1)}, \ldots, X_{\pi(i-1)}) \tag{3.2}$$

is an equivalent representation of $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$. In other words, we simply use a different ordering of the features to represent the discrete joint density, which then results in a different classifier chain. Although all such products represent the same joint density assuming the true conditional density estimates are known, the ordering may be important for the performance of our classifiers: Ideally, the ordering enables the classifiers to exploit conditional independence relationships, so that some of the features can be disregarded. Figure 3.1 provides an illustration for this statement. On the left is a Bayesian network with six variables. On the right are three possible orderings of the variables. Dependent on the ordering, certain interdependencies of the Bayesian network can be represented by the corresponding classifier chain. The remaining interdependencies can only be respected by learning *inverse relationships*,

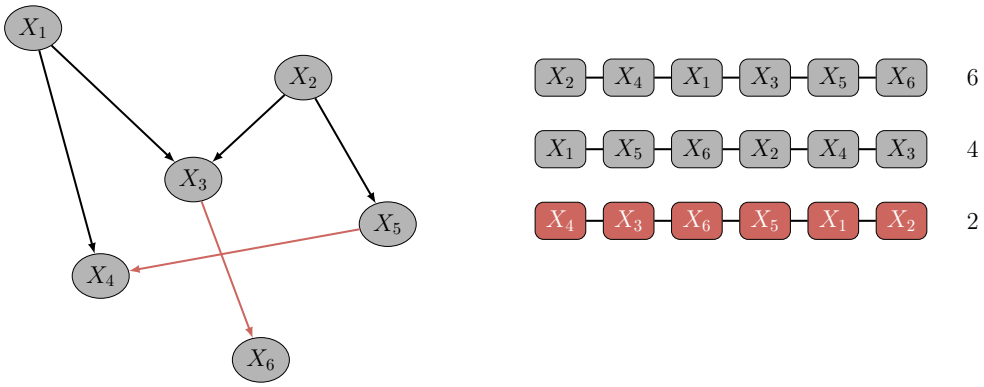


Figure 3.1: On the left-hand side, the figure shows a Bayesian network representing a joint density with six discrete variables. On the right-hand side are three possible orderings of the variables plus the corresponding number of interdependencies of the Bayesian network that are represented in the correct ordering (i.e., $x_i \leq x_j$ in the chain, if there is a sequence of directed edges $x_i \rightarrow \ldots \rightarrow x_j$ in the Bayesian network). As illustration, we highlighted the interdependencies represented by the last variable ordering.

which potentially makes learning more difficult for the classifier (e.g., $X_6 \rightarrow X_2$ instead of $X_2 \rightarrow X_3$ together with $X_3 \rightarrow X_6$). The experiments in Section 3.6.2 provide evidence for this claim.

Hence, to increase robustness, we consider a second algorithm that generates several classifier chains and combines their estimates to a single density estimate. This algorithm, which generates ensembles of classifier chains, simply samples chains from the set of possible feature orderings at random and averages the density estimates obtained.

Although the number of possible orderings is exponential in the number of features, ensembles do not seem to require an exponential number of chains. In a set of experiments (see Sections 3.6.1 and 3.6.2), we found that more than 10 classifier chains only yielded negligible improvements – if any. This is in line with other results from the ensemble literature. Bauer and Kohavi (1999) and Frank and Kramer (2004) reported, for example, that ensembles of at most 25 respectively 20 ensemble members were sufficient to obtain a solution sufficiently close to the optimum.

3.4 Ensembles of Weighted Classifier Chains

An ensemble of classifier chains computes the average over all classifier chains, thereby compensating for chains with insufficient performance. If we were able to weight the chains according to their current performance, it could further improve the density estimation. One possible approach is the *exponentiated gradient investment strategy* (EG investment strategy) (Cesa-Bianchi and Lugosi, 2006), known from the area of

Algorithm 3: Updating the weights of classifier chains.

Input: Let $cc_1, \ldots, cc_k$ be classifier chains. Further, let $(w_t[1], \ldots, w_t[k]) \in \mathbb{R}^k$ be the corresponding weight vector, $(\vec{y}_t, \vec{x}_t)$ be the previous instance, and $(\vec{y}_{t+1}, \vec{x}_{t+1})$ be the current instance.

```
// a parameter controlling the weight update
```
1 $\eta \leftarrow (c/C) \cdot \sqrt{(8 \cdot \ln k)/N}$

```
// the relative change of the density value
```
2 **for** $i = 1, \ldots, k$ **do**

3 $\quad p[i] \leftarrow \frac{\hat{f}_{cc_i}(\vec{y}_{t+1}, \vec{x}_{t+1})}{\hat{f}_{cc_i}(\vec{y}_t, \vec{x}_t)}$

4 **end**

```
// the weight update
```
5 $wp \leftarrow \sum_{j=1}^{k} w_t[j] \cdot p[j]$

6 **for** $i = 1, \ldots, k$ **do**

7 $\quad w_{t+1}[i] \leftarrow \frac{w_t[i] \cdot \exp(\eta \cdot p[i]/wp)}{\sum_{j=1}^{k} w_t[j] \cdot \exp(\eta \cdot p[j]/wp)}$

8 **end**

online learning, which is an investment strategy for the *portfolio selection problem*. The general setting is a stock market consisting of k stocks. Each day, a so-called *market vector* $s_{t+1} := (s_{t+1}[1], \ldots, s_{t+1}[k]) \in \mathbb{R}_+^k$ is released, where $s_{t+1}[i]$ is the relative price of the i-th stock, i.e., the increase of the price compared to the previous trading day. Based on this market vector, the investor is allowed to re-distribute her wealth among the k stocks every day, so that her wealth is maximized over a period of N trading days. The success of a strategy is measured as the relative gain or loss per trading day. When multiplied over all trading days, this yields the so-called *wealth factor*, which is used to compare investment strategies. Assuming that the market vectors are IID, the wealth factor of the best possible investment strategy (according to (Cesa-Bianchi and Lugosi, 2006; Cover and Thomas, 2006)) divided by the EG investment strategy is bounded by $\frac{C}{c} \cdot \sqrt{\frac{N}{2} \ln(k)}$, where c is the smallest observed relative price and C is the largest observed relative price (Cesa-Bianchi and Lugosi, 2006).

Algorithm 3 applies the EG investment strategy to the problem of weighting classifier chains. It is basically a re-interpretation of the portfolio selection as a method to weight ensemble members. In line 1, the parameter η is defined, which controls the weight update for each chain. Cesa-Bianchi and Lugosi (2006) proposed to set η to $(c/C) \cdot \sqrt{(8 \cdot \ln k)/N}$ (line 1), where c is the smallest value observed for $p[i]$ and C is the largest value, $1 \leq i \leq k$. c and C can both be estimated based on some initial buffer and can also be updated over time. The remaining part of the algorithm performs the weight update. In lines 2-4, the relative change is computed using the previous and the current density value of each chain. In lines 5-8, the next weight vector is computed

as proposed by Cesa-Bianchi and Lugosi (2006).

If we have to compute the probability of an instance, we select only those classifier chains whose weight does not deviate more than a certain percentage from the highest weight (for this thesis, we allow a deviation of 30%). This last step is supposed to reduce the influence of poorly performing classifier chains (see Subsection 3.6.1). Then we simply combine the probabilities of the individual classifier chains with respect to their weights, renormalizing again as necessary (lines 5-8). Notice that the ensemble of weighted classifier chains that we proposed is just one way of introducing weights into classifier chains. For instance, one can also use the weights to rank the classifier chains and pick the best or the three best classifier chains to construct a density estimate.

3.5 Consistency

In the context of density estimation, the aim is to obtain consistent estimators, where the estimate approaches the true density with increasing numbers of instances. In the following, we prove that the estimators from the previous sections fulfill this property. As each of them can be combined with many different base classifiers, the proof will be independent of a specific base classifier. Instead, we will prove that the estimators are consistent as long as the underlying base classifiers are.

First, we consider estimators employing a single classifier chain. We show that the conditional KL-divergence of the density f and its estimate $\hat{f}$ tends to 0 for increasing numbers of instances.

Theorem 1. *Let f be a discrete joint density with $f(\vec{x} \mid \vec{y}) = f_1(x_1 \mid \vec{y}) \cdots f_m(x_m \mid \vec{y}, x_1, \ldots, x_{m-1})$. Further, let $\hat{f}$ be an estimator employing a single classifier chain, and let $\hat{f}_i$ be the estimate of the i-th classifier in the classifier chain. If the number of instances tends to infinity and $KL_{cond}(f_i, \hat{f}_i) \to 0$, for all $i \in \{1, \ldots, m\}$, then $KL_{cond}(f, \hat{f}) \to 0$.*

Proof. For increasing numbers of instances, we will prove that $\mathrm{KL}_{\mathrm{cond}}(f, \hat{f}) \to 0$, if $\mathrm{KL}_{\mathrm{cond}}(f_i, \hat{f}_i) \to 0$ for all $i \in \{1, \ldots, n\}$ (in the following, $\sum_{\vec{x}}$ is a shorthand for the sum over all possible values of the vector $\vec{x}$):

$$\begin{aligned}
\mathrm{KL}_{\mathrm{cond}}(f, \hat{f}) &= \sum_{\vec{y}} \hat{f}(\vec{y}) \cdot \sum_{\vec{x}} \hat{f}(\vec{x} \mid \vec{y}) \cdot \ln \frac{\hat{f}(\vec{x} \mid \vec{y})}{f(\vec{x} \mid \vec{y})} \\
&= \sum_{\vec{y}} \hat{f}(\vec{y}) \cdot \sum_{\vec{x}} \hat{f}(\vec{x} \mid \vec{y}) \cdot \ln \frac{\hat{f}_1(x_1 \mid \vec{y}) \cdot \prod_{i=2}^{m} \hat{f}_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})}{f_1(x_1 \mid \vec{y}) \cdot \prod_{i=2}^{m} f_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})}.
\end{aligned}$$

Since $\ln(a \cdot b) = \ln(a) + \ln(b)$, $\mathrm{KL}_{\mathrm{cond}}(f, \hat{f})$ can be written as

$$\begin{aligned}
\mathrm{KL}_{\mathrm{cond}}(f, \hat{f}) &= \sum_{\vec{y}} \hat{f}(\vec{y}) \cdot \sum_{\vec{x}} \hat{f}(\vec{x} \mid \vec{y}) \cdot \ln \frac{\hat{f}_1(x_1 \mid \vec{y}) \cdot \prod_{i=2}^{m} \hat{f}_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})}{f_1(x_1 \mid \vec{y}) \cdot \prod_{i=2}^{m} f_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})} \\
&= \sum_{\vec{y}} \hat{f}(\vec{y}) \cdot \sum_{\vec{x}} \hat{f}(\vec{x} \mid \vec{y}) \cdot \sum_{i=1}^{m} \ln \frac{\hat{f}_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})}{f_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})} \\
&= \sum_{i=1}^{m} \sum_{\vec{y}} \hat{f}(\vec{y}) \cdot \sum_{\vec{x}} \hat{f}(\vec{x} \mid \vec{y}) \cdot \ln \frac{\hat{f}_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})}{f_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})}.
\end{aligned}$$

By definition, $\hat{f}(\vec{x} \mid \vec{y}) = \hat{f}_1(x_1 \mid \vec{y}) \cdots \hat{f}_m(x_m \mid \vec{y}, x_1, \ldots, x_{m-1})$ and, therefore, the following equalities hold for each i in the above sum:

$$\begin{aligned}
&\sum_{\vec{y}} \hat{f}(\vec{y}) \cdot \sum_{\vec{x}} \hat{f}(\vec{x} \mid \vec{y}) \cdot \ln \frac{\hat{f}_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})}{f_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})} \\
&= \sum_{\vec{y}} \hat{f}(\vec{y}) \cdot \sum_{\vec{x}} \left(\prod_{j=1}^{m} \hat{f}_j(x_j \mid \vec{y}, x_1, \ldots, x_{j-1}) \right) \cdot \ln \frac{\hat{f}_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})}{f_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})} \\
&= \sum_{\vec{y}} \hat{f}(\vec{y}) \cdot \sum_{\vec{x}} \left(\prod_{j=1, j \neq i}^{m} \hat{f}_j(x_j \mid \vec{y}, x_1, \ldots, x_{j-1}) \right) \cdot \\
&\quad \hat{f}_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1}) \cdot \ln \frac{\hat{f}_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})}{f_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})} \\
&\leq \sum_{\vec{y}} \hat{f}(\vec{y}) \cdot \sum_{\vec{x}} \hat{f}_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1}) \cdot \ln \frac{\hat{f}_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})}{f_i(x_i \mid \vec{y}, x_1, \ldots, x_{i-1})} \\
&= \mathrm{KL}_{\mathrm{cond}}(f_i, \hat{f}_i),
\end{aligned}$$

where $\prod_{j=1, j \neq i}^{m} \hat{f}_j(x_j \mid \vec{y}, x_1, \ldots, x_{j-1})$ has been replaced by 1 in the last but one step, which is possible, since $0 \leq \prod_{j=1, j \neq i}^{m} \hat{f}_j(x_j \mid \vec{y}, x_1, \ldots, x_{j-1}) \leq 1$. Also notice that $f(\vec{y}) = f_i(\vec{y})$ for all $i \in \{1, \ldots, m\}$, i.e., each classifier uses the same estimator for $\vec{y}$. Hence, if $\mathrm{KL}_{\mathrm{cond}}(f_i, \hat{f}_i) \to 0$ for all $i \in \{1, \ldots, m\}$, then $\mathrm{KL}_{\mathrm{cond}}(f, \hat{f}) \to 0$. □

Thus, estimators employing a single classifier chain are consistent as long as the underlying base classifiers are. However, proving this property for individual base classifiers is not always easy. A Hoeffding tree with leaf classifiers of type MajorityClass, for example, partitions the instances by the attributes on which splits took place. If the Hoeffding tree constitutes a complete tree, then, according to the law of large numbers, the leaf classifiers approach the true class probabilities of each partition with increasing numbers of instances. Otherwise, the consistency property is fulfilled, if there is a stream prefix of length $N_0 \in \mathbb{N}$, such that, for each branch of that tree, no further pruning takes place and either

- there is no further attribute on which the leaf node can be split, or
- for each path from the root to a leaf, any series of further splits does not change the distribution of the partition belonging to this path.

Next, we extend the statement of Theorem 1 to estimators employing an ensemble of (weighted) classifier chains.

Theorem 2. *Let f be a discrete joint density with $f(\vec{y}, \vec{x}) = f_1(x_1 \mid \vec{y}) \cdots f_m(x_m \mid \vec{y}, x_1, \ldots, x_{m-1})$. Further, let $\hat{f}$ be an estimator employing an ensemble of (weighted) classifier chains, and let $\hat{f}_{i,j}$ be the estimate of the j-th classifier in the i-th classifier chain. If the number of instances tends to infinity and $KL_{cond}(f_i, \hat{f}_i) \to 0$ for all $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, m\}$, then $KL_{cond}(f, \hat{f}) \to 0$.*

Proof. Follows immediately from the definition of an ensemble of (weighted) classifier chains and Theorem 1. Each classifier chain provides a consistent estimate, such that the weights of the classifier chains do not affect the consistency of the ensemble. □

3.6 Evaluation

In this section, we evaluate the algorithms presented in Sections 3.2, 3.3, and 3.4 on synthetic and real-world data. To facilitate comparisons with other methods, we will focus on joint densities, i.e., $f(X_1, \ldots, X_m)$. Conditional densities are not explicitly evaluated, but as they constitute a key component of the proposed density estimators, they are responsible for their overall performance.

The algorithms have been implemented as part of the MiDEO framework[1] (Mining Density Estimates inferred Online, see Chapter 6), which is based on MOA[2] (Bifet et al., 2010). To improve readability, we introduce some notations to refer to specific variants: $\text{EDDO}_T(L)$ is the online density estimators for discrete joint densities that we proposed in the previous sections, where $T \in \{\text{CC}, \text{ECC}, \text{EWCC}\}$, and $L \in \{\text{MC}, \text{NB}, \text{NBA}\}$. The index denotes the type of density estimator, which is either an estimator employing a classifier chain (CC), an ensemble of classifier chains (ECC), or an ensemble of weighted classifier chains (EWCC). If T is not specified, it refers to all types. The L specifies the leaf classifier of the HoeffdingTree, which is MajorityClass (MC), NaiveBayes (NB), or NaiveBayes adaptive (NBA).

In order to compare the performance of the online density estimators to existing ones, we integrated Bayesian structure learners from the `bnlearn` package (Scutari, 2010)[3]. At the time of writing, it contained

- Constraint-based algorithms: Grow-Shrink (GS), Incremental Association (IAMB), Interleaved-IAMB (Inter-IAMB), Fast-IAMB

[1] `https://github.com/kramerlab/mideo`

[2] version 2013.11

[3] Please notice that we also compared the online density estimator with a corresponding batch version. The results are available in Online Resource 1.

Table 3.1: The table summarizes some properties of the datasets. In general, we distinguish two types of data: synthetic and real-world. *Shortcut* is a shorter naming scheme that is mainly used in plots to make axis labels more readable.

Dataset	Shortcut	Type	#Variables	#Instances
Bayesian networks	bn-100		$4-10$	100
Bayesian networks	bn-1000	synthetic	$4-10$	$1,000$
Bayesian networks	bn-10000		$4-10$	$10,000$
movielens			23	$49,282$
pokerhand		real-world	11	$1,025,015$
us-census			68	$2,458,285$

- Score-based algorithms: Hill-Climbing greedy search (HC), Tabu Search (TABU)
- Hybrid algorithms: Max-Min Hill Climbing (MMHC), Two-Phase Restricted Maximization (RSMAX2)
- Local discovery algorithms: ARACNE, Max-Min Parents and Children (MMPC), Semi-Interleaved Hiton-PC (SIHPC), Chow-Liu

The networks returned by these algorithms possibly contain undirected arcs. Therefore, we employ the `pdag2dag` method to direct these arcs before estimating the conditional probability tables (CPT). They are estimated by the `fit` method of the Bayesian network, for which two alternatives are provided: maximum likelihood (`mle`) and Bayesian a posteriori (`bayes`).

In order to compare the density estimates, we used the KL-divergence for small and medium-sized synthetic datasets and the log-likelihood (LL) for larger synthetic datasets and real-world data. Although it is not necessary to compute the KL-divergence to compare density estimates, it has the advantage of providing information how close the density estimate is to the true density. Unfortunately, computing the KL-divergence is computationally expensive, since we have to consider every possible combination of feature values. For instance, if we have a Bayesian network with 8 nodes and 7 values each, then there are already $7^8 = 5,764,801$ combinations, for each of which we have to compute the probability given by the estimate. This computation is very expensive, so that we can only consider discrete joint densities with a small number of nodes and a small number of node values.[4] In case of real-world data, we divided the dataset into training and test set, forwarded the instances from the training set to the estimators, and computed the log-likelihood on the test set.

The properties of the data are summarized in Table 4.1. The Bayesian networks were randomly generated using the `random.graph` method of the `bnlearn` package. Its

[4]Notice that, in order to avoid rounding errors, we computed the KL-divergence using logarithms (Mann, 2006).

parameter `method` was set to `melancon`, which uses a Markov chain to draw acyclic, directed graphs uniformly at random, according to Melançon and Philippe (2004). The Bayesian networks had between 4 and 10 nodes, for each of which 15 networks were considered. From these discrete joint densities, we drew N instances with $N \in \{10^2, 10^3, 10^4\}$. As real-world datasets, we selected three publicly available datasets[5]: movielens, pokerhand, and us-census.

3.6.1 Chain Orderings

As discussed in Section 3.3, the ordering of the chain could have an effect on the performance of the density estimator. In our first experiment, we analyze the influence of the variable ordering with respect to the performance of the online density estimates. We randomly picked 15 Bayesian networks with seven nodes and computed the KL-divergence of 1000 randomly chosen classifier chains for the leaf classifiers MC, NB, and NBA.

Some representative results of this experiment are given in Figure 3.2 (for the remaining results, see Appendix A). In all of the presented cases, we observe large differences between the best and the worst classifier chain, which range between a KL-divergence of roughly 0.2 and 0.65. The percentage of chains performing generally well and chains performing generally poorly varies substantially between the different Bayesian networks and also depends on the base classifiers. For MC, the top-left plot shows roughly 30% of the chains performing relatively poorly and the rest performing well. For MC in the bottom-right plot, the opposite can be observed. Generally, the performance of NB and NBA seems to be more stable, so that the differences are rather small (an exception is NB in the bottom-left plot). The smoothness of the NB and NBA curves can be explained by the way MC classifies instances. Since MC only takes the majority class into account, the differences in the class distributions that are affected by minor changes in the chain orderings are often small, so that several classifier chains can have the same performance.

From the plots, we can conclude that employing a single classifier chain will generally be less effective compared to an ensemble employing several classifier chains. As the idea is to compensate for classifier chains with poor performance and the probability of drawing a good classifier chain is rather high in this experiment, we assume that a small ensemble size should be sufficient to achieve a good performance.

3.6.2 Ensemble Size

For further investigations of this matter, we conducted another experiment and analyzed how many chains are necessary to compensate for poorly performing chains. For this purpose, we generated 100 Bayesian networks with seven nodes and computed the KL-divergence for the ensemble sizes 1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35. Each ensemble was trained with $10^2, 10^3, 10^4, 10^5$ instances.

[5] `http://archive.ics.uci.edu/ml/`

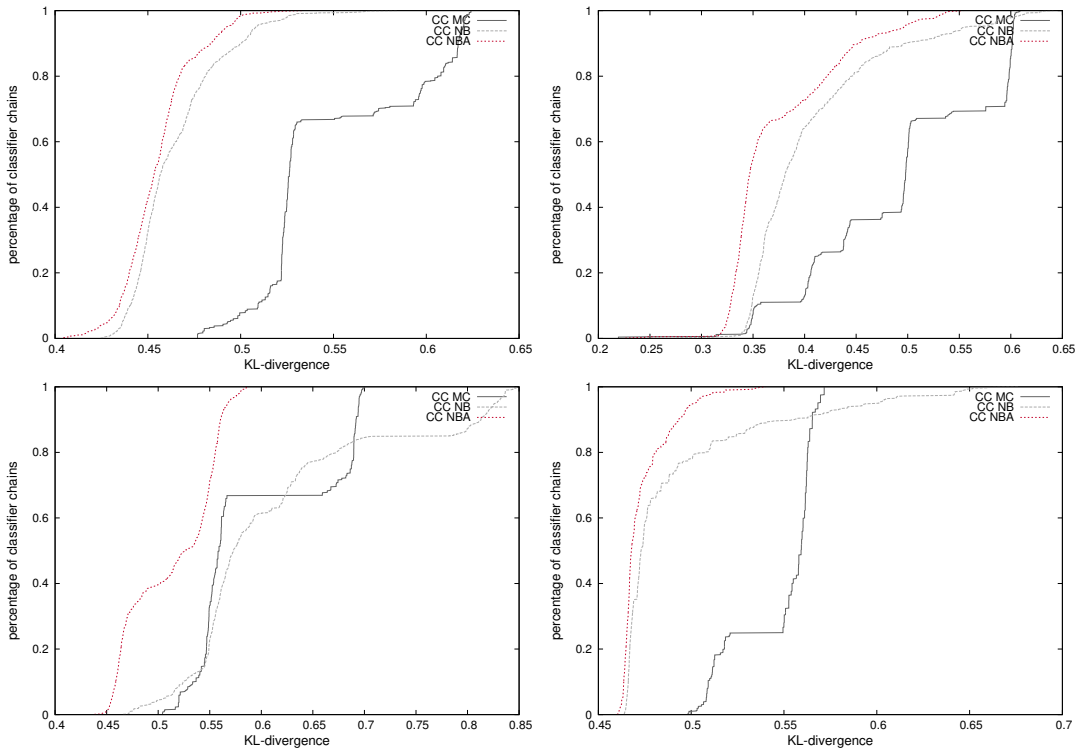


Figure 3.2: Each plot shows the performance of 1000 EDDO_{CC} density estimators for a single randomly-generated Bayesian network with 7 nodes. On the x-axis is the KL-divergence, and on the y-axis is the percentage of classifier chains having a smaller or equal KL-divergence.

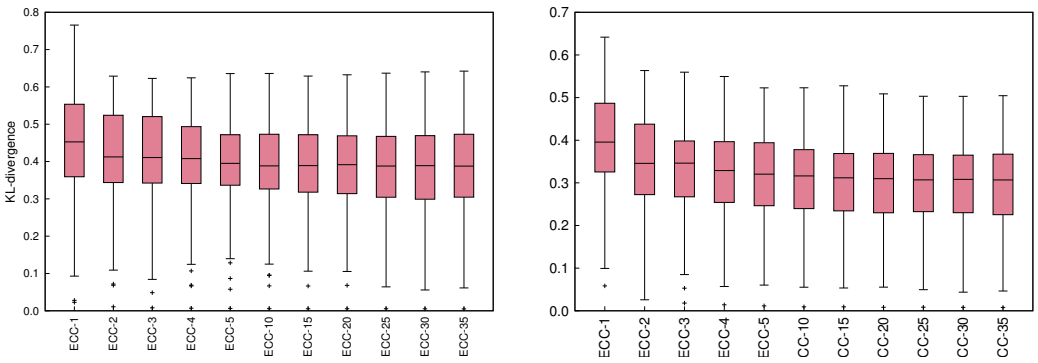


Figure 3.3: The figure shows the KL-divergence of ensemble estimators with various ensemble sizes. It shows the performance of $\text{EDDO}_{\text{ECC}}(\text{MC})$ (left) and $\text{EDDO}_{\text{ECC}}(\text{NBA})$ (right) trained on $100,000$ instances from Bayesian networks with 7 nodes.

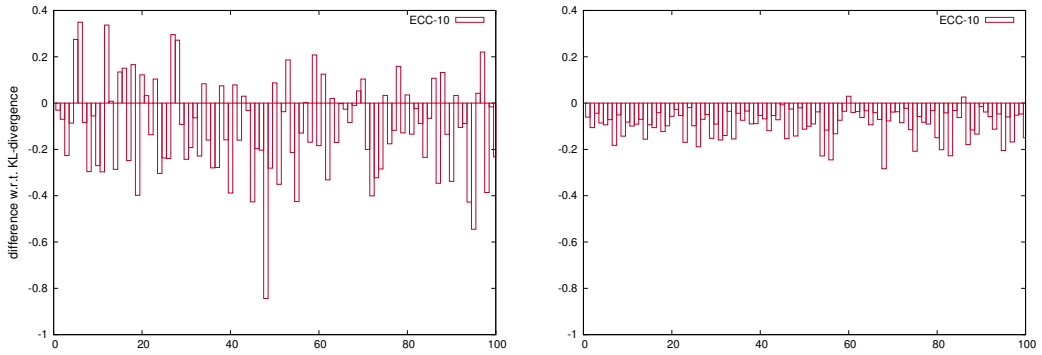


Figure 3.4: The figure shows a direct comparison of ECC-1 with ECC-10 using NBA as leaf classifier when trained with 1000 (left) and 100,000 (right) instances from Bayesian networks with 7 nodes. The x-axis of the plots shows the individual runs of the estimators. The y-axis shows the difference with respect to the KL-divergence.

Figure 3.3 shows the performance of the ensemble estimators when trained on Bayesian networks having 7 nodes and 10^5 instances. As stated earlier, more than ten chains do not seem to provide substantial performance gains – at least for the given synthetic data. In some cases, even four classifier chains seem to be sufficient, which is probably due to the large number of classifier chains showing a good performance.

How the number of instances affects the performance of the ensemble is shown in Figure 3.4, which provides a head-to-head comparison of ECC-10 and ECC-1 when trained on 10^3 respectively 10^5 instances, i.e., $KL(\text{ECC-10}) - KL(\text{ECC-1})$. In both cases, ECC-10 performs better than ECC-1 in the majority of cases. But the differences between the two estimators are rather large, if only few instances are available (plot on the left), and are rather small, if many instances are available (plot on the right). Hence, the more instances are available, the less important is the ordering of the variable. This is in line with our earlier discussion on the variable ordering: From a theoretical point of view, every ordering is equivalent. But some variable interdependencies are more difficult to be represented by the base estimators, if only few training instances are available.

3.6.3 Comparison with Bayesian Structure Learners

In this subsection, we compare EDDO to 12 Bayesian structure learners on synthetic and real-world datasets. Especially on the synthetic data, we expect the Bayesian structure learners to be strong competitors, as they process the instances in an offline fashion and the data is generated from Bayesian networks.

The synthetic datasets are generated from Bayesian networks with between 4 and 10 nodes. For networks with 4 to 7 nodes, we computed the KL-divergence and the average log-likelihood. For Bayesian networks with 8 to 10 nodes, we only computed the average log-likelihood due to running time constraints. As visible in the result plots

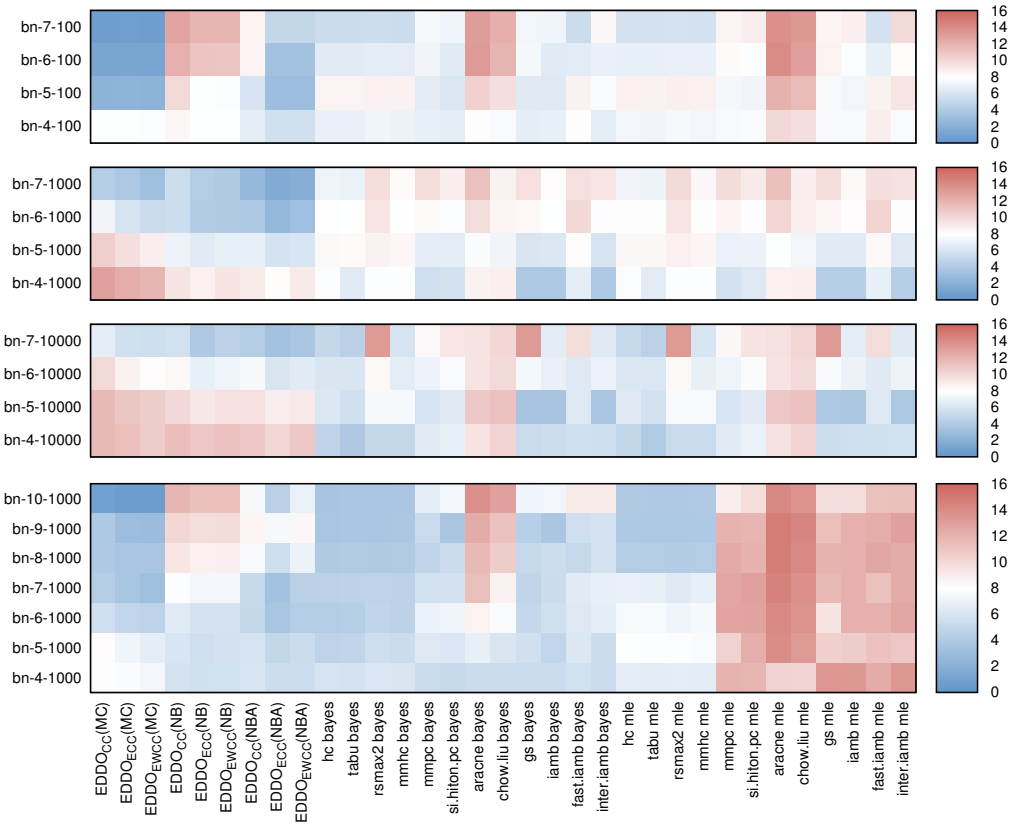


Figure 3.5: The plots show the average rank (lower is better, i.e, blue is better) of different EDDO density estimators compared to 12 Bayesian structure learners, which use either `mle` or `bayes` to estimate the a posteriori probabilities of the CPTs. In the three plots at the top, each estimator has been trained with 10^2, 10^3, or 10^4 instances on datasets generated from Bayesian networks with 4 to 7 nodes (the datasets are denoted as bn-#nodes-#instances). As performance measure, the KL-divergence has been used. In the plot at the bottom, each estimator has been trained with 10^3 instances on datasets generated from Bayesian networks with 4 to 10 nodes. As performance measure, the average log-likelihood has been used.

(see bn-[4|5|6|7]-1000 in Figure 3.5), the ranking of the given methods is affected by the performance measure. The KL-divergence includes every possible instance in the computation of the performance measure, whereas the average log-likelihood measures the performance on the instances available for testing. However, this does not affect the general trends that we observe for EDDO – although it can affect the ranking among the Bayesian structure learners.

The results are summarized in Figure 3.5. Generally, Bayesian structure learners show a better performance when either the networks underlying the datasets are small or many instances are available, which is due to the conditional probability tables of the networks. Each combination of parent values is represented in the table and in order to estimate the probabilities of each combination, sufficient numbers of instances are required.[6] For this reason, compared with Bayesian structure learners, EDDO performs better on datasets originating from networks with 7 nodes than from networks with only 4 nodes. This observation is further supported by the plot at the bottom of Figure 3.5, where the number of nodes in the Bayesian networks was increased up to 10. The larger and the more complex the Bayesian network, the better the performance of EDDO compared to the Bayesian structure learners – at least for MC as leaf classifier. For leaf classifiers following the Bayesian principle, the situation is different. For a fixed number of instances, they first perform better when the networks become more complex. However, if the networks become too complex, this trend is reversed. This is probably due to its independence assumption, which requires more instances to represent the density equally well.

Additionally, we compared EDDO with the Bayesian structure learner on three publicly available datasets: movielens, pokerhand, and us-census. In order to ensure that the instances are IID, we randomized the dataset and repeated the evaluation 15 times. Since movielens and us-census have a larger number of attributes (23 and 68 respectively), some of the Bayesian structure learners were not able to finish within 12 hours. If this affected only one run, we excluded this run. Otherwise, we removed the Bayesian structure learner.

The results are summarized in Figures 3.6, 3.7, and 3.8. On every dataset, the EDDO density estimators perform better on average than all Bayesian structure learners. Similar to the synthetic datasets, Bayesian structure learners show a good performance when many instances and only few attributes are available (pokerhand dataset), and show a worse performance on datasets with many attributes and relatively few instances – few in terms of possible variable-value combinations. On the pokerhand dataset, the benefits of ensembles of classifier chains and ensembles of weighted classifier chains are illustrated. EDDO_{ECC} shows a substantial improvement over EDDO_{CC}, and $\text{EDDO}_{\text{EWCC}}$ in turn shows a further improvement over EDDO_{ECC}. We also observe that the variance of EDDO_{ECC} and $\text{EDDO}_{\text{EWCC}}$ is lower than the variance of

[6] Please note that the problem of having too few examples for accurately estimating the CPTs could be less prominent when the CPTs are replaced by decision trees (Friedman and Goldszmidt, 1996; Su and Zhang, 2006).

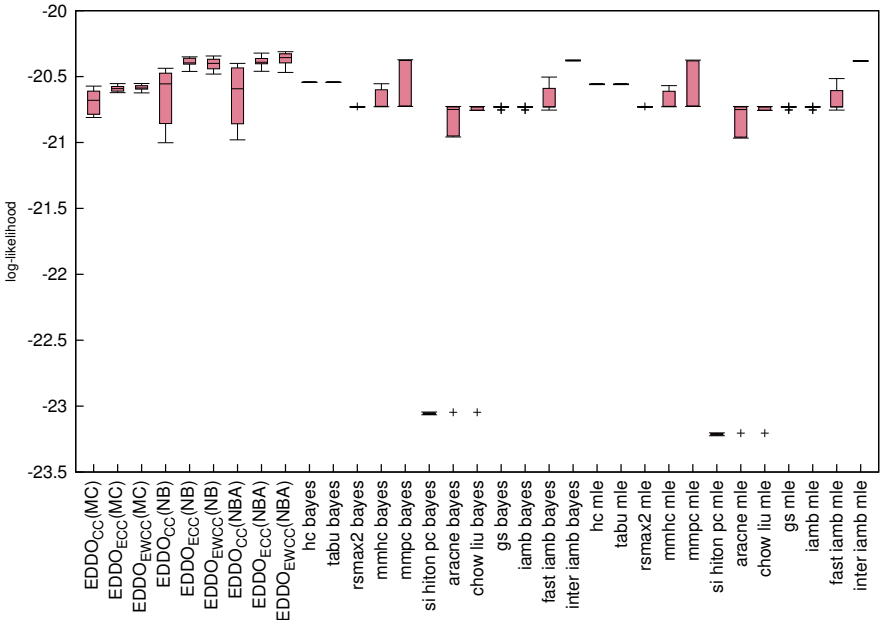


Figure 3.6: A comparison on the pokerhand dataset, where EDDO density estimators have been compared to 12 Bayesian structure learners, which use either `mle` or `bayes` to estimate the a posteriori probabilities of the CPTs.

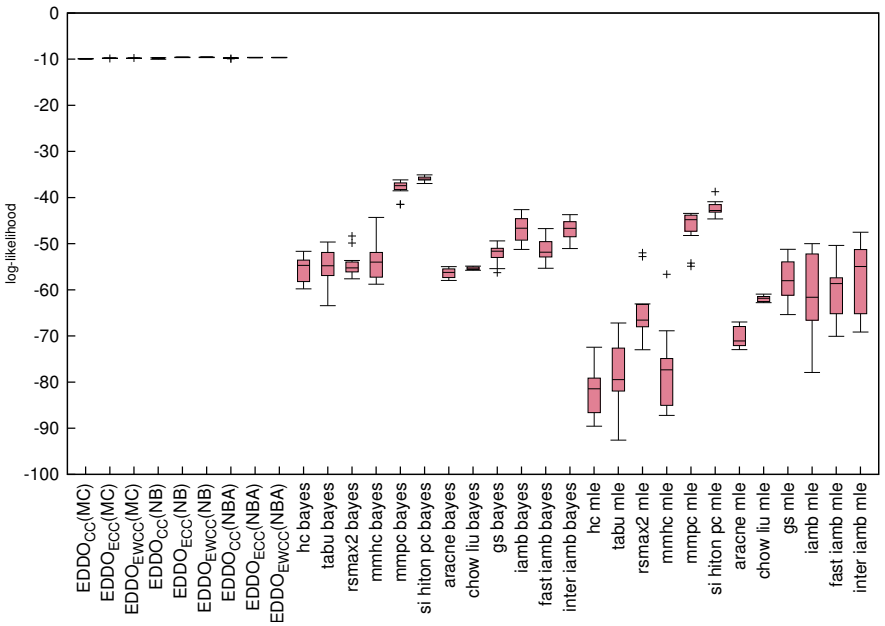


Figure 3.7: A comparison on the movielens dataset, where EDDO density estimators have been compared to 12 Bayesian structure learners, which use either `mle` or `bayes` to estimate the a posteriori probabilities of the CPTs.

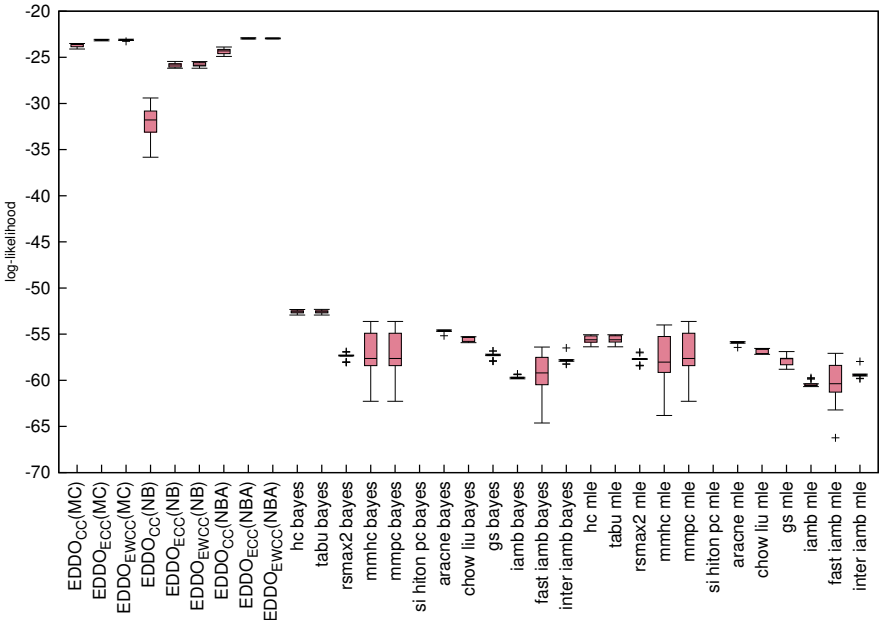


Figure 3.8: A comparison on the us-census dataset, where EDDO density estimators have been compared to 12 Bayesian structure learners, which use either `mle` or `bayes` to estimate the a posteriori probabilities of the CPTs.

most Bayesian structure learners, which makes their performance more stable in practice.

3.7 Conclusion

In this chapter, we proposed a family of online algorithms to estimate joint densities with discrete variables. In particular, we considered three variations: one that uses a random classifier chain, one that uses an ensemble of random classifier chains, and one that uses an ensemble of weighted random classifier chains. We proved the consistency of their estimates and analyzed the behavior of the density estimators by investigating how the ordering and the ensemble size influence the overall performance. It turned out that the difference between the classifier chains can be substantial and that an ensemble of up to 10 classifier chains can make the estimate more robust without demanding too much memory and running time. Moreover, we demonstrated with another experiment that the performance of the estimate is competitive to Bayesian structure learners on synthetic and real-world data.

Chapter 4

Densities with Mixed Types of Variables

In the previous chapter, we described a framework using classifier chains to estimate conditional joint densities. It partitions a density defined over several variables, i.e., $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$, into conditional densities with only one target variable, i.e., $f_i(X_i \mid Y_1, \ldots, Y_l, X_1, \ldots, X_{i-1})$ with $1 \leq i \leq m$, and estimates the f_i by Hoeffding trees that act as class probability estimators for the X_i. However, as the Hoeffding trees can only handle discrete X_i, we can only employ them for densities where X_i is discrete and the Y_i are discrete and / or continuous. Densities where X_i is a continuous random variable cannot be handled so far.

Therefore, we present an online density estimator for continuous target variables in this chapter, i.e., $f_i(X_i \mid Y_1, \ldots, Y_l, X_1, \ldots, X_{i-1})$ where X_i is continuous. Since the factors in Equation 3.1 are independent from each other, the univariate density estimators can be mixed arbitrarily. In the end, we obtain a framework that is able to estimate conditional joint densities with discrete variables, continuous variables, or mixed types of variables. To distinguish these cases, we refer to the discrete case as EDDO (Estimating Discrete Densities Online), to the continuous case as ECDO (Estimating Continuous Densities Online), and to the mixed case as EDO (Estimating Densities Online).

4.1 Class Probability Estimators

To provide an estimate for densities $f_i(X_i \mid Y_1, \ldots, Y_l, X_1, \ldots, X_{i-1})$ where X_i is a continuous random variable, we extend a method proposed by Frank and Bouckaert (2009). They proposed a batch algorithm that estimates conditional densities using class probability estimators. For a variable X and a set of variables $\mathcal{Y}$, they discretize the range of X into bins, weight each instance according to the likelihood of falling into a certain bin given the variables in $\mathcal{Y}$, and create an estimate for $f(X \mid \mathcal{Y})$ using these weights. Hence, the instantiations x_j of X are basically reweighted in order to respect that X is conditioned on the variables in $\mathcal{Y}$ – initially, all instantiations are assumed to have the same weight. For a specific value x_j of X, the weight is computed as follows:

$$w(x_j \mid \mathcal{Y}) = N \cdot \frac{p(c_{x_j} \mid \mathcal{Y})}{N_{c_{x_j}}}. \tag{4.1}$$

Here, c_{x_j} is the bin to which x_j belongs, N is the total number of instances, and $N_{c_{x_j}}$ is the number of instances falling into c_{x_j}. Subsequently, one can choose an estimator for $f(X \mid \mathcal{Y})$. Frank and Bouckaert considered a univariate normal estimator, a kernel estimator, and a histogram estimator. Their experiments showed that overall the kernel estimator provides the best performance, which is why we use this estimator for our purpose. It is given by a sum over all target values:

$$f_{kernel}(x \mid \mathcal{Y}) = \frac{1}{N} \sum_{j=0}^{N} w(x_j \mid \mathcal{Y}) \cdot \mathcal{N}(x; x_j, \sigma^2_{kernel}) \ . \tag{4.2}$$

The kernel bandwidth σ_{kernel} is computed using estimators for the mean and variance: $\sigma_{kernel} = \frac{\sigma_I}{N^{1/4}}$ with $\sigma_I^2 = \frac{1}{N} \sum_{j=0}^{N} w(x_j|\mathcal{Y}) \cdot (x_j - \mu_I)^2$ and $\mu_I = \frac{1}{N} \sum_{j=0}^{N} w(x_j|\mathcal{Y}) \cdot x_j$. Corresponding online versions simply rely on a sufficiently large number of previously seen instances (e.g., 100 or 1000 instances). This provides good estimates and takes into account possible concept drifts.

To return to our initial problem, this means that an online version of these estimators would be sufficient to estimate those f_i in Equation 3.1 for which X_i is continuous. But to obtain an online estimator, we have to solve two problems: First, we need to compute the weights in an online fashion, and, second, we need to regularly reduce the number of summands in Equation 4.2, because this number grows with the number of instances. For computing the weights in an online fashion, we will present an algorithm that discretizes the range of X_i and classifiers that provide class probability estimates. In particular, we will extend a discretization method for data streams that has been proposed by Gama and Pinto (2006) (Section 4.2) and extend Hoeffding trees to deal with this discretization method (Section 4.3). For reducing the number of summands in Equation 4.2, we adapt a compression method proposed by Goldberger and Roweis (2004) (Section 4.4).

4.2 Online Discretization

First, we deal with the problem of discretizing the target variable X_i in an online fashion. The algorithm by Gama and Pinto (2006) approaches the discretization for data streams using two layers. The first layer is a fine-grained discretization of $values(X_i)$. It consists of a large number of bins – many more than needed for the final discretization. Whenever the number of values within a bin exceeds a user-defined threshold, the bin is split into two bins or, if it is the first or last bin, a new bin is added. The second layer represents the final discretization. It is triggered by the user and enables an equal-width or equal-frequency discretization by merging the many smaller bins of the first layer into bigger bins.

With more and more data points, the positions of the interval borders change – substantially in the beginning and more subtly later. Hence, there are data points that can be used more safely, as it is unlikely that they will switch to another bin, and

Figure 4.1: $bin_0, bin_1, \ldots, bin_k$ are the bins, and the highlighted regions are the soft borders. For the data points that lie in the highlighted regions, it is unclear whether they belong to bin bin_j or to bin bin_{j+1}, $i \in [0, k-1]$.

there are data points for which it is unclear in which bin they will end up. For the Hoeffding tree, however, it is crucial to know to which class a data point has to be assigned. Otherwise, split decisions cannot be made with high confidence. Therefore, we extend the approach by Gama and Pinto and propose to classify data points into safe-to-use and not safe-to-use using Chernoff bounds. Regions containing the latter data points will be called *soft borders* (see Figure 4.1 for an illustration).

Assuming that the range of the target variable X_i is $[0, 1]$, the soft borders are computed as described by Algorithm 4. It is based on an existing discretization $d = (b_0, \ldots, b_k)$ of X_i, which is maintained by the equal-width or equal-frequency discretization method described above. This discretization can only take previously seen instances into account, so that future instances are possibly not represented correctly. In order to also account for these, we consider the value $\frac{\sum_{i=0}^{j} |bin_i|}{\sum_{i=0}^{k} |bin_i|}$ as the border of b_j and use Chernoff bounds for the expected deviation from this value. In particular, we consider the sequence of previously seen instances as a sequence of binary random variables $Z_1, Z_2, \ldots, Z_N$, i.e., IID Bernoulli experiments, where Z_i is the event that the instance x_i falls in the interval $[0, b_j]$, $1 \leq i \leq N$. Each Z_i has a success probability of $\frac{\sum_{i=0}^{j} |bin_i|}{\sum_{i=0}^{k} |bin_i|}$, which corresponds to the expected position of b_j in the interval $[0, 1]$ for an equal-frequency discretization. Together with the expected probability of the Z_i, a lower bound l and an upper bound u can be determined by computing the corresponding Chernoff bounds with confidence level θ. From the resulting estimates, we obtain the starting and end point of the soft border: $b_j + u$ and $b_j - l$. Theorem 3 shows the correctness of the described procedure.

Theorem 3. *Let $\theta \in [0, 1]$, let k be a user-defined number of intervals, and let $d = (b_0, l_1, b_1, u_1, \ldots, l_k, b_k, u_k)$ be a discretization produced by Algorithm 4 for confidence level θ. Then l_j and u_j, $1 \leq j \leq k$, are lower and upper bounds for b_j that hold with confidence level θ.*

Proof. First, we project the interval $[b_0, b_k]$ to $[0, 1]$. Now, consider an arbitrary border b_j that is projected onto the interval $[0, 1]$, which we denote by b'_j. We design a coin flipping experiment with for some instance *inst*: A is the event that *inst* is smaller than or equal to b_j. B is the event that *inst* is greater than b_j. Hence, the probability of A is $\frac{\sum_{i=0}^{j} |bin_i|}{\sum_{i=0}^{k} |bin_i|}$, the probability of B is $1 - Pr(A)$.

Using Chernoff bounds, we then compute the lower and upper bounds for b_j, which

Algorithm 4: computeSoftBorders

Input: Confidence level θ, an initial equal-frequency discretization with k bins $d = (b_0, \ldots, b_k)$, instances I

Output: $d = (b_0, l_1, b_1, u_1, \ldots, l_k, b_k, u_k)$

for $1 \leq j \leq k$ **do**

create a random variable X that is 1, if a value lies in $[0, b_j]$ and 0 otherwise

$p \leftarrow \dfrac{\sum_{i=0}^{j} |bin_i|}{\sum_{i=0}^{k} |bin_i|}$

$l \leftarrow$ find the minimal $0 < \lambda_Z \leq 1$ using a binary search, such that

$$Pr[Z < (1 - \lambda_Z) \cdot \mu] < e^{-\frac{\mu \lambda_Z^2}{2}} < \theta \ ,$$

where $\mu = |I| \cdot p$ and Z is the sum of independent Bernoulli trials with probabilities $Pr(Z_i = 1) = p$, $1 \leq i \leq |I|$.

$l_j \leftarrow b_j - l$

$u \leftarrow$ find the minimal $0 < \lambda_Z \leq 1$ using a binary search, such that

$$\theta < Pr[Z > (1 + \lambda_Z) \cdot \mu] < \left[\frac{e^{\lambda_Z}}{(1 + \lambda_Z)^{(1+\lambda_Z)}} \right]^{\mu} \ ,$$

where $\mu = |I| \cdot p$ and Z is the sum of independent Bernoulli trials with probabilities $Pr(Z_i = 1) = p$, $1 \leq i \leq |I|$.

$u_j \leftarrow b_j + u$

end

return $d = (b_0, l_1, b_1, u_1, \ldots, l_k, b_k, u_k)$

are l_j and u_j. Making worst- and best-case assumptions, we obtain

$$x \in [b_{j-1} + u_{j-1}, b_j - l_j], \tag{4.3}$$

which means that $x \in values(X_i)$ belongs to bin b_j with confidence level θ. Subsequently, $(b_j - l_j)$ is the lower bound of b_j and $(b_j + u_j)$ the upper bound. All instances smaller than $(b_j - l_j)$ and larger than $(b_{j-1} + u_{j-1})$ can be discarded, and we simply store the number of elements falling into this interval. Instances that lie between the lower and upper bound of b_j have to be stored. □

Hence, all values that are not in some interval $[l_j, u_j]$ can be used safely. For the remaining values, we cannot say in which bin they will finally end up. Therefore, we store them until we can make a decision. The safe-to-use values are not stored. We simply count how many times they are observed.

In extreme cases, it could happen that we have to store too many instances, because a lot of values may fall into soft border regions. In this situation, we could extend the approach proposed above by discretizing the border intervals and keeping statistics about them. In the end, we obtain an approximation that is memory-efficient.

Although we can use soft borders to classify instances into safe-to-use and not-safe-to-use, there is still a certain chance that this classification is wrong. By construction, the soft borders guarantee that the corresponding border lies within the specified interval with confidence level θ. Hence, the theoretical error of this classification is $1 - \theta$. In combination with Hoeffding trees that perform their split decisions with another confidence level θ', the overall error per split decision is $(1 - \theta \cdot \theta')$. This error is then propagated along the paths in the Hoeffding tree, until it stops at a leaf. However, this error is not propagated to other Hoeffding trees, since the trees are independent from each other – at least for the density estimation.

4.3 Extension of Hoeffding Trees

Based on the discretization proposed in the previous subsection, we modify the split criterion of Hoeffding trees (Hulten et al., 2001). Hoeffding trees use a heuristic measure to make split decisions. Typical examples are the information gain and the Gini index. To decide whether to split a given leaf node, the Hoeffding tree algorithm usually determines the attributes A and B with the highest and second highest observed heuristic measure, respectively. Subsequently, a split is performed, if

$$\overline{G}(A) - \overline{G}(B) > \sqrt{\frac{R^2 \ln(1/\delta)}{2 \cdot N}}$$

and A is not the pseudo attribute for pre-pruning, where R is the range of the variable, $1 - \delta$ is the confidence level, and N is the number of instances.

We modify this condition by only providing the counts of safe-to-use regions. Hence, data points from the soft borders are simply disregarded. Alternatively, one could also

base the split decision on worst-case assumptions of the soft border discretization. For example, the tree only branches on a leaf, if branching would take place in any possible scenario of setting the borders of the discretization to the lower and upper bounds of the soft borders. However, this increases the running time of the Hoeffding tree, since many possible scenarios have to be considered.

4.4 Compression

The last step in adapting the class probability estimator proposed by Frank and Bouckaert (2009) to the data stream setting is controlling the number of kernels. Since the number of instances is constantly growing, we need to perform some kind of compression to keep the number of summands of our kernel density estimator small (see Equation 4.2). To tackle this problem, we cluster the summands and merge all summands within one cluster into a single one.

For the clustering, we employ the *MStream* algorithm, which has been proposed by Wan and Wang (2010). It is an online algorithm that is able to handle evolving data streams and consists of two components: an online and an offline component. The online component maps data points into a hyperspace. If two points are mapped to the same point in the hyperspace, they are considered to belong to the same micro-cluster. The offline component is responsible for merging the micro-clusters into a final clustering, where two clusters are merged, if their distance is below some user-defined threshold.

The final clustering only specifies which data points can be grouped together. It does not make a statement about how to combine the kernels into a smaller number of kernels. Therefore, we propose to merge all kernels within one cluster to a single kernel. However, as we seek to maintain a proper representation of the data, we cannot perform this compression arbitrarily. For example, simply choosing a single representative of the summands would skew the contribution of the cluster. The same is true for choosing a representative and adding up the weights – though to a lesser extent.

To solve this problem, we employ a method proposed by Goldberger and Roweis (2004). They compress a mixture $g = \sum_{i=1}^{r} \alpha_i g_i$ to a mixture with fewer components $h = \sum_{j=1}^{r'} \beta_j h_j$ by applying a mapping π with $\pi : \{1, \ldots, r\} \to \{1, \ldots, r'\}$ and $r' < r$.

Using this method, we obtain the compressed kernel as follows: $h^\pi = \sum_{j=1}^{r'} \beta_i g_j^\pi$ with

$$h_j^\pi = \mathcal{N}(\mu_j', \Sigma_j') \tag{4.4}$$

$$\beta_j = \sum_{i \in \pi^{-1}(j)} \alpha_i \tag{4.5}$$

$$\mu_j' = \frac{1}{\beta_j} \sum_{i \in \pi^{-1}(j)} \alpha_i \mu_i \tag{4.6}$$

$$\Sigma_j' = \frac{1}{\beta_j} \sum_{i \in \pi^{-1}(j)} \alpha_i (\Sigma_i + (\mu_i - \mu_j') \cdot (\mu_i - \mu_j')^T) \tag{4.7}$$

In our case, r' is 1, and $i \in \pi^{-1}(j)$ can be replaced by $i = 1, \ldots, r$. Hence, the summands are compressed to

$$\left(\sum_{i=1}^{r} \alpha_i \right) \cdot \mathcal{N}(\mu_1', \Sigma_1'). \tag{4.8}$$

Since the α_i, $1 \leq i \leq r$, are not scalars but depend on the class probability estimators, $\sum_{i=1}^{r} \alpha_i$ cannot be computed at the time of compressing the summands. However, as X_i has been discretized, we can keep a vector $t := (t_1, t_2, \ldots, t_k)$, where k is the number of bins and t_i is the number of data points that fell into bin c_i, $1 \leq i \leq k$. With this information, we can rewrite $\sum_{i=1}^{r} \alpha_i$ as $\sum_{i=1}^{k} t_i \cdot w(c_i \mid Y)$. As a result, we obtain a compact representation of the weights that can be evaluated when required.

4.5 Consistency

In Section 3.5, we have shown that EDDO density estimates are consistent as long as the underlying base estimators are. In the discrete case, we employed Hoeffding trees and argued that certain assumptions are necessary to ensure their consistency. The same is true for the continuous case, as we will explain below.

The proposed online kernel density estimator consists of several components: a discretization, a conditional class estimator, and a compression algorithm. For the discretization, it is easy to see that it converges to a fixed binning with increasing numbers of instances. Hence, at some point, the binning does not change anymore and the consistency of the conditional class estimator only depends on the underlying algorithm, which we already discussed in Section 3.5.

With this assumption, the proposed kernel density estimator would be consistent, if $\lim_{N \to \infty} N \cdot \sigma_{kernel} = \infty$ (Wied and Weißbach, 2012). However, this does not hold, since the compression limits the number of kernels and the bandwidth computation is limited to a fixed window. To solve these issues, it suffices to update the bandwidth with every new instance and to remove the limitation on the number of kernels. This way, compression can still take place, but we only compress kernels that are in a certain

vicinity. If this vicinity is sufficiently small, the density estimator becomes consistent, as $\lim_{N\to\infty} N \cdot \sigma_{kernel} = \infty$. However, this comes at the cost of increased memory requirements, because the smaller vicinity of each kernel leads to a greater number of kernels in general.

4.6 Evaluation

In this section, we evaluate ECDO and EDO on real-world data. If it is clear from the context that all variables are continuous-valued, we will also refer to ECDO as EDO, which is possible since EDO subsumes ECDO. An implementation of both density estimators is provided as part of the MiDEO framework (see Chapter 6).

To facilitate comparisons with other methods, we will focus on joint densities, i.e., $f(X_1, \ldots, X_m)$ (see Chapter 3), and, to improve readability, we introduce some notation to refer to specific variants of the density estimators: $\text{EDO}_T(L)$, where $T \in \{\text{CC}, \text{ECC}, \text{EWCC}\}$, and $L \in \{\text{MC}, \text{NB}, \text{NBA}\}$. The index denotes the type of density estimator, which is either an estimator employing a classifier chain (CC), an ensemble of classifier chains (ECC), or an ensemble of weighted classifier chains (EWCC). If T is not specified, it refers to all types. The L specifies the leaf classifier of the HoeffdingTree, which is MajorityClass (MC), NaiveBayes (NB), or NaiveBayes adaptive (NBA).

To evaluate the performance of ECDO and EDO, we use the kernel density estimator oKDE[1] by Kristan et al. (2011), which is one of the few online density estimators for multivariate densities. The performance is measured by computing the average log-likelihood prequentially, i.e., the log-likelihood of a given instance has been computed before using it for training. The instances used to compute the initial estimator (the first 100) were excluded from this computation.

4.6.1 Comparison with oKDE

In our first experiment, we compare EDO with a state-of-the-art online density estimator for continuous densities[2], oKDE (Kristan et al., 2011). For EDO, we set its internal buffer size to 100 instances, the maximal number of kernels to 20,000, and the confidence for the soft borders to 90%. In order to study how the number of bins for discretizing the class attribute affects the performance of EDO, we set this parameter to 3, 5, 10, 15, and 20, respectively.

As datasets, we selected letter, electricity, shuttle, adult, and covertype[3]. Unfortunately, oKDE was not able to complete a single job within twelve hours on the covertype dataset and showed warnings regarding matrix computations on the adult dataset. Therefore, we excluded these datasets from the comparison. In case of the

[1] Its implementation is publicly available (MATLAB).

[2] Unfortunately, even after several emails, the authors of RS-Forest did not respond to our request to share their program.

[3] `http://archive.ics.uci.edu/ml/`

Table 4.1: The table shows some properties of the datasets that have been used for the evaluation. We distinguish two types of data: continuous and mixed. The former is used for datasets that only consist of continuous variables, the latter for datasets that contain discrete and continuous variables.

Dataset	Type	#Variables	#Instances
letter	continuous	17	19,999
electricity		9	45,313
shuttle		10	58,000
adult	mixed	15	30,163
covertype		54	581,012

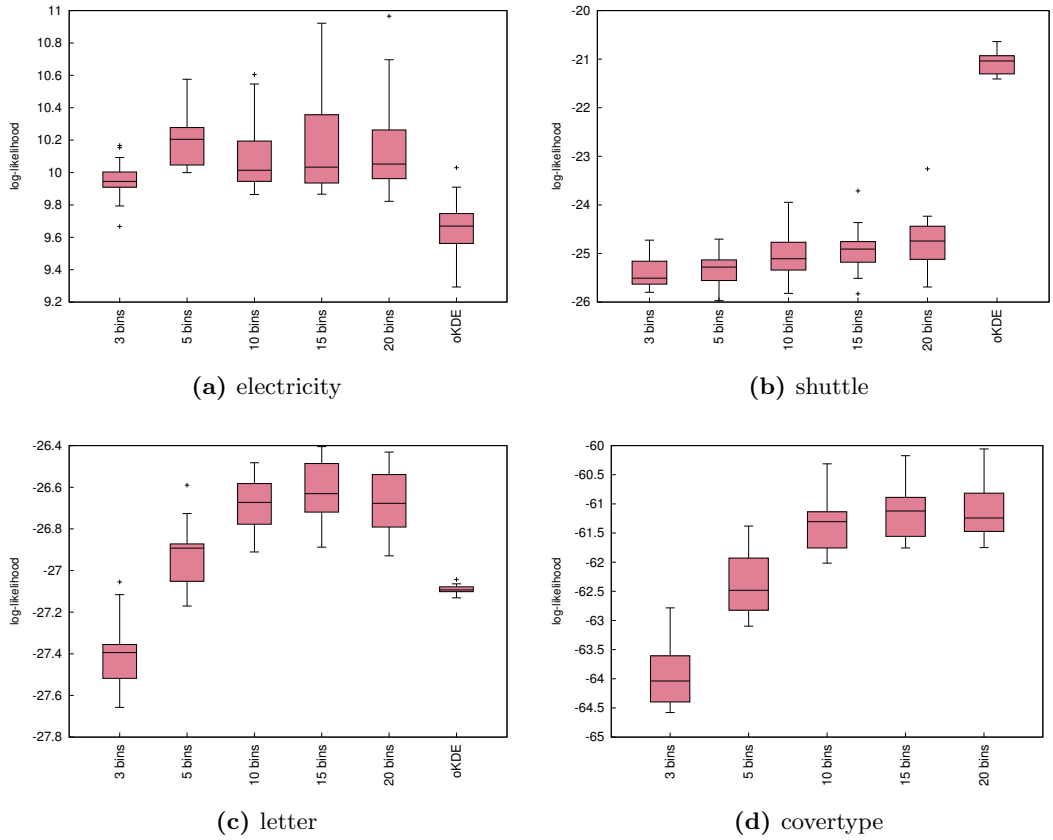
covertype dataset, the running time issues can be explained by the large number of variables. The problems with the adult dataset can be explained by the many discrete variables (note that we converted these variables to a numeric representation).

The results are summarized in Figure 4.2 and Appendix B. With the exception of the shuttle dataset, EDO performs better or equally well compared to oKDE, if at least five bins are used. On the shuttle dataset, EDO discards up to 1443 instances due to the soft border computations, which probably leads to less accurate class probability estimators and thereby to a less accurate estimate. On the electricity dataset, the performance of the estimators are comparable with a slight advantage for EDO. The poor performance of oKDE on the letter dataset can be explained by the values taken by the variables. They are mostly small integer values, so that the dataset acts more as a discrete than a continuous dataset. On the remaining datasets only EDO is able to provide results either due to running time constraints or due to internal errors of oKDE.

Regarding the number of bins, we observe that more bins are generally beneficial for the performance of EDO and that 10 bins is a good compromise between performance and memory usage. The only exception is the electricity dataset, where all estimators are roughly on the same level and five bins seem to be sufficient for separating the values of the variables. When the number of bins is getting too large, EDO's performance tends to decrease again. This can be explained by fewer instances per bin, which leads to fewer training instances for the conditional class probability estimators. Another reason are the number of discarded instances due to the soft borders. In our experiments, the number of these instances usually ranges between a couple of instances and 1443 instances.

4.6.2 Compressions

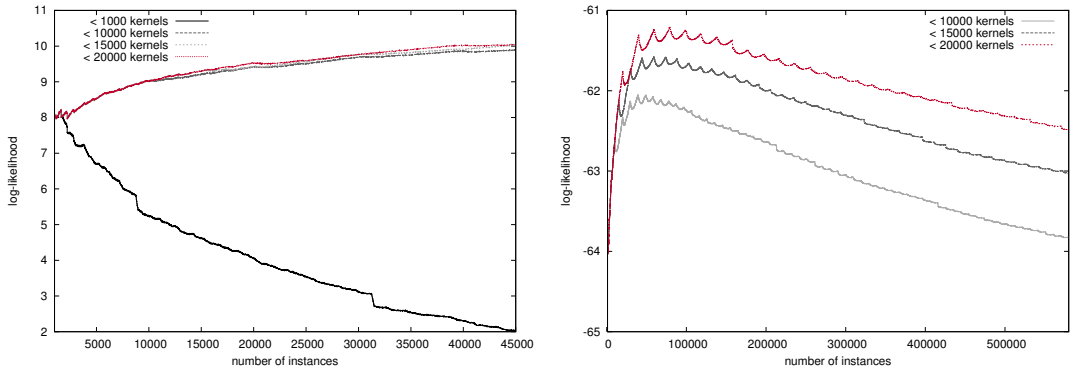
EDO is performing regular compressions to keep the number of kernels low. In order to analyze the behavior of EDO with respect to these compressions, we set the maximal number of kernels to 10,000, 15,000, and 20,000, which means that EDO



(a) electricity **(b)** shuttle

(c) letter **(d)** covertype

Figure 4.2: $\mathrm{EDO}_{\mathrm{CC}}(\mathrm{MC})$ compared with oKDE on the datasets electricity, shuttle, letter, and covertype. On the y-axis is the average log-likelihood (computed prequentially). *i bins* with $i \in \{3, 5, 10, 15, 20\}$ stands for the number of bins used for discretizing the class attribute.



(a) electricity **(b)** covertype

Figure 4.3: These plots illustrate the behavior of EDO when different levels of compression are used to reduce the number of kernels. On the left is the electricity dataset, on the right the covertype dataset. $< k$ means that a compression takes place, if there are more than k kernels.

performs a compression as soon as the corresponding limit is reached. Its behavior is illustrated with two plots (see Figure 4.3): On the electricity dataset, the prequential log-likelihood is constantly improving for a kernel limit of $10,000$ kernels or more, which shows that compression is working and does not substantially affect the accuracy of the estimate. As opposed to that, the performance decreases on the covertype dataset whenever a compression takes place. This happens when the number of kernels is too low compared to the number of possible variable-value combinations, i.e., the *instance space*. Especially the covertype dataset with its many attributes has a high-dimensional instance space (see Section 7.1.3 for further details regarding the complexity of datasets). If only few instances are observed for large subregions of that instance space, EDO will compress all kernels in that region to meet the kernel limit, and, in case these kernels are rather unrelated, EDO will predict the density values in that region less accurately. To solve this issue, one could increase the maximum number of kernels to make the aforementioned subregions more dense before another compression takes place.

The effect of this proposal is illustrated in Figure 4.3 for the electricity dataset. Whereas a kernel limit of $1,000$ leads to a constant decrease of EDO's performance, $10,000$ kernels or more lead to a constant improvement. Hence, by increasing the kernel limit, we can counteract the issues caused by a compression. However, as this problem is an instance of the curse of dimensionality, EDO will run out of memory, if the density has too many variables.

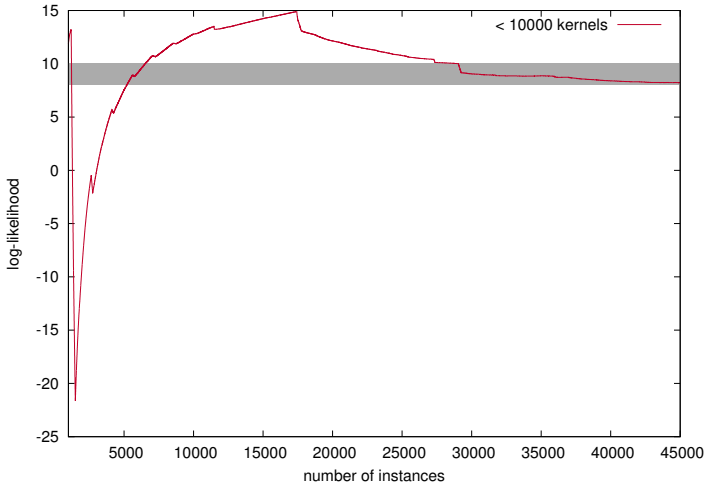


Figure 4.4: EDO exploits the assumption that the instances of the stream are IID. The effect of violating this assumption is shown at the example of the electricity dataset. The gray area represents the range in which EDO is performing, if the data is IID.

4.6.3 Non-IID Data Streams

EDO is making the assumption that the instances are drawn IID from the target distribution. This happens in two places: for split decisions in the Hoeffding trees and for the soft borders. The effect of violating this assumption is shown in Figure 4.4, where we considered the original data stream instead of a randomized version. Here, the performance of the density estimate becomes very unstable and varies a lot, since the split decisions of the Hoeffding trees are no longer guaranteed to be correct with high confidence.

Towards the end of the stream, the performance becomes more stable again and is roughly in the range of the IID data stream. This indicates that a larger number of instances could help to counteract the non-IIDness of the data stream. In fact, if there was not data distribution drift, the structure of the Hoeffding trees would become more and more fixed after a while, so that only the probability masses in the leaves would be changed. Even if this structure was not an ideal partitioning of the space, the density estimator would still approach the true density according to the law of large numbers. Therefore, we only have to make sure that data distribution drifts are handled separately, e.g., using an external drift detection (see Chapter 8). This solution is probably not ideal and possibly produces trees that are larger than necessary[4], but handling non-IID data streams with EDO natively is a different story that we plan to investigate in the future.

[4]If a Hoeffding tree ht becomes too large, one can train a new, smaller Hoeffding tree ht' from instances that are drawn IID from ht.

4.7 Conclusion

All in all, the proposed density estimators perform well on the selected datasets and can deal with purely continuous datasets as well as with datasets exhibiting discrete and continuous variables. However, it is important to notice two points for applications in practice. First, the performance is dependent on the interval between compressions. If the interval is too small compared to the size of the instance space, it causes a substantial decrease in performance. If it is too large, the density estimate consumes too much memory. Since the length of the interval can be controlled by a user-defined parameter, the estimator can be adapted for each dataset. But this is inconvenient for users and requires several runs over data stream. Therefore, we propose to control this parameter by monitoring the current log-likelihood. This way, we can adjust the interval length to be small enough to avoid causing a performance decrease. However, the parameter cannot be fine-tuned arbitrarily, as some performance changes can also be due to data distribution drift.

Second, the assumption that the instances of the data stream are IID is often made in the data stream literature but is unrealistic for many real-world applications. EDO suffers from this problem to a certain degree and it remains to find modifications that make EDO more robust on non-IID data streams.

Chapter 5

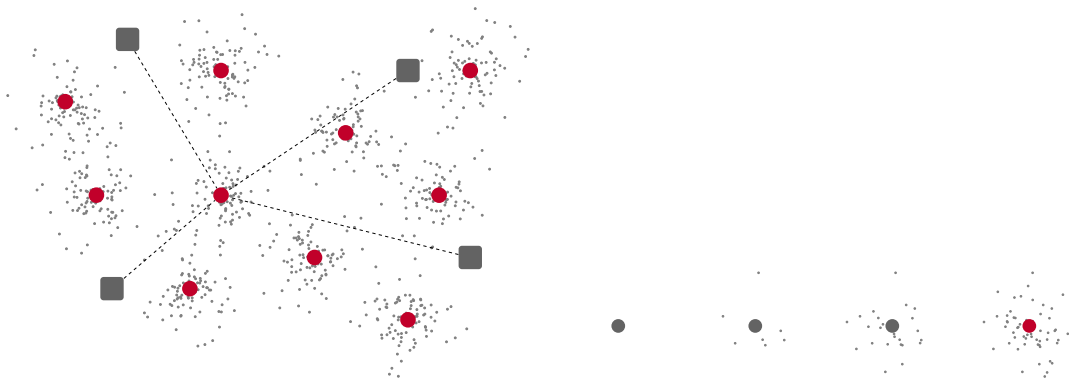
Higher-Dimensional Densities

EDO models the interdependencies between variables by a product of conditional densities, $f(X_1, \ldots, X_n) = \prod_{i=1}^{n} f_i(X_i \mid X_1, \ldots, X_{i-1})$. Each of these densities has a single target variable and is conditioned on all variables that occurred previously in the chain. Although this works well for densities with a small to medium number of variables, the size of the f_i grows quickly when the dimensionality of the data increases – making this approach unsuitable for this kind of data. To mitigate these limitations, we present a method that projects the original data stream into a vector space of lower dimensionality and uses a set of representatives to provide an estimate. Due to the structure of the estimates, it enables the estimation of higher-dimensional data and approaches the true density with increasing dimensionality of the vector space.

5.1 Introduction

With the constantly increasing number of interconnected devices that try to measure their environment to make intelligent decisions, high-dimensional data streams are becoming more and more frequent. A typical application are future smart homes, which are equipped with many sensors to measure various parameters of the house (e.g., temperature or humidity). By learning from past measurements, data mining algorithms have the possibility to distinguish between typical and abnormal behavior and can suggest appropriate actions to the user. For example, whereas an increase of the humidity in the basement can be explained by a tumble dryer that has been started some time ago, such an increase in the bed room could be due to water entering the room through an open window. The former situation is probably quite normal for a household and requires no action, the latter situation needs attention by the user. An estimate that captures the density of the sensor measurements and provides facilities to perform data mining tasks can be useful to develop such applications. To make a step towards this direction, we address the problem of estimating the joint density of heterogeneous data streams with many variables in this chapter and present an algorithm called RED (Representative-based online Estimation of Densities) for this purpose.

The main idea is to project the data stream into a vector space of lower dimensionality by computing distances to well-defined reference points. In particular, we distinguish between three types of objects (see Figure 5.1 for an illustration): *land-*



(a) object types **(b)** becoming a representative

Figure 5.1: On the left are the main object types: *landmarks* (dark gray squares), *representatives* (red), and *instances* (gray dots). On the right is the process of becoming a representative: If a distance vector cannot be assigned to a representative or candidate, it is first considered a candidate (big dark gray circle). Over time more instances appear in its neighborhood. If a predefined number is reached, the candidate is turned into a representative (big red circle).

marks, *representatives*, and *instances*. Landmarks are reference points spanning a vector space for representatives and instances, so that the position of each object can be defined in terms of distances to the landmarks (e.g., the position of the red circle in Figure 5.1 that is connected to the four landmarks (red circles) can be specified as a four tuple (d_1, d_2, d_3, d_4), where d_i is the distance to landmark i, $1 \leq i \leq 4$). Representatives stand for clusters of instances and will be the main components for estimating probabilities. Hence, the landmarks will be used to compute the relative distance of instances, and the representatives will maintain statistical information about instances that have been observed in their neighborhood. To maintain this information, we use EDO density estimates for each representative and estimate the distances to nearby instances. Compared to EDO, which directly estimates the density of the instances, RED reduces the dimensionality of the dataset to the number of landmarks. If this number is substantially smaller than the number of variables, the model size of the estimators can be substantially reduced, thereby making the approach suitable for data of higher dimensionality.

Although distances and representative instances have been used before to project data into a space of lower dimensionality (e.g., multidimensional scaling), the approach presented in this thesis is different. Whereas other techniques try to preserve the relevant characteristic properties of the data when embedding it into a space of lower dimensionality, RED characterizes the data using landmarks and provides a back translation to the original data. This back translation is a crucial and necessary part to enable density estimation. The approach pursued by RED is also different from micro-clustering (Aggarwal et al., 2003). Whereas micro-clusters maintain simple statistical

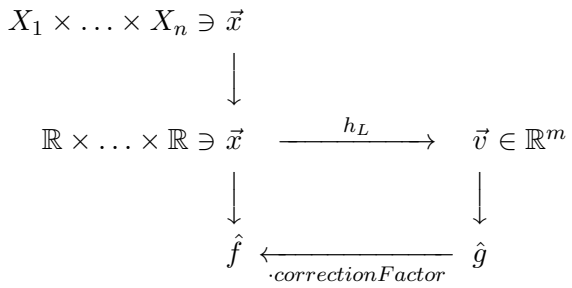


Figure 5.2: As we introduce quite some notation in the chapter, we provide an overview in this figure. Please notice that the arrows have different meanings and just indicate a relationship between the entities. Their meaning will be explained when describing the method in detail.

properties of the data such as the linear sum, RED uses landmarks and Gaussian mixtures to partition the data space and then estimates the full joint density of each partition.

5.2 Density Estimation using Representatives

Let $X_1, \ldots, X_n$ be a set of variables and let $\vec{x}$ be an instance defined over these variables. Given a possibly infinite stream of instances with many variables, we address the problem of estimating its density, $f : X_1 \times \ldots \times X_n \to [0, 1]$, in an online fashion. In order to determine a density estimate $\hat{f}$, we propose a method that reduces the dimensionality by using a small set of reference points $L := \{L_1, \ldots, L_m\}$, so-called landmarks. With these reference points, the method projects the data into a vector space of dimensionality $m < n$ by applying a mapping, $h_L : X_1 \times \ldots \times X_n \to \mathbb{R}^m$, to each instance $\vec{x}$. Here, the i-th component of $h_L(\vec{x})$ is defined as the distance between $\vec{x}$ and landmark L_i. The estimate $\hat{f}$ of f is then expressed as the product of two independent components: an online estimate $\hat{g}$ that captures the density in the new vector space and a *correctionFactor*, which is the expected number of instances that are mapped to the same distance vector – without the correction factor, we would only estimate g. Hence,

$$\hat{f}(\vec{x}) = \hat{g}(h_L(\vec{x})) \cdot correctionFactor.$$

An overview of this idea is summarized in Figure 5.2. In the remainder of this section, we give a detailed description of these components and provide a theoretical analysis.

5.2.1 The density of the vector space

In order to estimate the density g, RED distinguishes three kinds of objects in the vector space: *distance vectors*, *representatives*, and *candidates*. A distance vector is a projected data stream instance, which is determined by computing the distances to

the landmarks. A representative is a distance vector together with a density estimator and a covariance matrix, where the density estimator is supposed to provide a density estimate of nearby distance vectors. Whether a distance is nearby is decided based on its Mahalanobis distance to the representative. A candidate is a precursor of a representative. It will be turned into a representative, if it gathers enough distance vectors around it. The intuition behind these objects is that the landmarks provide a space with certain properties and guarantees, and the representatives and candidates are responsible for modeling the density.

Given the current $\hat{g}$, the estimate is updated as follows (illustrated by Figure 5.1): The next instance $\vec{x}$ is first projected into the vector space by applying h_L. Then the resulting distance vector $\vec{v}$ is tested against all representatives. If a representative is found, $\vec{v}$ is forwarded to the corresponding density estimator. Otherwise, $\vec{v}$ is tested against all candidates. If a candidate is found, $\vec{v}$ is assigned to the candidate. Otherwise, $\vec{v}$ becomes a candidate itself.

Algorithm 5 describes the update procedure of RED in more detail. In lines 1–2, the EDO estimates are updated, if the new instance is considered as a member of that representative. If it does not belong to any representative (line 3), we distinguish two cases: the instance does not belong to any existing candidate (lines 4–16) or it does (line 17–26). In the first case, the given instance becomes a candidate itself, as no matching representative nor candidate has been found. For this purpose, we determine the k closest representatives and compute a covariance matrix from their most recent samples. The result is a new covariance matrix, which will be used in the future to decide whether or not an instance belongs to the new candidate. In the second case (there are candidates to which the instance belongs), there is no need to create a new candidate. Instead, the buffers of matching candidates are simply updated. However, since the recent updates could result in buffer sizes of $\theta_{C\to R}$ instances, the update procedure finishes with checking the buffer size of all candidates (lines 21–25) and turning candidates to representatives that have more than $\theta_{C\to R}$ instances.

Membership tests

Whether an instance belongs to a candidate or representative is decided by employing a multivariate normal density $\mathcal{N}(\vec{v};\Sigma)$, where $\vec{v}$ is the distance vector of the representative instance and Σ is a covariance matrix computed from instances in the neighborhood – when a new candidate is created, it is computed from recent samples of neighboring representatives. The membership decision is based on the Mahalanobis distance, which is computed as follows: $\sqrt{(\vec{x}-\vec{r})^T\Sigma^{-1}(\vec{x}-\vec{r})}$. Any vector with a Mahalanobis distance less than a user-defined threshold is then considered a member.

Handling of noise

Almost all real-world applications suffer from certain degrees of noise. Hence, handling noise is of paramount importance for density estimators but in many cases difficult.

Algorithm 5: updateDensityEstimate

Input: landmarks L, instance $\vec{x}$, mapping h_L, number of neighbors $k \in \mathbb{N}$, candidate threshold $\theta_{C\to R} \in \mathbb{N}$, $C := \{(c, \Sigma, b) \mid$ candidate c, covariance, matrix Σ, recent instances $b\}$, $R := \{(r, \Sigma, e, b) \mid$ representative r, covariance matrix Σ, estimator e, recent instances $b\}$

```
// check representatives
```
Let $(r, \Sigma, e, b) \in R$ with r being closest to $h_L(\vec{x})$
if $r.isMember(\vec{x}, \Sigma)$ **then**
  e.update($h_L(\vec{x})$)
end
```
// no matching representative?
```
if $\nexists (r, \Sigma, e, b) \in R : r.isMember(inst, \Sigma)$ **then**
  ```
  // no matching candidate?
  ```
  if $\nexists (c, \Sigma, b) \in C : c.isMember(inst, \Sigma)$ **then**
    ```
    // find the k closest representatives
    ```
    $pq \leftarrow priorityQueue()$
    for $(r, \Sigma, e, b) \in R$ **do**
      $r' \leftarrow (r, e, \|r - h_L(\vec{x})\|, b)$
      pq.insert($r', \|r - h_L(\vec{x})\|$)
    end
    $representatives \leftarrow \{pq.popMin() \mid k \text{ times}\}$
    ```
    // compute covariance matrix
    ```
    $sample \leftarrow \emptyset$
    for $(r, e, \|r - h_L(\vec{x})\|, b) \in representatives$ **do**
      $sample \leftarrow sample \cup \{\vec{x}' \mid \vec{x}' \in b\}$
    end
    ```
    // initialize candidate with empty buffer
    ```
    C.append($(\vec{x}$, covariance($sample$), $[\vec{x}])$)
  end
  ```
  // matching candidates
  ```
  else
    for $(c, \Sigma, b) \in C$ **do**
      if c.isMember($h_L(\vec{x}), \Sigma$) **then** $b \leftarrow b \cup \{\vec{x}\}$; **break**
    end
    ```
    // turn candidates into representatives
    ```
    for $(c, \Sigma, b) \in C$ *with* $|b| \geq \theta_{C\to R}$ **do**
      $e \leftarrow$ initialize EDO estimator
      e.update($h_L(\vec{x})$)
      R.append((c, Σ, e, b))
    end
  end
end

In an online setting, the problem is even more severe, since future instances cannot be included into the decision making process. The EDO estimators employed by RED are able to handle noise, but in order to keep them as clean as possible, it is important that a noisy instance does not become a representative in the first place. Otherwise, this instance and every instance in its neighborhood becomes inevitably a part of the estimate. Therefore, RED distinguishes between candidates and representatives. If an instance cannot be assigned to an existing representative, it is first considered a candidate for becoming a representative. Only if enough instances are gathered around the candidate, it becomes an actual representative.

Concept drift

In real-world applications, the distribution of data streams is changing constantly and a density estimator has to adapt to these changes to provide reliable estimates. In order to address this problem, RED pursues a timestamp-based solution. However, old instances are not simply discarded when they become too old, but candidates and representatives are discarded, if no instance has been assigned to them within a certain period of time. This time period is specified as a parameter and can be adjusted according to the smallest probability values that should be covered by the estimate – using Chernoff bounds, the parameter can be computed with high confidence.

When setting this parameter, one should also consider the parameter $\theta_{C \to R}$, because a high value for $\theta_{C \to R}$ prevents rare instances from becoming a part of the density estimate. In our experiments, we usually set $\theta_{C \to R}$ to 100, which is large enough for a statistical test but not too large to exclude less frequent instances.

5.2.2 Distance measure

With landmarks, high-dimensional data can be mapped to a lower dimensional vector space. But dependent on the number and the choice of the landmarks, the resulting vector space could still be relatively large, so that the distance measure has to be chosen with care. For high-dimensional spaces, the Manhattan distance (1-norm) or a fractional distance measure is usually the best choice (Aggarwal et al., 2001), so that we prefer p-norms with small p. Employing a p-norm, the mapping $h_L : X_1 \times \ldots \times X_n \to V_1 \times \ldots \times V_m$ with $V_j \subseteq \mathbb{R}$, $1 \leq j \leq m$, is defined as

$$h_L(\vec{x})[V_i] := \left(\sum_{X_j \in X} \|l_i[X_j] - \vec{x}[X_j]\|^p \right)^{\frac{1}{p}} ,$$

where $\|\cdot\|$ computes the distance for the given variable values and is defined as the difference $\frac{l_i[X_j]-\vec{x}[X_j]}{\max(X_j)-\min(X_j)}$ for numeric values and $\frac{l_i[X_j]-\vec{x}[X_j]}{\#values} \in [0,1]$ for nominal values. For p, we select values from the range $(0, 2]$, which corresponds to the Euclidean distance for $p = 2$, the Manhattan distance for $p = 1$, and to fractional norms for

$0 < p < 1$. (For the evaluation, we will mostly select the Euclidean norm, as we only expect low dimensional vector spaces.) The denominator $\max(X_j) - \min(X_j)$ can only be estimated, as the minimum and maximum values cannot be determined with certainty in a streaming setting, making a correct normalization impossible. For typical applications, however, an estimate is probably more than sufficient, since extreme deviations are most likely due to a concept drift.

5.2.3 Choice of the landmarks

If the data stream is projected into a vector space of lower dimensionality, information about the original instances will possibly be lost. In particular, some of the variable interdependencies are no longer visible, as the mapping h_L only adds up the distances of individual variables. As a consequence, instances may be projected onto the same point of the vector space, that is there are $\vec{x}$ and $\vec{x}'$, such that $h_L(\vec{x}) = h_L(\vec{x}')$ but $\vec{x} \neq \vec{x}'$. If the original instances have only nominal variables ($\{X_1, \ldots, X_n\}$) and each variable has $|X_i|$, $1 \leq i \leq n$, many values, then there are already $\prod_{i=1}^{n} 2 \cdot (|X_i| - 1)$ many possible instances that are be mapped to the same distance value. RED would treat all of these instances equally when computing their density value, which poses no problem to the density estimate, as long as similar distances to the landmarks correspond to similar density values of the instances. But it implies that the information encoded by the distances to the landmarks needs to be sufficiently good. Therefore, we propose to choose the landmarks in such a way that the estimate approaches the accuracy of the underlying density estimator as $|L|$ approaches n:

Definition 15. Let $\mathcal{X} := \{X_1, \ldots, X_n\}$ be the set of variables, and let m be the requested number of landmarks. Then the first landmark L_1 is defined as $(1) \circ (0)_{2 \leq j \leq n}$ and landmark L_{i+1}, $1 \leq i < m$, is defined as

$$\left((i - j + 1) \cdot \frac{\max(X_j) - \min(X_j)}{m} \right)_{1 \leq j < i} \circ (1) \circ (0)_{j > i}$$

where $\max(X_j)$ and $\min(X_j)$ are the currently observed maximum and minimum, respectively. The set of landmarks is denoted by $L := \{L_i \mid 1 \leq i \leq m\}$.

By construction of h_L and by the choice of the landmarks, the mapping h_L projects any two instances $\vec{x}$ and $\vec{x}'$ to different points in the vector space as long as $\vec{x} \neq \vec{x}'$, $|L| = n + 1$, and certain assumptions hold. (An example is given below.) Hence, the mapping becomes injective:

Lemma 2. *If $|L| = n + 1$, landmark L_i, $i \in [1, n + 1]$, is defined as in Definition 15, and* max *is the actual maximum for all $X_j \in X$, then the projection mapping $h_L : X_1 \times \ldots \times X_n \to V_1 \times \ldots \times V_m$ is injective, for $V_j = \mathbb{R}$, $1 \leq j \leq m$.*

Proof. Under the assumption that $\max(X_j)$ is the actual maximum for all $X_j \in \mathcal{X}$, $L_j[X_j]$ is always larger than $h_L(\vec{x})[V_j]$. Hence, for $x_{j_1} \neq x_{j_2} \in X_j$, $h_L(\vec{x}_{j_1})[V_j]$ is not

equal to $h_L(\vec{x}_{j_2})[V_j]$. (Please notice that this would not have been the case, if we had chosen $L_j[X_j]$ to be $\frac{1}{2} \cdot \max(X_j)$, since h_L does not consider the sign of differences, e.g., $(\min(X_j) + 0.2) - \frac{1}{2} \cdot \max(X_j)$ and $(\max(X_j) - 0.2) + \frac{1}{2} \cdot \max(X_j)$ result in the same distance).

So individual variable values do not cause two different instances to have the same distance vector. It remains to show that this property is preserved when computing the summation over all variable differences in h_L. For this purpose, we project $X_1 \times \ldots \times X_n$ into $\mathbb{R}^n$ and first show that, if $|L| = n + 1$, all vectors $\vec{x}$ of $\mathbb{R}^n$ that are mapped to the same distance vector have the same length. Let $h_L(\vec{x}) = (v_1, v_2, \ldots, v_m)$ be the distance vector of $\vec{x}$. We can determine the possible lengths of all instance vectors $\vec{x}$ that are mapped to $h_L(\vec{x})$ by finding the solution for the following system of equations:

$$\left(\begin{array}{ccccc|c} 1 & 0 & 0 & \ldots & 0 & v_1 \\ 1 \cdot s_1 & 1 & 0 & \ldots & 0 & v_2 \\ 2 \cdot s_1 & 1 \cdot s_2 & 1 & \ldots & 0 & v_3 \\ & & & \vdots & & \\ n \cdot s_1 & (n-1) \cdot s_2 & (n-2) \cdot s_3 & \ldots & 1 & v_n \end{array}\right),$$

where s_j equals $\frac{\max(X_j) - \min(X_j)}{m}$. Due to the choice of the landmarks, the left-hand side is a square matrix, denoted by A, of rank n. Hence, the system of equations has only one unique solution: $\vec{x} = A^{-1}\vec{b}$, where $\vec{b} = \vec{v}^T$.

Dependent on the norm employed by h_L, there are fewer or more instances having the same length in $\mathbb{R}^n$. In case of the Euclidean norm, for example, the corresponding vectors having the same distances to the landmarks $L_2, \ldots, L_{n+1}$ lie on the border of a $(n-1)$-dimensional norm sphere (notice that L_1 is excluded here). In order to ensure that the mapping h_L is injective, we simply have to include the landmark L_1, which introduces another dimension and reduces the border of the $(n-1)$-dimensional norm sphere to a single point. The same approach is also valid for arbitrary p-norms, which we prove by constructing a contradiction: Let L be defined as in Definition 15. Assume that there are $\vec{x} \neq \vec{y}$ in the projected vector space, such that $\forall L_i \in L$: $\left(\sum_{X_j \in X} \|L_i[X_j] - x[X_j]\|^p\right)^{\frac{1}{p}} = \left(\sum_{X_j \in X} \|L_i[X_j] - y[X_j]\|^p\right)^{\frac{1}{p}}$. Due to the projection of $\vec{x}$ and $\vec{y}$ into $\mathbb{R}^n$ and due to the definition of $\|\cdot\|$, $\|\cdot\|$ becomes $|\cdot|$ in $\mathbb{R}^n$. From the first part of the proof, we can conclude that for all $L_i \in L$ and for all $X_j \in \mathcal{X}$:

$$\begin{aligned} & |L_i[X_j] - x[X_j]|^p = |L_i[X_j] - y[X_j]|^p \\ \Leftrightarrow\ & |L_i[X_j] - x[X_j]| = |L_i[X_j] - y[X_j]|, \end{aligned}$$

As this equation also has to hold for L_1 ($L_1 = (0)_{1 \leq i \leq n} \in L$) and as for all $X_j \in \mathcal{X}$: $x[X_j], y[X_j] \geq 0$, this implies that for all $X_j \in \mathcal{X}$ holds $x[X_j] = y[X_j]$, which contradicts the assumption that $\vec{x} \neq \vec{y}$. □

This theorem makes a valuable statement about the validity of the method: Although $n + 1$ landmarks are probably infeasible for extremely high-dimensional data

streams, we know that additional landmarks are beneficial for the accuracy of the method. Hence, it is up to the user whether the accuracy or memory consumption is more critical.

5.2.4 Correction factor

If $|L| < n + 1$, the projection mapping h_L possibly maps several instances from the original data stream onto the same point of the vector space. When RED estimates the density value of that point, it has actually estimated the density value of all instances that are mapped to it. Since we have no additional information on how to divide the density value among those points, we will divide it equally.

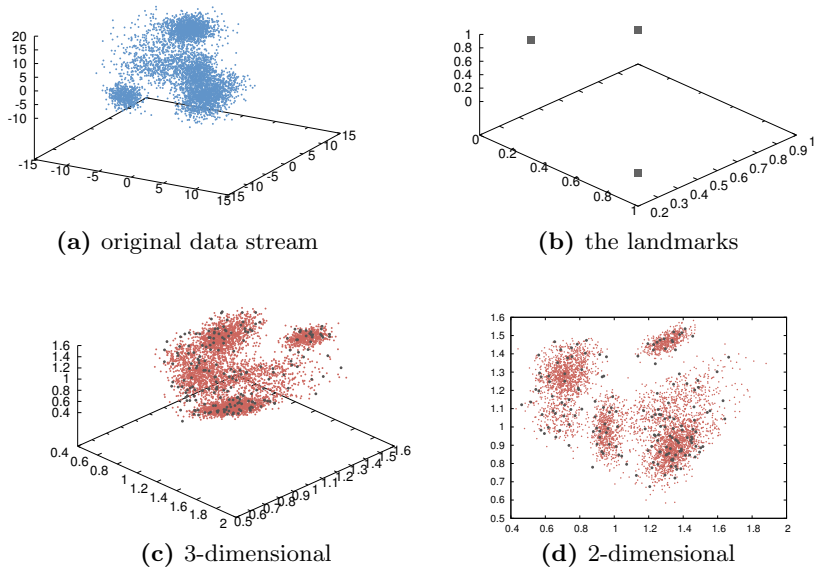
To obtain the density value from the original data stream, we have to multiply it by the integral $\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \dots \int_{-\infty}^{+\infty} f(x_i, x_{i+1}, \dots, x_n) dx_{i+1} dx_{i+2} dx_n$, which is the expected density value of the variables $X_i, X_{i+1}, \dots, X_n$. In case where all variables are discrete, this would simply be $\frac{1}{\prod_{i=1}^{k} |X_i|}$. For a continuous variable X_j, we will approximate the integral using a sampling technique. In particular, we employ a conditional density estimator for $f(X_j \mid V_1, V_2, \dots, V_m)$ and sum $f(x_j \mid v_1, v_2, \dots, v_m)$ over $\min(X_j) \leq x_j \leq \max(X_j)$ with step size $\frac{\max(X_j) - \min(X_j)}{|S|}$ and sample size $|S|$. Hence, the correction factor is 1, if $|L| \geq n + 1$.

5.2.5 Illustrative example

To illustrate the density estimation RED, we give a small example: We generated a synthetic data stream of dimensionality 3, for which we selected the landmarks $L_1 = (1, 0, 0)$, $L_2 = (0.2, 1, 0)$, and $L_3 = (0.3, 0.2, 1)$ and projected it into a vector space of dimensionality 3. Subsequently, we also projected the data into a vector space of dimensionality 2 – this time with the landmarks $L_1 = (1, 0, 0)$ and $L_2 = (0.2, 1, 0)$. As Figure 5.3 illustrates, the instance clusters are still visible after applying h_L to the data (see (a) and (d)). The relative positions remain roughly the same, but due to the landmarks, they capture a different area in the vector space. For the vector space that is induced by two landmarks, the instance clusters are arranged similarly. When the instances have been mapped to the vector space, it is up to the representatives to model the density. The density values of the original data stream can then be computed by retranslating the density value using the $correctionFactor$, where the $correctionFactor$ accounts for the instances that have been mapped to the same point in the vector space.

5.2.6 Consistency

The consistency of a density estimator is a desirable property, as it ensures that the estimator approaches the true density. We have already shown that the number of instances that are mapped to the same point in the vector space can be controlled by



(a) original data stream

(b) the landmarks

(c) 3-dimensional

(d) 2-dimensional

Figure 5.3: Illustrates a synthetic data stream (see Plot (a)) that is projected with three landmarks (see Plot (b)) into a vector space of dimensionality 3 (see Plot (c)) and with two landmarks to dimensionality 2 (see Plot (d)). Please notice the the dark gray dots in Plot (c) and (d) are the representatives. Due to the Mahalanobis distance, dense regions have more representatives than sparse regions, which helps to model the density more accurately.

the number of landmarks under certain assumptions. In the following, we will prove that RED estimates are consistent, if some further conditions hold:

Theorem 4. *If $|L| = n+1$, L is defined as in Definition 15,* min *is the actual minimum for all $X_j \in \mathcal{X}$,* max *is the actual maximum for all $X_j \in \mathcal{X}$, and $\hat{g}$ is consistent, then $\hat{f}$ is consistent.*

Proof. If $|L|$ equals $n+1$, then the correction factor is 1, so that it remains to show that $\hat{f}(\vec{x}) = \hat{g}(h_L(\vec{x}))$. Furthermore, as Algorithm 5 partitions the vector space into independent subregions where each subregion has its own online density estimator, it suffices to show that the estimator of each subregion is consistent.

Since $|L|$ equals $n+1$ and the assumptions for min and max hold, h_L is injective according to Lemma 2. Hence, for any two instances $\vec{x}_1$ and $\vec{x}_2$ that only differ in variable X_j by a small amount, it follows that

$$h_L(\vec{x_1}[V_i])^p - h_L(\vec{x_2}[V_i])^p = \sum_{k=1}^{p} c_k \cdot \|x_1 - x_2\|,$$

by definition of h_L, where $1 \leq i \leq n+1$ and the c_k are constants that have two factors: l_i and a constant that results from the binomial theorem. In other words, the density in the vector space is only shifted and compressed, but the original information of f is completely contained in g. Therefore, we can conclude that $\hat{f}(\vec{x}) = \hat{g}(h_L(\vec{x}))$. □

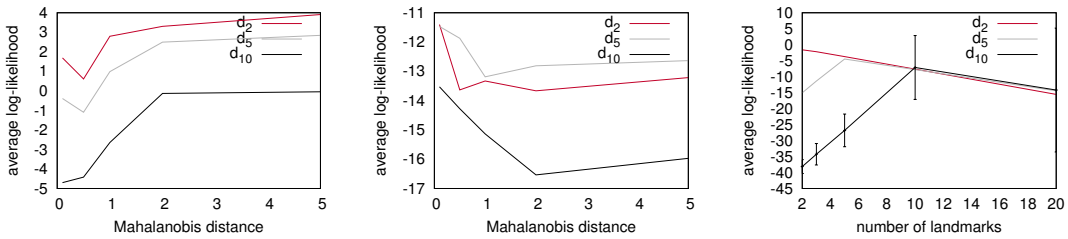
5.3 Evaluation

In this section, we analyze the behavior of RED with respect to its parameters on synthetic datasets and evaluate its capability to estimate joint densities on real-world datasets. RED has several parameters: the number of landmarks $|L|$, the Mahalanobis distance, the threshold of becoming a representative $\theta_{C \to R}$, and the distance measure p. As $\theta_{C \to R}$ is mostly relevant for drift detection or for application tasks such as outlier detection, we do not discuss this parameter here. Also not discussed is the parameter p. Hence, we focus our experimental analysis of the parameters on the number of landmarks $|L|$ and the Mahalanobis distance. For $|L|$, we consider the values 2, 3, 5, 10, and 20. For the Mahalanobis distance, we consider the values 0.1, 0.5, 1.0, 2.0, 5.0, and 10.0.

As datasets, we generated synthetic data consisting of 1, 2, 5, or 10 multivariate Gaussians in a d-dimensional vector space with $d \in \{2, 3, 5, 10, 20\}$. The mean variables and covariance matrices were drawn independently and uniformly at random, where the values for the mean have been drawn from the interval $[-10, 10)$ and the values for the covariance matrix from the interval $[0.5, 3)$.

5.3.1 Influence of Parameters

The influence of the number of landmarks is illustrated by Figure 5.4. The shapes of the curves show that the performance is increasing with the number of landmarks until it



(a) 1 Gaussian **(b)** 10 Gaussians **(c)** $|L|$

Figure 5.4: For data of different dimensionality, the figures give some details about the behavior of RED when the Mahalanobis or the number of landmarks is increased. All plots are aggregated over all synthetic datasets. d_i is the dimensionality of the data stream, for $i \in \{2, 5, 10\}$.

reached the dimensionality of the dataset. So, for d_2, the peak performance is reached at 2, for d_5, the peak performance is reached at 5, and, for d_{10}, the peak performance is reached at 10. The increase for $|L| \leq d$ is completely in line with Theorem 4. The decrease for $|L| > d$ can be explained by the increase of the vector space: Due to higher dimensionality of the vector space, fewer and fewer instances share the same space and, hence, fewer instances are available to provide an estimate for this region. This effect is also responsible for the increase of the variance for increasing numbers of landmarks (as visible for d_{10}). Due to the small number of instances per region, the density estimators are more sensitive to smaller changes, which results in an increased variance. So generally, up to d landmarks are beneficial for the performance, but the closer $|L|$ gets to d, the more instances are required to compensate for the higher variance.

The effect of the Mahalanobis distance is summarized by Figures 5.5 and 5.4. For lower dimensional datasets (e.g., d_2 and d_3), the performance is slightly degrading for minor increases of the Mahalanobis distance. For $M_{5.0}$ and $M_{10.0}$, however, we already see substantial improvements, which can be explained by fewer numbers of representatives having more instances at their disposal to provide a good estimate. For higher dimensional datasets (e.g., d_{10} and d_{20}), this trend is reverted, and we consistently observe that a low Mahalanobis distance is the better choice among the given selection. This can be explained by the possibility for each representative to specialize on regions with many instances. Otherwise, one representative would be responsible for a diverse set of instances. This observation is further supported by Figure 5.4. If there is only one Gaussian, a higher Mahalanobis distance is beneficial. But if we have several Gaussians (e.g., 10), a larger Mahalanobis distance leads to a degradation of the performance. So generally, one can say the higher the dimensionality of the data, the lower should be the Mahalanobis distance.

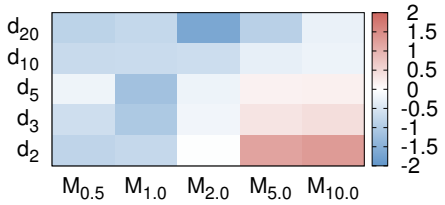


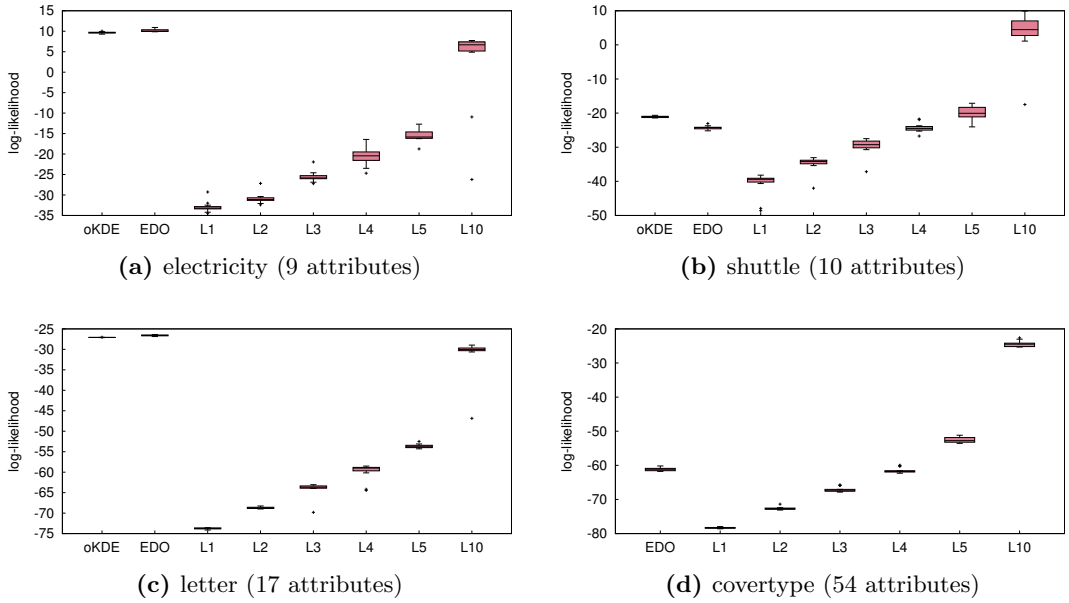
Figure 5.5: The heatmap summarizes the effects of the Mahalanobis distance for data of different dimensionality d_i ($i \in \{2, 3, 5, 10, 20\}$). It shows the improvement (red) and degradation (blue) in performance, if RED uses a specific Mahalanobis distance ($M_{0.5}, M_{1.0}, M_{2.0}, M_{5.0}, M_{10.0}$) compared with a Mahalanobis distance of 0.1.

5.3.2 Comparison to other Density Estimators

If data mining should be performed on RED estimates, the estimates have to describe the density of the data as accurately as possible. Since several compromises have been made to enable the estimation of data with many variables, we do not expect to outperform other density estimators. However, the performance should be in the same order of magnitude. To evaluate RED on real-world data, we selected EDO and the state-of-the art online density estimation method, i.e., the online kernel density estimator oKDE by Kristan et al. (2011). We measured the performance on four publicly available datasets: covertype (581,012 instances, 54 attributes), electricity (45,313 instances, 9 attributes), letter (19,999 instances, 17 attributes), and shuttle (58,000 instances, 10 attributes). For every parameter setting and for every dataset, the average log-likelihood is computed 15 times. In order to take possible concept drifts into account, the log-likelihood was computed in a prequential way, i.e., the log-likelihood of a given instance has been computed before using it for training. The instances used to compute the initial estimator (the first 250) were excluded from this computation.

The results are summarized in Figure 5.6. The most apparent observation is the increasing performance with increasing numbers of landmarks. As already observed on synthetic data, this increase in performance is accompanied with an increase in variance, which is visible to different degrees and does not even depend on the number of variables (electricity vs. shuttle). But most surprisingly is probably the abrupt increase on the covertype dataset. A more detailed analysis of this matter revealed that this is due to the nature of its instances. Covertype has 10 numerical variables, 43 binary variables, and one further nominal variable. As most of the binary variables are 0 for almost all instances, they do not provide sufficient additional information to justify more landmarks. Hence, with every new landmark, the estimators have fewer instances and provide estimates that are more sensitive to individual instances.

When we compare RED to oKDE and EDO, we observe that RED performs surprisingly well. For electricity and letter, RED is approaching the performance of oKDE and EDO. For shuttle and covertype, the performance is even better than that of oKDE



(a) electricity (9 attributes)

(b) shuttle (10 attributes)

(c) letter (17 attributes)

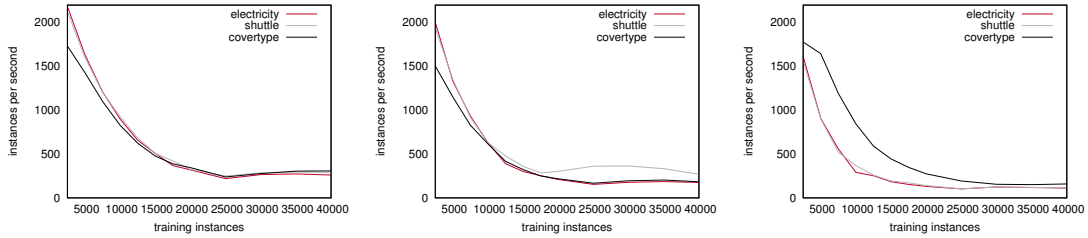
(d) covertype (54 attributes)

Figure 5.6: The figure shows a comparison of oKDE and EDO with RED. For RED, the number of landmarks has been varied ($|L| \in \{1, 2, 3, 4, 5, 10\}$). On the y-axis is the average log-likelihood computed in a prequential way.

and EDO, if RED uses 5 or 10 landmarks. This is surprising as EDO is supposed to produce exact estimates while RED uses EDO estimates to produce an approximation. But, on the shuttle and covertype dataset, the intrinsic dimensionality, i.e., the dimensions that increase the number of different density values over the instance space, is below ten, so that RED can take the advantage of having several specialized density estimates, compared to only one estimate that has to provide density values for all instances. For covertype, oKDE was not even able to process the dataset within 15 hours, whereas RED was able to produce results for all landmark sizes.

Hence, RED is able to compete with oKDE and EDO on low dimensional data, when a sufficient number of landmarks is chosen, while it can also handle high-dimensional data (e.g., more than 50 attributes). How many landmarks are sufficient for a specific datasets depends on two aspects: (1) its intrinsic dimension and (2) the selected landmarks (because the landmarks determine which dimensions are considered for evaluating the distance function). Considering that we made several compromises to enable the density estimation of data streams with many variables, RED performed very good on low- and medium-sized data streams.

The insights we gained about the parameters should be a useful guide for applications to other data streams. Alternatively, one could also follow a multi-layer approach where three or four RED estimators are initialized with different parameter settings. When enough instances are available, one could then choose the estimator with highest



(a) $|L| = 2$ **(b)** $|L| = 3$ **(c)** $|L| = 5$

Figure 5.7: The plots show the number of instances processed per second for different datasets and varying numbers of landmarks.

log-likelihood. This is enabled by favorable running time behavior.

5.3.3 Running Time

RED offers many opportunities for parallelism (e.g., one CPU core for each representative). For our evaluation, however, we wanted to keep the implementation simple and avoided any advanced optimizations. The current implementation is still fast enough for data stream applications, as Figure 5.7 summarizes. The general behavior is the same for all tested datasets (electricity, shuttle, and covertype). In the beginning, when almost no representatives are discovered and the density estimators are still very simple, the RED estimate is able to process 1400 or more instances per second. Then, with increasing numbers of training instances, it drops to several hundred instances per second before it stabilizes (at 100 to 300 instances per second, depending on the number of landmarks). This is in line with the expected behavior of the method. First, the instances are required to find a partitioning of the vector space. When this partitioning is converging and the corresponding density estimators have received a larger number of instances, the processing speed of the estimator becomes more and more constant.

5.4 Conclusions

We proposed a new approach for estimating densities of heterogeneous data streams with many variables, which reduces the dimensionality of the data by projecting it into a vector space. In particular, the algorithm creates a small number of instances, called landmarks, and creates a vector space in which the position of each instance is computed in terms of distances to the landmarks. Subsequently, the density is described by partitioning the vector space with representatives and estimating the density of each partition by employing online density estimators. In the theoretical analysis, we showed the validity and consistency of the presented density estimator. With experiments on synthetic and real-world data, we demonstrated that – despite

the compromises that had to be made to enable the estimation of densities with many variables – RED produces estimates having a comparable performance to that of state-of-the art density estimators. Keeping in mind that other approaches are possibly not able to handle large numbers of variables at all, RED could be, at this point, the only available option for some data streams.

Part II

A Probabilistic Condensed Representation of Data

Chapter 6

The MiDEO Framework

Traditional data mining algorithms operate on the data itself, which poses problems in data stream settings, where the amount of data is often too large to be kept in memory. Established solutions try to avoid memory limitations by pursuing window-based approaches, but they implicitly assume that collecting the data and performing the data mining task is either happening simultaneously or with a small temporal delay. Hence, the user has to know in advance, i.e., before collecting the data, whether he or she wants to perform a certain data mining task – something which is unrealistic in interactive environments where data streams may need to be analyzed after days or even weeks. For example, after performing pattern mining, the user discovers that she is only interested in patterns contained in a particular subset of the data. A window-based solutions would require the user to initiate another pattern mining process and to wait for new data instances to arrive.

To overcome these limitations, we introduce the MiDEO framework (Mining Density Estimates inferred Online) in this chapter, in which stream mining is no longer performed on original data in any form (e.g., windows, random samples, ...) but on a so-called *probabilistic condensed representation of data*. Its main components and their relationships are illustrated in Figure 6.1: At the center is the probabilistic condensed representation of data, denoted by F. It is basically an online density estimate $\hat{f}$ extended by inference capabilities, where $\hat{f}$ provides an estimate for the joint density of the data stream and the inference operations are algorithms that operate on $\hat{f}$ to retrieve statistical information. Since the density estimate is updated with every new instance, algorithms have the choice of working with the current estimate or to obtain a snapshot of it. The latter is especially important for more involved computations that assume a stationary data distribution. Before an algorithm performs a data mining task on $\hat{f}$, the user has tools to modify the density estimate – either to preserve the privacy of the entities described by the data or to restrict the estimate horizontally or vertically, i.e., selecting only a subset of the data or removing random variables. The result is a new probabilistic condensed representation F' on which algorithms can perform data mining without accessing the original data.

This clear separation between collecting data, preprocessing F, and performing a data mining task is one of the main features of this framework. It enables different user roles and a clear separation of concerns. For example, a company that does not have any expertise in the area of data mining could use an online density estimator

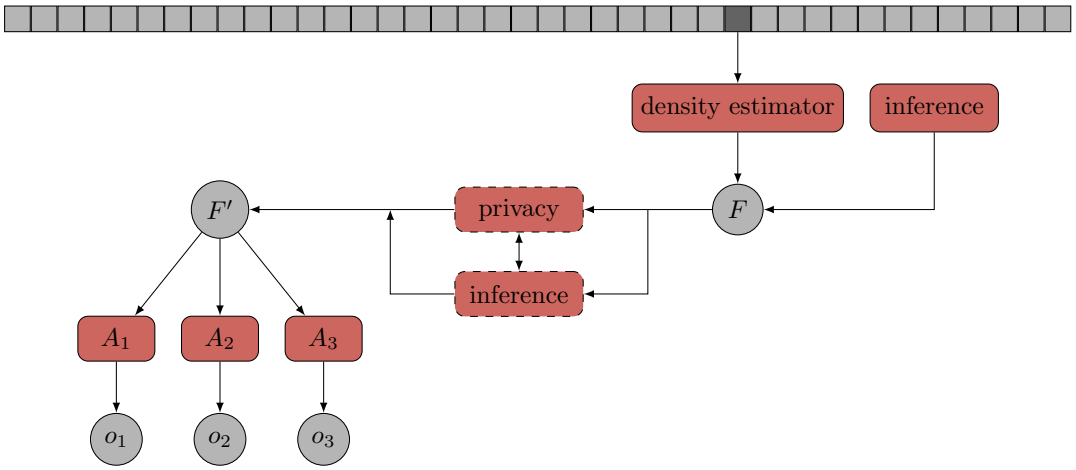


Figure 6.1: The main components of the MiDEO framework. *density estimator* is an online density estimator that produces a density estimate $\hat{f}$. Together with *inference*, $\hat{f}$ is extended by inference operations to F. The *privacy* and *inference* components to the left of F are optional. *privacy* is a set of algorithms that modify F until it meets certain privacy concerns. *inference* provides capabilities to restrict the estimate horizontally or vertically, i.e., selecting only a subset of the data or removing random variables. A_i is a data mining algorithm producing output o_i, $i \in \{1, 2, 3\}$.

to estimate the joint density of its data stream $\hat{f}$. After modifying $\hat{f}$ to meet certain privacy concerns, the company sends the resulting condensed representation of data to a data scientist, who can then use her expertise to provide an in-depth analysis of the data. In other words, the proposed framework decouples the process of density estimation from the extraction of human-understandable data mining results (patterns, rules, outliers, clusters, ...), thereby embracing two important principles of data mining and knowledge discovery in databases: the use of probabilistic methods to model the data and at the same time the value of human-readable patterns and models to gain a deeper understanding of the data. Compared to many traditional data mining algorithms, this has several benefits:

1. It can analyze data of much greater *volume* than could ever fit into main memory. As the density estimates can be updated continuously and the sizes of the estimates are a fraction of the size of the represented data, potential memory limitations do not play a major role anymore.

2. Along the same lines, the *speed of the stream* and the *speed of analysis* need not be coupled anymore. While the online density estimator constantly updates the density estimate $\hat{f}$, this density can be accessed by the user according to their own needs and analysis requirements.

3. *Unknown task at the time of collecting the data*: A probabilistic condensed rep-

resentation of the data can be useful, if the data mining task is not known at the time of collecting the data or several tasks need to be performed – possibly even depending on each other. As density estimates are "universal representations" of data, they can be used to derive any information needed, be it patterns, rules, outliers, clusters, or the like. To make this possible, inference tasks need to be supported for this specific type of density estimate.

4. *Privacy concerns*: To protect the privacy of the entities described by the data, it could be required that algorithms must not operate on the original data but some condensed representation of the data. This could be relevant if contracts between companies prohibit or restrict the access to data but allow to perform certain evaluation tasks. Density estimates are suitable for this task, because they keep all the relevant statistical properties of the data, *in principle* without revealing information about individual entities

5. Basis for *parallelization*: If we had an anytime density estimator instead of an online density estimator, the whole approach could provide a basis for parallelization on many common architectures and platforms. Fractions of the stream could be distributed, results could be gathered given the available time budget, and combined (mixed) again to obtain the overall density that is queried by the analysts.

6. The probabilistic condensed representation of data enables a clear separation of concerns, such that each step in the knowledge discovery process could be performed by a specialists without disclosing information that is irrelevant for the task at hand.

In the remainder of this chapter, we describe how the online density estimator EDDO (Chapter 3) can be extended to a probabilistic condensed representation of data F (Section 6.1) and how F needs to be modified to meet certain privacy concerns (Section 6.2). In Section 6.1, we will present algorithm for inference operations such as drawing instances, incorporating hard evidence, incorporating soft evidence, marginalizing out variables, and determining the density value of an instance (with respect to the given evidence). In Section 6.2, we discuss how to achieve k-anonymity (Sweeney, 2002) and t-closeness (Li et al., 2007) on F. Extensions to RED density estimates (see Chapter 5) are omitted, since the extension are rather a matter of managing the representatives than presenting new algorithmic ideas.

6.1 Inference

The algorithms presented in Part I yield density estimators providing a compact representation of the density. In its current state, however, information about the density value of a given instance can only be obtained with respect to the full density. This can severely limit the use of the estimate, as the user's interests probably change over

time (e.g., the user is only interested in a subset of the random variables after analyzing the density estimate). To overcome this limitation, we extend density estimates to so-called *probabilistic condensed representations of data*[1], which are density estimators together with infrastructure to pose queries on the densities. Among the most basic queries are inference operations such as drawing instances, incorporating hard evidence, incorporating soft evidence, marginalizing out variables, and determining the density value of an instance (with respect to the given evidence). On a higher level, these can be combined to more complex tasks such as pattern mining (see Chapter 7). In order to illustrate the inference operations, we consider three scenarios, which we call *inference scenarios*:

1. Let f be a joint density defined on variables $\mathcal{X} := \{X_1, \ldots, X_m\}$. Further, let $\mathcal{Y} := \{Y_1, \ldots, Y_l\} \subset \mathcal{X}$ and $\mathcal{Z} := \{Z_1, \ldots, Z_{m'}\} \subset \mathcal{X}$ be subsets with $\mathcal{Y} \cap \mathcal{Z} = \emptyset$. Then we can determine a new density
$$f'(Z_1, Z_2, \ldots, Z_{m'} \mid Y_1 = y_1, \ldots, Y_l = y_l), \tag{6.1}$$
where $y_j \in values(Y_j)$, $j \in \{1, \ldots, l\}$.

2. Let f, $\mathcal{X}$, and $\mathcal{Z}$ be as in 1. Further, let Y be a variable from $\mathcal{X}$ for which the user knows that it takes value y_1 with probability p_1, value y_2 with probability p_2, ..., and value $y_{|values(Y)|}$ with probability $p_{|values(Y)|}$. Then we determine a new density
$$f'(Z_1, Z_2, \ldots, Z_{m'} \mid Y'), \tag{6.2}$$
where Y' takes the value y_j with probability p_j, $j \in \{1, \ldots, |values(Y)|\}$.

3. Let f be as in Setting 1 and θ be a user-defined value that is between 0 and 1 in the discrete case and greater than 0 in the continuous case. Then we determine a new density f', such that, for $T = \sum_{\vec{x}' : f(\vec{x}') \geq \theta} f(\vec{x}')$, $f'(\vec{x})$ equals $\frac{f(x)}{T}$ for all instances $\vec{x}$ with $f(\vec{x}) \geq \theta$, and $f'(\vec{x}) = 0$ for all other instances. The continuous case is defined analogously.

Notice that, for the first two settings, $\mathcal{Y} \cup \mathcal{Z}$ is not necessarily $\mathcal{X}$ but could be a proper subset. In Setting 1, the user has hard evidence for certain features and is interested in the density defined over the features $Z_1, \ldots, Z_{m'}$. The situation in Setting 2 is almost the same as in Setting 1, but instead of hard evidence only soft evidence is available. In Setting 3, we determine a density that contains all instances exceeding a certain probability.

In this section, we present algorithms for the aforementioned inference operations and distinguish two cases: (1) the density estimators are employed with the base estimators proposed in this thesis (e.g., Hoeffding trees) and (2) different base estimators

[1] If it is clear from the context that we are referring to a *probabilistic condensed representation of data* as proposed in this thesis, we will also write *probabilistic condensed representation* and *condensed representation*. This is supposed to improve the readability and should not be confused with condensed representations known from the area of pattern mining.

are employed that do not necessarily support inference. For the first case, we exploit the base estimators and perform most of the operations on top of them. For the second case, we do not take knowledge of the base estimators into account and provide general procedures. These are less effective but can be used as a backup in case the first case is not applicable. Subsequently, we illustrate the inference operations using the three inference scenarios presented above and briefly show how accurately the inference operations are performed using two real-world datasets as example. Please notice that, for reasons of readability, we consider densities $f(X_1, \ldots, X_m)$. All results can be easily extended to $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$.

6.1.1 Hoeffding Trees as Base Estimator

The structure of Hoeffding trees contain implicit information about the frequency of certain variable-value combinations. Although each tree is only responsible for predicting the conditional probability of a given variable, it also provides information about other variables through its inner nodes: Let $f(X_i \mid X_1, \ldots, X_{i-1})$ be a conditional density that is represented by a Hoeffding tree ht. Further, let $path := node_1, edge_1, node_2, edge_2, \ldots, edge_l, node_l$ be a path from the root ($node_1$) to a leaf ($node_l$), where each $node_i$ corresponds to a variable and each $edge_i$ corresponds to a value that $node_i$ can take. Then $f(X_i \mid X_1, \ldots, X_{i-1})$ is the density of the values of X_j given the variable values of $path$. By construction of ht, each variable on the path tries to make $f(X_i \mid X_1, \ldots, X_{i-1})$ as discriminating as possible, i.e., choosing variables that allow to distinguish between values with low probability and those with high probability. The inner nodes of the resulting tree yield a partitioning of the data instances and the paths specify which values the variables should take to obtain certain probabilities. In this section, we will show how this structure can be modified and exploited to support the inference operations of a probabilistic condensed representation.

Incorporating Evidence

Hard evidence and soft evidence can directly be incorporated into Hoeffding trees. If hard evidence $Y = y$ is given, we can simply disable or remove all branches of variable Y that correspond to a value that is not equal to y. If soft evidence Y' is given, where Y' takes the value $Y'[k]$ with probability p_k, $k \in \{1, \ldots, |values(Y)|\}$, we have to make two changes to the structure of the Hoeffding trees: First, the edges are extended by weights representing the fraction of instances that passed through the edge divided by the instances that passed through the node. Second, the class distribution of the leaf classifiers is corrected according to the given soft evidence. If the weight of the edge is q and the soft evidence of the edge is p, then $\frac{p}{q}$ is the multiplier for that edge. The multipliers of all edges on the path, starting at the edge with the soft evidence and ending at the leaf classifier, are first multiplied and then multiplied with each value of the class distribution (see Algorithm 6).

Theorem 5. *Let $\hat{f}$ be a density estimate for a density $f(X_1, \ldots, X_m)$, Y be a random*

Algorithm 6: Incorporating evidence for Hoeffding tree estimators

Input: estimator $\hat{f}$, random variables $X_1, \ldots, X_m$, evidence for random variable Y where $Y[k]$ has probability p_k
Output: updated estimator

```
1 for n in f̂.nodes() do
2     if randomVariable(n) = Y then
3         (e_1, ..., e_|values(Y)|) ← edges(n)
4         for 1 ≤ k ≤ |values(Y)| do
5             weight(e_k) ← p_k / weight(e_k)
6         end
7     end
8 end
```

variable, and $(p_k \mid 1 \leq k \leq |values(Y)|)$ *be soft evidence given for* Y*. Algorithm 6 correctly modifies* $\hat{f}$ *to incorporate the soft evidence.*

Proof. Let $d = f(x_1, \ldots, x_m)$ be the density value of an instance $\vec{x} = (x_1, x_2, \ldots, x_m)$. The estimator $\hat{f}$ computes d by following the corresponding paths in the conditional density estimates $\hat{f}_i(x_i \mid x_1, \ldots, x_{i-1})$ from the root to a leaf, $1 \leq i \leq m$. In the following, we will show that the soft evidence is correctly incorporated into each $\hat{f}_i$.

Let $\hat{f}_i$ be arbitrary. Then there is a path $n_1, e_1, n_2, e_r, \ldots, e_L, n_{L+1}$, such that n_j are the nodes $1 \leq j \leq L$, e_j are the edges, $1 \leq j \leq L$, n_1 is the root of $\hat{f}_i$, n_{L+1} are the values of the target variable, and $\exists j \in \mathbb{N} : randomVariable(n_j) = Y$. Due to the structure of $\hat{f}_i$, the density value $d_i = \hat{f}_i(x_i \mid x_1, \ldots, x_{i-1})$ can be expressed as $d_i = weight(e_1) \cdot \ldots \cdot weight(e_L)$. By multiplying $weight(e_j)$ with $\frac{p_k}{weight(e_k)}$, as defined in Line 5 of Algorithm 6, one obtains $d_i = weight(e_1) \cdot \ldots \cdot weight(e_{k-1}) \cdot p_k \cdot weight(e_{k+1}) \cdot \ldots \cdot weight(e_L)$, which means that the soft evidence of Y is correctly reflected in d_i. □

6.1.2 Marginalization

When it comes to marginalizing out variables, there are only a few special cases in which the current structure can be manipulated to represent the marginal density. The problem lies with the children of the variable to be marginalized out, as it can easily happen that inner nodes of tree become a forest: For example, if there is a variable $X_i \in \mathcal{X}$ with $X_i \notin \mathcal{Z}$ where the descendant of X_i for its value v_1 is D_1 and the descendant of X_i for its value v_2 is D_2, then it may easily happen that $D_1 \cap D_2 = \emptyset$. Hence, there are no common variables on which the branches can be merged, and we end up with a forest instead of a single tree. Even if it was possible to find a common root, we would only have the instance counters for the target variable but not of the underlying instances.

Algorithm 7: Marginalization for drawing instances

Input: estimator $\hat{f} := [(w_1, cc_1), \ldots, (w_k, cc_k)]$ with k chains, where w_j are the weights and cc_j are the chains, random variables $\mathcal{Z} := \{Z_1, \ldots, Z_{m'}\}$

Output: *inst* drawn from $\hat{f}$

$nodes \leftarrow []$ `// tuples consisting of a weight and corresponding node`
for $1 \leq j \leq k$ **do**
  let $f_1, \ldots, f_m$ be the factors of cc_j
  for $1 \leq i \leq m$ *with* $X_i \in \mathcal{Z}$ **do**
    $distribution \leftarrow \left(\frac{1}{|values(X_i)|}, \ldots, \frac{1}{|values(X_i)|}\right)$
    $nodes \leftarrow \{(w_j, cc.root())\}$
    `// a breadth-first search to find all matching nodes`
    do
      $nodes' \leftarrow []$
      `// only one child is possible, because we have evidence for` $n.randomVariable()$
      while $(w, n) \in nodes$ *with* $n.randomVariable() \in \mathcal{Z}$ **do**
        $(w', n') \leftarrow$ child matching evidence
        $nodes' \leftarrow nodes' \setminus \{(w, n)\}$
        $nodes' \leftarrow nodes' \cup \{(w', n')\}$
      end
      `// every child is possible, because there is no evidence for` $n.randomVariable()$
      for $(w, n) \in nodes$ *with* $randomVariable(n) \notin \mathcal{Z}$ **do**
        if $n.isLeaf() == false$ **then**
          `// add all children to` $nodes'$
          for $edge \in edges(n)$ **do**
            $nodes' \leftarrow nodes' \cup \{(w \cdot weight(edge), targetNode(edge))\}$
          end
        else
          $distribution \leftarrow distribution + w \cdot n.distribution()$
        end
      end
      $nodes \leftarrow nodes'$
    while $nodes \neq \{\}$
    $z_j \leftarrow$ draw value according to $normalize(distribution)$
  end
end
return $(z_1, \ldots, z_{m'})$

Therefore, we propose to perform marginalizations on the fly, i.e., it is performed while drawing instances or computing density values. This can be achieved by aggregating all paths from the root to the leaves that match the requirements. Algorithm 7 demonstrates this idea at the example of drawing instances. The algorithm basically iterates over all relevant Hoeffding trees of the ensemble (some trees are not relevant, because their target variable is not an element of $\mathcal{Z}$). Then for each node, it distinguishes two cases: (1) The node has only one successor, because we have a value for the random variable of that node. (2) The node has all its children as successor, because no value is given for the random value of that node. In this fashion, the algorithm visits nodes until it reaches a leaf. At a leaf, the distribution of the target variable is extracted (in the case of drawing instances) and subsequently aggregated with the distribution of other leaves of that tree.

Drawing Instances

If we need to draw an instance from an estimator that uses a single classifier chain, we simply iterate over the classifiers from $f_1(X_1)$ to $f_m(X_m \mid X_1, \ldots, X_{m-1})$, draw a value based on the probability distribution of the represented estimate, and use the output as input for the next classifier.

Algorithm 8: Drawing instances from an ensemble

Input: estimator $\hat{f} := [(w_1, cc_1), \ldots, (w_k, cc_k)]$ with k chains, where w_j are the weights and cc_j are the chains, random variables $X_1, \ldots, X_m$
Output: *inst* drawn from $\hat{f}$

```
// select chain determining the ordering of the variables
```
$cc \leftarrow$ draw chain according to $(w_1, \ldots, w_k)$
```
// draw attribute values
```
for $X_i \in X_{cc.\pi(1)}, \ldots, X_{cc.\pi(m)}$ **do**
  $dist_i \leftarrow (0 \mid 1 \leq j \leq values(X_i))$
  for $1 \leq j \leq k$ **do**
    let f_j be the factor of cc_j with target variable X_i
    $dist_i \leftarrow dist_i + w_j \cdot f_j(X_i \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1})$
  end
  $normalize(dist_i)$
  $x_i \leftarrow$ draw value from $dist_i$ (using Algorithm 7)
end
return $(x_1, \ldots, x_m)$

Although it is straightforward to obtain a density estimate for a particular instance from the ensemble, it is no longer straightforward to generate data samples based on the estimated density. The simple process that can be used in the case of a single chain no longer applies, as every chain has a different variable ordering. Therefore, the in-

stance will be drawn based on a single ordering and the other densities will be adapted using marginalization (see Algorithm 8). At the beginning, the algorithm randomly selects an ordering $X_{\pi_{cc(1)}}, \ldots, X_{\pi_{cc(m)}}$ based on the classifier chains in the ensemble (line 1). In principle, one could use any ordering, but the chains are more accurate when using their own ordering (e.g., in case some instances have been discarded due to split decisions). Based on $X_{\pi_{cc(1)}}, \ldots, X_{\pi_{cc(m)}}$, the algorithm draws the values iteratively from $f_1(X_{\pi_{cc(1)}})$ to $f_m(X_{\pi_{cc(m)}} \mid X_{\pi_{cc(1)}}, \ldots, X_{\pi_{cc(m-1)}})$ by marginalizing out the variables $X_{\pi_{cc(i+1)}}, \ldots, X_{\pi_{cc(m)}}$ for each conditional density with target variable $X_{\pi_{cc(i)}}$. As a result, we obtain a probability distribution for each variable $X_{\pi_{cc(i)}}$ and for each ensemble member, which are then combined to a single probability distribution $dist_i$ for each $X_{\pi_{cc(i)}}$ (lines 3-8), $1 \leq i \leq m$. Finally, the value for each $X_{\pi_{cc(i)}}$ can be drawn from $dist_i$.

Theorem 6. *Let $\hat{f} = [(w_1, cc_1), \ldots, (w_k, cc_k)]$ be the k chains of an estimate $\hat{f}$ for the density $f(X_1, \ldots, X_m)$, where w_j are the weights and cc_j are the chains. Algorithm 8 draws instances according to f, if $\hat{f}$ is a consistent estimate and the number of training instances tends to infinity.*

Proof. Let $\vec{x} = (x_1, \ldots, x_m)$ be an arbitrary instance. We will show that $\vec{x}$ is drawn according to $f(X_1, \ldots, X_m)$, if $\hat{f}$ is a consistent estimate for f. According to the product rule, every ordering of variables represents f equally well, i.e.,

$$f(X_1, \ldots, X_m) = f_1(X_{\pi(1)}) \cdot \prod_{i=1}^{m} f_i(X_{\pi(i)} \mid X_{\pi(1)}, \ldots, X_{\pi(i-1)}),$$

where $\pi : \{X_1, \ldots, X_m\} \to \{X_1, \ldots, X_m\}$ is a one-to-one mapping. Hence, if $\hat{f}$ is a consistent estimate, line 1 can be considered a heuristic without an effect on the correctness of the algorithm, and line 3 through 8 compute the the weighted average over all classifier chains for all variables, which is only affected by the marginalization in line 6.

The marginalization, on the other hand, is described by Algorithm 7 and is a breadth-first search through the Hoeffding tree where all matching paths are combined: Each branch of the tree represents a partition of the variable-value space. When marginalizing out variables $\mathcal{X} \setminus \mathcal{Z} := \{Z_1, \ldots, Z_{m'}\}$ for a conditional density estimate $\hat{f}_i$, a variable $X \in \mathcal{X} \setminus \mathcal{Z}$ can take any value in $values(X)$, so that the density values of all paths starting at a node n with $randomVariable(n) \in \mathcal{X} \setminus \mathcal{Z}$ have to be considered and the density values of the corresponding leaves have to be combined. By performing a breadth-first search and averaging over these leaves, one obtains the density value of $\hat{f}_i(Z_i \mid Z_1, \ldots, Z_{m'})$, which is exactly what Algorithm 7 does. □

6.1.3 Continuous Base Estimators

Since the continuous base estimator presented in Chapter 4 are based on a slightly modified Hoeffding tree, which does not clash with the inference operations, most of the algorithms presented in the previous subsections can be easily extended:

Evidence With a slight modification evidence is already supported due to the Hoeffding tree (i.e., the class probability estimator). We only need to correct the $N_{c_{x_j}}$ to comply with the evidence.

Drawing Instances Iterate over all Gaussians, draw a value x'_i from each Gaussian $\mathcal{N}(x; x_i, \sigma^2_{kernel})$, and compute $\frac{1}{N} \sum_{i=0}^{N} w(x_i \mid X) \cdot x'_i$.

Marginalization is already supported.

6.1.4 Arbitrary Base Estimator

If Hoeffding trees should not or cannot be used as base estimator, one can design new algorithms that are suitable for the selected base classifier or use algorithms that are independent from the base estimator. Whereas the first requires additional time for developing these algorithms, the later will probably lead to longer running times. Longer running times are only acceptable, however, if a prototype is being developed or no other algorithms exist yet. In this section, we propose some general algorithms that are suitable for any base estimator.

Incorporating Evidence

If the density estimator is not based on Hoeffding trees, we pursue a more general approach, which is independent from the base estimator and only requires the two operations: drawing instances and computing density values. Since both of these operations are fast in the presented framework, we can train a new density estimator that fulfills the desired properties. It suffices to draw instances and discard all those which do not match the evidence. The training finishes as soon as the new density estimator differs not more than d percent from the original density estimate on the most recent sample in terms of log-likelihood and at least m instances have been used for training, where m ensures that enough instances have reached the new density estimator. A more detailed description is given by Algorithm 9. Please notice that the notation follows that of soft evidence, but we can simulate hard evidence by giving a probability of 1 to the hard evidence and setting the probability of the remaining values to 0.

Due to this general approach with only few requirements imposed on the base estimators, the algorithm is less effective in some cases. If the evidence makes an unlikely value likely, this easily requires a tremendous number of instances – especially if one assigns a high probability to a value that actually has a low probability. Moreover, it can no longer be guaranteed that the sequence of instances is independent, which is often a requirement for the base estimator.

Algorithm 9: Incorporating evidence for arbitrary base estimators

Input: estimator $\hat{f}$, random variables $X_1, \ldots, X_m$, evidence $Y_1, \ldots, Y_l$ where $Y_j[k]$ has probability p_k, minimum number of training instances m (e.g., 10000), allowed deviation of the average log-likelihood d (e.g., 10%)

Output: updated estimator

$\hat{f}' \leftarrow$ initialize density estimator
$counter \leftarrow 0$
$LL, LL' \leftarrow 0$
do
  `// Requirements imposed on instance given by evidence`
  for $1 \leq j \leq l$ **do**
    $v_j \leftarrow Y_j[k]$ where k is drawn according to $(p_k \mid k \in \{1, \ldots, |values(Y_j)|\})$
  end
  `// Sample an instance fullfilling these requirements`
  do
    $inst \leftarrow$ sample instances from $\hat{f}$
    $\hat{f}'.update(inst)$
    $LL \leftarrow LL$ - $\log(\hat{f}(inst))$
    $LL' \leftarrow LL'$ - $\log(\hat{f}'(inst))$
  while $inst[Y_j] \neq v_j, j \in [1 : l]$
  $count \leftarrow counter + 1$
while $\frac{LL' - LL}{LL} < d \wedge counter < m$
return $\hat{f}'$

Marginalization

Even for arbitrary base estimators, marginalizing out variables can be performed quite efficiently: First, the hard or soft evidence is set in the estimator $\hat{f}$, such that all instances drawn from $\hat{f}$ respect the given evidence. Then a new estimator $\hat{f}'$ is initialized and trained with N instances from $\hat{f}$. Before the instances are forwarded to $\hat{f}'$, all variables are removed that are supposed to be marginalized out. After processing N instances, the resulting estimator is returned. This procedure is easily extended to ensembles of (weighted) classifier chains by adding an outer loop iterating over the classifier chains.

6.1.5 Illustration of the Inference Operations

In this section, we discuss how the inference operations can be used to perform queries on the probabilistic condensed representation and briefly show how the accuracy of the inference operations depends on the representation using two real-world datasets as example.

Inference Scenarios

At the beginning of this section, we proposed three inference scenarios to illustrate the inference operations imposed on a probabilistic condensed representation. In the following, we will show how these operations can be used to answer the queries described in the inference scenarios.

To evaluate Equation 6.1 and 6.2 of the first two inference scenarios, there are two main tasks that need to be performed: (1) specifying the variables Y_i (either to specify values or to specify the probabilities of specific values), and (2) marginalizing out variables from $\mathcal{X} \setminus \{Z_1, \ldots, Z_{m'}\}$. In the context of Bayesian networks, both tasks are usually performed by operating on conditional probability tables. Here, we perform these tasks on the estimators presented in the previous sections. With Algorithm 6 and 8, both tasks are already taken care of.

For Inference Scenario 3, we need to infer a density from our current estimate that only contains instances exceeding a certain probability θ. For that purpose, we do the following:

1. Draw instances from the current density estimate.
2. Check whether the value of the density estimate for this instance exceeds θ. If yes, use this instance to train the new target distribution. Otherwise, disregard this instance and go to 1.

As stopping criterion, we need a condition that guarantees a certain quality of the resulting density estimate. For that purpose, we compute how many instances need to be observed, so that an instance with probability $\theta_1 + \epsilon$ is included in a sample of size N, where ϵ is a small, positive number close to 0. Hence, using the upper

Chernoff bound, the sample size can be computed with high confidence by computing the most extreme deviation from the mean $N \cdot (\theta_1 + \epsilon)$. Using this sample size, we can then decide whether all relevant instances should have been drawn from the density estimate already.

Given a confidence level $0 < \theta_2 < 1$ and a $\lambda > 0$, we can apply the Chernoff bound (Motwani and Raghavan, 1995) and compute the minimal N, such that

$$\theta_2 < Pr[X > (1 + \lambda) \cdot \mu] < \left[\frac{e^{\lambda}}{(1 + \lambda)^{(1+\lambda)}}\right]^{\mu},$$

where $\mu = N \cdot (\theta_1 + \epsilon)$ and and X is the sum of independent Poisson trials with $Pr(X = 0) = 1 - \theta_1 - \epsilon$, $Pr(X = 1) = \theta_1 + \epsilon$. For continuous random variable, θ is treated as a density value instead of a probability.

Accuracy of Inference Operations

If Algorithms 8, 6, and 7 are used to perform inference, their results are mainly dependent on the accuracy of the density estimate. In the following, we briefly illustrate how much the result deviates from the actual frequencies by generating instances with marginalized out variables. For this purpose, we take a density estimate $\hat{f}$ and draw 1000 instances as follows:

1. select an integer m' from $[1; 0.5 \cdot m]$ uniformly at random,
2. select a subset $\mathcal{Z}$ from $\mathcal{X}$ of size s uniformly at random (i.e., the variables $\mathcal{X} \setminus \mathcal{Z}$ will be marginalized out from $\hat{f}$),
3. draw 100 instances from $\hat{f}(Z_1, \ldots, Z_{m'})$ with $Z_i \in \mathcal{Z}$, $1 \leq i \leq m'$,
4. go to Step 1 until 1000 instances have been generated.

Figure 6.2 illustrates the results of this experiment on the datasets movielens and us-census. If the density estimate would correspond to the frequencies of the instances, all dots would lie on the gray line, meaning that the density value equals the true density. On the movielens dataset, the density values generally only deviate slightly from the true density, where the deviation is smaller for large density values and larger for small density values. This is in line with our expectations, since instances with a smaller density value occur less often and are therefore often missing when making split decision in the Hoeffding trees. This is further supported by the us-census dataset. Compared to its instance space, the number of available instances is relatively low, so that instances with a low density value are occasionally not represented accurately in the tree (observe the large deviation for a density value below 0.5).

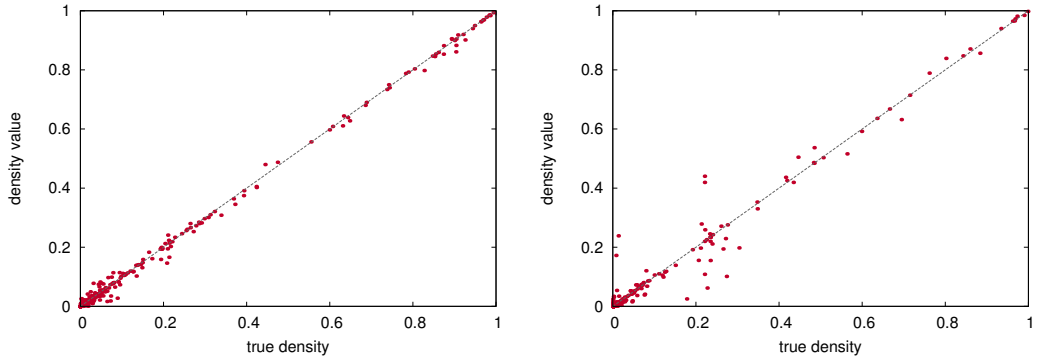


Figure 6.2: The plots illustrate the density values of instances from the movielens and the us-census dataset, where variables have been marginalized out at random. On the x-axis is the frequency of the instance, and on the y-axis is the estimated density value.

6.2 Privacy-Preserving Data Mining

When mining data, one is looking for general patterns. But the more specific they become, the smaller is the set of data instances that match the pattern and the higher is the risk that confidential information is disclosed. This risk is already lowered when, instead of the data, only statistical information is used, but confidential information can still be disclosed using inference. The same applies to probabilistic condensed representations of data as proposed in Part I and Section 6.1. In this section, we discuss how EDDO estimates can be modified to preserve the privacy of the entities described by the data.

6.2.1 k-Anonymity

EDDO allows to separate the process of collecting the data and the process of analyzing it. Consequently, a company could take their current probabilistic condensed representation as a snapshot and send it to a data scientist of a possibly different company for mining purposes. However, before sending it, they should ensure that confidential information about individuals is not disclosed, which can easily happen if only a few people share a set of features, e.g., current female heads of government. In the context of data, Sweeney (2002) discussed the problem of preserving the privacy of individuals and introduced the notion of *k-anonymity*. It basically demands that, for a set of attributes $\mathcal{Q}$ that could be used to identify entities, there are at least k instances for every possible attribute-value combination over $\mathcal{Q}$. The attributes in $\mathcal{Q}$ are called quasi-identifiers and are typically something like age, gender, or occupation. In the following, we will show how to achieve k-anonymity for an EDDO estimator.

Since the joint density is estimated by chains of Hoeffding trees and the factors estimated by each Hoeffding tree are independent from each other (according to the

product rule), it suffices to check each Hoeffding tree (see Theorem 7). A Hoeffding tree ht preserves k-anonymity, if k instances have been observed for every attribute $Q \in \mathcal{Q}$ and for every value belonging to Q. Hence, we only have to iterate over the tree and check the instance counts of nodes corresponding to Q. If all counts are greater than $k-1$, k-anonymity is preserved. Otherwise, it is not.

If a tree does not preserve k-anonymity, the tree needs to be modified. Sweeney (2002) suggested two strategies to ensure k-anonymity: removing attributes from the data and grouping attribute values (e.g., age ranges instead of individual ages). Both strategies are supported by EDDO estimators. Removing an attribute can be achieved by deleting all branches that have Q as its source and by deleting the Hoeffding tree that has Q as its class attribute. Grouping attribute values can be achieved by retraining the branches that have a node associated with Q as its source. Algorithm 10 applies these strategies in a greedy way until k-anonymity is achieved. It iterates over the parts of the Hoeffding trees that lead to a violation of the k-anonymity and tries to solve the problem by merging attribute values (lines 6-14). If the merging does not work (i.e., only a single branch is left), it deletes the attribute from the original Hoeffding tree (lines 11-14) to avoid any errors introduced by merging branches. This procedure terminates, since $|\mathcal{Q}|$ and $|\mathcal{N}|$ are finite and all $node \in \mathcal{N}$ will be deleted in the worst case.

Theorem 7. *Let $\hat{f}_{ECC}$ be an EDDO estimator and $\mathcal{Q}$ be a set of quasi identifiers. Algorithm 10 ensures that $\hat{f}_{ECC}$ preserves k-anonymity for $Q \in \mathcal{Q}$.*

Proof. (proof by contradiction) Let $\mathcal{Q}$ be the set of quasi identifiers and let G be a group of entities that is given by some evidence $(Z_1, v_1), \ldots, (Z_l, v_l)$ with $Z_i \in \mathcal{X}$ and $v_i \in values(Z_i)$, $1 \leq i \leq l$. Assume that there is a random variable $Q \in \mathcal{Q}$ where an instance count of $\hat{f}'_{ECC}(Q \mid Z_1 = v_1, \ldots, Z_l = v_l)$ is less than k. Algorithm 10 guarantees that there is no child of a $Q \in \mathcal{Q}$ with an instance count of less than k in any $ht \in \hat{f}_{CC} \in \hat{f}_{ECC}$. Hence, in order to enforce a count of less than k, one needs to combine paths from several trees. Let $(Z_1, v_1), \ldots, (Z_l, v_l)$ be a combination for which $\hat{f}'_{ECC}(Q \mid Z_1 = v_1, \ldots, Z_l = v_l)$ has an instance count of less than k. Relevant paths are those that start at the root, end at a leaf, and contain all elements from $\mathcal{P} := \{(Z_1, v_1), \ldots, (Z_l, v_l)\}$. Relevant nodes $\mathcal{N}$ are the ones contained in the largest suffix not containing any element from $\mathcal{P}$. Hence, for each Hoeffding tree ht in $\hat{f}'_{ECC}$, the conditional probability $\hat{f}'_{ECC}(Q \mid Z_1 = v_1, \ldots, Z_l = v_l)$ results from $\sum_{C \in \mathcal{N}} weight(C) \cdot weight(ht) \cdot distribution(node)$, where $weight(C)$ is the weight induced by soft evidence and $weight(ht)$ is the weight of the classifier chain to which ht belongs. Translating the density values of EDDO to counts, we obtain $\sum_{C \in \mathcal{N}} w_C \cdot counts(C)$. Hence, the only way to construct a count smaller than or equal to k is that $counts(C)$ fulfills this property already for at least one $C \in \mathcal{N}$. This is, however, prevented by Algorithm 10. Since all choices were arbitrary, this contradicts the initial assumption. □

So, after estimating the density of a data stream, the user has tools at her disposal to check whether the probabilistic condensed representation preserves k-anonymity and

Algorithm 10: k-anonymity

Input: $EDDO_{ECC}$ f_{ecc}, sensitive attributes $\mathcal{Q}$
Output: f' preserving k-anonymity

```
1  for Q ∈ 𝒬 do
2      for ht ∈ f_ecc do
3          ht' ← copy of ht
4          𝒩 ← collect nodes associated with Q in ht'
5          for node ∈ 𝒩 do
6              for A ∈ node.branches() s.t. Q violates k-anonymity at A do
7                  while Q violates k-anonymity at A do
                       // group attribute values
8                      B ← node.branches()[i + 1], where node.branches()[i] = A
9                      A ← merge(A, B)
10                 end
                   // remove attribute, if only one branch is left
11                 if node.branches() = 1 then
12                     ht' ← ht
13                     ht'.delete(node)
14                 end
15             end
16         end
17     end
18 end
```

to modify it if necessary. Due to the nature of the modifications, the representation is possibly less accurate. Regarding data privacy, however, this is not only acceptable in many applications but actually desired – at least from the viewpoint of the entities described by the data

6.2.2 t-Closeness

Although demanding k-anonymity is a first step towards privacy-preserving data mining, it still gives ways to different types of attacks with which confidential data can be disclosed (Soria-Comas and Domingo-Ferrer, 2013). To address this problem, several extensions have been developed and the *t-closeness* model is one of the strongest approaches (Li et al., 2007). It demands that, for a certain variable-value combination, the distribution of a sensitive variable does not deviate more than a threshold t, in the following denoted by δ, from the corresponding global distribution. To ensure t-closeness for an EDDO estimator, it suffices to perform two types of actions:

Adapting the edge weights Between a node *node* and its children are edges. Each edge has a weight representing the probability of following this edge and going to the corresponding child. Hence, the successor of *node* is described by a probability distribution d over its children. If $node \in \mathcal{Q}$ and d deviates by more than δ from the global distribution g, d can be adapted accordingly.

Adapting the class distributions Some Hoeffding trees have a sensitive variable as a class attribute. If the class distribution d deviates by more than δ from the global distribution g, d is adapted accordingly.

The global distribution of a variable $Q \in \mathcal{Q}$ can be easily estimated by computing $\hat{g} = \hat{f}(X)$, i.e., the variables $\mathcal{X} \setminus \{Q\}$ have been marginalized out. To compute the distance between the estimate of the global distribution $\hat{g}$ and $\hat{f}$, there are several well-known distance measures on discrete probability distributions such as the variational distance or the KL-divergence. However, the authors who proposed the notion of t-closeness (Li et al., 2007), suggested to use the *Earth Mover's Distance* instead (also known as the Wasserstein metric or Kantorovich metric (Gibbs and Su, 2002)) In this thesis, we do not assume a specific distance measure and simply write $\|\cdot\|$ for the distance.

Algorithm 11 formalizes these ideas. It corrects the probability distributions for every quasi identifier $Q \in \mathcal{Q}$ (line 1). In particular, it determines the global distribution $\hat{g}$ of Q (line 2) and then considers every node that is associated with Q (line 3-4). If it is a leaf, Q is the class attribute of the Hoeffding tree and it takes the class probabilities (line 6). If it is an inner node, it considers the distribution of its outgoing edges (line 8). Subsequently, it measures the distance between the global distribution $\hat{g}$ and the node distribution d (line 10). If it is larger than the given threshold δ, it corrects d until $\|\hat{g} - d\|$ is below δ. To do so, it changes each component of d towards $\hat{g}$ by iteratively correcting it by a factor of $\frac{step}{100}$ (lines 10-14), where *step* is a user-defined

Algorithm 11: t-closeness

Input: $EDDO_{ECC}$ f_{ecc}, sensitive attributes $\mathcal{Q}$, the accepted deviation from the global distribution δ, the percentaged difference by which a distribution is changed *step*

Output: f' preserving t-closeness

```
1  for Q ∈ 𝒬 do
2      ĝ ← f̂(Q)
3      𝒩 ← collect nodes associated with Q
4      for node ∈ 𝒩 do
           // determine node distribution
5          if node.isLeaf() then
6              d ← node.getClassDistribution()
7          else
8              d ← node.getEdgeDistribution()
9          end
           // reduce distance ‖ĝ − d‖
10         while ‖ĝ − d‖ ≥ δ do
11             for 1 ≤ i ≤ values(Q) do
12                 c ← ĝ[i] − d[i]
13                 d[i] ← d[i] + c · step/100
14             end
15         end
16     end
17 end
```

threshold that is chosen according to the intended precision. Theorem 8 states that Algorithm 11 turns a given EDDO estimator into an estimator preserving t-closeness.

Theorem 8. *Let $\hat{f}_{ECC}$ be an EDDO estimate, $\mathcal{Q}$ be a set of sensitive attributes, and δ be a threshold for the accepted deviation from the global distribution. Algorithm 11 ensures that $\hat{f}_{ECC}$ preserves t-closeness for all $Q \in \mathcal{Q}$.*

Proof. (proof by contradiction) Let $\mathcal{Q}$ be the set of sensitive attributes and let G be a group of entities that is given by some evidence $(Z_1, v_1), \ldots, (Z_l, v_l)$ with $Z_i \in \mathcal{X}$ and $v_i \in values(Z_i)$, $1 \leq i \leq l$. Assume that the probability distribution of a random variable $Q \in \mathcal{Q}$ deviates more than δ from the probability distribution $\hat{f}'_{ECC}(Q \mid Z_1 = v_1, \ldots, Z_l = v_l)$, i.e.,

$$\|\hat{f}'_{ECC}(Q) - \hat{f}'_{ECC}(Q \mid Z_1 = v_1, \ldots, Z_l = v_l)\| \leq \delta.$$

Algorithm 11 guarantees that the corresponding deviation for every node in $\hat{f}'$ is always smaller than or equal to δ. Hence, in order to enforce a deviation of more

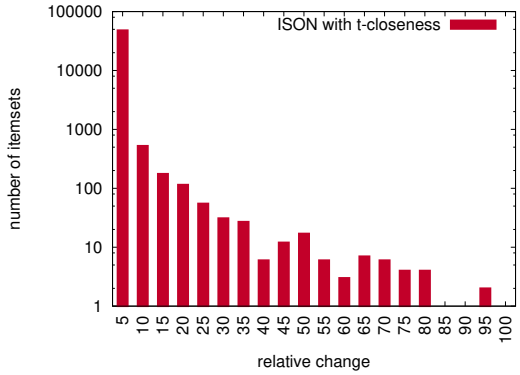


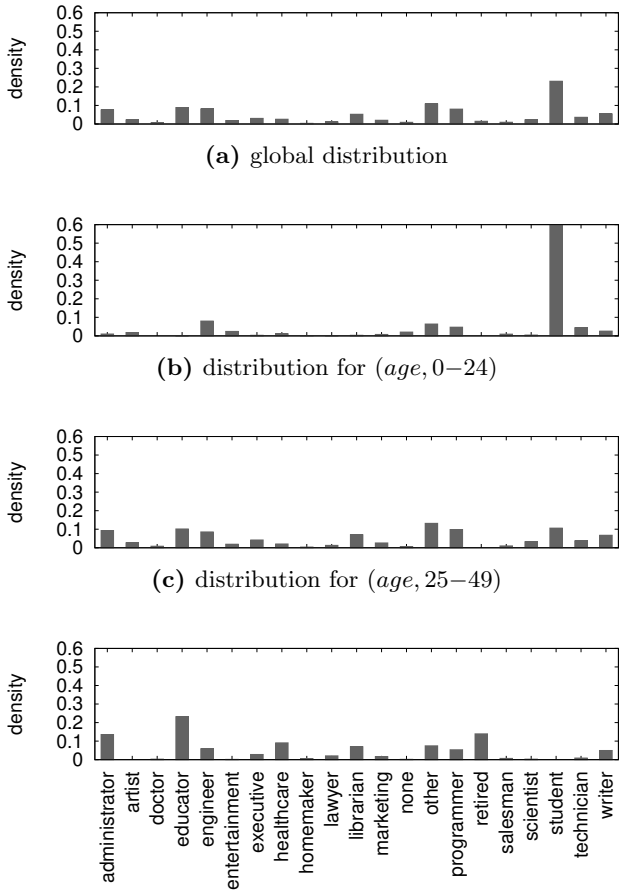
Figure 6.3: This histogram shows how a t-closeness algorithm with $\delta = 0$ affects the supports of itemsets containing the attributes *age* and *occupation*. On the x-axis is the percentaged deviation of the support, i.e., $x_{i-1} \leq \frac{f(itemset)}{f'(itemset)} \cdot 100 \leq x_i$, where f is the original estimate, f' is the estimate after applying the t-closeness algorithm, and *itemset* is an itemset. On the y-axis is the number of itemsets matching this percentaged deviation.

than δ, one needs to combine paths from several trees. Let $(Z_1, v_1), \ldots, (Z_l, v_l)$ be a combination for which $\hat{f}'_{ECC}(Q \mid Z_1 = v_1, \ldots, Z_l = v_l)$ deviates more than δ from $\hat{f}'_{ECC}(Q)$. Relevant paths are those that start at the root, end at a leaf, and contain all elements from $\mathcal{P} := \{(Z_1, v_1), \ldots, (Z_l, v_l)\}$. Relevant nodes $\mathcal{N}$ are the ones contained in the largest suffix not containing any element from $\mathcal{P}$. Hence, for each Hoeffding tree ht in $\hat{f}'_{ECC}$, the conditional probability $\hat{f}'_{ECC}(Q \mid Z_1 = v_1, \ldots, Z_l = v_l)$ results from $\sum_{C \in \mathcal{N}} weight(C) \cdot weight(ht) \cdot distribution(node)$, where $weight(C)$ is the weight induced by soft evidence and $weight(ht)$ is the weight of classifier chain to which ht belongs. Hence, $\|\hat{f}'_{ECC}(Q) - \hat{f}'_{ECC}(Q \mid Z_1 = v_1, \ldots, Z_l = v_l)\| =$

$$\sum_{C \in \mathcal{N}} w_C \cdot \|\hat{f}'_{ECC}(Q) - distribution(C)\|,$$

where $w_C = \frac{weight(C) \cdot weight(ht)}{\sum_{C \in \mathcal{N}} weight(C) \cdot weight(ht)}$. Since $\|\hat{f}'_{ECC}(Q) - distribution(C)\| \leq \delta$ for all $C \in \mathcal{N}$ and $w_c \leq 1$, it follows that $\|\hat{f}_{ECC}(Q) - \hat{f}'_{ECC}(Q \mid Z_1 = v_1, \ldots, Z_l = v_l)\| \leq \delta$. Since all choices were arbitrary, this contradicts the initial assumption. □

In addition to Theorem 8, we also evaluated the effects of the t-closeness property on real-world data. The plot in Figure 6.3 illustrates how many and how strongly itemsets are changed, if one allows no deviation from the global distribution for the attributes *age* and *occupation* of the movielens dataset. It shows that most of the density values are modified compared to the initial density estimator, which is in line with our expectation, because the t-closeness algorithm makes sure that more extreme distributions are brought closer to the global distribution (Figure 6.4 provides additional insights into this matter). This can also affect variable-value combinations not



(a) global distribution

(b) distribution for $(age, 0{-}24)$

(c) distribution for $(age, 25{-}49)$

(d) distribution for $(age, 50{-}74)$

Figure 6.4: The figure illustrates how the probability distributions of the movielens dataset differ for certain entity groups. At the top is the global distribution of the sensitive attribute *occupation*. Below are the distributions of the same attribute for entities of age 0−24, 25−49, and 50−75, respectively.

Table 6.1: When training an EDO density estimate on the movielens dataset, ISON extracts itemsets that can be associated with a single entity, a student of age 50-74. Without our t-closeness algorithm, the preferences of this student would be perfectly visible. After modifying the estimate with Algorithm 11, the itemsets are still visible, but the supports suggests that at least several hundreds of entities are associated with them, making it less likely that certain itemsets are associated with individual persons. Please notice that '¬' indicates that this person did not choose this genre.

before	after	itemset
0.00002	0.02406	$\neg film{-}noir$
0.00002	0.02291	$\neg mystery$
0.00002	0.01871	$\neg thriller$
0.00002	0.01854	$\neg film{-}noir$, $\neg thriller$
0.00002	0.01843	$\neg mystery$, $\neg thriller$
0.00002	0.01816	$\neg action$
0.00001	0.01782	$\neg action$, $\neg film{-}noir$
0.00001	0.01695	$\neg action$, $\neg mystery$
0.00001	0.01665	$\neg comedy$
0.00001	0.01472	$\neg action$, $\neg thriller$
0.00001	0.01333	$\neg film{-}noir$, $\neg thriller$
0.00001	0.01136	$\neg comedy$, $\neg thriller$
0.00001	0.01132	$\neg action$, $\neg comedy$
0.00001	0.01116	$\neg comedy$, $\neg film{-}noir$, $\neg mystery$
0.00001	0.01112	$\neg comedy$, $\neg mystery$, $\neg thriller$
0.00001	0.01089	$\neg action$, $\neg comedy$, $\neg film{-}noir$
0.00001	0.01028	$\neg action$, $\neg comedy$, $\neg mystery$

containing any sensitive attributes, since their density values is computed by marginalizing out variables from $\mathcal{Q}$, which influences the weighting of Hoeffding tree branches. These changes are clearly intended, as information about the age and occupation of entity groups are no longer different from other entity groups, and this protects smaller entity groups.

Table 6.1 illustrates how smaller groups can benefit from this effect. In this particular example, 17 itemsets with low support thresholds have been discovered from the movielens dataset. Given the number of instances, one can easily infer that each of these itemsets can be attributed to a single person. Because of the given age and the additional information that each person provided at least 20 ratings, one can also infer that all of them belong to the same person, a student between age 50 and 74. After applying the t-closeness algorithm, the frequency of these itemsets have been changed, so that they appear to originate from more than 400 persons.

6.2.3 Continuous Variables

In principle, Algorithms 10 and 11 can be extended to deal with continuous variables. It involves other strategies to achieve certain properties (e.g., achieving k-anonymity by micro-clustering (Domingo-Ferrer and Torra, 2005)) and involves other tools to identify which parts of the estimate needs to be changed (e.g., using an appropriate distance measure $\|\cdot\|$). However, we will not go into details here, since a translation to continuous or mixed variables is not different from other approaches such as the one proposed by Domingo-Ferrer and Torra (2005). The key difference is that the focus does not lie on instances but on the conditional and continuous probability densities. So instead of working with instances, one needs to work with kernels and their weights.

Chapter 7

Pattern Mining

In the previous chapter, we introduced the MiDEO framework and claimed that data mining can be performed on a probabilistic condensed representation without having access to the original data. Although we presented algorithms to perform inference and showed how to meet certain privacy-preserving properties, we did not support this claim with an actual data mining algorithm. In this chapter, we will complete our argumentation by presenting algorithms for one of the main data mining tasks: frequent itemset mining (Calders et al., 2004).

Additionally, we will show that the density estimates offer new possibilities of finding more relevant patterns. One of the main issues of frequent itemset mining is that the user is often confronted with a large number of itemsets and that is it is difficult to distinguish relevant itemsets from irrelevant ones. To master the sheer volume of itemsets, the user would probably restrict the search to longer itemsets or itemsets that contain particular items. But, even in that case, the result set could still consist of thousands of itemsets – a number too large to be scanned by a human user or further meaningfully processed by machine (e.g., for prediction, visualization, or dimensionality reduction). Hence, a user would rather be interested in fewer itemsets that are characteristic for the given data than obtaining all itemsets that are frequent. To demonstrate that probabilistic condensed representation can be used to mitigate these issues, we will introduce so-called *characteristic itemsets* and show how to compute them.

7.1 Frequent Itemsets

Traditionally, an itemset specifies the items that are included in a given transaction. In this thesis, we consider a more general setting where an item is a variable that could take several values. More formally, let $\mathcal{X} := \{X_1, X_2, \ldots, X_m\}$ be a set of discrete variables with finite domains $values(X_i)$, i.e., X_i takes the values $v \in values(X_i)$. Further, let $S := \vec{x}_1, \vec{x}_2, \ldots$ be a stream of instances coming from the same probability distribution, where $\vec{x}_i = \{(X_j, v_j) \mid 1 \leq j \leq m\}$ is a set of variable-value combinations. An itemset is a subset $I \subseteq \vec{x}$ and its support is the relative frequency of I in S, denoted as $freq(I)$, where $freq(I)$ is the expected value in case S is unbounded. To show that F_S can be determined from probabilistic condensed representation without accessing the original data, we present two approaches: The first approach relies on the EDDO

density estimates and exploits the tree structure of the underlying Hoeffding trees to construct the required itemsets. The second approach is more general, as it does not make any assumption about the density estimator and solely uses the inference operations that come with a probabilistic condensed representation.

7.1.1 ISON

Frequent itemsets are variable-value combinations that exceed a probability threshold θ. The probabilities of these combinations are contained in the classifier chains of EDDO estimates and can be computed by following the correct paths in its Hoeffding trees – from the roots to the leaves. Although each tree is only responsible for predicting the conditional probability of a given variable, it implicitly provides information about other variables through its inner nodes: Let $f(X_i \mid X_1, \ldots, X_{i-1})$ be a conditional density that is represented by a Hoeffding tree ht. Further, let $path := node_1, edge_1, node_2, edge_2, \ldots, edge_l, node_l$ be a path from the root ($node_1$) to a leaf ($node_l$), where each $node_i$ corresponds to a variable and each $edge_i$ corresponds to a value that $node_i$ can take. Then $f(X_i \mid X_1, \ldots, X_{i-1})$ is the density of the values of X_i given the variable values of $path$. By construction of ht, each node on the path tries to make $f(X_i \mid X_1, \ldots, X_{i-1})$ as discriminating as possible, i.e., choosing variables that allow to distinguish between values with low probability and those with high probability. The inner nodes of the resulting tree yield a partitioning of the data instances and the paths specify which values the variables should take to obtain certain probabilities. Hence, combining the paths from all trees would enable a direct construction of itemsets that is guided by the structure of the density estimate.

Pursuing this idea, we propose an algorithm in this section, called *ISON* (Itemset Mining on the Structure of Online deNsity estimates), that exploits this structure to guide the construction of itemsets (illustrated by Figure 7.1 and Algorithm 12): Unlike the well-known Apriori algorithm, which first constructs 1-itemsets, then 2-itemsets, ..., ISON immediately constructs itemsets of different length. It iterates over all Hoeffding trees and turns the path of these trees into itemset candidates (lines 3-10). Candidates for which $filter$ returns false are neither relevant for the output nor for future iterations and are therefore removed as early as possible. Extracting paths from the Hoeffding trees and filtering them is considered as Iteration 1. Then it starts merging already found itemsets to larger itemsets by computing the union over the items of each merge partner (lines 12-25). In the beginning, when only one iteration exists, the itemsets from the Hoeffding trees are merged pairwise. For iteration $i + 1$, ISON will take every itemset from iteration i and merges it with every itemset from the iterations $j < i$. As in the case of Iteration 1, each candidate needs to pass the given filter.

In principle, we could have also searched the space of itemsets the same way as Apriori, but the proposed approach has several benefits: (1) We immediately obtain itemsets of different lengths, so that the user can easily interrupt the mining process and already has itemsets of different length. (2) By constructing the density estimates,

Algorithm 12: *ISON*

Input: $EDDO_{ECC}$ $\hat{f}_{ecc}$, itemset filter $filter$
Output: the itemsets for which $filter$ is true

$iterations \leftarrow \emptyset$ `// generated candidates`
`// Iteration 0`
$C' \leftarrow \emptyset$
for $ht \in \hat{f}_{cc}$ *and* $\hat{f}_{cc} \in \hat{f}_{ecc}$ **do**
  $\mathcal{P} \leftarrow \{node_1, \ldots, node_l \in ht \mid node_1.isRoot() \wedge node_l.isLeaf()\}$
  for $path \in \mathcal{P}$ **do**
    $c \leftarrow$ turn $path$ into an itemset
    `// iterate over candidates by size and prune using` $filter$
    $it \leftarrow subsets(c, \textbf{pruning = true})$
    $C' \leftarrow C' \cup \{s \mid s \in it\}$
  end
end
$iterations.append(C')$
`// Iteration` > 0
while $|C'| > 0$ **do**
  $C_{curr} \leftarrow C'$, $C' \leftarrow \emptyset$
  if $|iterations| = 1$ **then**
    $mergePartners \leftarrow iterations[1]$
  else
    $mergePartners \leftarrow iterations[1:i]$
  end
  for $c_1 \in mergePartners$ *and* $c_2 \in C_{curr}$ **do**
    if $filter(merge(c_1, c_2))$ **then**
      $C' \leftarrow C' \cup \{merge(c_1, c_2)\}$
    end
  end
  $iterations.append(C')$
end
return $\bigcup_{1 \leq i \leq |iterations|} iterations[i]$

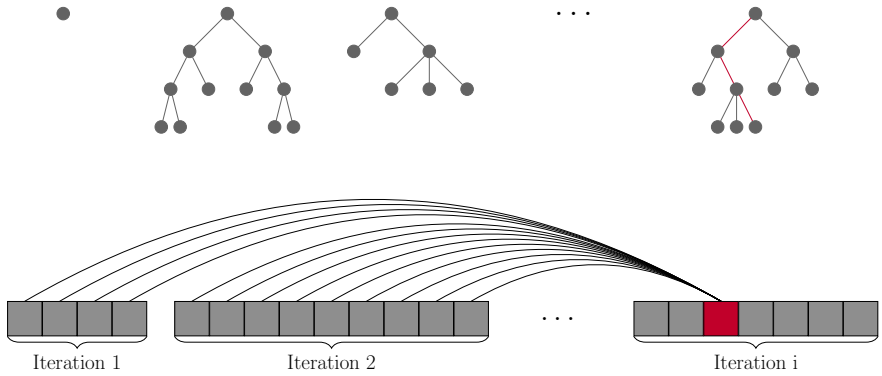


Figure 7.1: The two most important steps of ISON are illustrated by this figure. At the top are the Hoeffding trees of a classifier chain. The red path in the right-most Hoeffding tree shows one path that is turned into a candidate of a frequent itemset. At the bottom is an illustration of the merge operation of ISON, where the itemsets for iteration $i+1$ are constructed by combining every itemset of iteration i with every itemset from the previous iterations. The red square illustrates the itemset that is currently considered.

Algorithm 13: *isFrequent*

Input: $EDDO_{ECC}\hat{f}_{ecc}$, threshold $\theta \in [0,1]$, itemset C
Output: true iff C is frequent

$w \leftarrow []$ `// heuristic: weight chains`
for $\hat{f}_{cc} \in \hat{f}_{ecc}$ **do**
  `// determine the indices of the attributes in the chain ordering`
  $O \leftarrow [\texttt{argmin}_{j=1,\dots,m}(\hat{f}_{cc}[j] = X_i) \mid (X_i, v_i) \in C]$
  `// weight` $\hat{f}_{cc}$ `according to these indices`
  $w.append(\sum_{i=1}^{|O|} |O| - O[i] + \sum_{i=2}^{|O|} |O| - O[i] - O[i-1])$
end
$w \leftarrow normalize(w)$
return $\sum_{\hat{f}_{cc} \in \hat{f}_{ecc}} w[\hat{f}_{cc}] \cdot \hat{f}_{cc}(candidate) < \theta$

a lot of statistical information is already contained in the Hoeffding trees, which is a more informed way of searching and pruning the space of itemsets. (3) Other kinds of itemsets can be mined from the statistical information (e.g., the *characteristic itemsets* in Section 7.2).

Algorithm 12 is only the general framework of ISON that can be combined with many different filters. To mine frequent itemsets, we need a filter that estimates the

frequency of an itemset based on the given online density estimate $\hat{f}_{ecc}$. An example for such a filter is provided by Algorithm 13. It computes the frequency as $\sum_{\hat{f}_{cc} \in \hat{f}_{ecc}} w[\hat{f}_{cc}] \cdot \hat{f}_{cc}(candidate)$, where w is a weight vector for the classifier chains. Instead of using the original weights, however, it reweighs the chains using the itemset candidate and the chain ordering, because the ordering of a classifier chain can have an impact on how accurately it estimates certain density values (see Chapter 3). The closer a variable is to the end of the ordering, the more variables are considered during split decisions, which possibly yields less accurate results for itemset candidates consisting of only a couple of variables. To take this into account, the $isFrequent$ filter reweighs the classifier chains of $\hat{f}_{ecc}$ for each itemset candidate in dependence of the position of the variables in the classifier chain (lines 1-9).

If $\hat{f}_{ecc}$ is an accurate representation of f, i.e., the density values correspond to the true density values, then $ISON(\hat{f}_{ecc}, isFrequent(\hat{f}_{ecc}, \theta))$, denoted as $ISON \circ isFrequent$, is able to discover all frequent itemsets:

Theorem 9. *Let f be the joint density of the data stream S, and let $\theta \in [0,1]$ be the threshold for frequent itemsets. If $\hat{f}_{cc}$ is an accurate representation of f, then $ISON(\hat{f}_{cc}, isFrequent(\hat{f}_{cc}, \theta)) = \mathcal{F}_S$.*

Proof. $\mathcal{F}_S \subseteq ISON \circ isFrequent$: (proof by induction) There is a Hoeffding tree in $\hat{f}_{cc}$ with target variable X for all variables $X \in \mathcal{X}$. Hence, we have the discrete probability distribution of all $X \in \mathcal{X}$. This implies that $(X, v) \in \mathcal{F}(S)$ for all $X \in \mathcal{X}$ and for all $v \in values(X)$, since $\hat{f}_{cc}$ is an accurate representation of f.

Now, assume that $ISON \circ isFrequent$ contains all frequent itemsets of length k. Further assume that $itemset \in \mathcal{F}(S)$ be an arbitrary frequent itemset of length $k+1$ and that $itemset \notin ISON \circ isFrequent$. By the induction assumption, there is an iteration i in ISON where every proper subset of $itemset$ has been constructed by ISON. Hence, at latest in iteration $i+1$, $itemset$ will be constructed using one of its possible decomposition into subsets. Since $\hat{f}_{cc}$ is an accurate representation of f, $itemset \in ISON \circ isFrequent$, which contradicts our assumptions and, therefore, implies that $\mathcal{F}_S \subseteq ISON \circ isFrequent$.

$ISON \circ isFrequent \subseteq \mathcal{F}_S$: Follows immediately, because $\hat{f}$ is an accurate representation of f. □

From a theoretical point of view, a single classifier chain is sufficient. In practice, however, one classifier chain could easily miss important attribute interdependencies due the underlying ordering of the attributes. By using an ensemble of classifier chains, the probability of missing important attribute interdependencies is reduced, thereby increasing its robustness and accuracy:

Corollary 1. *Let f be the joint density of the data stream S, and let $\theta \in [0,1]$ be the threshold for frequent itemsets. If $\hat{f}_{ecc}$ is an accurate representation of f, then $ISON(\hat{f}_{ecc}, isFrequent(\hat{f}_{ecc}, \theta)) = \mathcal{F}_S$.*

7.1.2 POEt

ISON is specifically designed to extract frequent itemsets from the structure of EDDO density estimates. If the user wants to use another estimator, the algorithm needs to be adapted accordingly or needs to be replaced by a completely new approach. However, knowing the underlying structure of the density estimate is actually not necessary, since a probabilistic condensed representation already provides inference operations to access statistical information. Therefore, we propose another, simpler algorithm, called *POEt* (Pattern mining on Online dEnsity estimaTes), that only uses inference operations to mine frequent itemsets.

To evaluate the interestingness of an itemset, POEt pursues the same approach as ISON and estimates the frequency using the density estimate $\hat{f}$. In contrast to ISON, however, it does not use some threshold θ to select frequent itemsets but requires a parameter N specifying the number of desired candidates. These candidates are supposed to be itemsets with a high probability and are generated by drawing samples according to $\hat{f}$. But drawing them directly from $\hat{f}$ is not practicable, since the itemsets are usually of different lengths and only consists of a small number of variables. Therefore, we suggest to derive density estimate $\hat{f}'$ from $\hat{f}$ that are specifically designed for a corresponding subset of variables. In particular, we randomly create density estimates $f'(Z_1, \ldots, Z_{m'})$ with $\{Z_1, \ldots, Z_{m'}\} \subseteq \{X_1, \ldots, X_m\}$ and draw itemsets from f'. To get a variety of different itemsets, we restrict the number of itemsets per $\hat{f}'$ to a user-defined internal parameter M and create $\lceil \frac{N}{M} \rceil$ many densities.

The main procedure is given in Algorithm 14. As input parameters, it takes the density estimate $\hat{f}$, the variables $\mathcal{X}$ over which $\hat{f}$ is defined, the maximal number of candidates N, and Chernoff bound parameters. Additionally, it has an internal parameter M specifying the number of instances that are drawn from each generated density estimate. In lines 1-4, variable subsets are chosen at random, from which new densities $\hat{f}'$ are generated. Their size is chosen according to a geometric distribution, and the elements are chosen uniformly at random. As a consequence, there is bias preferring smaller itemsets over bigger ones, but, in our opinion, this is closer to what we have to expect from real-world applications (Zheng et al., 2001). In the second part of the algorithm (lines 5-12), a density estimate $\hat{f}'$ is derived for each subset of variables $\mathcal{Z}$ by marginalizing out the variables in $\mathcal{X} \setminus \mathcal{Z}$. From $\hat{f}'$ up to M instances are drawn and stored into $\mathcal{C}'$. After processing all subsets, we randomly select N itemsets from $\mathcal{C}'$, because $\lceil \frac{N}{M} \rceil \cdot M$ is possibly larger than N.

The subprocedure *createCandidateGeneratingDensity* is responsible for creating the densities $\hat{f}'$. In Chapter 6, we proposed algorithms that marginalized out variables by drawing instances from the original density estimate and removing the variables to be marginalized out. To determined the number of required training instances, we monitored the average log-likelihood and compared it to original density estimate. The situation for POEt, on the other hand, is more specific, as we actually do not need an accurate description of $f(Z_1, \ldots, Z_{m'})$ but an accurate description of $\{\vec{z} \mid f(\vec{z}) \geq \theta\}$, i.e., the itemsets exceeding a given probability threshold. This corresponds to Inference

Algorithm 14: POEt

Input: density estimate $\hat{f}$, set of all variables $\mathcal{X}$, maximal number of candidates N, and parameters of Chernoff bound $0 < \theta_c < 1$ and $\lambda > 0$

Output: set of candidates $\mathcal{C}$

Require: maximal number of candidates per subset M

```
// choose a subset of variables
```

$\mathcal{Z} \leftarrow \emptyset$

for $i = 0$ *to* $\lceil \frac{N}{M} \rceil$ **do**

  $\mathcal{Z} \leftarrow \mathcal{Z} \cup Z$, where $Z \subseteq \mathcal{X}$, $|Z|$ is chosen according to a geometric distribution with $Pr(l) = 0.5 \cdot (1 - 0.5)^{l-2}$ for $l = 2, 3, \ldots$ and the elements are chosen uniformly at random

end

```
// create a density estimator for each subset and draw instances
```

$\mathcal{C}' \leftarrow \emptyset$

for $Z \subseteq \mathcal{Z}$ **do**

  $\hat{f}' \leftarrow createCandidateGeneratingDensity(\hat{f}, X, Z, \theta_c, \lambda)$

  for $i = 1$ *to* M **do**

    $inst \leftarrow \hat{f}'.drawInstance()$

    $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{inst\}$

  end

end

return $\mathcal{C} \subseteq \mathcal{C}'$ *uniformly at random, such that* $|\mathcal{C}| = N$

Algorithm 15: createCandidateGeneratingDensity

Input: density estimate $\hat{f}$, set of all variables $\mathcal{X}$, $\mathcal{Z} \subseteq \mathcal{X}$, and parameters of Chernoff bound $0 < \theta_c < 1$ and $\lambda > 0$
Output: density estimate $\hat{f}'$
Require: mininum number of training instances N_{train} (e.g., 10^5)

`// Determine minimum observed probability from a sample`
$\theta_{sample} \leftarrow 1.0$
for $i = 1$ *to* $sampleSize$ **do**
  $inst \leftarrow \hat{f}.drawInstance()$
  $inst \leftarrow (inst[Z_1], \ldots, inst[Z_{m'}])$
  $\theta_{sample} \leftarrow \min(\theta_{sample}, \hat{f}(inst))$
end
`// Compute a lower bound for`
$\lambda \leftarrow$ find the minimal $0 < \lambda \leq 1$ using a binary search, s.t.

$$Pr[T < (1-\lambda) \cdot \mu] < e^{-\frac{\mu\lambda_T^2}{2}} < \theta_c \ ,$$

where $\mu = sampleSize \cdot \theta$ and T is the sum of independent Poisson trials with probabilities $Pr(T_i = 0) = 1 - \theta_{sample}$, $Pr(T_i = 1) = \theta_{sample}$.
$\theta \leftarrow \theta_{sample} \cdot (1 - \lambda)$
find the minimal N using a binary search starting at $N = 1$ with $N \in \mathbb{N}$, s.t.

$$\theta_2 < Pr[T > (1+\lambda) \cdot \mu] < \left[\frac{e^\lambda}{(1+\lambda)^{(1+\lambda)}}\right]^\mu \ ,$$

where $\mu = N \cdot \theta_1$ and T is the sum of independent Poisson trials with probabilities $Pr(T_i = 0) = 1 - \theta_1$, $Pr(T_i = 1) = \theta_1$.
`// Train estimator`
$\hat{f}' \leftarrow$ initialize a new density estimator
while $\hat{f}'.getNumberOfProcessedInstances() \leq \max(N, N_{train})$ **do**
  $inst \leftarrow \hat{f}.drawInstance()$
  if $\hat{f}[Z_1, \ldots, Z_{m'}](inst) \geq \theta$ **then**
    $\hat{f}'.update(inst)$
  end
end
return $\hat{f}'$

Scenario 3, for which we used Chernoff bounds to determine the number of required instances.

Algorithm 15 summarizes these ideas. It first estimates θ_{sample} based on a sample drawn from f (line 1-6). This is necessary, since the probabilities of the itemsets contained in the sample depend on the dimensionality of the dataset and the variance of the probabilities. However, as we only use a sample to determine θ_{sample}, it may happen that the actual threshold is below θ. Therefore, we correct θ_{sample} by computing the maximal deviation with respect to a given confidence level (line 7 and 8). Based on the resulting θ, the algorithm then computes the required number of training instances (line 9). Finally (lines 10-16), a new density estimator $\hat{f}'$ is initialized that is trained with instances drawn from $\hat{f}$ that exceed the probability threshold θ.

7.1.3 Evaluation

In this section, we evaluate the capabilities of ISON and POEt to discover frequent itemsets. For the experiments, we compare to Apriori, as an algorithm delivering the ground truth, and to Moment[1], as one of the few stream mining algorithms for which a functioning implementation is available.

ISON and POEt have been implemented as part of MiDEO[2], which builds on the MOA framework (Bifet et al., 2010). For Apriori we used the implementation by Christian Borgelt[3] and, for Moment, the MOA extension provided on the MOA website[4]. Since the implementation of Moment is not able to deal with non-Boolean variables and only produces closed itemsets, we added a wrapper that flattened the data stream and computed the frequent itemsets based on the closed frequent itemsets. As window sizes, we considered $1 \cdot 10^2$, $1 \cdot 10^3$, $1 \cdot 10^4$, $2 \cdot 10^4$, $4 \cdot 10^4$, which we selected according to the datasets with the smallest number of instances.

The evaluation is conducted on several datasets, the properties of which are summarized in Table 7.1. To generate synthetic datasets, we used the well-known IBM dataset generator (Agrawal and Srikant, 1994), where we set (1) the number of patterns to 5000, (2) the average transaction size to 4, (3) the average itemset lengths to 2 or 4, and (4) the number of instances to a value between $50,000$ and $50,000,000$. The real-world datasets are based on three publicly available datasets[5]: skin, pokerhand, and movielens. To obtain more meaningful itemsets and to make some trends more visible, we modified the movielens dataset slightly: Instead of having the age of the users as attribute in the movielens dataset, we introduce a binning and grouped users into the four groups 0-25, 26-50, 51-75, 76-100. Moreover, as the original dataset has relatively few instances compared to the number of possible combinations, we consider

[1] A comparison to Moment is strictly speaking neither necessary (because Apriori already delivers the ground truth) nor entirely fair (because Moment is intended to determine the currently closed frequent itemsets).

[2] `https://github.com/kramerlab/mideo`

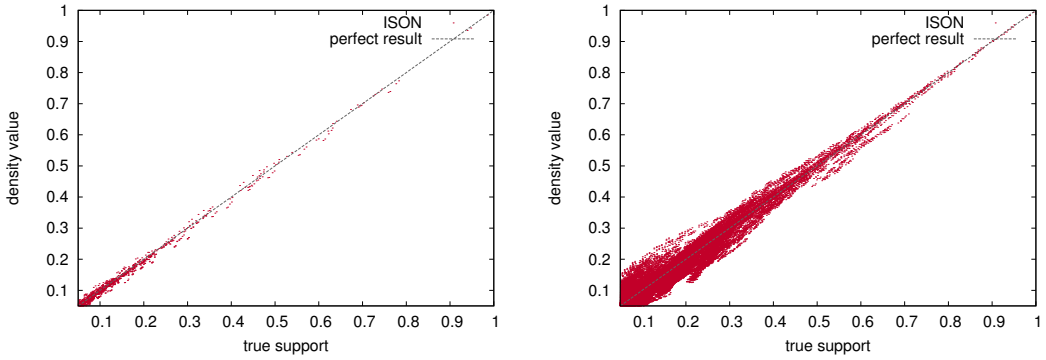
[3] `http://www.borgelt.net/pyfim.html`

[4] `http://moa.cms.waikato.ac.nz/moa-extensions/`

[5] `http://archive.ics.uci.edu/ml/`

Table 7.1: The table summarizes some properties of the datasets. In general, we distinguish two types of data: synthetic and real-world. *Shortcut* is a shorter naming scheme that is mainly used in plots to make axis labels more readable. *Purpose* specifies in which contexts the data has been used. It can be *itemset discovery*, *running time*, and *general*, where the first two refer to itemset discovery and running time. The AP in IBM_{AP} stands for average pattern length, i.e., the average itemset length.

Dataset	Shortcut	Purpose	Type	#Variables	#Instances
$\text{IBM}_{AP\in\{2,4\}}$	ibm01	general	synthetic	10	$50,000$
	ibm02				$500,000$
	ibm03				$5,000,000$
$\text{IBM}_{AP=4}$		running time	synthetic	10	$10,000,000$
					$20,000,000$
					$30,000,000$
					$40,000,000$
					$50,000,000$
pokerhand	poker	itemset discovery	real-world	11	$1,025,015$
skin				4	$225,009$
movielens-01	mov01			9	$49,282$
movielens-02	mov02			16	$49,282$
movielens-03	mov03			23	$49,282$



(a) mov01 (9 attributes and 49,282 instances) **(b)** mov02 (16 attributes and 49,282 instances)

Figure 7.2: These plots illustrate how the combination ratio affects the accuracy of the density estimate. Each dot represents an itemset. On the x-axis is the true support of this itemset; on the y-axis is the density value of the density estimate. Comparing both plots, we observe that the smaller the combination ratio, i.e., movielens-01 with 9 attributes and 49,282 instances compared to movielens-02 with 16 attributes and 49,282 instances, the more accurate is the representation of the supports.

two variants of the movielens dataset having fewer variables, which will help to determine how the accuracy of the representation affects the ability to discover frequent itemsets.

Accuracy of Density Estimate

ISON discovers itemsets based on a probabilistic condensed representation instead of the original data stream. Therefore, it requires an accurate description of the data, which is mainly determined by two factors: the number of δ-tolerant closed frequent itemsets Δ and the number of available instances N. For small δ, Δ basically characterizes the number of variable-value combinations that have differing probabilities. As each of these probabilities need to be captured by the density estimate, it cannot treat them the same way without compromising the accuracy of the estimate. If $|\Delta|$ is large and only few instances are available, an accurate description is difficult. The more instances are available, the easier it gets to approximate the probabilities.

To describe this dependence on Δ and N, we define the ratio of Δ and N as *combination ratio* and use it as a measure of how complex a dataset is with respect to estimating its density. Figure 7.2 illustrates the effect of this ratio on two version of the movielens dataset: movielens-01 and movielens-02. Due to the fewer variables in the movielens-01 dataset, there are fewer variable-value combinations and it is easier to estimate the density.

Itemset Discovery

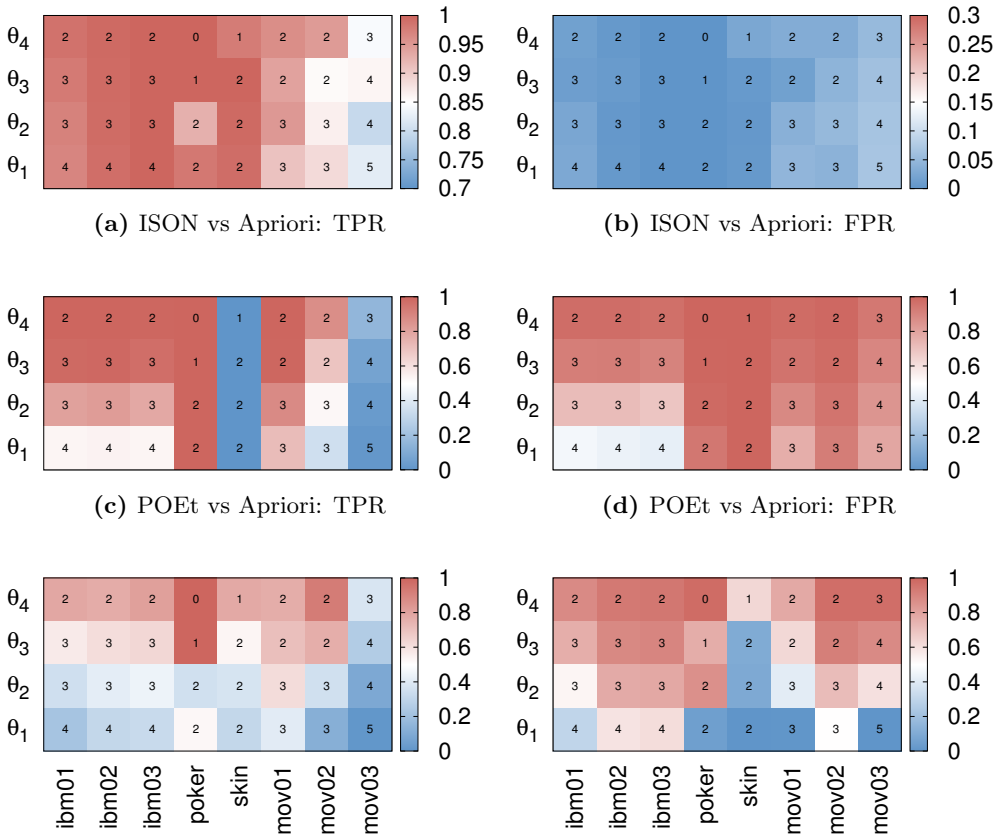
To assess the performance of the algorithms, we measure the true positive rate (TPR), i.e., $\frac{|\mathcal{F}_{ISON}(S) \subseteq \mathcal{F}_{Apriori}(S)|}{|\mathcal{F}_{Apriori}(S)|}$, and false positive rate (FPR), i.e., $\frac{|\{I \in \mathcal{F}_{ISON}(S) | I \notin \mathcal{F}_{Apriori}(S)\}|}{|\mathcal{F}_{Apriori}(S)|}$.

How many itemsets are discovered by ISON is illustrated in Figure 7.3. The TPR is very high on datasets with a low combination ratio (ibm01, ibm02, ibm03, pokerhand, mov01) and high on datasets with a higher combination ratio (mov02 and mov03). Compared to POEt, ISON is showing an overall better TPR, but on datasets with fewer frequent itemsets (i.e., ibm01, ibm02, ibm03, skin, and mov01 with $\theta \geq 0.20$) both algorithms are roughly on par. Regarding the FPR, ISON exhibits very low values on all datasets, whereas POEt exhibits high values. This due to the few assumptions that POEt imposes on the density estimate. POEt constantly retrains density estimates and draws a fixed number of itemsets from them. Many of these itemsets are not frequent, which causes the low FPR values. These itemsets can clearly be filtered out using the original density estimate, as in the case of ISON, but it shows that POEt has to generate a multiple of itemsets to achieve high TPR values. Moreover, many of the itemsets are also duplicates, which increases the number of required itemsets even further. This is not a problem for datasets with a low combination ratio but leads to a substantially increased running time for datasets with higher ratios. Hence, POEt can be successful at discovering extremely frequent itemsets but struggles with itemsets having a medium to low frequency.

ISON also exhibits a higher TPR and lower FPR than Moment on most datasets. If the window size is large enough and θ is large, we observe that ISON and Moment are on a similar level with respect to the TPR. These datasets have a rather low combination ratio, so that a fraction of the instances is already sufficient to determine the frequent itemsets precisely – at least for sufficiently large minimum support thresholds. Since Moment measures the frequency using a window, this helps to achieve a good performance. If the combination ratio is high, as in the case of the movielens dataset, the window size needs to be large to capture the frequencies of the itemsets properly. Surprisingly, however, increasing the window size on movielens seems to decrease the coverage. We assume that this is due to small distribution drifts in the data that cause Moment to correct the frequency of itemsets that are frequent with respect to the window but not frequent overall. Although this leads to a low coverage compared to Apriori, this behavior is actually intended – because Moment focuses on the currently frequent itemsets. Otherwise, an increase of the window size should increase the coverage of Moment. Similarly to POEt, Moment has a substantially higher FPR.

Number of Classifier Chains

In a separate experiment, we also analyzed how the number of classifier chains of the density estimate affect the discovery of frequent itemsets. The results are summarized in Figure 7.4. It shows that the TPR does not change much, whereas the FPR can be reduced substantially. When having several classifier chains to extract itemsets, the



(a) ISON vs Apriori: TPR

(b) ISON vs Apriori: FPR

(c) POEt vs Apriori: TPR

(d) POEt vs Apriori: FPR

(e) Moment-40000 vs Apriori: TPR

(f) Moment-40000 vs Apriori: FPR

Figure 7.3: These plots summarize how well ISON, POEt, and Moment discover frequent itemsets on various datasets. The plots present the TPR and FPR values with respect to several datasets and support thresholds. For ibm01, ibm02, ibm03, poker, skin, and mov01, $\theta_1 = 0.05$, $\theta_2 = 0.10$, $\theta_3 = 0.20$, $\theta_4 = 0.30$. For mov02 and mov03, $\theta_1 = 0.50$, $\theta_2 = 0.60$, $\theta_3 = 0.70$, $\theta_4 = 0.80$. On the left are the TPR values (red is better, i.e., larger values); on the right are the FPR values (blue is better, i.e., smaller values). The numbers in the heatmap are logarithms to base 10 of the number of itemsets. Please notice that we set the window size of Moment equally for all datasets. Therefore, it is sometimes greater than the number of instances. This is, however, beneficial for the TPR, if no drift occurs. Also notice that POEt did not finish on the skin dataset within 24 hours, so that we set the TPR to 0 and the FPR to 100 in these cases.

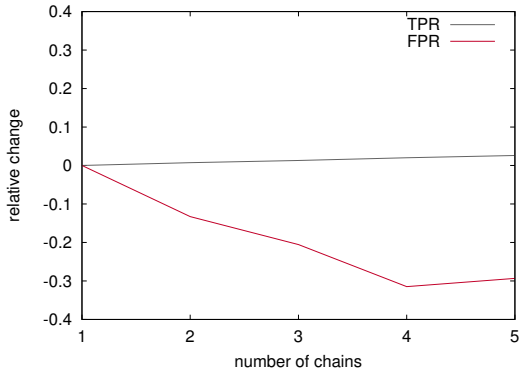
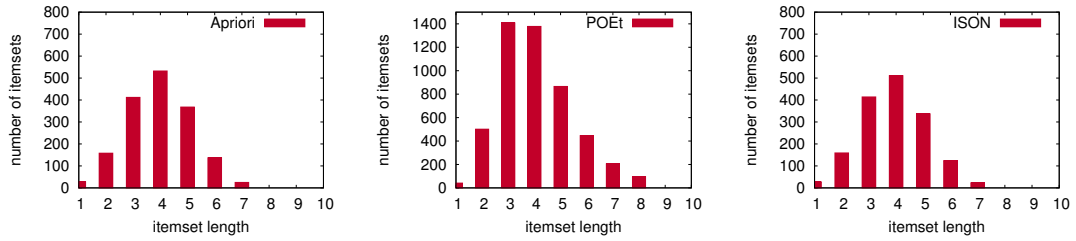


Figure 7.4: The plot shows how much the TPR and FPR values of ISON increase respectively decrease, if more than one classifier chain is used.



(a) Apriori **(b)** POEt **(c)** ISON

Figure 7.5: Histograms showing how the itemset lengths of Apriori, POEt, and ISON are distributed on the movielens-01 dataset. The minimum support threshold is 0.05. The distributions of Apriori and ISON are very similar, whereas the distribution of POEt is deviating substantially.

algorithm has a higher confidence on the itemsets that are represented in all classifier chains, whereas itemsets only available in one or two classifier chains are more or less not taken into account.

Itemset Length

If the estimate does not describe the joint density of the data stream accurately, ISON and POEt will both miss itemsets. In this case, one would expect that itemsets returned by the algorithms at least follow the same distribution as the ground truth. But we actually observed (see Figure 7.5) that the distributions of Apriori and ISON are very similar, whereas the distribution of POEt is deviating substantially. The latter is due to the way POEt is producing itemsets: namely by first picking the itemset length according to a geometric distribution and then drawing instances. Hence, POEt can only imitate the distribution of the ground truth, if it is actually provided.

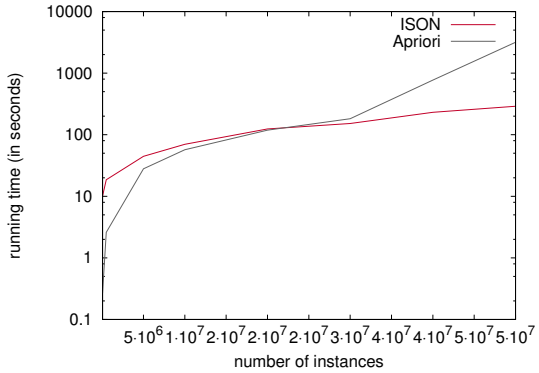


Figure 7.6: This plot compares the running time of Apriori and ISON, if the number of instances is increased. As datasets, we used the IBM datasets with an average transaction size of 4, an average itemset length of 4, and 5000 itemsets. The number of instances has been varied from 50,000 instances up to 50,000,000 instances.

Running Time

To analyze ISON's running time behavior, we conducted an experiment on IBM datasets that have equal properties and between 50,000 and 50,000,000 instances (see Table 7.1). The experiment has been conducted on a standalone computer with an Intel® Core™ i7-4770K CPU (4 cores, 3.50 GHz, 8 MB cache) and 32 GB RAM (of which 15 GB have been assigned to each job). As baseline, we used Apriori instead of Moment, since Moment only produces the currently closed frequent itemsets.

The results are summarized in Figure 7.6. In the beginning, when the dataset consists of fewer than 2 million instances, Apriori extracts the frequent itemsets faster than ISON – especially, when only few instances are available. But, for increasing numbers of instances, ISON's running time only increases moderately, whereas Apriori's running time increases substantially. Considering the approach taken by ISON, this was to be expected, because ISON performs the itemset discovery in two phases: It first estimates the joint density of the dataset and then extracts the itemsets from the estimate. Hence, the itemset discovery is completely independent of the size of the dataset, and ISON only requires longer running times for the construction of the density estimate, which is in general fast. Apriori, on the other hand, has to scan the dataset to discovery frequent itemsets, which takes longer, the larger the dataset.

Summary

All in all, ISON successfully discovers itemsets when the underlying probabilistic condensed representation is an accurate description of the data. POEt, on the other hand, is only competitive on datasets with a low combination ratio and on itemsets having a high probability. Moreover, POEt requires the user to specify the probability distribution of the itemset lengths to obtain a representative result set.

7.2 Itemsets based on Statistical Information

Frequency is only one property of potentially interesting itemsets and becomes less important when the number of frequent itemsets increases. For example, among the most frequent itemsets are those that consist of only one variable-value pair (i.e., (X_i, v_i)). Although they often have the highest frequency, they are not really relevant for the user, if there are several thousands other frequent itemsets. In order to reduce the size of the result set, the user can increase the threshold θ, restrict the search to itemsets of a given length, or restrict the search to itemsets with certain attributes. But, in the end, it could be a long process of adjusting the input parameters until the result set is small enough to be scanned by a human user for relevant itemsets.

Previous research addressing this problem can be divided into three main directions: (1) The first class of approaches uses so-called *condensed representations* to reduce the size of the result set. A condensed representation consists of subsets of frequent itemsets from which the other frequent itemsets can be derived without rescanning the database again. This allows itemset mining algorithms to maintain a smaller memory footprint and to avoid unnecessary and expensive database scans. However, as the objective is to find subsets from which the other frequent itemset can be derived, they are not necessarily interesting from a user perspective. (2) The second class of approaches reduces the size of the result set by allowing the user to define constraints, e.g., enforcing that a certain attribute has a specific value. However, if little is known about the data or the user cannot really identify the properties of the itemsets she is interested in, little is gained by applying constraints. (3) The third direction of research focuses on more informed measures of interestingness to measure whether an itemsets provides knowledge that is novel, unexpected, or actionable (McGarry, 2005). Although this directly addresses the problem from a user perspective, it completely disregards the context of the user. For example, customers of a restaurant are possibly ordering other dishes for their lunch break than for a candle light dinner with their partners. If the owner of the restaurant wants to increase her revenue made at lunch time, she is rather interested in patterns related to this time period than in patterns that are frequently occurring for the evenings. This information could be clearly modeled using constraints, but either requires someone knowing the restaurant well or someone who has already performed some preliminary analysis.

Hence, trying to objectively measure the interestingness of an itemset disregards that interestingness is in fact subjective and depends on the context and intention of the user. Adding this information to the pattern mining process, however, can be difficult and assumes that the user knows exactly what she wants. Since this is unrealistic in general, we pursue a different direction in this section where itemsets are selected based on statistical information. Although selecting such itemsets with traditional approaches would be possible, it could easily require the algorithms to generate large numbers of frequent and infrequent itemsets, making these approaches infeasible for large datasets. A probabilistic condensed representation, on the other hand, can be much more effective in this case, as it already provides statistical information about

possible itemsets, which is available without generating any itemsets (see Section 7.1.1).

In this section, we will show that the probabilistic condensed representation used by ISON is not only suitable to find frequent itemsets but to mine completely new types of itemsets. As proof of concept, we introduce so-called *characteristic itemsets*, which are supposed to be small set of itemsets that empower the user to get an overview of all itemset, i.e., by definition, they also provide information about non-characteristic itemsets. Additionally, we show how these itemsets can be mined with the ISON framework.

7.2.1 Characteristic Itemsets

A small set of itemsets that empowers the user to get an overview and is still small enough to be scanned by humans requires itemsets that are in a certain way representative or characteristic for the given data. We provide an example of how this can be achieved by using the statistical information contained in some probabilistic condensed representations and call the resulting itemsets *characteristic itemsets*. As a basis, we use conditional densities to divide the itemsets into groups and choose representatives. Compared to other itemsets in its group, representatives share the same variables, differ in exactly one variable value, and have a substantially lower support. This allows to make statements about the remaining itemsets, because changing the value of any of its variables has substantial impact on its frequency – unless changing that value results in another characteristic itemset. Hence, it is representative for all non-characteristic itemsets that can be obtained by changing the value of a single variable.

There are several conditions that have to be fulfilled for an itemset to be characteristic. The first condition is that the discrete probability distribution of every variable given the remaining variable-value pairs has to be characteristic, which basically means that there are substantial differences in the discrete probability distribution. In order to measure these differences, we apply the 80-20 rule and split the values in two halves: one half consisting of 80% of the values, and one half consisting of 20% of the values (see Figure 7.7 for an illustration):

Definition 16. Let X be a variable, let f_X be the discrete probability distribution over X, and let $v_1, v_2, \ldots, v_d$ be the values that X can take, such that $f_X(v_1) > f_X(v_2) > \ldots > f_X(v_d)$. Then f_X is called γ-characteristic, if $\sum_{i=1}^{\lfloor 0.2 \cdot d \rfloor} f_X(v_i) \geq \gamma$.

In other words, 80% of the values that X can take have an accumulated density value of $1-\gamma$ and 20% have an accumulated density value of γ. If $\gamma = 0.5$, a relatively small fraction of these values account for the same accumulated density value as a relatively large fraction. Hence, replacing the value v of an attribute X by another value v' should in most cases result in a substantial increase or decreases of the support of the itemset. This means that we also gain insights into related itemsets, if variables have a characteristic probability distribution.

The second condition demands that the characteristic variable takes one of the values of $\{v_{\lceil 0.8 \cdot d \rceil}, \ldots, v_d\}$ (e.g., values v_1 or v_2 in Figure 7.7), so that its density value is larger

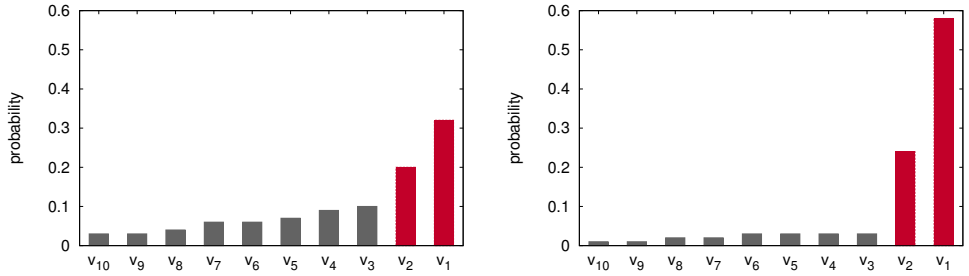


Figure 7.7: On the left is an example of a 0.5-characteristic probability distribution and on the right an example of 0.8-characteristic probability distribution. The bars colored red belong to the half consisting of 20% of the values, the bars colored gray belong to the half consisting of 80% of the values.

than the density value of most of the other values. If the first and the second condition hold for every variable-value pair of the itemset, we obtain itemsets where any change of a variable-value pair has a high probability of decreasing the support of the itemset substantially, so that it is rather important that the itemset has this value for the given variable:

Definition 17. Let $\mathcal{X}' := \{X_1, X_2, \ldots, X_k\}$ be the attributes of the items of a k-itemset (X_1, v_1), $(X_2, v_2), \ldots, (X_k, v_k)$. We call this itemset γ-characteristic, if $f_{X_i} := f(X_i \mid (X_j, v_j)$ for $X_j \in \mathcal{X} \setminus \{X_i\})$ is γ-characteristic and $f_X(v_i) \geq f_X(v_l)$ for all $X_i \in \mathcal{X}'$, such that $|\{v_j \mid f_X(v_j) \leq f_X(v_l)\}| = \lceil 0.8 \cdot |X_i| \rceil$.

If the characteristic itemsets are even *maximal*, i.e., no other attribute-value pair can be added without losing the property of γ-characteristic, the user should receive a relatively small set of itemsets, which also provides feedback about many non-characteristic itemsets. To illustrate the definitions given above, we provide an example from the movie domain.

Example 6. Assume that a cinema provides a special card for frequent customers with which they keep track of cinema-related information: age (i.e., discretized in bins 21-30, 31-40, 41-50, 51-60), gender (i.e., male, female), occupation (i.e., twenty occupations such as student or scientist), and the number of visits per genre (i.e., discretized in bins 0, 1-5, 6-10, 11-20, $\geq$ 21). Further assume that the following γ-characteristic itemsets, $\gamma = 0.75$, are contained in the database and that their conditional probability distributions do not contain any extreme probabilities for the 80% of the values with lower probability (for simplicity):

```
age.21-30, gender.male, occupation.student, romance.0
age.21-30, gender.male, occupation.student, scifi.11-20
age.21-30, gender.male, occupation.scientist, romance.0
age.21-30, gender.male, occupation.scientist, scifi.11-20
```

If there is no characteristic itemset where only one of these variables have a different value, then we have very specific information about this particular group of customers. (At this point, it is important that the customers' privacy has been preserved before starting the mining process.) Changing any of the variable values would result in a substantial drop of the itemset's support. For example, since there are at most five values for each genre, we already know that male customers who are students or scientists and are between 21 and 30 years old mostly watch science fiction (because the support for the variable genre, in this case, is less than 0.25 for `scifi.0`, `scifi.1-5`, `scifi.6-10`, and `scifi.>20` combined) and do not like romance (because the support for the variable genre is in this case less than 0.25 for `romance.1-5`, `romance.6-10`, `romance.11-20`, and `romance.>20` combined).

Please notice, however, that characteristic itemsets do not provide any information when two variables values are replaced at the same time (e.g., we have no information about the same itemsets but with female customers who are between 31 and 40 years old). In this case, one has to check other characteristic itemsets to obtain information for this particular combination.

7.2.2 Similarities to Other Types of Itemsets

Characteristic itemsets are quite different from other types of itemsets such as closed itemsets, δ-tolerant closed itemsets, or δ-free itemsets. Whereas these itemsets make statements about the relation to their subsets or supersets, characteristic itemsets considers the frequency distribution of individual variables of the itemsets. Hence, the focus lies on itemsets with the same variables but different values.

This is in certain aspects similar to a method proposed by Sese and Morishita (2004). They did not propose an new condensed representation but used itemsets to cluster a set of transactions. In addition to the items, each transaction has some *objective attributes*, which are basically numerical attributes that are considered special. Based on these attributes, Sese and Morishita aim at finding the optimal clusterings that maximizes the multi-dimensional interclass variance of each clusters. Although the problem is NP-hard, they propose a method that iteratively uses itemsets to divide the transactions into two groups: one group containing the given itemset and one group not containing this itemset. This means that unlike condensed representations such as δ-free itemsets, it actually characterizes the subsets of transactions by itemset. In particular, for binary variables $Z_1, \ldots, Z_k \in \mathcal{X}$, an itemset $(Z_1, v_1), \ldots, (Z_k, v_k)$ represents a set of transactions containing $\{(Z_1, true), \ldots, (Z_k, true)\}$. The complement, on the other hand, contains the itemset $\{(Z_1, false), \ldots, (Z_k, false)\}$. Hence, similar to characteristic itemsets, the approach by Sese and Morishita also uses the values of variables to characterize sets. But characteristic itemsets differ in two main aspects:

- They make statements about the set of itemsets and not the set of transactions.
- They use the probability distributions of itemsets with respect to a given variable instead of the multi-dimensional interclass variance.

Consequently, the resulting itemsets are in general different and not comparable to the itemsets characterizing clusters.

7.2.3 ISON for Characteristic Itemsets

Algorithm 16: *isCharacteristic*

Input: $EDDO_{ECC}$ density estimate $\hat{f}_{ecc}$, γ, itemset candidate *candidate*
Output: true iff *candidate* is characteristic

`// heuristic: weight chains according to their ordering`
$w \leftarrow []$
$char \leftarrow true$
for $(X_i, v_i) \in candidate$ **do**
  $O_{cc} \leftarrow [\texttt{argmin}_{j=1,\dots,m}(\hat{f}_{cc}[j] = X_i) \mid (X_i, v_i) \in candidate]$
  $w[\hat{f}_{cc}] \leftarrow \sum_{i=1}^{|O_{cc}|} |O_{cc}| - O_{cc}[i] + \sum_{i=2}^{|O_{cc}|} |O_{cc}| - O_{cc}[i] - O_{cc}[i-1]$
end
$w \leftarrow normalize(w)$
`// check Condition (1) and (2) for all` (X_i, v_i)
for $(X_i, v_i) \in candidate$ **do**
  `// heuristic: choose best chain`
  for $\hat{f}_{cc} \in \hat{f}$ **do**
    $attIndex \leftarrow chain\_ordering(\hat{f}_{cc}).indexOf(X_i)$
    $w[\hat{f}_{cc}] \leftarrow w[\hat{f}_{cc}] + \sum_{j=1}^{|O_{cc}|} \mathbb{I}_{I_{cc}[j] \leq attIndex}$
  end
  $w \leftarrow normalize(w)$
  $\hat{f}_{cc} \leftarrow \text{argmax}_{\hat{f}_{cc} \in \hat{f}} w[\hat{f}_{cc}]$
  `// check Condition (1) and (2) for` X_i
  $dist \leftarrow \hat{f}_{cc}[X_i].getDistribution(itemset)[v_i]$
  $density \leftarrow dist[v_i]$
  $d \leftarrow sorted(dist)$
  if $\sum_{j=1}^{\lfloor 0.8 \cdot |d| \rfloor} d[j] > 1 - \gamma \wedge density < d[\lfloor 0.8 \cdot |d| \rfloor]$ **then**
    $char \leftarrow false$
  end
end
return *char*

In order to discover the characteristic itemsets form a stream S, denoted by $\mathcal{C}(S)$, we provide another filter for ISON (see Algorithm 16), which distinguishes characteristic from non-characteristic itemsets. It basically checks Condition (1) and (2) for potential itemsets. However, with two additional heuristics, it also takes into account that some

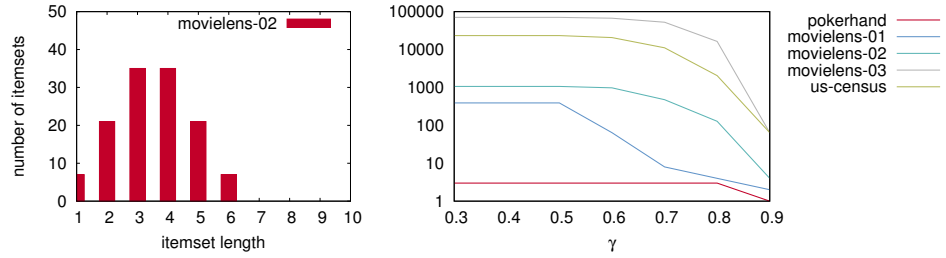


Figure 7.8: On the left is the distributions of the itemset lengths (cp. with Figure 7.5). On the right is the number of characteristic itemsets for different γ on the real-world datasets.

classifier chains are better suited to represent certain interdependencies and, therefore, better at representing the discrete probability distribution of a given attribute. Its correctness is stated by Theorem 10 (the proof of which is omitted, because it can be conducted in a similar fashion as the proof for Theorem 9).

Theorem 10. *Let f be the joint density of the data stream S, and let $\theta \in [0,1]$ be the threshold for frequent itemsets. If $\hat{f}_{ecc}$ is an accurate representation of f, then $ISON(\hat{f}_{ecc}, isFrequent(\hat{f}_{ecc}, \theta) \circ isCharacteristic(\hat{f}_{ecc}, \gamma)) = \mathcal{C}(S)$.*

That we achieved our goal with the definition of characteristic itemsets is illustrated in Figure 7.8. By adjusting γ, the number of itemsets is reduced to a human-readable size while still providing sufficient information about the remaining itemsets. For a γ value of about 0.30, we obtain many itemsets for movielens-02, movielens-03, and us-census. By increasing γ (and thereby increasing the importance of relatively few attribute-values), the number of itemsets is reduced to a human-readable size. For $\gamma = 0.7$, this means that 20% of the attribute values take 0.70 of the probability distribution, i.e., together they have the same weight as the remaining attribute-values.

7.3 Conclusion

In this chapter, we introduced a novel way of mining itemsets. Instead of scanning the original data stream, we proposed to first estimate its joint density and then to perform itemset mining on the density estimate. This enables the application of some preprocessing steps before the actual mining task such as adding constraints in the form of evidence or enforcing privacy-preserving properties. In particular, we developed two different approaches to address the problem: For density estimates as introduced in Part I, we proposed the ISON framework, which exploits the structure of a density estimate to extract frequent itemsets. If the representation is a perfect description of the density, it yields all frequent itemsets. For cases where the density estimate is unknown or no specific mining algorithm exists yet, we proposed POEt, an algorithm that only assumes a probabilistic condensed representation without any

assumptions about its structure. It uses inference operations to constantly transform the density estimate and discovers itemsets by sampling variable-value combinations. Although POEt is successful in discovering extremely frequent itemsets, it struggles with itemsets having a medium to low frequency – as those require a large number of samples. ISON, on the other hand, is able to discovery itemsets with arbitrary supports, as long as the density estimate reflects the frequencies of the itemsets well.

To show that probabilistic condensed representations are not only suitable to handle well-known pattern mining tasks but open up new opportunities, we introduced a new kind of itemset, so-called characteristic itemsets, that is defined in terms of conditional probability distributions. They offer a novel kind of itemsets that are defined in terms of conditional probability distributions and their relationships to other itemsets.

In the future, we would like to extend this work in several aspects: (1) ISON and POEt only handle discrete variables so far, but the presented ideas can also be applied to densities with continuous variables and densities with discrete and continuous variables. (2) Moreover, it would be interesting to consider other frequency-related constraints such as class-correlation. (3) Finally, since many aspects of the characteristic itemsets have been chosen heuristically and may not be suitable for every dataset (e.g., the 80-20 rule or that exactly one variable value is changed), they should be rather considered as a proof of concept. Comparisons with other concepts and a more detailed analysis could guide future research in understanding interesting itemsets.

Chapter 8

Recurrent Data Distributions

In the previous chapters, we proposed probabilistic condensed representations capturing the joint density of a data stream. If they employ base estimators that are able to adapt to drifts in the data (e.g., CVFDT (Hulten et al., 2001)), the representation will change towards the new distributions with increasing numbers of instances. However, this leads to a loss of information, since it is no longer available to the data mining algorithms and therefore poses a problem for the historical analysis of the data. To address this problem, we propose a condensed representation in this chapter that captures the various – possibly recurrent – data distributions of the stream by extending the notion of so-called *possible worlds*. The representation enables queries concerning the whole stream, thereby serving as a tool for supporting decision-making processes or serving as a basis for implementing data mining algorithms on top of it.

8.1 Introduction

A real-world example where recurrent data distributions are likely to occur is the smart home example we introduced earlier. In a smart home, every part of the house is equipped with sensors that constantly measure all kinds of parameters such as light intensity, temperature, or humidity. If we collect these measurements for a given time point, we obtain an instance and if we do this every second, we obtain a stream of instances. As with many stream applications, the data distribution of the resulting stream is usually not fixed but subject to changes – a typical example for our smart home would be a new resident who moved into the house and caused a change in the data distribution. As a result, the instances of this stream do not belong to one stationary data distribution but can be segmented into parts with stationary data distributions and into parts with changing data distributions (see Figure 8.1 for an illustration). Since the condensed representations proposed in the previous chapters constantly adapt their density estimate $\hat{f}$ to reflect these changes, $\hat{f}$ can only represent the most recent data distribution (e.g., S_{10}). All the information originating from historical data is simply lost, which unfortunately prevents a detailed analysis of the residents' behavior.

This is even more severe, if data distributions are recurrent. Especially in the smart home example, we expect many data distributions to occur again and again, which could be caused by differences in the residents' behavior during special time periods

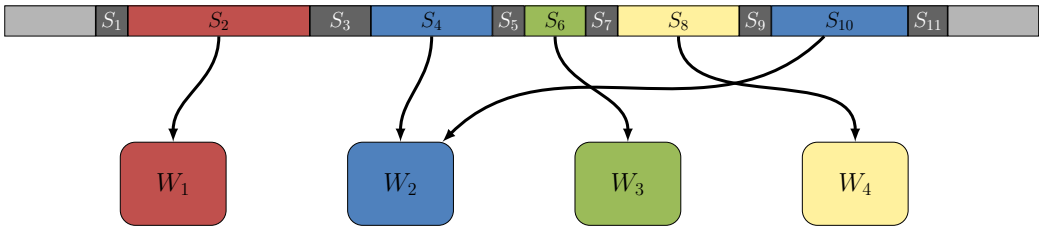


Figure 8.1: At the top, a data stream is illustrated that is divided into a sequence of segments $S_1, \ldots, S_{11}$. Colored segments represent a data distribution and segments that are dark gray represent data distribution drifts. Below the stream are the macro-worlds $W_1, \ldots, W_4$ that are associated with the data distributions.

such as day and night, working days and weekends, or the seasons of the year. If several of these causes come together, the pattern of recurrence becomes more complex and may also be limited to certain parts of the house. For example, the office room and living room are possibly more strongly affected by working days and weekends than the bed room.

To represent the recurrent data distribution of a data stream, we propose a new condensed representation in this chapter that builds on so-called *macro-worlds*. This representation will allow users to query the stream, so that they obtain statements holding for all macro-worlds, some macro-worlds, or no macro-world. In the example given above, this could be useful to identify typical behaviors of the residents and to adapt the energy usage of the smart home accordingly.

Formally, we address the problem of performing online density estimation on a stream $stream(f^{j_1})[1:N_{j_1}] \circ stream(f^{j_2})[1:N_{j_2}] \circ \ldots$, where $f^{j_i} \in \mathcal{F} = \{f^i(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l) \mid 1 \leq i \leq k \in \mathbb{N}\}$, $j_i, N_{j_i} \in \mathbb{N}$, and $\exists j_a, j_b \in \mathbb{N}$ with $j_a \neq j_b$, such that $f^{j_a} = f^{j_b}$. We approach this problem in several steps (see Figure 8.2 for an illustration): First, we identify the stationary data distributions, associate them with a macro-world, and represent the macro-world with an online density estimate. If a data distribution reoccurs, the corresponding estimate is reused. In cases where a novel data distribution is identified, a new density estimate is initialized. To obtain the best possible estimate and to reuse as much as possible of an existing estimate, we also identify independent parts of the macro-worlds, which we call *micro-worlds*. For the relationships between the worlds, we construct a graph representing the transition probabilities between the macro-worlds and the recurrence probabilities of the micro-worlds. Hence, the resulting condensed representation models the densities of the macro- and micro-worlds, their transition probabilities, as well as the recurrence probabilities of micro-worlds.

In the following, we introduce the notion of possible worlds and extend them for the purpose of representing data streams with recurrent macro- and micro-worlds (Section 8.2). In Section 8.3 and 8.4, we present several algorithms that provide a condensed representation of the data. Finally, we evaluate them in Section 8.5 and conclude this

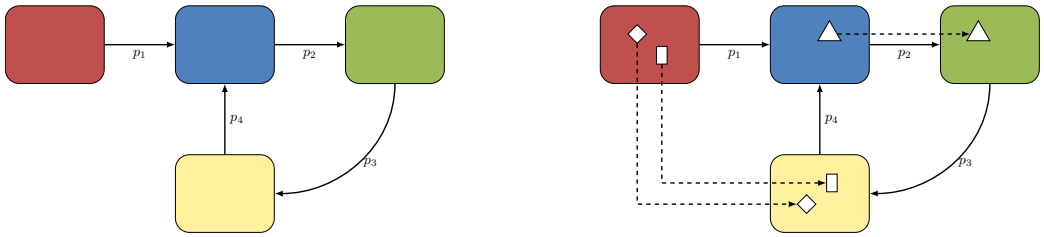


Figure 8.2: After data distributions have been associated with macro-worlds (see Figure 8.1), the condensed representation is created in two main steps: determining transition probabilities and identifying micro-worlds. The colored boxes are the macro-worlds, the white shapes are the micro-worlds, and the labeled arrows and the dashed arrows are the transition probabilities, where the labeled arrows are transitions between macro-worlds and the dashed arrows are transitions between micro-worlds.

chapter with an outlook and some final remarks (Sections 8.6 and 8.7).

8.2 From Streams to Possible Worlds

Formally, let $\vec{x}_1, \vec{x}_2, \ldots$ be a stream of instances defined over variables $X_1, \ldots, X_m$. A data streams can be divided into a sequence of segments $S_1, \ldots, S_k$, such that the S_i with $i \in \{2, 4, 6, \ldots\}$ are maximal segments having a stationary data distribution f_i and the S_i with $i \in \{1, 3, 5, \ldots\}$ are maximal segments between two data distributions, which are segments where a so-called *data distribution drifts* occurs. We say that the stream is recurrent, if there are $i, j \in [1, k]$ with $i \neq j$ and S_i has the same data distribution as S_j.

To model the different macro-worlds and the relations between them, we employ the notion of possible worlds (Gamut, 1991). Possible worlds consist of propositional letters to which truth values are assigned. The same letter can exist in different possible worlds but may have a different truth value. Given two worlds W and W', an accessibility relation indicates whether W' is accessible from W. This allows to evaluate the truth value of a given statement with respect to several interconnected worlds. In this context, two operators, $\Box$ and $\Diamond$, are important. $\Box$ indicates whether something is necessary (i.e., holds in all worlds), and $\Diamond$ indicates whether something is possible (i.e., holds in at least one world).

The condensed representation has two levels of abstraction. The first level describes the structure of the streams, i.e., which segments exist and how they are related to each other. On this level, each segment with a stationary data distribution (segments $S_2, S_4, S_6, S_8, S_{10}$ in Figure 8.1) is associated with a possible world. The worlds themselves are described on the second level of the abstraction, for which we require a representation that estimates the underlying data distribution and supports inference tasks. The latter are basic operations, which will enable queries on the condensed representation.

In most cases, only parts of the world will reoccur. For example, the office room and living room are possibly more strongly affected by working days and weekends than the bed room. Therefore, we introduce the notions of what we call macro- and micro-worlds as addition to possible worlds (the white shapes in Figure 8.2 are the micro-worlds). The following definition formalizes these ideas and are based on Gamut (1991). Novel compared to Gamut are the notion of macro- and micro-worlds and the introduction of probabilities.

Definition 18. Let $\mathcal{W}$ be a non-empty set of possible worlds with $\Omega \subseteq \mathcal{W}$ (macro-worlds), $\Theta \subseteq \mathcal{W}$ (micro-worlds), $\Omega \cap \Theta = \emptyset$, $\Omega \cup \Theta = \mathcal{W}$, and $\forall \theta \in \Theta \; \exists W \in \Omega : \theta \subsetneq w$. Further, let $R : \mathcal{W} \times \mathcal{W} \to [0, 1]$ be a binary relation representing the transition probabilities between possible worlds (micro- and macro-worlds), and let V be a valuation function that assigns a probability $V_W(a)$ for every proposition letter a in every context of $W \in \mathcal{W}$. Then (Ω, Θ, R, V) is a model for propositional logic.

Micro-worlds have several benefits: They are independent components, which can be treated separately. This means that only the relevant parts are considered when trying to capture them, which yields an overall smaller and probably more accurate representation. Also, similarities between macro-world are easier to detect, since worlds can be compared on a micro-world basis.

8.3 Condensed Representation: Level One

The first step in constructing a condensed representation of streams is to determine segments of the stream that belong to macro-worlds. A stream can be divided into two types of segments: *macro-world segments* containing *macro-world instances* (i.e., instances that clearly belong to a certain macro-world) and *transition segments* containing *transition instances* (i.e., instances that belong to transitions from one macro-world to another one). Whereas macro-world instances are used to construct the possible worlds, transition instances are usually not very interesting, since it is not clear to which world they belong and are, therefore, unsuitable for improving the online density estimate of the macro-world.

In order to split the stream into these segments, we will use two algorithms: $detectDrift$ and $equalDensities$. $equalDensities$ compares the distribution between a sample of instances and a density estimate by employing a Wilcoxon rank-sum test, and $detectDrift$ identifies changes in the distribution of the stream using $equalDensities$. The pseudocode of both are left out, as they are explained in detail in Section 8.3.2. Please notice that these algorithms are not the main focus of this thesis, but rather tools that we employ to accomplish the task of building a condensed representation of a stream with recurrent macro-worlds. It is basically possible to use any concept-drift detection method that is based on the data distribution of the underlying stream.

After the stream is segmented into macro-world segments and transition segments, the first level of the condensed representation can be built. This requires to capture

the dependencies between the worlds by observing transitions from one macro-world to another and to represent individual macro-worlds.

8.3.1 Relationships between Macro-Worlds

In order to capture the relationships between the macro-worlds, we basically construct a graph where the nodes are the macro-worlds of the stream – represented by online density estimates – and an edge between two nodes N_1 and N_2 with label p means that the probability of going from macro-world W_1 to macro-world W_2 is p. Algorithm 17 is responsible for maintaining the relationships between the macro-worlds (lines 1-8) and for passing instances to the second level of the condensed representation (line 25), which are the online density estimates. To accomplish these tasks, it maintains a window (consisting of the sequences S_B, S_M, S_E), the instances that just arrived ($insts'$), and an additional buffer ($insts$), which we use to ensure that we possess enough instances to detect data distribution drifts. As soon as enough instances for drift detection are available (i.e., $|insts| \geq |S_B| + |S_M| + |S_E|$), the algorithm moves the window instance by instance (line $17 - 20$) and performs a test to detect data distribution drifts. If a drift has been detected (line 21), all instances up to this point are removed, which includes all instances from S_B, S_M, and S_E (see Figure 8.3), and the algorithm waits until enough instances are available again (line $1 - 2$). If no drift has been detected (line 24), it updates the counters of μ, which will be used to estimate the probabilities of the edges. For any two density estimates $\hat{f}, \hat{f}' \in \mathcal{F}$, μ stores the number of times we transitioned from the macro-world represented by $\hat{f}$ to the macro-world represented by $\hat{f}'$. Using μ, the transition probability p of going from $\hat{f} \in \mathcal{F}$ to $\hat{f}' \in \mathcal{F}$ is computed as follows (simply maximum likelihood, i.e., the relative frequency): $p = \frac{\mu(\hat{f},\hat{f}')}{\sum_{g,g' \in \mathcal{F}} \mu(g,g')}$. In lines 5-9, the algorithm tries to match the emerging density estimates to one of the already existing ones. If none is found, a new density estimate is initialized. In the last step of the algorithm, the instances that belonged to S_B in the previous iteration (see t_1 in Figure 8.3) are forwarded to the current online density estimate (line 25).

8.3.2 Identifying Macro-Worlds

The procedure described above needs to identify segments with macro-world instances and segments with transition instances. In an supervised setting, there exist several methods and frameworks to detect concept drift and recurrent concepts (e.g., Gonçalves Jr and de Barros (2013), Gama and Kosina (2013)), which usually measure some kind of error of the classifiers and use a statistical test or property to detect a concept drift. However, since we want to use joint density estimates as condensed representation, the supervised setting does not apply in our case. Other approaches such as methods proposed by Kifer et al. (2004) or Dries and Rückert (2009) consider the data distribution of the stream to detect data distributions drift.

Since the main focus of this chapter is a condensed representation for streams with recurrent macro-worlds, we decided in favor of a simple approach, in our case: the

Algorithm 17: updateLevelOneRepresentation

Input: unprocessed instances $insts$, new instances $insts'$, the windows S_B, S_M, S_E, density estimate $\hat{f}$, the set of density estimates $\mathcal{F}$, a set of edges $\mu : \mathcal{F} \times \mathcal{F} \to \mathbb{N}$, significance level α

if $|insts| + |insts'| < |S_B| + |S_M| + |S_E|$ **then**
  append $insts'$ to $insts$
else
  $matchFound \leftarrow false$
  for $\hat{f}' \in \mathcal{F}$ *with* $equalDensities(\hat{f}', S_E, \alpha)$ **do**
    $matchFound \leftarrow true$
    $\mu(\hat{f}, \hat{f}') \leftarrow \mu(\hat{f}, \hat{f}') + 1$
    $\hat{f} \leftarrow \hat{f}'$
  end
  if $matchFound = false$ **then**
    $\hat{f}' \leftarrow$ initialize density estimator
    $\mu(\hat{f}, \hat{f}') \leftarrow 1$
    append $\hat{f}'$ to $\mathcal{F}$
    $\hat{f} \leftarrow \hat{f}'$
  end
  for $inst_1 \in insts'$ **do**
    $q \leftarrow$ queue with elements S_B, S_M, S_E
    $inst_2 \leftarrow q.pop()$
    $q.push(inst_1)$
    split q into S_B, S_M, S_E
    if $detectDrift(\hat{f}, S_B, S_M, S_E, \alpha)$ **then**
      $insts \leftarrow insts'[index(inst_1) + 1 :]$
      break loop
    else
      $\hat{f}.update(inst_2)$
    end
  end
end

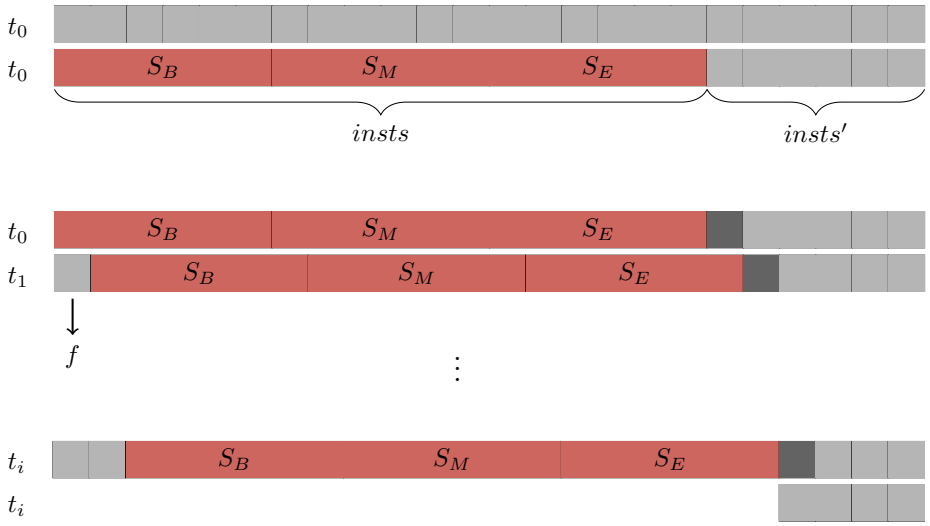


Figure 8.3: In order to detect drifts in the data distribution of the stream, a window is used that is split into three segments of equal size: S_B, S_M, and S_E. If enough instances are available to perform drift detection, *insts* are exactly the instances contained in $S_B \circ S_M \circ S_E$, whereas *insts*$'$ are the new instances that need to be processed. In order to process the next instance of *insts*$'$ (dark gray), the algorithm treats $S_B \circ S_M \circ S_E$ as a queue and removes the oldest instance whenever a new instance is appended. The instance that has been removed is then forwarded to the current density estimate $\hat{f}$. If a drift is detected (see t_i), all previous instances are discarded and the algorithm waits until enough instances are available for further drift detections.

CNF density estimation test (Dries and Rückert, 2009), for which the authors pursued a window-based approach. They split the window into three parts and determine whether the first two parts have the same distribution as the last one. In order to capture changes in the distribution, they transform the instances to a binary representation and compare the distributions of the different parts. For this purpose, they introduce a predicate *covers*, use this predicate to describe subsets of variables that are consistent with a given set of instances, and perform a Mann-Whitney test to decide whether there is a significant difference between the distributions.

Following this idea, we employed a window-based approach (a buffer) to ensure that only those instances reach the condensed representation that really belong to the corresponding concept. Therefore, we keep instances in a buffer as long as the density estimation test could still detect a data distribution drift. Hence, we forward only the instances of S_B to the density estimate of the current macro-world. Instances from S_M and S_E will only be forwarded, if they become part of an S_B of another partition, which also means that we probably do not capture all the macro-world instances that

are available. To detect changes in the distribution, we perform a Wilcoxon rank-sum test (Wilcoxon, 1945; Mann and Whitney, 1947; Moore et al., 2009) on S_B and S_E using the current density estimate – instead of the predicate *covers*.

8.4 Condensed Representation: Level Two

In the previous section, we described how to identify the macro-worlds of a stream and how to model the relationships among each other using the possible worlds semantics. In this section, we consider the second level and explain how the macro-worlds can be represented in a condensed way. For this purpose, we introduce *modules* to density estimates that will serve as an estimate for the *micro-worlds*.

Definition 19. Let $\mathcal{X} := \{X_1, \ldots, X_m\}$ be a set of variables and let $f(X_1, \ldots, X_m)$ be a discrete joint density. If $\mathcal{X} = A_1 \dot\cup \ldots \dot\cup A_k$, $k > 1$, and $A_i \subsetneq \mathcal{X}$ for $1 \leq i \leq k$, then the factors of the product $\prod_{i=1}^{k} f_i(A_i)$ are called *modules* of f if, for each $1 \leq i \leq k$, f_i is a discrete joint density and there is no decomposition $A_i = B_1 \dot\cup \ldots \dot\cup B_l$, such that $l > 1$ and $f_i(A_i)$ can be represented by a product $\prod_{j=1}^{l} f_{i,j}(B_j)$, where $f_{i,j}$ is a discrete joint density for $1 \leq j \leq l$.

In other words, modules are the independent factors of a density. For example, if a discrete joint density $f(X_1, X_2, \ldots, X_8)$ can be represented as a product $f_1(X_1, X_3, X_8) \cdot f_2(X_2, X_4, X_5) \cdot f_3(X_6) \cdot f_4(X_7)$ and the f_i, $i \in \{1, 2, 3, 4\}$, cannot be decomposed any further, then f_1, f_2, f_3, and f_4 are the modules of f.

8.4.1 Modules in Density Estimates

EDO density estimates can be used to derive modules for the represented density. For example, if a Hoeffding tree is used as base classifier, then the Hoeffding tree algorithm implicitly chooses a subset of variables to construct the tree. If a graph is constructed that has the classifiers of the chain as nodes and connects two nodes if the classifiers belonging to that node have common variables, then the cliques in the graph and all nodes that do not belong to a clique are the modules of the estimate.

In this chapter we will pursue a more explicit approach to modules, which is applicable to data streams consisting of discrete and/or continuous variables. The algorithm we are about to present constantly estimates the structure of the density by grouping the variables into modules, which is achieved by determining connected components of the variables based on the normalized mutual information – a measure for the dependence of two random variables. For each variable group, it then employs an online density estimate to represent the module. When new instances arrive, it may turn out that the proposed module does no longer match the data, because it is not a module at all, a part of a larger module, or only a part of it is a module. In this case, the module is entirely removed, and the estimate of each newly introduced or modified module is then replaced by a new online density estimate.

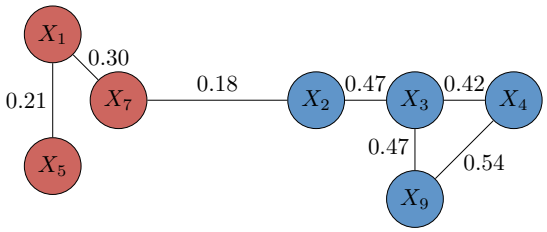


Figure 8.4: The figure shows variables (circles) that are dependent on each other according to the normalized mutual information. The values on the edges are the normalized mutual information for a given pair of variables. The algorithm creates two clusters: the variables X_1, X_5, and X_7 are clustered and the variables X_2, X_3, X_4, and X_9. As the connection between X_7 and X_2 is relatively weak in terms of the normalized mutual information, this connection is treated as a conditional dependence between these groups of variables.

In the discrete case, the mutual information (Cover and Thomas, 2006) of two variables X and Y is defined as $I(X,Y) := \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \left(\frac{p(x,y)}{p(x) \cdot p(y)} \right)$, where $p(x,y)$ is the probability of $(x,y) \in X \times Y$ and $p(x)$ and $p(y)$ are the corresponding marginal probabilities. For continuous variables, the summations are replaced by integrals. However, instead of computing integrals for continuous variables, we will discretize continuous variables using the online discretization method proposed by Gama and Pinto (2006), which is also used in Chapter 4. Since the mutual information does not necessarily lie between 0 and 1, which would be advantageous to define a general threshold that enables us to decide whether two variable are considered dependent on each other, we will use a normalized version of the mutual information (NMI) (Manning et al., 2008):

$$NMI(X,Y) := \frac{I(X,Y) \cdot 2}{H(X) + H(Y)},$$

where H is the entropy. The threshold to decide whether two variables are dependent on each other is set to some value θ, that is a value less than θ means not dependent, and a value greater than or equal to θ means dependent.

However, not all variables that are considered as dependent on each other should be treated equally. In some initial experiments, we noticed that the range of the normalized mutual information is often quite different and that often only parts of the modules are actually reoccurring. Hence, the modules themselves seemed to have a structure of their own. Figure 8.4 provides a typical example. Here, all variables are dependent on each other, but it is quite likely that only the density of the variables X_1, X_5, and X_7 or the density of the variables X_2, X_3, X_4, and X_9 are recurrent, as these two groups of variables are connected by a weaker link than the ones between the other variables. In order to make this structure available and to increase the detection of recurrences, we extend the notion of modules to *conditional modules*, which group variables by the strength of their dependence:

Definition 20. Let $\mathcal{X} := \{X_1, \ldots, X_m\}$ be a set of variables, let $f(X_1, \ldots, X_n)$ be a discrete joint density, and let $\{f_i(A_i) \mid A_i \subseteq X \wedge 1 \leq i \leq k\}$ be the modules of f, where the variables in A_i have a pairwise NMI of at least $0 \leq \theta_1 \leq 1$. Each module $f_i(A_i)$ can be decomposed into *conditional modules*, which are the factors of the product $f_{i,1}(B_1) \cdot f_{i,2}(B_2 \mid B_1) \cdot \ldots \cdot f_{i,l}(B_l \mid B_1, \ldots, B_{l-1})$, where $B_1 \dot{\cup} \ldots \dot{\cup} B_l$ is a decomposition of A_i and the variables in B_i have an NMI of at least $0 \leq \theta_2 < \theta_1$.

Example 7. In the example given in Figure 8.4, $f_i(X_1, X_2, X_3, X_4, X_5, X_7, X_9)$ would be a module. θ_1 would be something below 0.18 and θ_2 is something between θ_1 and 0.21. The conditional modules would be $f_1(X_2, X_3, X_4, X_9)$ and $f_2(X_1, X_5, X_7 \mid X_2, X_3, X_4, X_9)$. $f_1(X_2, X_3, X_4, X_9) \cdot f_2(X_1, X_5, X_7 \mid X_2, X_3, X_4, X_9)$ is the density of the module f_i.

Algorithm 18 maintains the modules and their estimates. It first updates the normalized mutual information NMI for all pairs of variables (lines 2-3). Then, every ρ instances (line 5), the structure of the modules is recomputed. In lines 6-7, D is determined, which defines dependencies between two variables X_i and X_j, iff their normalized mutual information is greater than or equal to θ_1. In the remainder, the algorithm reuses density estimates for every module existing before (lines 10-16) and passes the instance *inst* to all modules (line 23).

8.4.2 Shared Modules

A module is an estimate for a micro-world that can be shared among several possible worlds. If a new macro-world emerges, we need to examine whether it contains already discovered modules. Since macro-worlds are represented by estimates of joint densities, we require an algorithm that tests whether the estimates are constructed from the same distribution. A good choice for this task is usually the KL-divergence, which is, however, not suitable in our setting for several reasons: (1) Computing the KL-divergence is computationally expensive. (2) The estimates can differ with respect to their precision, since one estimate has been constructed from millions of instances, whereas the other one could have been constructed from a few thousand instances only. (3) Even if the estimates have reached the same level of precision, it remains unclear which KL-divergence is acceptable. As an alternative, we will use *equalDensities*, which employs a Wilcoxon rank-sum test to compare two distributions (see Section 8.3.2 for reference). Using this, we can compare the distribution of the already discovered module and the newly emerging module.

8.4.3 Divergence of Recurrent Micro-Worlds

Since a module that is shared among several possible worlds $\mathcal{W}'$ is the same object, updating the corresponding module would always affect every world in $\mathcal{W}'$. Hence, if a shared module develops differently, it can no longer be shared. Therefore, we make individual copies and assign them to each world belonging to $\mathcal{W}'$. It is crucial,

Algorithm 18: updateLevelTwoRepresentation

Input: variables $\mathcal{X}$, recomputation threshold $\rho \in \mathbb{N}$, instance counter *count*, set of modules *modules*, instance *inst*, object NMI, NMI threshold θ_1, clustering threshold θ_2

$modules' \leftarrow \emptyset$
for $(X_i, X_j) \in \mathcal{X} \times \mathcal{X}$, $1 \leq i < j \leq n$ **do**
  update $NMI(X_i, X_j)$
end
if *count* mod $\rho = 0$ **then**
  initialize $D_1 : \mathcal{X} \times \mathcal{X} \mapsto \{0, 1\}$
  set $D_1(X_i, X_j) = 1$ iff $NMI(X_i, X_j) >= \theta_1$
  $\mathcal{A} \leftarrow \{A_1, \ldots, A_k \mid A_i$ is a maximal independent component in $D_1\}$
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{\{X_i \mid \nexists X_j : D(X_i, X_j) < \theta_1\}\}$
  for $A_i \in \mathcal{A}$ **do**
    initialize $D_2 : \mathcal{X} \times \mathcal{X} \mapsto \{0, 1\}$
    set $D_2(X_i, X_j) = 1$ iff $NMI(X_i, X_j) >= \theta_2$
    $\mathcal{B} \leftarrow \{B_1, \ldots, B_k \mid B_i$ is a maximal independent component in $D_2\}$
    $m \leftarrow$ create a module from A_i
    $\hat{f} \leftarrow$ initialize density estimate $modules'.append((m, \hat{f}))$
  end
end
for $(m, \hat{f}) \in modules'$ **do**
  if $\exists \hat{f}' : (m, \hat{f}') \in modules$ **then**
    $\hat{f} \leftarrow \hat{f}'$
  end
  $modules'.append((m, \hat{f}))$
  $m.update(inst)$
end
$count \leftarrow count + 1$

however, that the copies are created before a deviation occurred. Otherwise, this deviation would be a part of all worlds in $\mathcal{W}'$. In order to solve this, we perform data distribution drift detection in combination with a buffering system as described in Section 8.3.

8.5 Evaluation

In this section, we evaluate the algorithms presented in Section 8.3 and Section 8.4. We first measure how modules affect the performance of the density estimate using synthetic and real-world datasets – the corresponding online density estimator will be denoted by REC (RECurrent densities estimated online). Subsequently, the construction of possible worlds is evaluated on real-world data.

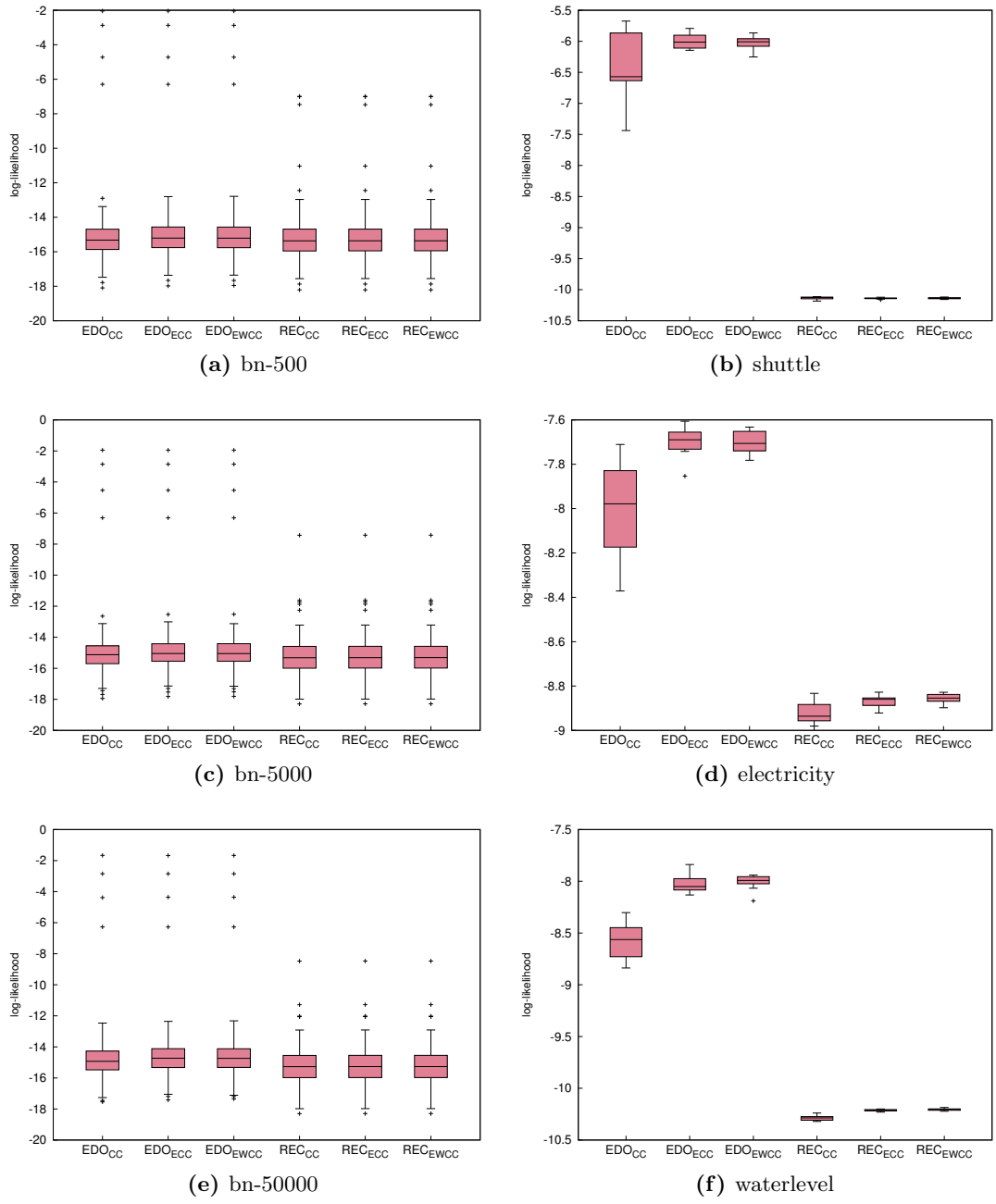
8.5.1 Evaluating Modules

Compared to EDDO, REC provides a more explicit representation of the density by grouping variables into independent components (modules). This leads to a representation that is easier to interpret by users, since variables that are dependent on each other are grouped together. Despite this more explicit representation, we will show that both methods have a similar performance. As we already compared EDDO to 12 Bayesian structure learners with a comprehensive set of experiments and showed that EDDO outperforms standard Bayesian structure learners, we focus our presentation on a comparison of EDDO and REC.

As synthetic data, we generated streams from several Bayesian networks and drew N many instances ($N \in \{5 \cdot 10^2, 5 \cdot 10^3, 5 \cdot 10^4\}$). As real-world datasets, we used several publicly available datasets: electricity, shuttle, and covertype. Additionally, we modified the water level dataset prepared by Schlüter and Conrad (2012), where the water level of rivers in Germany were measured over twelve years. Since the data is very sparse, we only considered two stations (Arloff and Meschede) and removed instances where parts of the measurements were missing. For each station, we had nine variables: the values of the closest four neighboring stations of the previous two days (eight values) and the class attribute indicating whether the water level increased, decreased or stayed the same.

In order to measure the performance of each estimator, we computed the average log-likelihood prequentially, i.e., the log-likelihood of a given instance has been computed before using it for training. The instances used to compute the initial estimator (the first 100) were excluded from this computation.

The results are summarized in Figure 8.5. Generally, EDDO is performing better than REC on all datasets. On smaller datasets and datasets with a low combination ratio (e.g. bn-500, bn-5000, electricity), the differences are smaller, but on the shuttle and waterlevel dataset, which have a high combination ratio, the difference is rather large. Since REC is using EDDO density estimates as a basis, these difference can only be explained by the modules and drift detection method.



(a) bn-500 **(b)** shuttle

(c) bn-5000 **(d)** electricity

(e) bn-50000 **(f)** waterlevel

Figure 8.5: The figure shows the average log-likelihoods (computed prequentially) for EDDO and REC on various datasets.

When we had a closer look on the individual runs, we noticed that REC detects many of the drifts in the data stream and produces separate density estimates for each of the segments (see Figure 8.6) while only detecting few recurrences. For example, on the waterlevel dataset, REC detects 412 drifts and can, therefore, only provide a few hundred instances to each density estimate. EDDO, on the other hand, benefits from having all previously seen instances for estimating the density value of the next instance – although some of these instances are possibly making the estimate less accurate, since they belong to a different data distribution. This effect could be reduced or reversed, if either fewer instances are available or data distributions reoccur more often. To validate this idea, we reduced the number of instances on the waterlevel dataset to half of the available instances. In this case, the performance difference between EDDO and REC is substantially smaller, i.e., -10.18 and -10.73 respectively, which supports our claim. Hence, we can expect REC to perform equally well as EDDO on datasets with more recurrences while providing a more explicit representation of the data stream.

8.5.2 Recurrent Macro- and Micro-Worlds

Several publications monitor data distributions of streams to detect concept drifts (e.g, Gonçalves Jr and de Barros (2013)) and use the detected drifts to train new classifier or to continue training previous classifiers. However, constructing a universal condensed representation that enables queries – and as a result enables data mining and machine learning tasks on top of it – has to the best of our knowledge not been considered before. Hence, there is currently no other method trying to detect something like recurrent macro- and micro-worlds, and we could only perform the evaluation of recurrent worlds based on a real-world data such as the water level dataset.

How quickly the distribution of the dataset changes is visualized by Figure 8.6 (see the plot at the left-hand side), which shows the distance of each instances to the vector $\vec{0}$ – this is not to be confused with the distribution of the data, it only indicates changes in the distribution. Between two data distribution drifts there are often only a few hundred instances, which makes the detection of macro- and micro-worlds a challenging task. However, even in this difficult setting, the proposed algorithm provides a rather accurate description of the stream, as illustrated at the right-hand side of Figure 8.6. Large parts of this stream segment are almost exactly captured by the macro-worlds, whereas other parts show only minor deviations. In total, the algorithm finds 412 macro- and 438 micro-worlds. Although it is difficult to measure the meaningfulness of the micro-world detection, its classification is performed according to our intuition of the data. For example, it groups variables belonging to the same stations (i.e., measurements from different days) or variables indicating the current trend of the stations (i.e., whether the water level decreased, increased, or stayed the same).

In order to evaluate the detection of recurrent micro- and macro-worlds, we connected the water level dataset two times in a row. Hence, all macro- and micro-worlds have to reoccur again and should be detected by the algorithm. In this setting, the proposed method rediscovers 100% of the macro-worlds and 100% of the micro-worlds,

which indicates that REC is, in principle, able to rediscover macro- and micro-worlds.

8.6 Outlook: Queries on Possible Worlds

The proposed condensed representation is not only able to represent streams with recurrent macro-worlds, but also enables queries on top of it, which can be used to analyze the underlying data. In order to model this, we introduce the following operators:

Definition 21. Let $M = (\Omega, \Theta, R, V)$ be a model for propositional logic, and let p be some probability. Then $V_{M,w}(\phi)$, the probability of ϕ in w given M, is defined by the following clauses, which are an extension of the clauses by Gamut (1991):

1. $V_{M,w}(a) = V_w(a)$, $\forall$ variable-value pairs a
2. $V_{M,w}(\neg\phi) = p$ iff $V_{M,w}(\phi) = 1 - p$
3. $V_{M,w}(\phi \vee \psi) = min(V_{M,w}(\phi), V_{M,w}(\psi))$
4. $V_{M,w}(\phi \wedge \psi) = max(V_{M,w}(\phi), V_{M,w}(\psi))$
5. $V_{M,w}(\phi \rightarrow \psi) = min(V_{M,w}(\neg\phi), V_{M,w}(\psi))$
6. $V_{M,w}(\Box\phi) = min_{w' \in W:R(w,w')}(V_{M,w'}(\phi))$
7. $V_{M,w}(\Diamond\phi) = max_{w' \in W:R(w,w')}(V_{M,w'}(\phi))$

Definition 21 provides the foundation to perform queries on the condensed representation. With inference algorithms, one can first restrict the condensed representation to the parts that are interesting to the user. Subsequently, questions like: *Given world* W*, what is the probability that* a *is true?* can be posed. If the $\Box$ operator is used, this probability is computed with respect to all worlds directly or indirectly connected to W, and if the $\Diamond$ operator is used, this probability is computed with respect to the world that is directly or indirectly connected to W and yields the largest probability for a. To compute those probabilities, inference algorithms as the ones proposed in Chapter 6 can be used.

8.7 Conclusion

We proposed a novel approach of representing data from a stream with recurrent macro-worlds (stationary data distributions) using possible worlds and online density estimates. As the density estimates enable data mining tasks without access to the raw data (see Chapter 6 and 7), the proposed representation allows to perform these tasks on a much broader scale. In the experiments, we showed that the representation is accurate, to a certain degree, and can capture not only recurrent macro-worlds but

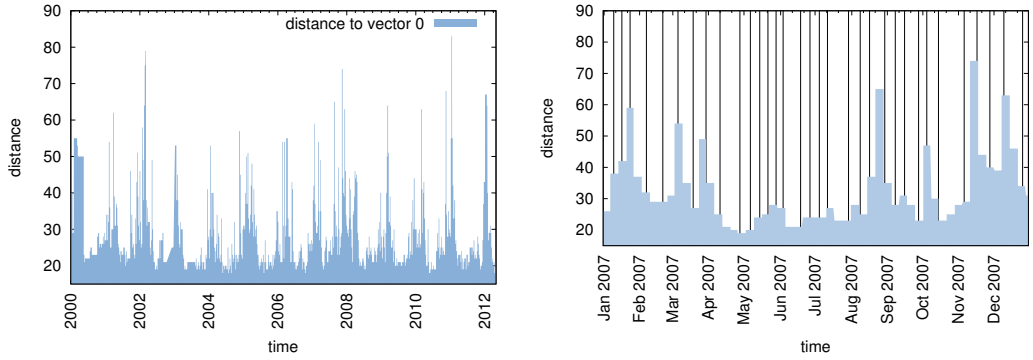


Figure 8.6: On the left-hand side, changes in the data distribution are illustrated by plotting the distance to the vector $\vec{0}$ for each measurement. On the right-hand side, macro-worlds are shown that have been detected between January 2007 and February 2008.

also enables the detection of micro-worlds. This is, to the best of our knowledge, the first time that possible worlds and micro-worlds are employed in the context of streams with recurrent data distributions.

In the future, we intend to reuse existing estimates more efficiently. If a module changes, the estimate is discarded and replaced by a completely new one. In some cases, however, a module might be split into several smaller modules, such that the current estimate can be reused by marginalizing out the variables. Hence, we could use an estimate that has been constructed from possibly thousands of instances instead of using an estimate that is constructed from only a few instances. Moreover, one may optimize the drift detection method for the proposed online density estimators.

Chapter 9

Conclusion and Future Work

An increasing number of devices are equipped with sensors, producing streams of measurements. Leveraging knowledge from these streams requires infrastructure to analyze them in real-time and involves challenging tasks such as cleaning the data, handling large amounts of data, and preserving their privacy. The state of the art in data mining already addressed some of these challenges, but the proposed methods are typically designed for a specific task (e.g., predicting a certain variable or finding frequent patterns) and perform this task while scanning the data stream.

The MiDEO framework presented in this thesis first estimates the joint density of the data and then performs data mining on the online density estimate. This way, the process of collecting the data and the actual analysis are decoupled, which allows to select the data mining algorithms after collecting the data. Thanks to the inference capabilities, it is also possible to modify the density estimate according to the current needs of the user or machine, which includes selecting subsets of variables, incorporating evidence, sampling a subset, or enforcing privacy-preserving properties. As a result, we can support several new data mining workflows that put the emphasis on the user – as part of an interactive process – and support task-independent privacy-preserving techniques.

9.1 Density Estimation

The cornerstone of the MiDEO framework (Mining Density Estimates inferred Online) is a collection of online density estimators, which are responsible for estimating the joint density of the data stream in an online fashion. In this thesis, we proposed three density estimators: EDO (see Chapters 3 and 4), RED (see Chapter 5), and REC (see Chapter 8), each of them designed for different kinds of data streams.

9.1.1 EDO

EDO (Estimating Densities Online) is able to handle (conditional) joint densities with discrete variables, continuous variables, and mixtures thereof. It exploits the product rule known from probability theory to split a joint density defined over several variables into several conditional densities with only one target variable. Each of these conditional densities is then estimated by online classifiers acting as univariate density

estimators. This way, the estimate can be updated on an instance-by-instance basis and also enables the application of a broad range of fast and efficient classifiers. For densities with discrete target variables, we used Hoeffding trees to estimate the probability mass of each variable value. For continuous target variables, we extended a method designed for conditional density estimation to the online setting (Frank and Bouckaert, 2009). As part of EDO, we proposed three variants of this scheme including a single random classifier chain (CCs), ensembles of classifier chains (ECCs), and ensembles of weighted classifier chains (EWCCs). To provide a theoretical foundation for all of these estimators, we proved that the resulting estimates are approaching the true density as long as the underlying univariate density estimators are consistent.

In the experiments, we demonstrated that EDO's performance is comparable to that of state-of-the-art density estimators, if the datasets consist of only discrete or only continuous variables. On datasets with mixed types of variables, it even outperforms these methods. However, the experiments also revealed that the IID assumption, which is inherited from the univariate density estimators, is too strong for many real-world data streams. In the future, we will work on a new type of univariate density estimator to alleviate this shortcoming.

9.1.2 RED

To model interdependencies between variables, EDO picks a variable ordering and conditions each variable on all preceding variables. This works well as long as the density has only a few variables, but, as soon as the dimensionality increases, the number of classifiers and their size grows quickly – making this approach unsuitable for data of high dimensionality. To propose an alternative density estimator for these cases, we designed RED (Representative-based online Estimation of Densities), an estimator that is able to represent joint densities defined over a large number of mixed types of variables. For this purpose, it projects the original data stream into a lower dimensional vector space and uses a set of representatives to provide an estimate. Each representative is a data instance and the joint density of its neighborhood is estimated by an EDO density estimate. Due to this structure, it enables the density estimation of higher-dimensional data and approaches the true density with increasing dimensionality of the vector space.

With a sufficiently large vector space, RED is able to compete with EDO, while it is also able to deal with many more variables. However, setting the parameters of RED is not always clear and the performance depends on the choice of the landmarks, i.e., the data points that span the lower dimensional vector space. Therefore, we will analyze the choice of the landmarks in more depth and plan to develop an algorithm that chooses the landmarks based on a data stream prefix.

9.1.3 REC

In principle, EDO and RED are both able to deal with data distribution drifts. In case of a drift, they simply adapt the current estimate to match the emerging joint density. If one is interested in a historical analysis of the data stream, however, this approach is not suitable. Therefore, we proposed yet another density estimator with a more explicit representation of data distributions, which we called REC (RECurrent densities estimated online). It captures the various – possibly recurrent – data distributions of the stream by extending the notion of possible worlds. The estimate enables queries concerning the whole stream, so that it can serve as a tool for supporting decision-making processes or can serve as a basis for implementing data mining algorithms on top of it. As it uses online density estimators such as EDO or RED to model the stationary data distributions, its applicability to real-world data depends on these base estimators. Further improvements such as obtaining a more accurate representation with fewer instances mainly require suitable real-world data streams. Therefore, we plan to continue the development of REC as part of a collaboration, where the collaborator provides suitable real-world data, i.e., an industry partner or a research team.

9.2 Probabilistic Condensed Representations of Data

The proposed online density estimates are not only capable of representing the joint density of a data stream, but also offer inference capabilities to perform queries. This distinguishes them from other density estimates that typically focus on representing the density – Bayesian networks are one of the exceptions but are limited to the offline setting. In particular, we proposed a specific set of inference operations and called the resulting density estimates *probabilistic condensed representations of data*.

9.2.1 Pattern Mining

To show that they are sufficient to perform more complex data mining tasks, we developed an itemset mining algorithm, which comes with theoretical guarantees regarding the correctness of its output. Moreover, we introduced the notion of characteristic itemsets to show that the presented mining framework not only allows to mine frequent itemsets effectively but also enables mining of new types of itemsets that are based on statistical information without generating all involved itemsets.

In the experiments, we described the performance of the algorithm with respect to the properties of the data stream. In general, one can say that the higher the dimensionality of the data stream and the more complex the density, the more instances are required to estimate the support of the itemsets precisely. In the future, we would like to extend our work to continuous variables and to pattern mining under constraints. Additionally, we will continue our work to define new types of itemsets that

are based on statistical information. In particular, we are interested in capturing the interestingness of itemsets with respect to a user-defined context.

9.2.2 Privacy

Another highlight of the MiDEO framework is its capability to protect the privacy of the entities described by the data. In particular, we proposed algorithms that modify EDO density estimates to meet privacy-preserving properties such as k-anonymity or t-closeness. We provided proofs showing that these properties can be guaranteed and illustrated the effect of the t-closeness algorithm on real-world data. In the future, we extend this work towards privacy-preserving methods that also protect the data itself. The density estimates do that to a certain degree, because they only maintain statistical information of the data. But, as we have seen with the itemset mining algorithm, this can already be enough to extract valuable patterns. Therefore, we plan to develop a protocol that allows multiple parties to share their knowledge without sharing their data or statistical information thereof. Following similar developments as in other communities, this could be considered as multi-party data mining[1].

9.3 Outlook

Knowledge discovery is a complex process often requiring extensive user interaction. In fact, there are probably few cases where the application of a single algorithm yields the desired result. In the remaining cases, the user is part of a creative process involving steps such as tuning parameters, selecting subsets of the data, or applying a chain of algorithms before repeating a certain step. With MiDEO, we presented a framework that supports this process for large amounts of data and enables privacy-preserving data mining. In its current state, it can be seen as a proof of concept, which could open up interesting new research directions:

Online Density Estimation Motivated by the new possibilities that MiDEO offers, various aspects of online density estimate could be explored that would bring MiDEO closer to systems for real-world applications. For example, the IID assumption needs to be removed to handle autocorrelation – something, which is often exhibited by actual sensor measurements, e.g., as in the smart home example. Moreover, for a correct interpretation of events and causalities, density estimates need to be aware of location and time. This could be modeled using variables, but an effective and efficient representation for density estimates needs to be developed.

Privacy The more data is collected and the more powerful the methods are to gain insights into the data, the more reluctant are people to share their data. Therefore, we believe that privacy-preserving data mining will play a major role in the future to

[1] This term has been initially coined by Prof. Dr. Stefan Kramer.

protect the entities described by the data and to collaboratively gain insights without sharing the data itself. In both cases, probabilistic condensed representations of data as proposed in this thesis could provide a powerful solution that is reliable and interpretable.

List of Figures

List of Tables

List of Algorithms

Part III

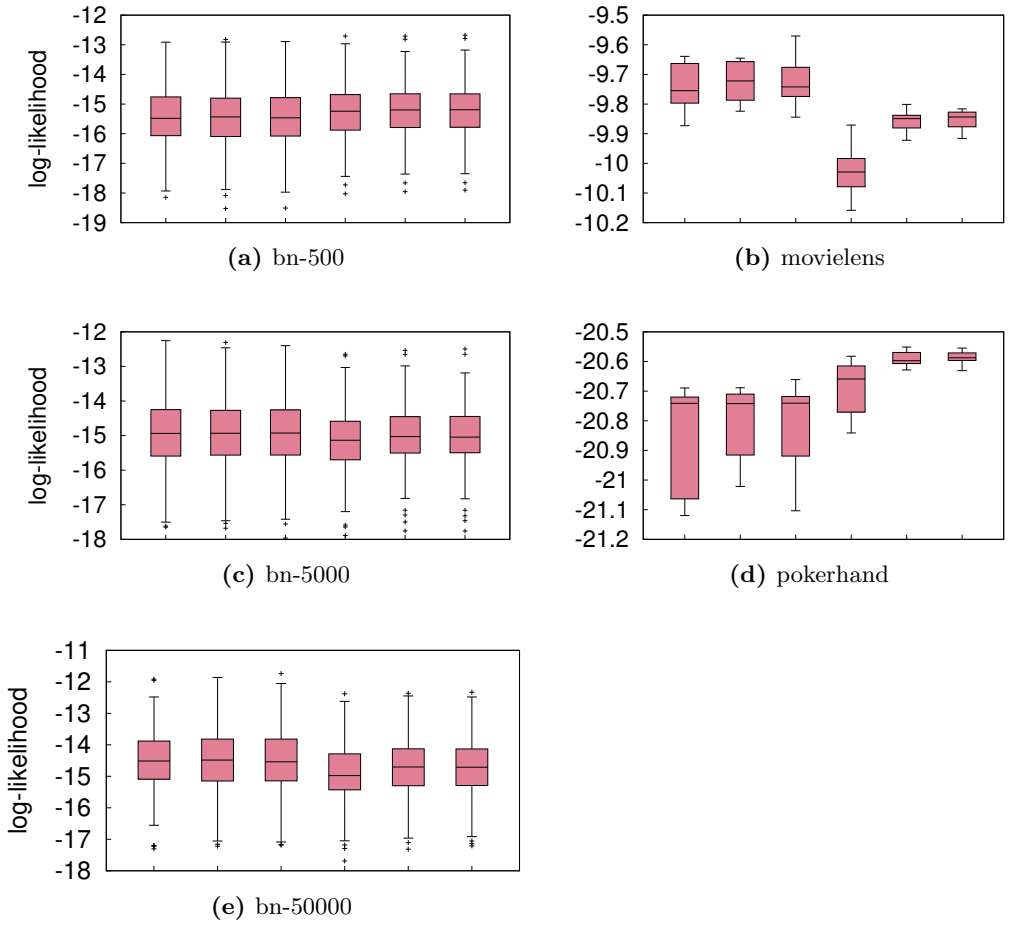
Appendix

Appendix A

Discrete Joint Densities

Figure A.1 shows how EDDO performs in comparison with its batch version (J48 classifiers from WEKA instead of Hoeffding trees from MOA).

Figures A.2, A.3 illustrate that the performance of the EDDO density estimates depends on the variable ordering of the classifier chains. They show the distribution of 1000 randomly sampled classifier chains on 15 Bayesian networks with 7 nodes.



(a) bn-500 **(b)** movielens

(c) bn-5000 **(d)** pokerhand

(e) bn-50000

Figure A.1: The performance of EDDO and its batch version on the Bayesian networks with 500, 5000, and 50000 instances and on the movielens and pokerhand dataset. From left to right, the algorithms are $\text{Batch}_{\text{CC|ECC|EWCC}}$ and $\text{EDDO}_{\text{CC|ECC|EWCC}}$.

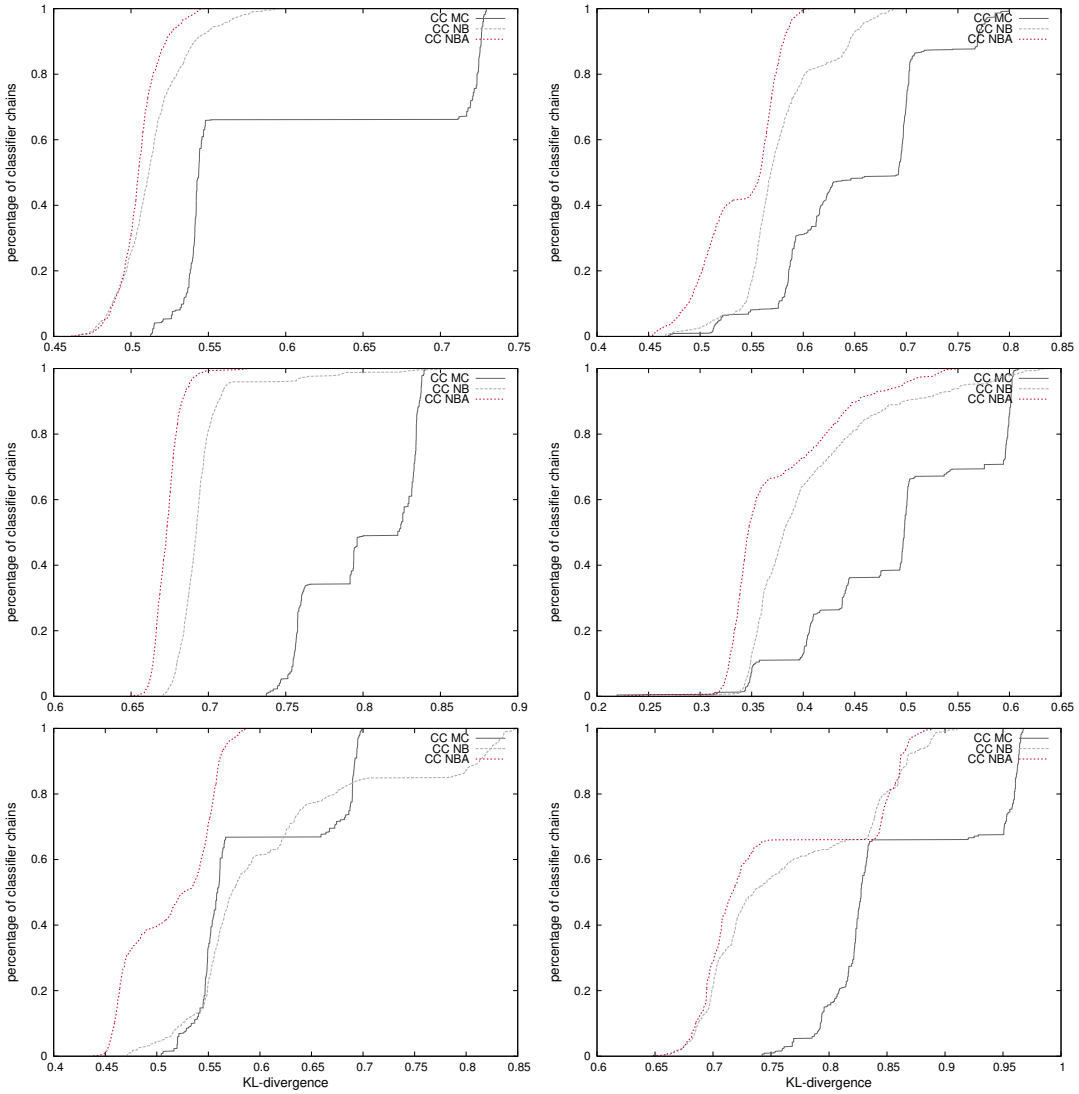


Figure A.2: Each plots shows the performance of 1000 EDDO_{CC} density estimators for a single Bayesian network with 7 nodes. On the x-axis is the KL-divergence, and on the y-axis is the percentage of classifier chains having a smaller or equal KL-divergence.

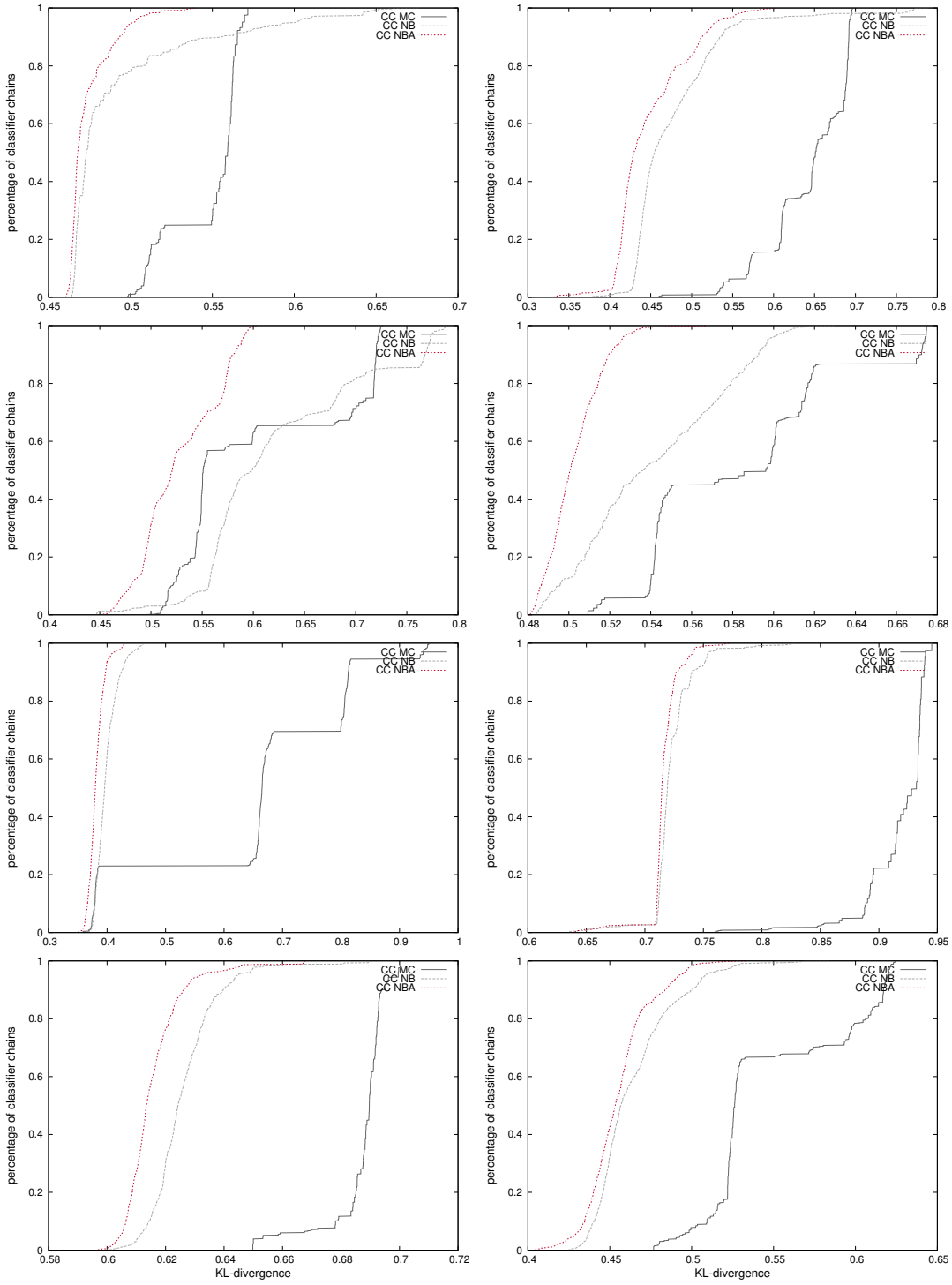
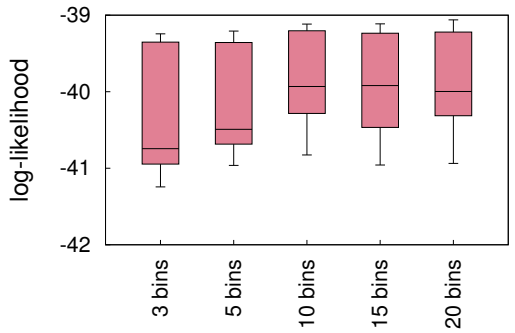


Figure A.3: Each plots shows the performance of 1000 EDDO_{CC} density estimators for a single Bayesian network with 7 nodes. On the x-axis is the KL-divergence, and on the y-axis is the percentage of classifier chains having a smaller or equal KL-divergence.

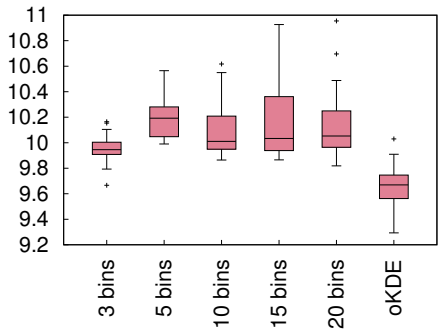
Appendix B

Continuous Joint Densities

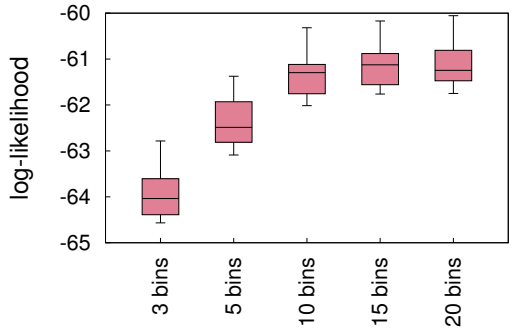
We compared EDO_{CC}(MC) with oKDE on five publicly available datasets: letter, electricity, shuttle, covertype, adult. oKDE had problems with the covertype and adult dataset: Either it was not able to complete a single job within twelve hours (covertype) or showed warnings regarding matrix computations (adult).



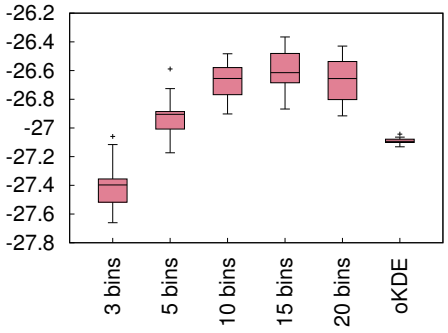
(a) adult



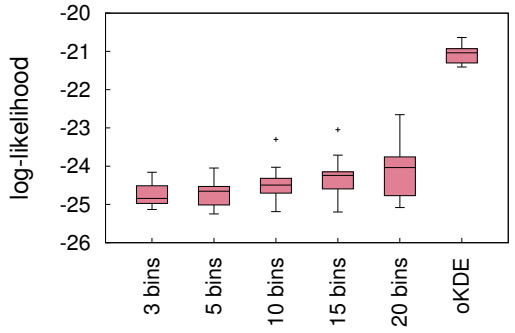
(b) electricity



(c) covertype



(d) letter



(e) shuttle

Figure B.1: EDO_{CC}(MC) compared with oKDE on the datasets electricity, shuttle, letter, covertype, and adult. On the y-axis is the average log-likelihood (computed prequentially). *i bins* with $i \in \{3, 5, 10, 15, 20\}$ stands for the number of bins used for discretizing the class attribute.

Bibliography

Adam, N. R. and J. C. Wortmann (1989). Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys 21*(4), 515–556.

Aggarwal, C. C. (2005). On k-anonymity and the curse of dimensionality. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, pp. 901–909.

Aggarwal, C. C. (2007). On randomization, public information and the curse of dimensionality. In *Proceedings of the 23rd International Conference on Data Engineering ICDE*, pp. 136–145.

Aggarwal, C. C., J. Han, J. Wang, and P. S. Yu (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases (VLDB)*, pp. 81–92.

Aggarwal, C. C., A. Hinneburg, and D. A. Keim (2001). On the surprising behavior of distance metrics in high dimensional spaces. In *Proceedings of the 8th International Conference on Database Theory*, pp. 420–434.

Aggarwal, C. C. and P. S. Yu (2004). A condensation approach to privacy preserving data mining. In *Proceedings of the 9th International Conference on Extending Database Technology Technology (EDBT)*, pp. 183–199.

Aggarwal, C. C. and P. S. Yu (2008). A general survey of privacy-preserving data mining models and algorithms. In C. C. Aggarwal and P. S. Yu (Eds.), *Privacy-Preserving Data Mining*, Volume 34 of *Advances in Database Systems*, pp. 11–52. Springer.

Agrawal, R., T. Imielinski, and A. N. Swami (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp. 207–216.

Agrawal, R. and R. Srikant (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of 20th International Conference on Very Large Data Bases (VLDB)*, pp. 487–499.

Agrawal, R. and R. Srikant (2000). Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pp. 439–450.

Archambeau, C. and M. Verleysen (2003). Fully nonparametric probability density function estimation with finite gaussian mixture models. In *Proceedings of the 5th International Conference on Advances in Pattern Recognition (ICAPR)*, pp. 81–83.

Atallah, M., A. Elmagarmid, M. Ibrahim, E. Bertino, and V. Verykios (1999). Disclosure limitation of sensitive rules. In *Proceedings of the 1999 Workshop on Knowledge and Data Engineering Exchange*, Washington, DC, USA, pp. 45–52. IEEE Computer Society.

Bach, S. H. and M. A. Maloof (2010). A bayesian approach to concept drift. In *Proceedings of the 24th Annual Conference on Neural Information Processing Systems*, pp. 127–135.

Bastide, Y., R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal (2000). Mining frequent patterns with counting inference. *SIGKDD Explorations 2*(2), 66–75.

Bauer, E. and R. Kohavi (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning 36*(1-2), 105–139.

Bifet, A., G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl (2010). MOA: Massive online analysis, a framework for stream classification and clustering. *Journal of Machine Learning Research - Proceedings Track 11*, 44–50.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bonchi, F. and C. Lucchese (2006). On condensed representations of constrained frequent patterns. *Knowledge and Information Systems 9*(2), 180–201.

Boulicaut, J. and A. Bykowski (2000). Frequent closures as a concise representation for binary data mining. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PADKK)*, pp. 62–73.

Boulicaut, J., A. Bykowski, and C. Rigotti (2003). Free-sets: A condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery 7*(1), 5–22.

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

Brin, S., R. Motwani, J. D. Ullman, and S. Tsur (1997). Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pp. 255–264.

Buchwald, F., T. Girschick, E. Frank, and S. Kramer (2010). Fast conditional density estimation for quantitative structure-activity relationships. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pp. 1268–1273.

Bykowski, A. and C. Rigotti (2001). A condensed representation to find frequent patterns. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*.

Bykowski, A. and C. Rigotti (2003). DBC: a condensed representation of frequent patterns for efficient mining. *Information Systems 28*(8), 949–977.

Calders, T. and B. Goethals (2002). Mining all non-derivable frequent itemsets. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, pp. 74–85.

Calders, T., C. Rigotti, and J. Boulicaut (2004). A survey on condensed representations for frequent sets. In *Proceedings of the European Workshop on Inductive Databases and Constraint Based Mining*, pp. 64–80.

Cesa-Bianchi, N. and G. Lugosi (2006). *Prediction, learning, and games*. Cambridge University Press.

Chakraborty, S. (2008). Some applications of dirac's delta function in statistics for more than one random variable. *Applications and Applied Mathematics: An International Journal (AAM) 3*(1), pp. 4254.

Chen, F., P. Deng, J. Wan, D. Zhang, A. V. Vasilakos, and X. Rong (2015). Data mining for the internet of things: Literature review and challenges. *IJDSN 2015*, 431047:1–431047:14.

Chen, K. and L. Liu (2011). Geometric data perturbation for privacy preserving outsourced data mining. *Knowledge and Information Systems 29*(3), 657–695.

Cheng, J., R. Greiner, J. Kelly, D. Bell, and W. Liu (2002, May). Learning bayesian networks from data: An information-theory based approach. *Artifical Intelligence 137*(1-2), 43–90.

Cheng, J., Y. Ke, and W. Ng (2006). delta-tolerance closed frequent itemsets. In *Proceedings of the 6th IEEE International Conference on Data Mining*, pp. 139–148.

Cheng, J., Y. Ke, and W. Ng (2008). A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems 16*(1), 1–27.

Chi, Y., H. Wang, P. S. Yu, and R. R. Muntz (2006). Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. *Knowledge and Information Systems 10*(3), 265–294.

Cover, T. M. and J. A. Thomas (2006). *Elements of information theory (2. ed.)*. Wiley.

Davies, S. and A. W. Moore (2002). Interpolating conditional density trees. In *Uncertainty in Artificial Intelligence*, pp. 119–127.

De, I. and A. Tripathy (2013). A secure two party hierarchical clustering approach for vertically partitioned data set with accuracy measure. In *Recent Advances in Intelligent Informatics - Proceedings of the Second International Symposium on Intelligent Informatics*, pp. 153–162.

Dembczynski, K., W. Cheng, and E. Hüllermeier (2010). Bayes optimal multilabel classification via probabilistic classifier chains. In *International Conference on Machine Learning*, pp. 279–286.

Demsar, J., Z. Bosnic, and I. Kononenko (2014). Visualization and concept drift detection using explanations of incremental models. *Informatica (Slovenia) 38*(4).

Domingo-Ferrer, J. and V. Torra (2005). Ordinal, continuous and heterogeneous k-anonymity through microaggregation. *Data Mining and Knowledge Discovery 11*(2), 195–212.

Domingos, P. and G. Hulten (2000). Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pp. 71–80.

Dowd, J., S. Xu, and W. Zhang (2006). Privacy-preserving decision tree mining based on random substitutions. In *Proceedings of the 2006 International Conference on Emerging Trends in Information and Communication Security (ETRICS)*, pp. 145–159.

Dries, A. and U. Rückert (2009). Adaptive concept drift detection. *Statistical Analysis and Data Mining 2*(5-6), 311–327.

Du, W. and Z. Zhan (2002). Building decision tree classifier on private data. In *Proceedings of the IEEE International Conference on Privacy, Security and Data Mining - Volume 14*, pp. 1–8. Australian Computer Society, Inc.

Dwork, C. (2006). Differential privacy. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming*, pp. 1–12.

Dwork, C., K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor (2006). Our data, ourselves: Privacy via distributed noise generation. In *Proceedings of the 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques Advances in Cryptology (EUROCRYPT)*, pp. 486–503.

Elgammal, A., R. Duraiswami, and L. S. Davis (2003). Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25*, 1499–1504.

Emekçi, F., O. D. Sahin, D. Agrawal, and A. El Abbadi (2007). Privacy preserving decision tree learning over multiple parties. *Data and Knowledge Engineering 63*(2), 348–361.

Even, S., O. Goldreich, and A. Lempel (1985). A randomized protocol for signing contracts. *Communications of the ACM 28*(6), 637–647.

Evfimievski, A. V., R. Srikant, R. Agrawal, and J. Gehrke (2002). Privacy preserving mining of association rules. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 217–228.

Frank, E. and R. R. Bouckaert (2009). Conditional density estimation with class probability estimators. In *Proceedings of First Asian Conference on Machine Learning*, pp. 65–81.

Frank, E. and S. Kramer (2004). Ensembles of nested dichotomies for multi-class problems. In *Proceedings of the 21st International Conference of Machine Learning*, pp. 305–312.

Friedman, N. and M. Goldszmidt (1996). Learning Bayesian networks with local structure. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 252–262.

Galushka, M., D. W. Patterson, and N. Rooney (2006). Temporal data mining for smart homes. In *Designing Smart Homes, The Role of Artificial Intelligence*, pp. 85–108.

Gama, J. and P. Kosina (2013). Recurrent concepts in data streams classification. *Knowledge and Information Systems ro. 2013*.

Gama, J., P. Medas, G. Castillo, and P. P. Rodrigues (2004). Learning with drift detection. In *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (SBIA)*, pp. 286–295.

Gama, J. and C. Pinto (2006). Discretization from data streams: applications to histograms and data mining. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC)*, pp. 662–667.

Gamut, L. (1991). *Logic, Language, and Meaning: Intensional logic and logical grammar*. Logic, Language, and Meaning. University of Chicago Press.

Gao, J., W. Fan, J. Han, and P. S. Yu (2007). A general framework for mining concept-drifting data streams with skewed distributions. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, pp. 3–14.

Gibbs, A. L. and F. E. Su (2002). On choosing and bounding probability metrics. *International Statistical Review*, 419–435.

Gkoulalas-Divanis, A. and V. S. Verykios (2009). Exact knowledge hiding through database extension. *IEEE Transactions on Knowledge and Data Engineering 21*(5), 699–713.

Goethals, B. (2003). Survey on frequent pattern mining. Technical report, University of Helsinki.

Goldberger, J. and S. T. Roweis (2004). Hierarchical clustering of a mixture model. In *Advances in Neural Information Processing Systems 17*, pp. 505–512.

Gomes, J. B., E. M. Ruiz, and P. A. C. Sousa (2011). Learning recurring concepts from data streams with a context-aware ensemble. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC)*, pp. 994–999.

Gonçalves Jr, P. M. and R. S. M. de Barros (2013). RCD: A recurring concept drift framework. *Pattern Recognition Letters 34*(9), 1018–1025.

Guo, Y., Y. Tong, S. Tang, and D. Yang (2006). A FP-tree-based method for inverse frequent set mining. In *Proceedings of the 23rd National Conference on Databases on Flexible and Efficient Information Handling*, pp. 152–163. Springer Berlin Heidelberg.

Han, J., H. Cheng, D. Xin, and X. Yan (2007). Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery 15*(1), 55–86.

Han, J. and M. Kamber (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

Han, J., J. Pei, Y. Yin, and R. Mao (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery 8*(1), 53–87.

Hancock, T. R., T. Jiang, M. Li, and J. Tromp (1996). Lower bounds on learning decision lists and trees. *Information and Computation 126*(2), 114–122.

Harel, M., S. Mannor, R. El-Yaniv, and K. Crammer (2014). Concept drift detection through resampling. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*, pp. 1009–1017.

Holmes, M. P., A. G. Gray, and C. L. I. Jr. (2012). Fast nonparametric conditional density estimation. *CoRR abs/1206.5278*.

Hulten, G., L. Spencer, and P. M. Domingos (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pp. 97–106.

Hwang, J.-N., S.-R. Lay, and A. Lippman (1994). Nonparametric multivariate density estimation: a comparative study. *IEEE Transactions on Signal Processing 42*(10), 2795–2810.

Izenman, A. J. (1991). Recent developments in nonparametric density estimation. *Journal of the American Statistical Association 86*(413), 205–224.

Jain, Y. K., V. K. Yadav, and G. S. Panday (2011). An efficient association rule hiding algorithm for privacy preserving data mining. *International Journal on Computer Science and Engineering (IJCSE) 3*(7), 2792–2798.

Jha, S., L. Kruger, and P. McDaniel (2005). Privacy preserving clustering. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, pp. 397–417.

Johnson, W. B. and J. Lindenstrauss (1984). Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics 26*(1), 189–206.

Kantarcioglu, M. and C. Clifton (2004). Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering 16*(9), 1026–1037.

Kargupta, H., S. Datta, Q. Wang, and K. Sivakumar (2003). On the privacy preserving properties of random data perturbation techniques. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*, pp. 99–106.

Kifer, D., S. Ben-David, and J. Gehrke (2004). Detecting change in data streams. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB)*, pp. 180–191.

Kim, J. and C. D. Scott (2012). Robust kernel density estimation. *Journal of Machine Learning Research 13*, 2529–2565.

Krawczyk, B. and M. Wozniak (2013). Incremental learning and forgetting in one-class classifiers for data streams. In *Proceedings of the 8th International Conference on Computer Recognition Systems (CORES)*, pp. 319–328.

Kristan, M. and A. Leonardis (2010). Online discriminative kernel density estimation. In *International Conference on Pattern Recognition*, pp. 581–584.

Kristan, M., A. Leonardis, and D. Skocaj (2011). Multivariate online kernel density estimation with Gaussian kernels. *Pattern Recognition 44*(10-11), 2630–2642.

Lambert, C. G., S. E. Harrington, C. R. Harvey, and A. Glodjo (1999). Efficient on-line nonparametric kernel density estimation. *Algorithmica 25*(1), 37–57.

Lazarescu, M. (2005). A multi-resolution learning approach to tracking concept drift and recurrent concepts. In *Proceedings of the 5th International Workshop on Pattern Recognition in Information Systems (PRIS)*, pp. 52.

Li, H., M. Shan, and S. Lee (2008). DSM-FI: an efficient algorithm for mining frequent itemsets in data streams. *Knowledge and Information Systems 17*(1), 79–97.

Li, N., T. Li, and S. Venkatasubramanian (2007). t-closeness: Privacy beyond k-anonymity and l-diversity. In *Proceedings of the 23rd International Conference on Data Engineering ICDE*, pp. 106–115.

Li, W. and J. Liu (2009). Privacy preserving association rules mining based on data disturbance and inquiry limitation. In *Proceedings of the Fourth International Conference on Internet Computing for Science and Engineering (ICICSE)*, pp. 24–29.

Lin, X., C. Clifton, and M. Y. Zhu (2005). Privacy-preserving clustering with distributed EM mixture modeling. *Knowledge and Information Systems 8*(1), 68–81.

Lindell, Y. and B. Pinkas (2002). Privacy preserving data mining. *J. Cryptology 15*(3), 177–206.

Liu, B., Y. Xiao, P. S. Yu, L. Cao, Y. Zhang, and Z. Hao (2014). Uncertain one-class learning and concept summarization learning on uncertain data streams. *IEEE Transactions on Knowledge and Data Engineering 26*(2), 468–484.

Liu, H., J. D. Lafferty, and L. A. Wasserman (2007). Sparse nonparametric density estimation in high dimensions using the rodeo. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, pp. 283–290.

Liu, K., C. Giannella, and H. Kargupta (2006). An attacker's view of distance preserving maps for privacy preserving data mining. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pp. 297–308.

Liu, K., H. Kargupta, and J. Ryan (2006). Random projection-based multiplicative data perturbation for privacy preserving distributed data mining. *IEEE Transactions on Knowledge and Data Engineering 18*(1), 92–106.

Liu, L., M. Kantarcioglu, and B. M. Thuraisingham (2008). The applicability of the perturbation based privacy preserving data mining for real-world data. *Data and Knowledge Engineering 65*(1), 5–21.

Liu, L., M. Kantarcioglu, and B. M. Thuraisingham (2009). Privacy preserving decision tree mining from perturbed data. In *Proceedings of the 42nd Hawaii International Conference on Systems Science (HICSS-42)*, pp. 1–10.

Machanavajjhala, A., D. Kifer, J. Gehrke, and M. Venkitasubramaniam (2007). *L*-diversity: Privacy beyond *k*-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD) 1*(1).

Manku, G. S. and R. Motwani (2002). Approximate frequency counts over data streams. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB)*, pp. 346–357.

Mann, H. B. and D. R. Whitney (1947, March). On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics 18*(1), 50–60.

Mann, T. P. (2006). Numerically stable hidden Markov model implementation. *An HMM scaling tutorial*, 1–8.

Mannila, H. and H. Toivonen (1997). Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery 1*(3), 241–258.

Manning, C. D., P. Raghavan, and H. Schütze (2008). *Introduction to information retrieval*. Cambridge University Press.

Masud, M. M., T. Al-Khateeb, L. Khan, C. C. Aggarwal, J. Gao, J. Han, and B. M. Thuraisingham (2011). Detecting recurring and novel classes in concept-drifting data streams. In *Proceedings of the 11th IEEE International Conference on Data Mining (ICDM)*, pp. 1176–1181.

Matloff, N. S. (1986). Another look at the use of noise addition for database security. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, pp. 173–181.

McGarry, K. (2005). A survey of interestingness measures for knowledge discovery. *Knowledge Engineering Review 20*(1), 39–61.

McLachlan, G. and D. Peel (2004). *Finite Mixture Models*. Wiley Series in Probability and Statistics. Wiley.

Melançon, G. and F. Philippe (2004). Generating connected acyclic digraphs uniformly at random. *Information Processing Letters 90*(4), 209–213.

Mitchell, T. M. (1997). *Machine learning*. McGraw Hill series in computer science. McGraw-Hill.

Moore, D., G. McCabe, and B. Craig (2009). *Introduction to the Practice of Statistics*. W.H. Freeman.

Motwani, R. and P. Raghavan (1995). *Randomized algorithms*. New York, NY, USA: Cambridge University Press.

Muralidhar, K., D. Batra, and P. J. Kirs (1995). Accessibility, security, and accuracy in statistical databases: The case for the multiplicative fixed data perturbation approach. *Management Science 41*(9), 1549–1564.

Oliveira, S. R. M. and O. R. Zaïane (2010). Privacy preserving clustering by data transformation. *Journal of Information and Data Management (JIDM) 1*(1), 37–52.

Oliveira, S. R. M., O. R. Zaïane, and Y. Saygin (2004). Secure association rule sharing. In *Proceedings of the 8th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pp. 74–85.

Parzen, E. (1962, 09). On estimation of a probability density function and mode. *The Annals of Mathematical Statistics 33*(3), 1065–1076.

Peherstorfer, B., D. Pflüger, and H. Bungartz (2014). Density estimation with adaptive sparse grids for large data sets. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pp. 443–451.

Pinkas, B. (2002). Cryptographic techniques for privacy-preserving data mining. *SIGKDD Explorations 4*(2), 12–19.

Powell, J. L. (1986). Estimation of semiparametric models. In R. F. Engle and D. McFadden (Eds.), *Handbook of Econometrics* (1 ed.), Volume 4, Chapter 41, pp. 2443–2521. Elsevier.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning 1*(1), 81–106.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies 27*(3), 221–234.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Rajalaxmi, R. and A. Natarajan (2008). An effective data transformation approach for privacy preserving clustering. *Journal of Computer Science 4*, 320–326.

Ram, P. and A. G. Gray (2011). Density estimation trees. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 627–635.

Rau, M. M., S. Seitz, F. Brimioulle, E. Frank, O. Friedrich, D. Gruen, and B. Hoyle (2015). Accurate photometric redshift probability density estimation - method comparison and application. *Monthly Notices of the Royal Astronomical Society 452*(4), 3710–3725.

Raykar, V. C. and R. Duraiswami (2006). Fast optimal bandwidth selection for kernel density estimation. In *Proceedings of the Sixth SIAM International Conference on Data Mining*, pp. 524–528.

Read, J., B. Pfahringer, G. Holmes, and E. Frank (2011). Classifier chains for multi-label classification. *Machine Learning 85*(3), 333–359.

Rokach, L. and O. Maimon (2005). Top-down induction of decision trees classifiers - a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C 35*(4), 476–487.

Rosenblatt, M. (1956, 09). Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics 27*(3), 832–837.

Russell, S. J. and P. Norvig (2010). *Artificial Intelligence - A Modern Approach*. Pearson Education.

Sakthithasan, S. and R. Pears (2014). Mining recurrent concepts in data streams using the discrete fourier transform. In *Proceedings of the 16th International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, pp. 439–451.

Saygin, Y., V. S. Verykios, and C. Clifton (2001). Using unknowns to prevent discovery of association rules. *SIGMOD Record 30*(4), 45–54.

Schlüter, T. and S. Conrad (2012). Hidden markov model-based time series prediction using motifs for detecting inter-time-serial correlations. In *Proceedings of the ACM Symposium on Applied Computing (SAC 2012)*, pp. 158–164.

Scott, D. W. and S. R. Sain (2004). *Multi-Dimensional Density Estimation*, pp. 229–263. Amsterdam: Elsevier.

Scutari, M. (2010). Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software 35*(3), 1–22.

Sese, J. and S. Morishita (2004). Itemset classified clustering. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pp. 398–409.

Shamir, A. (1979). How to share a secret. *Communications of the ACM 22*(11), 612–613.

Sheather, S. J. and M. C. Jones (1991). A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society. Series B (Methodological) 53*(3), pp. 683–690.

Silverman, B. W. and M. C. Jones (1989). E. Fix and J.L. Hodges (1951): An important contribution to nonparametric discriminant analysis and density estimation: Commentary on fix and hodges (1951). *International Statistical Review / Revue Internationale de Statistique 57*(3), 233–238.

Soria-Comas, J. and J. Domingo-Ferrer (2013). Differential privacy via t-closeness in data publishing. In *Proceedings of the Eleventh Annual International Conference on Privacy, Security and Trust (PST)*, pp. 27–35.

Su, J. and H. Zhang (2006). Full Bayesian network classifiers. In *Machine Learning, Proceedings of the Twenty-Third International Conference*, pp. 897–904.

Sun, X. and P. S. Yu (2007). Hiding sensitive frequent itemsets by a border-based approach. *Journal of Computing Science and Engineering JCSE 1*(1), 74–94.

Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10*(5), 557–570.

Tassa, T. (2014). Secure mining of association rules inhorizontally distributed databases. *IEEE Transactions on Knowledge and Data Engineering 26*(4), 970–983.

Traub, J. F., Y. Yemini, and H. Woźniakowski (1984). The statistical security of a statistical database. *ACM Transactions on Database Systems 9*(4), 672–679.

Vaidya, J. and C. Clifton (2002). Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 639–644.

Vaidya, J. and C. Clifton (2003). Privacy-preserving k-means clustering over vertically partitioned data. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 206–215.

Vaidya, J., C. Clifton, M. Kantarcioglu, and A. S. Patterson (2008). Privacy-preserving decision trees over vertically partitioned data. *TKDD 2*(3).

Vapnik, V. and S. Mukherjee (1999). Support vector method for multivariate density estimation. In *Neural Information Processing Systems*, pp. 659–665.

Verykios, V. S., E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis (2004). State-of-the-art in privacy preserving data mining. *SIGMOD Record 33*(1), 50–57.

Verykios, V. S., A. K. Elmagarmid, E. Bertino, Y. Saygin, and E. Dasseni (2004). Association rule hiding. *IEEE Transactions on Knowledge and Data Engineering 16*(4), 434–447.

Wan, R. and L. Wang (2010). Clustering over evolving data stream with mixed attributes. *Journal of Computational Information Systems*.

Wang, K., Y. Xu, R. She, and P. S. Yu (2006). Classification spanning private databases. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, pp. 293–298.

Wang, X. and Y. Wang (2015). Nonparametric multivariate density estimation using mixtures. *Statistics and Computing 25*(2), 349–364.

Warner, S. L. (1965). Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association 60*(309), 63–69.

Wied, D. and R. Weißbach (2012). Consistency of the kernel density estimator: a survey. *Statistical Papers 53*(1), 1–21.

Wilcoxon, F. (1945, December). Individual comparisons by ranking methods. *Biometrics Bulletin 1*(6), 80–83.

Wu, K., K. Zhang, W. Fan, A. Edwards, and P. S. Yu (2014). RS-forest: A rapid density estimator for streaming anomaly detection. In *Proceedings of the 14th International Conference on Data Mining*, pp. 600–609.

Wu, Y., C. Chiang, and A. L. P. Chen (2007). Hiding sensitive association rules with limited side effects. *IEEE Transactions on Knowledge and Data Engineering 19*(1), 29–42.

Xu, L., C. Jiang, J. Wang, J. Yuan, and Y. Ren (2014). Information security in big data: Privacy and data mining. *IEEE Access 2*, 1149–1176.

Xu, S., J. Zhang, D. Han, and J. Wang (2006). Singular value decomposition based data distortion strategy for privacy protection. *Knowledge and Information Systems 10*(3), 383–397.

Yao, A. C. (1986). How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pp. 162–167.

Yu, J. X., Z. Chong, H. Lu, and A. Zhou (2004). False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB)*, pp. 204–215.

Zaki, M. J., S. Parthasarathy, M. Ogihara, and W. Li (1997). New algorithms for fast discovery of association rules. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 283–286.

Zheng, Z., R. Kohavi, and L. Mason (2001). Real world performance of association rule algorithms. In *Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 401–406.

Zhou, A., Z. Cai, L. Wei, and W. Qian (2003). M-kernel merging: Towards density estimation over data streams. In *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, pp. 285–292. IEEE Computer Society.