Data Min Knowl Disc
https://doi.org/10.1007/s10618-017-0546-6

Online estimation of discrete, continuous, and conditional joint densities using classifier chains

Michael Geilke[1] · **Andreas Karwath**[1] · **Eibe Frank**[2] · **Stefan Kramer**[1]

Received: 18 December 2015 / Accepted: 9 November 2017

Abstract We address the problem of estimating discrete, continuous, and conditional joint densities online, i.e., the algorithm is only provided the current example and its current estimate for its update. The family of proposed online density estimators, estimation of densities online (EDO), uses classifier chains to model dependencies among features, where each classifier in the chain estimates the probability of one particular feature. Because a single chain may not provide a reliable estimate, we also consider ensembles of classifier chains and ensembles of weighted classifier chains. For all density estimators, we provide consistency proofs and propose algorithms to perform certain inference tasks. The empirical evaluation of the estimators is conducted in several experiments and on datasets of up to several millions of instances. In the discrete case, we compare our estimators to density estimates computed by Bayesian structure learners. In the continuous case, we compare them to a state-of-the-art online

Responsible editor: Hendrik Blockeel.

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10618-017-0546-6) contains supplementary material, which is available to authorized users.

✉ Michael Geilke
geilke@informatik.uni-mainz.de

Andreas Karwath
karwath@informatik.uni-mainz.de

Eibe Frank
eibe@cs.waikato.ac.nz

Stefan Kramer
kramer@informatik.uni-mainz.de

[1] Johannes Gutenberg-Universität Mainz, Staudingerweg 9, 55128 Mainz, Germany

[2] Department of Computer Science, The University of Waikato, Hamilton 3240, New Zealand

Published online: 25 November 2017

density estimator. Our experiments demonstrate that, even though designed to work online, EDO delivers estimators of competitive accuracy compared to other density estimators (batch Bayesian structure learners on discrete datasets and the state-of-the-art online density estimator on continuous datasets). Besides achieving similar performance in these cases, EDO is also able to estimate densities with mixed types of variables, i.e., discrete and continuous random variables.

Keywords Data streams · Density estimation · Classifier chains · Inference

1 Introduction

Density estimation is a well-known problem, which has received considerable attention in the offline (or batch) setting. With the emergence of data streams, the necessity of online density estimation has arisen, i.e., the need for density estimates that can be updated on a per-instance-basis and enable the representation of large amounts of data in a compact way. There is some recent work in this area (Lambert et al. 1999; Kristan and Leonardis 2010; Kristan et al. 2011; Kim and Scott 2012; Zhou et al. 2003; Wu et al. 2014), but it focuses on the online estimation of continuous densities, which is not suitable for datasets with mixed attribute types. Please notice that we use the term density estimation to refer to the estimation of probability masses, densities, and mixtures thereof.

In this paper, an extension of work we presented previously (Geilke et al. 2013), we propose and evaluate a family of online density estimators, called EDO (Estimation of Densities Online), which are suitable for the online estimation of discrete densities, continuous densities, conditional densities, and any mixture of them. They are designed to represent the densities in a compact way and also enable inference tasks. These inference facilities can be used to perform data mining without having access to the data stream (Geilke et al. 2014), which has several benefits: (1) *Run-time*: Many problems such as itemset mining become computationally expensive if performed on large amounts of data. With density estimates, these algorithms can operate on a compact representation of the data. In previous work (Geilke et al. 2014), we have shown that the density estimates presented in this paper provide sufficient information for this task. (2) *Privacy*: Since the algorithms operate on density estimates without incorporating any data from the data stream, the task of density estimation can be decoupled from the data mining or machine learning task. This allows companies to provide other entities with a view of their data without giving access to their data. (3) *Unknown task*: The decoupling has the additional benefit that the target data mining task does not need to be known at the time of collecting the data, as the density estimate is not specific to any task. The estimates simply represent the data and provide corresponding inference operations.

As this paper is an extension of a conference paper (Geilke et al. 2013), we distinguish between contributions already made in the conference paper and new ones:

Conference:

– We propose the use of classifier chains (CCs) and ensembles of classifier chains (ECCs) for the estimation of discrete densities.

– We introduce the use of ensembles of classifier chains to the world of unsupervised online learning.
– We propose ensembles of weighted classifier chains (EWCCs) as an alternative to regular, unweighted classifier chains. Our experiments show that EWCCs exhibit favorable behavior in many settings.
– We provide consistency proofs for the density estimators employing classifier chains, ensembles of classifier chains, and ensembles of weighted classifier chains.
– To illustrate potential applications of the estimated densities, we present inference algorithms that process queries based upon them.

Journal:

– We extend a method for conditional density estimation (Frank and Bouckaert 2009) to the online setting and employ this method to estimate continuous variables of joint densities.
– We generalize some of the inference algorithms presented in the conference version of this paper, and add an algorithm that incorporates soft evidence into Hoeffding trees. Moreover, we briefly show that the inference tasks proposed in this paper are also supported by the continuous density estimators we propose.
– The experiments extend and complement previous experiments in order to provide deeper insights into the online density estimators; we focus on stationary environments and do not consider changing data distributions as in the conference paper. Furthermore, all experiments are conducted with a broader range of datasets—in the synthetic cases (datasets generated from Bayesian networks with up to ten nodes) as well as in the real-world cases (eight real-world datasets).

The remainder of the paper is structured as follows. First, we formally define the problem addressed in this paper (Sect. 2), discuss related work (Sect. 3), and explain tools that we use throughout this paper (Sect. 4). Subsequently, we present the online density estimator for discrete densities (Sect. 5), the online density estimator for densities with mixed types of random variables (Sect. 6), and describe how inference tasks can be performed (Sect. 7). In Sect. 8, we present and discuss an extensive set of experiments, which have been conducted on synthetic and real-world data. Conclusions are drawn in Sect. 9.

2 Problem statement

Let $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$ be a joint density where the random variables are discrete, continuous, or a mixture thereof. In this paper, we address *online density estimation* from data streams $stream(f)$, where the $\mathbf{x}_i \in stream(f)$ are drawn independently and identically distributed (IID) from f.[1] Additionally, we define inference operations that allow users to perform queries regarding the density of the data stream. An online density estimator together with these inference operations will be called a

[1] Below we define the problem in a more general way to consider also drift and recurrent distributions, but we focus only on the most fundamental problem of estimating a single distribution from a stream in this paper.

probabilistic condensed representation of data. For a formal definition of the problem, we introduce the notion of online density estimation in this section and then describe the inference operations that need to be supported by an online density estimator to be called a probabilistic condensed representation of data.

2.1 Online density estimation

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.

Let X be a random variable with a set of possible outcomes $values(X)$. We call a random variable continuous if it can take on any value in the range $[a; b]$ for $a, b \in values(X)$. Otherwise, we call the random variable discrete. 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 $f(X) = \sum_{i=1}^{|V|} Pr(X = v_i) \cdot \delta(X - v_i)$ in the univariate case (Chakraborty 2008), where $V = \{v_1, \ldots, v_{|V|}\} = values(X)$. In case the density is conditioned on some random variables $Y_1, \ldots, Y_l$, we write $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$. Given a joint density f defined over variables $X_1, \ldots, X_m$, an *instance* of f is a variable assignment $X_1 = v_1, \ldots, X_m = v_m$, such that $v_i \in values(X_i)$ for $1 \leq i \leq m$. For reasons of readability, we also denote variable assignments $X_1 = v_1, \ldots, X_m = v_m$ as $\mathbf{x}$.

Definition 1 Let $\mathscr{F} := \{f^i(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l) \mid 1 \leq i \leq k \in \mathbb{N}\}$ be a set of joint densities. A *data stream of* $f \in \mathscr{F}$, denoted as $stream(f)$, is a possibly infinite sequence of instances $\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \ldots, \mathbf{x}_N$ that are drawn according to the probability distribution induced by f, and $stream(f)[1 : N]$ is the sequence $\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \ldots, \mathbf{x}_N$. A *data stream over* $\mathscr{F}$, denoted as $stream(\mathscr{F})$, is a possibly infinite sequence of instances $stream(\mathscr{F}) := stream(f^{j_1})[1 : N_{j_1}] \circ stream(f^{j_2})[1 : N_{j_2}] \circ \ldots$, where $f^{j_i} \in \mathscr{F}$ 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*. We assume that the stream is generated from a finite number of densities and generally seek *consistent* estimators, i.e., estimators that approach these densities with increasing numbers of instances per density.

Definition 2 Let $\mathscr{F} := \{f^i(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l) \mid 1 \leq i \leq k \in \mathbb{N}\}$ be a set of joint densities and let $stream(\mathscr{F})$ be a data stream over $\mathscr{F}$. 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 $\mathbf{x}_i$, 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., $\mathrm{KL}_{\mathrm{cond}}(f, \hat{f}) = \sum_{\mathbf{y}} \hat{f}(\mathbf{y}) \cdot \sum_{\mathbf{x}} \hat{f}(\mathbf{x} \mid \mathbf{y}) \cdot \ln \frac{\hat{f}(\mathbf{x}|\mathbf{y})}{f(\mathbf{x}|\mathbf{y})} = \mathbb{E}_f(\log(f(\mathbf{x} \mid \mathbf{y})) - \log(\hat{f}(\mathbf{x} \mid \mathbf{y}))) = \mathbb{E}_f(\log(f) - \log(\hat{f}))$, 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 $f \in \mathscr{F} : \mathrm{KL}_{\mathrm{cond}}(f, \hat{f}_N) \overset{N\to\infty}{\longrightarrow} 0$ 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 $f \in \mathscr{F}$ and estimating the corresponding transition probabilities for each pair $(f_i, f_j) \in \mathscr{F} \times \mathscr{F}$. 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 density estimate becomes active and receives the forthcoming instances until another drift occurs.

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

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

The consistency property demands that, *for all* $f \in \mathscr{F}$, the estimator approaches the true density, if $\sum_{h_i \in \{j_i \in \mathbb{N} \mid f^{j_i} = f\}} N_{h_i} \to \infty$. 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 $\mathscr{F}' = \{f_3\} \subset \mathscr{F}$. 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 $\mathbf{x}_{3,1}, \ldots, \mathbf{x}_{3,N_3}, \mathbf{x}_{3,N_3+1}, \ldots$ 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 $\mathscr{F}$ are approached with increasing numbers of instances.

Remark 3 In a data stream, the transition from one density to another is called *drift* in the literature. It can be abrupt (i.e., an immediate switch) or gradual (i.e., there is a transition phase). 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 $transition(f^{j_i}, f^{j_{i+1}}, T_i) := \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T_i}$ with $\mathbf{x}_i \in values(X_1) \times \ldots \times values(X_m)$ and $1 \leq i \leq T_i$. While the definition above is general enough to cover also drift and recurrent distributions, we focus on the most fundamental case of non-drifting data streams in the remainder of this paper and present detailed theoretical and empirical properties in this setting and for one specific approach. For a density estimator always adapting to the current data distribution, i.e. adapting to drift, we refer the reader to Geilke et al. (2013). For a density estimator modeling the history of possibly recurrent data distributions, we refer the reader to Geilke et al. (2015).

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 Inference

As we assume that users are not only interested in the full density but also specific parts of it, we also require infrastructure to pose queries on the densities 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 order to illustrate these inference operations, we consider three scenarios, which we call *inference scenarios*:

1. Let f be a joint density defined on variables $X := \{X_1, \ldots, X_m\}$. Further, let $Y := \{Y_1, \ldots, Y_l\} \subset X$ and $Z := \{Z_1, \ldots, Z_{m'}\} \subset X$ be subsets with $Y \cap Z = \emptyset$. Then we can determine a new density

$$f'(Z_1, Z_2, \ldots, Z_{m'} \mid Y_1 = y_1, \ldots, Y_l = y_l), \tag{1}$$

   where $y_j \in values(Y_j)$, $j \in [1; l]$.
2. Let f, X, and Z be as in 1. Further, let Y be a variable of 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

$$f'(Z_1, Z_2, \ldots, Z_{m'} \mid Y'), \tag{2}$$

where Y' takes the value y_j with probability p_j, $j \in [1; |values(Y)|]$.

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 $T = \sum_{\mathbf{x}': f(\mathbf{x}') \geq \theta} f(\mathbf{x}')$, $f'(\mathbf{x})$ equals $\frac{f(x)}{T}$ for all instances $\mathbf{x}$ with $f(\mathbf{x}) \geq \theta$, and $f'(\mathbf{x}) = 0$ for all other instances. The continuous case is defined analogously.

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. Notice that for the first two settings $Y \cup Z$ is not necessarily X, but could be a proper subset.

Density estimators supporting these inference operations constitute a probabilistic condensed representation of data on which data mining tasks can be performed such as pattern mining (Geilke et al. 2014). To support these inference operations, we propose online algorithms for estimating probabilistic condensed representations of $f(X_1, \ldots, X_m | Y_1, \ldots, Y_l)$ from data streams using classifier chains.

3 Related work

The paper focuses on non-parametric density estimation (Table 1). Non-parametric density estimation is a well-established problem that has received a lot of attention in the offline setting. This includes recent work based on decision trees, 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, which are estimated by piecewise constant estimators in the leaves. A similar approach was pursued by Davies and Moore (2002) as part of a conditional density estimator. Here, a tree represents a conditional density and the leaves may contain non-constant densities.

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 paper. 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 has been tackled using many different techniques. For example, Vapnik and Mukherjee (1999) employed support vector machines (SVM) to perform density estimation. They approached the problem by solving the integral over the density function using SVMs. More frequent approaches for multivariate

Table 1 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

Method	Setting	Multivariate	Conditional	Variable types
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	
EDO		Yes	Yes	Mixed

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 *Variable Types*, we mean whether the random variables are discrete and / or continuous. The method proposed in this paper is EDO

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 that build the given random process. Each component is usually represented by a parametric model such as a Gaussian density, which is influenced by the mean and its variance. The variance is a smoothing parameter, which is generally denoted as the bandwidth in kernel density estimates. The 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, see for example Raykar and Duraiswami (2006) or Sheather and Jones (1991) in the univariate case. 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 the focus in an approach 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.

Another aspect of density estimation is the set of properties that are fulfilled by the density estimate. For example, for financial data, the median of the data is often more important than the mean (Hall and Presnell 1999), which is often not the focus of density estimates such as kernel density estimators. To address this problem, some authors target density estimation under constraints, e.g., Hall and Presnell (1999) and Cheng et al. (1999).

4 Background

In this section, we introduce the concepts that are relevant for this paper. In particular, we briefly describe *online learning* and describe two tools that we use to model densities in an online fashion, namely Hoeffding Trees (Domingos and Hulten 2000) and probabilistic classifier chains (Dembczynski et al. 2010).

4.1 Online learning

Online learning emerged from the context of function learning (Littlestone 1987). Motivated by positive learning results on classes of Boolean functions (Valiant 1984), Littlestone (1987) proposed a problem setting in which learning is not performed on a separate training set but instance by instance. In his particular case, there is a learner L and a class of Boolean functions $\mathscr{F}$. From this class, the learner is supposed to learn an unknown function $f : \{0,1\}^n \rightarrow \{0,1\}$ from instances belonging to f, i.e., $\mathbf{x} \in \{0,1\}^n$ with $f(\mathbf{x}) \in \{0,1\}$. The instances are provided in rounds to L, such that L first receives an instance $\mathbf{x}$, makes a prediction for $f(\mathbf{x})$, and then receives the actual value of $f(\mathbf{x})$. To assess the performance of L, one can measure the number of mistakes.

Later on, many different variations of this setting have been proposed and analyzed (Blum 1996; Cesa-Bianchi and Lugosi 2006). One example is the *prediction from expert advice* framework, in which an unknown sequence is predicted by a *forecaster* with the help of *experts*. In this framework, there is a variable X that can take certain values, $values(X)$. The set of all possible values is called the *outcome space*, and the set of all possible values predicted by the forecaster is called the *decision space*, D. Similarly to the online setting described above, the forecaster has to predict the next element of the sequence, x_N. But this time, there are several experts $\mathscr{E}$ who provide

predictions of their own and the forecaster has access to them. Hence, the forecaster can examine the predictions of the experts before making a final decision, $\hat{x}_F$. When the true value is revealed, predictions are scored with a given loss function $loss : X \times D \to \mathbb{R}_+$, so that the forecaster knows the correct answer and also knows which experts were actually correct. The goal of the forecaster is to minimize its *regret* with respect to all experts, i.e., minimizing $\sum_{i=1}^{N} \sum_{E \in \mathscr{E}} loss(\hat{x}_{F,i}, x_i) - loss(\hat{x}_{E,i}, x_i)$, where $\hat{x}_{F,i}$ is the prediction of the forecaster for the element x_i and $\hat{x}_{E,i}$ is the corresponding prediction of the expert E.

In this paper, we will use an existing online learning algorithm to improve our own estimates. If the density estimate is an ensemble of consistent density estimates, one can use the so-called *simulatable experts* from the online learning setting to speed up the convergence of the density estimate (see Sect. 5.4).

4.2 Hoeffding trees

Hoeffding trees are an extension of decision trees bringing tree classifiers to the world of data streams. The primary difference is the sequential training of the tree that is achieved by using the Hoeffding bound to make split decisions. A split decision is no longer made by evaluating a heuristic measure on the full dataset, denoted by G, but by evaluating a heuristic measure based on the current counts of variable values, denoted by $\hat{G}$. Since only a subset of the data has been observed at the time of a split decision, one can only estimate the true heuristic measure, thereby introducing a certain error ε. In order to correct for this error and to make a decision that holds with high confidence, the Hoeffding bound provides an estimate for ε that holds with high confidence: Let X_a and X_b be the random variables that have the highest and second highest values with respect to $\hat{G}$. A split is performed if

$$\hat{G}(X_a) - \hat{G}(X_b) > \sqrt{\frac{R^2 \cdot \ln(1/\delta)}{2 \cdot N}},$$

where R is the range of the heuristic measure, $1 - \delta$ is the probability that ε is correct, and N is the number of instances. Several extensions of this basic idea lead to the VFDT (Very Fast Decision Tree learner) system, which has later been implemented as part of the MOA framework (Bifet et al. 2010).

4.3 Classifier chains

Probabilistic classifier chains (Dembczynski et al. 2010), which are based on the deterministic chains proposed by Read et al. (2011), are used in multi-label classification to model the dependencies between the labels and exploiting them for improving the prediction (Dembczynski et al. 2012; Kumar et al. 2013; Dembczynski et al. 2016). Similarly to the online density estimator proposed in this paper, they use the product rule to split the joint density into smaller parts, i.e.,

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

and then use classifiers such as decision trees to estimate the f_i, where $X_1, \ldots, X_m$ are the features and $Y_1, \ldots, Y_l$ are the labels that have to be predicted. So far, however, classifier chains have not been considered in the context of computing density estimates from data stream. In particular, there are no density estimates that are based on classifier chains and allow to perform inference tasks that are interesting from a density perspective, e.g., drawing instances, providing evidence, or marginalizing out variables.

5 Online density estimation using classifier chains

To enable online estimation of densities, we employ classifier chains to model dependencies between random variables. It not only factors the density into smaller—hopefully easier to understand—parts but also enables the application of a broad range of fast and efficient univariate online density estimators. In this section, we discuss how to obtain estimate $\hat{f}$ for a density $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$, where $X_1, \ldots, X_m$ and $Y_1, \ldots, Y_l$ are discrete. An extension to mixed types of random variables is presented in Sect. 6.

5.1 Online density estimation using classifiers

The online density estimators proposed in this paper represent joint densities using classifiers. 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

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

and each of the factors can be represented by a classifier providing class probability estimates. 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 Sect. 7), and are suitable for the goal of online density estimation, i.e., the minimization of the KL-divergence. 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 Sect. 5.5).

Because a single density estimator may not provide a reliable estimate, we also apply ensemble techniques (see Sect. 5.3) and weight the ensemble members according to

their current performance using the log-likelihood (see Sect. 5.4). 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 Sect. 5.5).

5.2 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:

$$f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l) = f_1(X_1 \mid Y_1, \ldots, Y_l) \cdot \prod_{i=2}^{m} f_i(X_i \mid Y_1, \ldots, Y_l, X_1, \ldots, X_{i-1}). \quad (3)$$

In other words, in order to model the density f, it is sufficient to model the density $f_1(X_1 \mid Y_1, \ldots, Y_l)$ and the densities $f_i(X_i \mid Y_1, \ldots, Y_l, X_1, X_2, \ldots, X_{i-1})$, $i \in \{2, \ldots, m\}$. To model the individual densities $f_i, 1 \leq i \leq m$, we employ classifiers that return class probability estimates. In particular, we employ Hoeffding trees (Domingos and Hulten 2000), which provide the probability masses of the classes of X_i. The Hoeffding trees allow us to estimate the density in an online fashion. 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), but we do not investigate this scenario in this paper.

Based on the classifier chain implied by Eq. 3, the proposed algorithm initializes the base classifiers for $f_1, \ldots, f_m$. When the algorithm receives an example $(\mathbf{y}, \mathbf{x})$ 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 $(\mathbf{y}, x_1)$ is forwarded to the classifier for f_1 and the example $(\mathbf{y}, x_1, \ldots, x_i)$ is forwarded to the classifier for f_i, $i \in \{1, \ldots, m\}$. Subsequently, each classifier processes its example and updates its current estimate.

5.3 Ensembles of classifier chains

The product on the right-hand side of Eq. 3 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

$$f_1(X_{\pi(1)} \mid Y_1, \ldots, Y_l) \cdot \prod_{i=2}^{m} f_i(X_{\pi(i)} \mid Y_1, \ldots, Y_l, X_{\pi(1)}, \ldots, X_{\pi(i-1)}) \quad (4)$$

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,

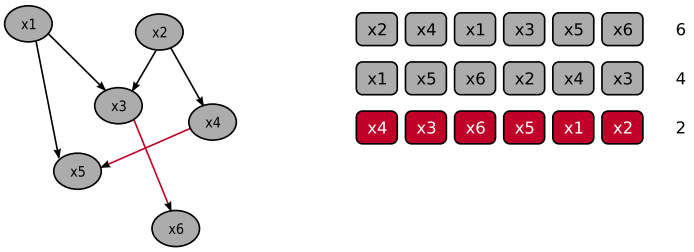

Fig. 1 On the left, the figure shows a Bayesian network representing a joint density with six discrete variables. On the right are three possible orderings of the variables plus the corresponding number of interdependencies of the Bayesian network that are represented in the correct ordering (i.e., $x_i \preceq x_j$ in the chain if there is a sequence of directed edges $x_i \to \cdots \to x_j$ in the Bayesian network). As illustration, we highlighted the interdependencies represented by the last variable ordering

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 1 provides an illustration for this statement. On the left is an Bayesian network with six variables. On the right are three possible orderings of the variables. Dependent on the variable 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* ($X_6 \to X_2$ instead of $X_2 \to X_3$ together with $X_3 \to X_6$), which potentially makes learning more difficult for the classifier. The experiments in Sect. 8.1 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 Sect. 8.1), 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.

5.4 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 of the ensemble. One possible approach is the *exponentiated gradient investment strategy* (EG investment strategy) (Cesa-Bianchi and Lugosi 2006), known

Algorithm 1: 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 and $(\mathbf{y}_{t+1}, \mathbf{x}_{t+1})$ be the current instance.

```
// a parameter controlling the weight update
```
1 $\eta \leftarrow (c/C) \cdot \sqrt{(8 \cdot \ln k)/N}$
```
// the current density values
```
2 **for** $i = 1, \ldots, k$ **do**
3 $\quad p[i] \leftarrow f_{cc_i}.densityValue(\mathbf{y}_{t+1}, \mathbf{x}_{t+1})$
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 $\quad w_{t+1}[i] \leftarrow \dfrac{w_t[i] \cdot exp(\eta \cdot p[i]/wp)}{\sum_{j=1}^{k} w_t[j] \cdot exp(\eta \cdot p[j]/wp)}$
8 **end**

from the area of 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 $s_{t+1}[i] := \frac{s_{t+1}[i]}{s_t[i]}$ 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 $\frac{C}{c} \cdot \sqrt{\frac{N}{2} \ln(m)}$, where c is the smallest observed value and C is the largest observed value (Cesa-Bianchi and Lugosi 2006).

Algorithm 1 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 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 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 current density value of each chain is calculated. 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 paper, we allow a deviation of 30%). This last step is supposed to reduce the influence of poorly performing classifier chains (see Sect. 8.1). Then we simply combine the probabilities of the individual classifier chains with respect to

their weights, renormalizing again as necessary. Notice that the ensemble of weighted classifier chains that we just 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.

5.5 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 section 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 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.

Proposition 1 *Let f be a discrete joint density with $f(\mathbf{x} \mid \mathbf{y}) = f_1(x_1 \mid \mathbf{y}) \cdot \ldots \cdot f_m(x_m \mid \mathbf{y}, x_1, \ldots, x_{m-1})$. 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 $i \in [1; m]$, 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 $i \in [1; n]$ (in the following, $\sum_{\mathbf{x}}$ is a shorthand for the sum over all possible values of the vector $\mathbf{x}$):

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

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

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

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

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

where $\prod_{j=1, j \neq i}^{m} \hat{f}_j(x_j \mid \mathbf{y}, x_1, \ldots, x_{j-1})$ has been replaced by 1 in the last but one step, which is possible, since $0 \leq \prod_{j=1, j \neq i}^{m} \hat{f}_j(x_j \mid \mathbf{y}, x_1, \ldots, x_{j-1}) \leq 1$. Also notice that $f(\mathbf{y}) = f_i(\mathbf{y})$ for all $i \in [1; n]$, i.e., each classifier uses the same estimator for $\mathbf{y}$. Hence, if $\mathrm{KL}_{\mathrm{cond}}(f_i, \hat{f}_i) \to 0$ for all $i \in [1; n]$, 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 Majority Class leaf classifiers, for example, partitions the instances by the attributes based on which splits took place and the values of the attributes. 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 Proposition 1 to estimators employing an ensemble of (weighted) classifier chains.

Proposition 2 *Let f be a discrete joint density with $f(\mathbf{y}, \mathbf{x}) = f_1(x_1 \mid \mathbf{y}) \cdot \ldots \cdot f_m(x_m \mid \mathbf{y}, x_1, \ldots, x_{m-1})$. 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 $i \in [1; k]$ and $j \in [1; m]$, then $KL_{cond}(f, \hat{f}) \to 0$.*

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

6 Mixed types of random variables

In the previous section, we described a framework for estimating conditional joint densities using classifier chains. To estimate the conditional densities $f_i(X_i \mid Y_1, \ldots, Y_l, X_1, \ldots, X_{i-1})$, which result from applying the product rule, we employed Hoeffding trees acting as class probability estimator for the X_i. 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.

Since the factors in Eq. 3 are independent from each other, the univariate density estimators can be mixed arbitrarily. Hence, we can employ Hoeffding trees for the f_i where X_i is discrete and some other univariate density estimator for the f_i where X_i is continuous. The first case has been already described in Sect. 5. In this section, we propose a density estimator for the latter case.

6.1 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 Y, they discretize the range of X, weight each instance according to the likelihood of falling into a certain bin of this discretization given the variables in Y, and create an estimate for $f(X \mid Y)$ using these weights. Hence, the instantiations x_j are basically reweighted in order to respect that X is conditioned on the variables in Y. For a specific value x_j of X, the weight is computed as follows:

$$w(x_j \mid Y) = N \cdot \frac{p(c_{x_j} \mid Y)}{N_{c_{x_j}}} \tag{5}$$

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}. Afterwards, one can choose an estimator for $f(X \mid Y)$. Frank and Bouckaert considered an 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:

$$f_{kernel}(x \mid Y) = \frac{1}{N} \sum_{j=0}^{N} w(x_j \mid Y) \cdot \mathcal{N}(x; x_j, \sigma^2_{kernel}) \ . \tag{6}$$

Fig. 2 $bin_0, bin_1, \ldots, bin_k$ 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}, $i \in [0; k-1]$

The kernel bandwidth σ_{kernel} is computed using estimators for the mean and variance: $\sigma_{kernel} = \frac{\sigma_I}{N^{1/4}}$ with $\sigma_I^2 = \frac{1}{N}\sum_{j=0}^{N} w(x_j|Y) \cdot (x_j - \mu_I)^2$ and $\mu_I = \frac{1}{N}\sum_{i=0}^{N} w(x_j|Y) \cdot x_j$. 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 Eq. 3 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 Eq. 6, since 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 present classifiers that provide class probability estimates. We will extend a discretization method for data streams that has been proposed by Gama and Pinto (2006) (Sect. 6.2) and extend Hoeffding trees to deal with this discretization method (Sect. 6.3). In order to reduce the number of summands in Eq. 6, we adapt a compression method proposed by Goldberger and Roweis (2004) (Sect. 6.4).

6.2 Discretization

First we deal with the problem of discretizing the target variable X_i in an online fashion. The algorithm by Gama and Pinto approaches the discretization for data streams using two layers. The first layer consists of a large number of intervals—many more than needed for the final discretization. Whenever the number of values within an interval exceeds a user-defined threshold, the bin is split into two bins or, if it is the first or last interval, 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 intervals into bigger intervals.

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 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 Fig. 2 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 2. It is based on an existing discretization

Algorithm 2: computeSoftBorders

Input: Confidence level θ, an initial equal-frequencey 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 V that is 1 if a value lies in $[0; b_j]$ and 0 otherwise

$p \leftarrow \dfrac{\sum_0^j |bin_i|}{\sum_0^k |bin_i|}$

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

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

where $\mu = |I| \cdot p$ and T is the sum of independent Bernoulli trials with probabilities $Pr(T_j = 0) = 1 - p$, $Pr(T_j = 1) = p$.

$l_j \leftarrow b_j - l$

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

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

where $\mu = |I| \cdot p$ and T is the sum of independent Bernoulli trials with probabilities $Pr(T_j = 0) = 1 - p$, $Pr(T_j = 1) = p$.

$u_j \leftarrow b_j + u$

end

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

$d = (b_0, \ldots, b_k)$ 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 b_j as the mean of the probability $\frac{\sum_0^j |bin_i|}{\sum_0^k |bin_i|}$ and use Chernoff bounds for the expected deviation from this mean. 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. Each Z_j has a success probability of $\frac{\sum_0^j |bin_i|}{\sum_0^k |bin_i|}$, 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_j, 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 1 shows the correctness of the described procedure.

Theorem 1 *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 2 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 to the interval $[0; 1]$, which we denote by b'_j. We design a coin flipping experiments with a two-sided coin 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 $\frac{\sum_0^j |bin_i|}{\sum_0^k |bin_i|}$, the probability of B is $1 - Pr(A)$.

Using Chernoff bounds, we then compute the lower and upper bounds for b_j, which are l_j and u_j. Making worst- and best-case assumptions, we obtain

$$x \in [b_{j-1} + u_{j-1}; b_j - l_j], \tag{7}$$

which means that x belongs to bin b_j with confidence level θ. Afterwards, $(b_j - l_j)$ 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 Hoffding 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.

6.3 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 $\overline{G}$, respectively. Afterwards, 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 null attribute, where R is the range of the variable, $1 - \delta$ is the confidence level, N is the number of instances, and the null attribute is a pseudo attribute for pre-pruning.

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 run-time of the Hoeffding tree, since many possible scenarios have to be considered.

6.4 Compression

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 Eq. 6). 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 $\pi : \{1, \ldots, r\} \to \{1, \ldots, r'\}$. Using this method, we obtain the compressed kernel as follows: $h^\pi = \sum_{j=1}^{r'} \beta_i g_j^\pi$ with

$$h_j^\pi = \mathcal{N}(\mu'_j, \Sigma'_j) \tag{8}$$

$$\beta_j = \sum_{i \in \pi^{-1}(j)} \alpha_i \tag{9}$$

$$\mu'_j = \frac{1}{\beta_j} \sum_{i \in \pi^{-1}(j)} \alpha_i \mu_i \tag{10}$$

$$\Sigma'_j = \frac{1}{\beta_j} \sum_{i \in \pi^{-1}(j)} \alpha_i (\Sigma_i + (\mu_i - \mu'_j) \cdot (\mu_i - \mu'_j)^T) \tag{11}$$

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

$$\left(\sum_{i=1}^{r} \alpha_i \right) \cdot \mathcal{N}(\mu_1', \Sigma_1') \tag{12}$$

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.

6.5 Consistency

In Sect. 5.5, we have shown that EDO 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 Sect. 5.5.

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 and a higher sensitivity to concept drifts.

7 Inference

The algorithms presented in the previous sections 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).

Algorithm 3: 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

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

To overcome this limitation, we propose to extend density estimates to so-called *probabilistic condensed representations of data*, 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, as previously presented in another paper (Geilke et al. 2014).

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 paper (e.g., Hoeffding trees) and (2) different base estimators 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 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. Please notice that, for reasons of readability, we consider densities $f(X_1, \ldots, X_m)$. All results can be easily extended to $f(X_1, \ldots, X_m \mid Y_1, \ldots, Y_l)$.

7.1 Hoeffding trees as base estimator

7.1.1 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, $k \in [1; |values(Y)|]$, 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 from the edge with the soft evidence down to

Algorithm 4: 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 ← draw chain according to (w_1, ..., w_k)
// draw attribute values
for X_i ∈ X_{π_cc(1)}, ..., X_{π_cc(m)} do
    dist_i ← (0 | 1 ≤ j ≤ values(X_i))
    for 1 ≤ j ≤ k do
        let f_j be the factor of cc_j with target variable X_i
        dist_i ← dist_i + w_j · f_j(X_i | X_1 = x_1, ..., X_{i-1} = x_{i-1})
    end
    normalize(dist_i)
    x_i ← draw value from dist_i (using Algorithm 5)
end
return (x_1, ..., x_m)
```

the leaf classifier are first multiplied and then multiplied with each value of the class distribution (see Algorithm 3).

Theorem 2 *Let $\hat{f}$ be a density estimate for a density $f(X_1, \ldots, X_m)$, Y be a random variable, and $(p_k \mid 1 \leq k \leq |values(Y)|)$ be soft evidence given for Y. Algorithm 3 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 $\mathbf{x} = (x_1, x_2, \ldots, x_m)$. 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 3, 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. □

7.1.2 Drawing instances

If we need to draw an instance from an online density 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 an estimate from each classifier, sample a value based on the distribution obtained, and use the output as input for the next classifier.

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 instance will be drawn based on a single ordering and the other densities will be adapted using marginalization (see Algorithm 4). 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 3 *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 4 draws instances according to f, if $\hat{f}$ is a consistent estimate and the number of training instances approaches infinity.*

Proof Let $\mathbf{x} = (x_1, \ldots, x_m)$ be an arbitrary instance. We will show that $\mathbf{x}$ 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 $\pi : \{1, \ldots, m\} \rightarrow \{1, \ldots, m\}$ 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 5 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 $\mathscr{X} \backslash \mathscr{Z} := \{Z_1, \ldots, Z_{m'}\}$ for a conditional density estimate $\hat{f}_i$, a variable $X \in \mathscr{X} \backslash \mathscr{Z}$ can take any value in $values(X)$, so that the density values of all paths starting at a node n with $randomVariable(n) \in \mathscr{X} \backslash \mathscr{Z}$ 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 5 does. □

7.2 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

Algorithm 5: 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 $\mathscr{Z} := \{Z_1, \ldots, Z_{m'}\}$

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

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

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 $X_i \in X$ with $X_i \notin Z$ 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 were possible to find a common root, we would only have the instance counters for the target variable but not of the underlying instances.

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 aggre-

gating all paths from the root to the leaves that match the requirements. Algorithm 5 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 Z). Then for each node, two cases need to be distinguished: (1) The node can only have 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 the nodes until it reaches a leaf. At the leaf some relevant information is extracted, which is the distribution of the target variable in the case of drawing instances, and aggregated with the other leaves of that tree.

7.3 Continuous base estimators

Since the continuous base estimator presented in Sect. 6 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.

7.4 Arbitrary base estimator

7.4.1 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 is finished 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. The parameter m ensures that enough instances have reached the new density estimator. A more detailed description is given by Algorithm 6. Please notice that the notation follows that of soft evidence, but we can simulate hard evidence by giving a

Algorithm 6: 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., 10,000), 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 $(p_k \mid k \in [1 : |values(Y_j)|])$
  end
  `// Sample an instance fullfilling these requirements`
  do
    $inst \leftarrow$ sample instances from $\hat{f}$
    $\hat{f}'.update(inst)$
    $LL \leftarrow LL$ - log($e.densityValue(inst)$)
    $LL' \leftarrow LL'$ - log($e'.densityValue(inst)$)
  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 $\hat{f}'$

probability of 1 to the hard evidence and set 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 classifiers.

7.4.2 Marginalization

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

7.5 Inference scenarios

In Sect. 2, 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 Eqs. 1 and 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 $\mathscr{X}\backslash\{Z_1, \ldots, Z_{m'}\}$. 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 Algorithms 3 and 4, 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 $\theta_1 + \varepsilon$ 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 $N \cdot (\theta_1 + \varepsilon)$. 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 $\mu = N \cdot (\theta_1 + \varepsilon)$ and and X is the sum of independent Poisson trials with $Pr(X = 0) = 1 - \theta_1 - \varepsilon$, $Pr(X = 1) = \theta_1 + \varepsilon$. For continuous random variable, θ is treated as a density value instead of a probability.

8 Evaluation

In this section, we evaluate the algorithms presented in Sects. 5, 6, and 7 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[2] (Mining Density Estimates inferred Online), which is based on MOA (version 2013.11) (Bifet et al. 2010). To improve readability, we introduce some notation to refer to specific variants: $EDDO_T(L)$ and $ECDO_T(L)$, where $T \in \{CC, ECC, EWCC\}$, and $L \in \{MC, NB, NBA\}$. $EDDO_T(L)$ represents online density estimators for discrete joint densities, and $ECDO$ represents online density estimators for densities with mixed random variables. The index denotes the type of density estimator, which is either an estimator employing a classifier chain (CC), an estimator employing an ensemble of classifier chains (ECC), or an estimator employing 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
- 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 aposteriori (bayes). For mixed densities, we employed the implementation of oKDE by Kristan et al. (2011), which is one of the few online density estimators for multivariate densities. Its implementation is publicly available (MATLAB).

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

[2] https://github.com/geilke/mideo.

[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 2 The table shows some properties of the data that has been used in this paper

Data	Type	#Attributes	#Instances
Bayesian networks	synthetic	4–10	100
Bayesian networks		4–10	1000
Bayesian networks		4–10	10,000
Movielens	Discrete	23	49,282
Pokerhand		11	1,025,015
US census		68	2,458,285
Letter	Continuous	17	19,999
Electricity		9	45,313
Shuttle		10	58,000
Adult	Mixed	15	30,163
Covertype		54	581,012

We distinguish four types of data: synthetic, discrete, continuous, and mixed. The latter is used for datasets that contain discrete and continuous variables

data, we distinguished two cases: For EDDO, 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. For ECDO, 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 properties of the data that we used are summarized in Table 2. The Bayesian networks were randomly generated using the `random.graph` method of the `bnlearn` package. Its parameter `method` was set to `melancon`, which uses a Markov chain to draw acyclic, directed graphs uniformly at random (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 being one of the values of $\{10^2, 10^3, 10^4\}$.

8.1 Chain orderings and ensemble size

As discussed in Sect. 5.3, 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 Fig. 3 (for the remaining results, see Online Resource 2). In all of the presented cases, we observe large differences between the best and the worst classifier chain, which range between 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

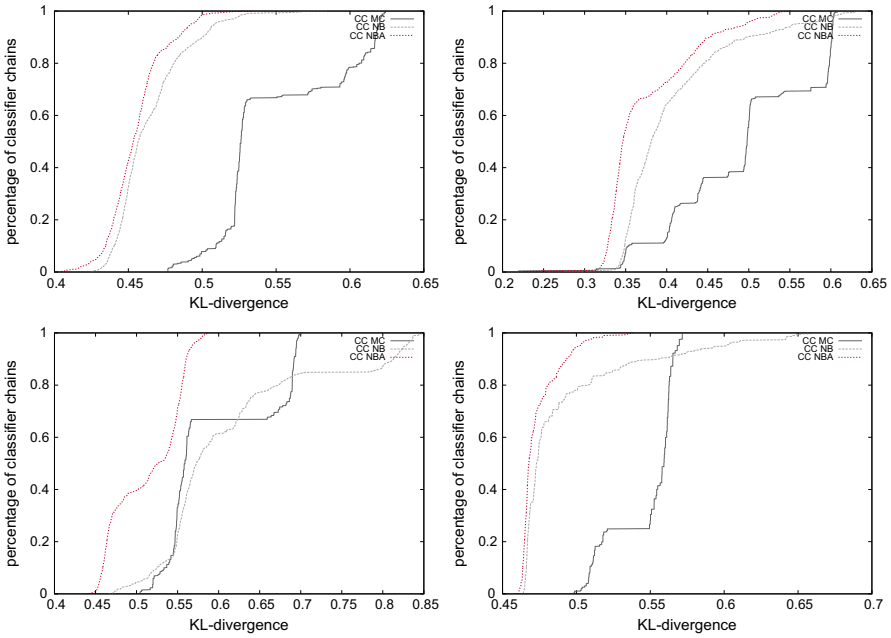

Fig. 3 Each plot shows the performance of 1000 $EDDO_{CC}$ density estimators for a single randomly-generated Bayesian network with 7 nodes. On the x-axis is the KL-divergence, and on the y-axis is the percentage of classifier chains having a smaller or equal KL-divergence

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 often have only a small effect. Hence, 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 of classifier chains employing several classifier chains. As the idea of ensembles of classifier chains 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 good performance. For further investigations of this matter, we conduct another experiment and analyze 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.

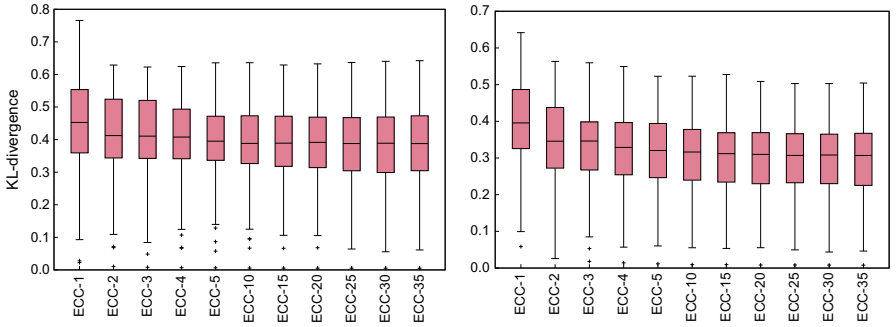

Fig. 4 The figure shows the KL-divergence of ensemble estimators with various ensemble sizes. It shows the performance of $EDDO_{ECC}(MC)$ (left) and $EDDO_{ECC}(NBA)$ (right) trained on 100,000 instances from Bayesian networks with 7 nodes

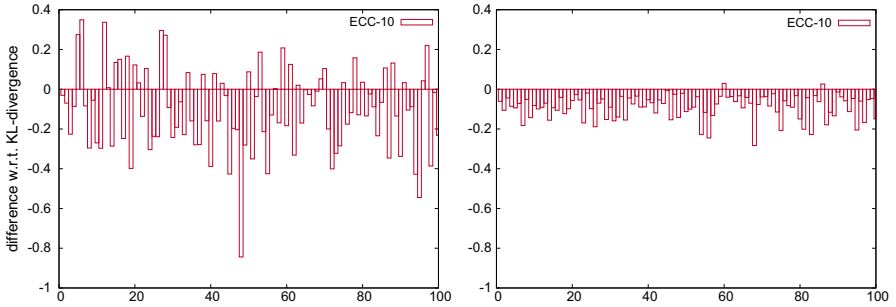

Fig. 5 The figure shows a direct comparison of ECC-1 with ECC-10 using NBA as leaf classifier when trained with 1000 (left) and 100,000 (right) instances from from Bayesian networks with 7 nodes. The x-axis of the plots shows the individual runs of the estimators. The y-axis shows the difference with respect to the KL-divergence

Figure 4 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 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 Fig. 5, 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 represent by the base estimators, if only few training instances are available.

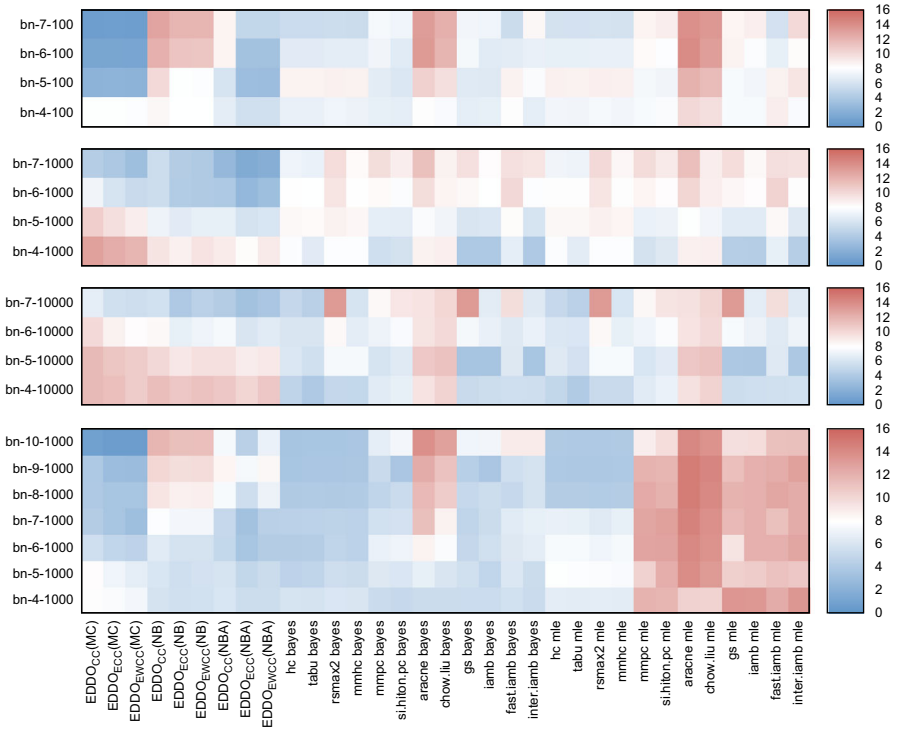

Fig. 6 The plots show the average rank (lower is better, i.e, blue is better) of different EDDO density estimators compared to 12 Bayesian structure learners, which use either mle or bayes to estimate the a posteriori probabilities of the CPTs. In the three plots at the top, each estimator has been trained with 10^2, 10^3, or 10^4 instances on datasets generated from Bayesian networks with 4–7 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 4–10 nodes. As performance measure, the average log-likelihood has been used (Color figure online)

8.2 Discrete densities

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 a strong competitor, 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 node sizes between 4 and 7, we computed the KL-divergence and the average log-likelihood. For Bayesian networks with 8 to 10 nodes, we only compute the average log-likelihood due to run-time constraints. As visible in the result plots (see bn-[4|5|6|7]-1000 in Fig. 6), 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 Fig. 6. 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.[4] For this reason, EDDO performs better on datasets originating from networks with 7 nodes than from networks with only 4 nodes, compared with Bayesian structure learners. This observation is further supported by the plot at the bottom of Fig. 6, where the number of nodes in the Bayesian networks were 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 is 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 h. If this affected only one run, we excluded this run. Otherwise, we removed the Bayesian structure learner.

The results are summarized in Fig. 7. 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 nicely illustrated. ECC shows a substantial improvement over CC, and EWCC in turn shows a further improvement over ECC. We also observe that the variance of $EDDO_{ECC}$ and $EDDO_{EWCC}$ is lower than the variance of most Bayesian structure learners.

8.3 Continuous and mixed densities

In our last experiment, we compare EDO with a state-of-the-art online density estimator for continuous densities,[5] oKDE (see Sect. 3). For EDO, we set its internal buffer size to 100 instances, the maximal number of kernels to 20,000, and the confidence

[4] 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).

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

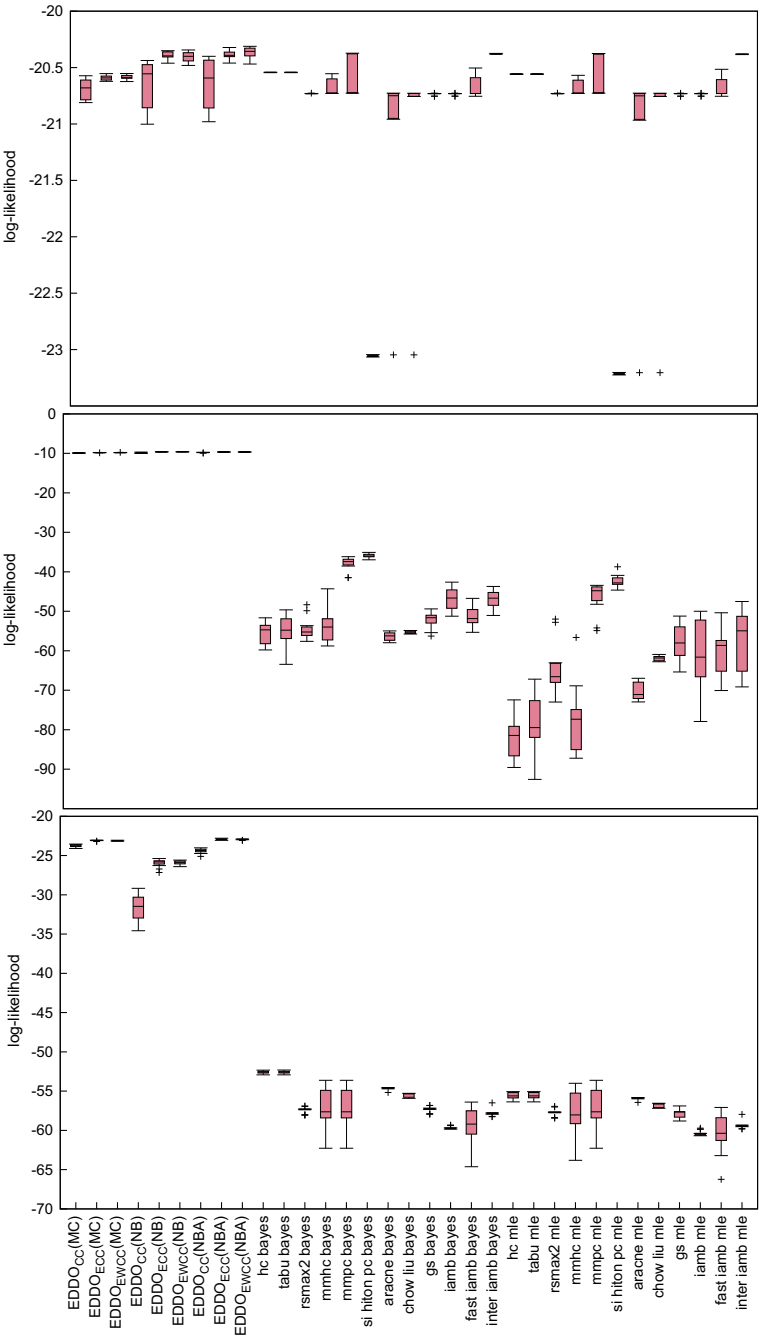

Fig. 7 EDDO density estimators compared to 12 Bayesian structure learners, which use either mle or bayes to estimate the a posteriori probabilities of the CPTs. Each estimator has been trained on the first half of the datasets [(Pokerhand (top), Movielens (middle), and US Census (bottom)]. The average log-likelihood has been measured on the second half

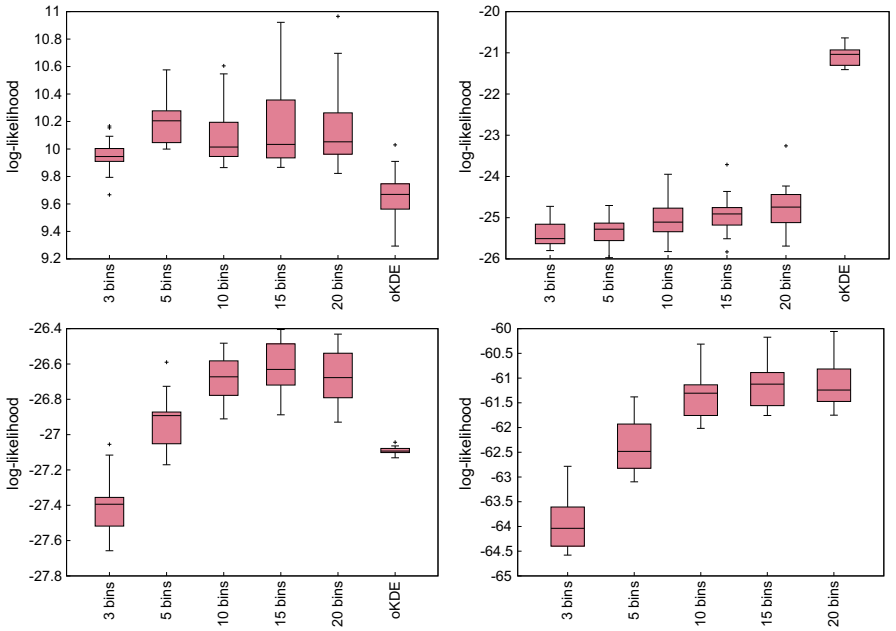

Fig. 8 $EDO_{CC}(MC)$ compared with oKDE on the datasets Electricity (top left), Shuttle (top right), Letter (bottom left), and Covertype (bottom right). On the y-axis is the average log-likelihood (computed prequentially). *i bins* with $i \in \{3, 5, 10, 15, 20\}$ stands for the number of bins used for discretizing the class attribute

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 Electricity, Shuttle, Letter, Adult, and Covertype. Unfortunately, oKDE was not able to complete a single job within 12 h 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 Covertype dataset, the run-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 Fig. 8 and Online Resource 3. 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 performances of the estimators are comparable with 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 runtime constraints or due to internal errors of oKDE.

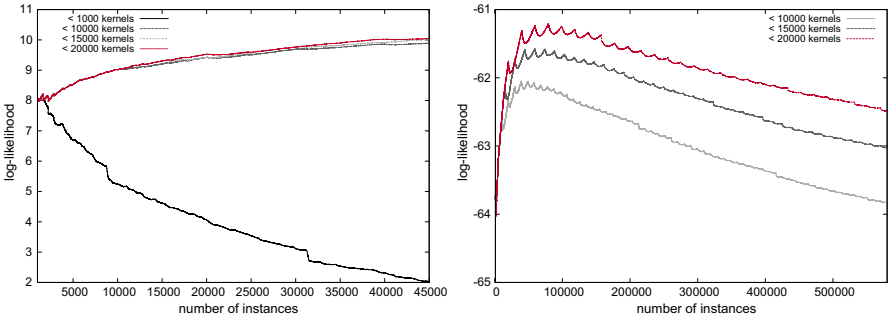

Fig. 9 These plots illustrate the behavior of EDO when different levels of compression are used to reduce the number of kernels. On the left is the electricity dataset, on the right the covertype dataset. $< k$ means that a compression takes place, if there are more than k kernels

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 to large, EDO tends to decrease in its performance 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 range between a couple of instances and 1443 instances.

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 performs a compression as soon as the corresponding limit is reached. Its behavior is illustrated with two plots (see Fig. 9): 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, we observe a substantial decrease on the Covertype dataset whenever a compression takes place. This happens when the number of kernels is too low compared to the possible number of combinations, i.e., the *instance space*. Especially the Covertype dataset with its many attributes has a high-dimensional instances space. 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—even if they are rather unrelated. As a consequence, 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 effects of this proposal are nicely illustrated in Fig. 9 for the Electricity dataset. Whereas a kernel limit of 1000 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.

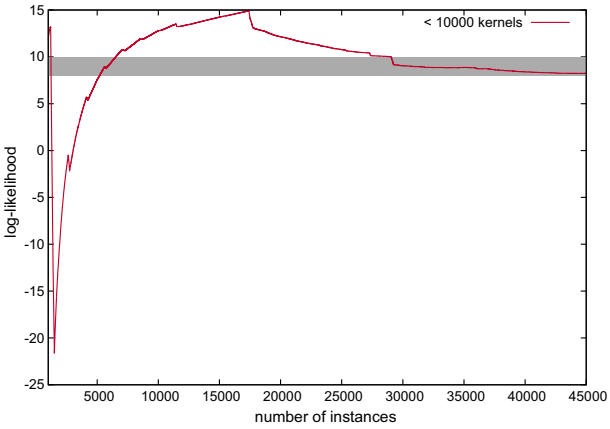

Fig. 10 EDO exploits the assumption that the instance of the stream are IID. What happens if this is not the case is shown at the example of the Electricity dataset. The gray area represents the range in which EDO is performing, if the data is IID

All in all, the proposed density estimator performs well on the selected datasets and can deal with purely continuous datasets as well as with datasets exhibiting discrete and continuous variables. The performance is, however, dependent on the number of maximally allowed kernels, which depends on the underlying instance space.

8.4 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. What happens if this assumption is violated is shown in Fig. 10, 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 is no data distribution drift, the structure of the Hoeffding trees will become more and more fixed after a while, so that only the probability masses in the leaves would be changed. Even if this structure is not an ideal partitioning of the space, the density estimator will still approach the true density according to the law of large numbers. However, this has two disadvantages: (1) the produced trees are possibly substantially larger than necessary and (2) for small numbers of instances, the density estimate is probably very inaccurate (as illustrated in Fig. 10). To avoid these problems and to handle non-IID data streams with EDO, one could model temporal dependencies between the instances using additional features (e.g., Zliobaite et al. 2015) and shuffle instances batch-wise.

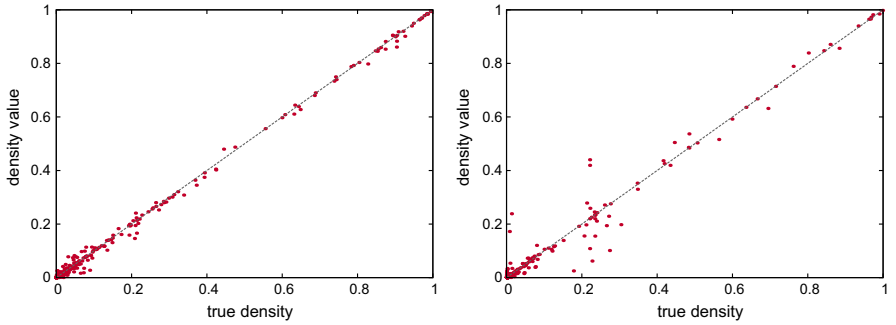

Fig. 11 The plots illustrate the density values of instances with marginalized out variables on the Movielens and the US Census dataset. On the x-axis is the frequency of the instance, and on the y-axis is the estimated density value

8.5 Inference operations

If Algorithms 3, 4, and 5 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 $\mathscr{Z}$ from $\mathscr{X}$ of size s uniformly at random (i.e., the variables $\mathscr{X} \setminus \mathscr{Z}$ will be marginalized out from $\hat{f}$),
3. draw 100 instances from $\hat{f}(Z_1, \ldots, Z_{m'})$ with $Z_i \in \mathscr{Z}$, $1 \leq i \leq m'$,
4. go to Step 1 until 1000 instances have been generated.

Figure 11 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).

9 Conclusions

In this paper, we proposed a family of online algorithms to estimate joint densities with discrete variables, continuous variables, or both. 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 proposed algorithms to perform certain inference tasks. The results of the experiments showed that their performance on synthetic data and real-world data is competitive to offline density estimators in the discrete case and to a state-of-the-art online density estimator in the continuous case.

In addition to the ability to represent joint densities, the estimators enable basic inference tasks that can be combined to perform complex data mining and machine learning tasks (Geilke et al. 2014). In the future, we would like to pursue this direction further and study which tasks can be addressed by operating on density estimates without having access to the original data. It would also be interesting to see whether a general framework can be designed that allows to specify properties of density estimates with respect to specific domains (constraint-based density estimation).

Acknowledgements We would like to thank the editor and the anonymous reviewers for their comments. They improved the presentation, readability, and quality of this paper substantially. We are particularly grateful to the anonymous reviewer who proposed the exponentiated gradient investment strategy for weighting the classifier chains.

References

Bauer E, Kohavi R (1999) An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Mach Learn 36(1–2):105–139

Bifet A, Holmes G, Pfahringer B, Kranen P, Kremer H, Jansen T, Seidl T (2010) MOA: massive online analysis, a framework for stream classification and clustering. J Mach Learn Res Proc Track 11:44–50

Blum A (1996) On-line algorithms in machine learning. In: Proceedings of the workshop on On-line Algorithms, Dagstuhl. Springer, pp 306–325

Buchwald F, Girschick T, Frank E, Kramer S (2010) Fast conditional density estimation for quantitative structure-activity relationships. In: Proceedings of the twenty-fourth AAAI conference on artificial intelligence, pp 1268–1273

Cesa-Bianchi N, Lugosi G (2006) Prediction, learning, and games. Cambridge University Press, Cambridge

Chakraborty S (2008) Some applications of dirac's delta function in statistics for more than one random variable. Appl Appl Math Int J (AAM) 3(1):4254

Cheng MY, Gasser T, Hall P (1999) Nonparametric density estimation under unimodality and monotonicity constraints. J Comput Graph Stat 8(1):1–21

Cover TM, Thomas JA (2006) Elements of information theory, 2nd edn. Wiley, New York

Davies S, Moore AW (2002) Interpolating conditional density trees. In: Uncertainty in artificial intelligence, pp 119–127

Dembczynski K, Cheng W, Hüllermeier E (2010) Bayes optimal multilabel classification via probabilistic classifier chains. In: International conference on machine learning, pp 279–286

Dembczynski K, Waegeman W, Hüllermeier E (2012) An analysis of chaining in multi-label classification. In: Proceedings of the 20th European conference on artificial intelligence (ECAI 2012), pp 294–299

Dembczynski K, Kotlowski W, Waegeman W, Busa-Fekete R, Hüllermeier E (2016) Consistency of probabilistic classifier trees. In: Proceedings of the 2016 European conference on machine learning and knowledge discovery in databases (ECML PKDD 2016), pp 511–526

Domingos P, Hulten G (2000) Mining high-speed data streams. In: Knowledge discovery and data mining, pp 71–80

Elgammal A, Duraiswami R, Davis LS (2003) Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking. IEEE Trans Pattern Anal Mach Intell 25:1499–1504

Frank E, Bouckaert RR (2009) Conditional density estimation with class probability estimators. In: Proceedings of first Asian conference on machine learning, pp 65–81

Frank E, Kramer S (2004) Ensembles of nested dichotomies for multi-class problems. In: Proceedings of the 21st international conference of machine learning, pp 305–312

Friedman N, Goldszmidt M (1996) Learning bayesian networks with local structure. In: Proceedings of the twelfth annual conference on uncertainty in artificial intelligence (UAI '96), pp 252–262
Gama J, Pinto C (2006) Discretization from data streams: applications to histograms and data mining. In: SAC, pp 662–667
Geilke M, Karwath A, Frank E, Kramer S (2013) Online estimation of discrete densities. In: Proceedings of the 13th IEEE international conference on data mining, pp 191–200
Geilke M, Karwath A, Kramer S (2014) A probabilistic condensed representation of data for stream mining. In: Proceedings of the 2014 international conference on data science and advanced analytics (DSAA 2014), IEEE, pp 297–303
Geilke M, Karwath A, Kramer S (2015) Modeling recurrent distributions in streams using possible worlds. In: Proceedings of the 2015 international conference on data science and advanced analytics (DSAA 2015), pp 1–9
Goldberger J, Roweis ST (2004) Hierarchical clustering of a mixture model. Adv Neural Inf Process Syst 17:505–512
Hall P, Presnell B (1999) Density estimation under constraints. J Comput Graph Stat 8(2):259–277
Holmes MP, Gray AG, Isbell CL Jr (2012) Fast nonparametric conditional density estimation. CoRR arXiv:abs/1206.5278
Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Knowledge discovery and data mining, pp 97–106
Hwang JN, Lay SR, Lippman A (1994) Nonparametric multivariate density estimation: a comparative study. IEEE Trans Signal Process 42(10):2795–2810
Kim J, Scott CD (2012) Robust kernel density estimation. J Mach Learn Res 13:2529–2565
Kristan M, Leonardis A (2010) Online discriminative kernel density estimation. In: International conference on pattern recognition, pp 581–584
Kristan M, Leonardis A, Skocaj D (2011) Multivariate online kernel density estimation with gaussian kernels. Pattern Recogn 44(10–11):2630–2642
Kumar A, Vembu S, Menon AK, Elkan C (2013) Beam search algorithms for multilabel learning. Mach Learn 92(1):65–89
Lambert CG, Harrington SE, Harvey CR, Glodjo A (1999) Efficient on-line nonparametric kernel density estimation. Algorithmica 25(1):37–57
Littlestone N (1987) Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Mach Learn 2(4):285–318
Liu H, Lafferty JD, Wasserman LA (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
Mann TP (2006) Numerically stable hidden Markov model implementation. HMM Scaling Tutor, pp 1–8.
Melançon G, Philippe F (2004) Generating connected acyclic digraphs uniformly at random. Inf Process Lett 90(4):209–213
Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press, New York
Peherstorfer B, Pflüger D, Bungartz H (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
Ram P, Gray AG (2011) Density estimation trees. In: Knowledge discovery and data mining, pp 627–635
Rau MM, Seitz S, Brimioulle F, Frank E, Friedrich O, Gruen D, Hoyle B (2015) Accurate photometric redshift probability density estimation—method comparison and application. Monthly Notices R Astron Soc 452(4):3710–3725
Raykar VC, Duraiswami R (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, Pfahringer B, Holmes G, Frank E (2011) Classifier chains for multi-label classification. Mach Learn 85(3):333–359
Scott DW, Sain SR (2004) Multi-dimensional density estimation. Elsevier, Amsterdam, pp 229–263
Scutari M (2010) Learning Bayesian networks with the bnlearn R package. J Stat Softw 35(3):1–22
Sheather SJ, Jones MC (1991) A reliable data-based bandwidth selection method for kernel density estimation. J R Stat Soc Ser B (Methodol) 53(3):683–690
Su J, Zhang H (2006) Full Bayesian network classifiers. In: Proceedings of the twenty-third international conference on machine learning, pp 897–904
Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

Vapnik V, Mukherjee S (1999) Support vector method for multivariate density estimation. In: Neural information processing systems, pp 659–665

Wan R, Wang L (2010) Clustering over evolving data stream with mixed attributes. J Comput Inf Syst 6:1555–1562

Wang X, Wang Y (2015) Nonparametric multivariate density estimation using mixtures. Stat Comput 25(2):349–364

Wied D, Weißbach R (2012) Consistency of the kernel density estimator: a survey. Stat Papers 53(1):1–21

Wu K, Zhang K, Fan W, Edwards A, Yu PS (2014) RS-forest: a rapid density estimator for streaming anomaly detection. In: Proceedings of the 14th international conference on data mining, pp 600–609

Zhou A, Cai Z, Wei L, Qian W (2003) M-kernel merging: towards density estimation over data streams. In: Proceedings of the eighth international conference on database systems for advanced applications, IEEE computer society, pp 285–292

Zliobaite I, Bifet A, Read J, Pfahringer B, Holmes G (2015) Evaluation methods and decision theory for classification of streaming data with temporal dependence. Mach Learn 98(3):455–482