Online Estimation of Discrete Densities

Michael Geilke*, Eibe Frank†, Andreas Karwath* and Stefan Kramer*

* Johannes Gutenberg-Universität Mainz, Staudingerweg 9, 55128 Mainz, Germany
† Department of Computer Science, The University of Waikato, Hamilton 3240, New Zealand
Email: {geilke,karwath,kramer}@informatik.uni-mainz.de, eibe@cs.waikato.ac.nz

***Abstract*—We address the problem of estimating a discrete joint density online, that is, the algorithm is only provided the current example and its current estimate. The proposed online estimator of discrete densities, *EDDO (Estimation of Discrete Densities Online)*, uses classifier chains to model dependencies among features. 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 data sets of up to several million instances: We compare them to density estimates computed from Bayesian structure learners, evaluate them under the influence of noise, measure their ability to deal with concept drift, and measure the run-time performance. Our experiments demonstrate that, even though designed to work online, EDDO delivers estimators of competitive accuracy compared to batch Bayesian structure learners and batch variants of EDDO.**

I. Introduction

Whereas many data mining tasks have received considerable attention in the context of stream mining recently, only little is known about the estimation of joint densities in an online setting. Offline density estimation includes recent work based on decision trees [1], where the leaves contain piecewise constant estimators. A similar approach was pursued by Davies and Moore [2] as part of a conditional density estimator. Vapnik and Mukherjee [3] used SVMs to perform density estimation. Work towards estimation of conditional density estimation has been pursued among others by Holmes *et al.* [4], Frank and Bouckaert [5], and Buchwald *et al.* [6].

Multi-variate densities are frequently estimated using kernel density estimators [7], [8]. Kernel density estimation is also the predominant direction of the few online variants of density estimation so far. For example, Kristan *et al.* [9], [10] proposed a method yielding results that are comparable to corresponding batch approaches. Lambert *et al.* [11] suggest 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., Zhou *et al.* [12] and Elgammal *et al.* [13].

In this paper, we consider the problem of *estimating discrete joint densities online* (notice that we use discrete densities as a synonym for probability mass functions), a challenging task, which requires to estimate the probabilities of a large number of discrete values, the distribution of which may change over time (concept drift). Even if no concept drift would be present, representing the probabilities of these values in a compact way requires sophisticated estimators. If the probabilities were simply stored into a table, e.g., by counting the number of occurrences, even small joint densities with 10 attributes and 10 values for each attribute would already require 10^{10} entries.

The approach we propose is based on so-called classifier chains [14], [15] and uses a set of probabilistic online classifiers to model a discrete joint probability distribution. In this way, one can build on existing work in online learning and hence take advantage of scalable, well-performing algorithms. The classifiers in a chain aim to model the probabilities of a particular feature, and the overall chain aims to model the dependencies among the features, which factors the density into smaller parts and enables to perform inference tasks on it. The individual estimates are combined using the product rule. Because a single classifier chain may not be sufficiently robust, we also provide a variant that uses ensembles of classifier chains and ensembles of weighted classifier chains. We evaluate our density estimators using discrete joint densities that were generated from Bayesian networks as well as on real-world data sets. For each density estimator, we also provide consistency proofs and propose algorithms to perform certain inference tasks.

The main contributions of this paper are as follows:

- We study the estimation of discrete densities in the online setting. The study is worked out in detail both theoretically and experimentally.
- We propose the use of classifier chains (CCs) and ensembles of classifier chains (ECCs) for the estimation of discrete densities. Up to now, CCs and ECCs were studied solely for multi-label classification.
- We propose the use of ensembles of classifier chains in the world of online learning. Up to now, ensembles of classifier chains were studied solely in batch learning.
- The paper introduces 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.
- We present a comprehensive set of experiments investigating the quality of the density estimates generated by our method on (noisy) synthetic data, synthetic data with concept drift, and real-world data sets. They are compared to twelve Bayesian structure learners, which are combined with both maximum likelihood and Bayesian aposteriori to estimate the conditional probability tables. The experiments show that our new approach EDDO (estimation of discrete densities online) produces estimators that are highly competitive with existing Bayesian network structure learners, while working efficiently in the online setting.

The remainder of the paper is organized as follows. In Section II, we describe a method to perform online estimation of discrete joint densities and provide proofs showing that these estimators are consistent as long as their base classifiers are. In Section III, we propose algorithms to perform certain inference tasks on these estimators. This includes queries containing hard evidence and queries that create densities containing specific variables or instances. In Section IV, the experimental setup and the results of six experiments are presented. In these experiments, we compare our density estimators to density estimates computed from Bayesian structure learners, measure the run-time, evaluate the influence of noise, and measure their ability to deal with concept drift. Moreover, we also investigate how the ordering of individual classifier chains affect their performance. Section V concludes the paper.

II. Online Density Estimation

Let $X_1, \ldots, X_n$ be nominal features and $f(X_1, \ldots, X_n)$ be an unknown discrete joint density. In this section, we present algorithms that, given an infinite stream of data that is distributed according to f, determine a density estimate $\hat{f}$ for f. The data stream is processed in an online fashion, that is, an algorithm is only provided the current example and its current density estimate. The estimates that are produced by these algorithms are represented using classifier chains, which not only factor the joint density into smaller parts but also enable application of a broad range of fast and efficient online base classifiers.

A. Classifier Chain

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

$$f(X_1, \ldots, X_n) = f_1(X_1) \cdot \prod_{i=2}^{n} f_i(X_i \mid X_1, \ldots, X_{i-1}). \quad (1)$$

In other words, in order to model the discrete joint density f, it is sufficient to model the density $f_1(X_1)$ and the conditional densities $f_i(X_i \mid X_1, X_2, \ldots, X_{i-1})$, $i \in \{2, \ldots, n\}$. The product over these estimates yields an estimate of f. To model the individual densities f_i, $1 \leq i \leq n$, we employ classifiers that return class probability estimates. For $f_1(X_1)$, we employ a Majority Class classifier, which provides the probability masses of the classes of X_1, and for $f_i(X_i \mid X_1, \ldots, X_{i-1})$, $i \in \{2, \ldots, n\}$, we employ Hoeffding trees [16]. Both classifiers allow us to estimate the density in an online fashion. In case the density changes over time, we employ classifiers that are able to deal with concept drift such as the Concept-adapting Very Fast Decision Tree learner [17].

Based on the classifier chain implied by Equation 1, our algorithm initializes the base classifiers for $f_1, \ldots, f_n$. When the algorithm receives an example $(x_1, \ldots, x_n)$ from an instance stream, it produces n examples, where example i contains the features $X_1, \ldots, X_i$. The example (x_1) is forwarded to the classifier for $f_1(X_1)$ and the example $(x_1, \ldots, x_i)$ is forwarded to the classifier for $f_i(X_i \mid X_1, \ldots, X_{i-1})$, $i \in \{1, \ldots, n\}$. Subsequently, each classifier processes its example and updates its current estimate.

If we need to draw an instance from our density estimate, we simply iterate over the classifiers from $f_1(X_1)$ to $f_n(X_n \mid X_1, \ldots, X_{n-1})$, draw an estimate from each classifier, sample a value based on the distribution obtained, and use the output as input for the next classifier. In a similar fashion, we can also compute the probability of a given instance.

B. Ensembles of Classifier Chains

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

$$f(X_1, \ldots, X_n) = f(X_{m(1)}) \cdot \prod_{i=2}^{n} f(X_{m(i)} \mid X_{m(1)}, \ldots, X_{m(i-1)}). \quad (2)$$

In other words, we simply use a different ordering of the features to represent the discrete joint density, which then results in a different classifier chain. Although all such products represent the same joint density assuming the true conditional density estimates are known, the ordering may be important for the performance of our classifiers: Ideally, the ordering enables the classifiers to exploit conditional independence relationships, so that some of the features can be ignored. 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 (detailed results omitted due to lack of space), we found that doubling and tripling the number of classifier chains (starting with as many classifier chains as features) only yielded negligible improvements. This is in line with other results from the ensemble literature. Bauer and Kohavi [18] and Frank and

Kramer [19], for instance, reported that ensembles of at most 25 respectively 20 ensemble members were sufficient to obtain a solution sufficiently close to the optimum.

Notice that 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. One possibility to sample instances from the estimator is to employ a Metropolis-Hastings algorithm [20]. It requires a candidate-generating density, from which samples are generated, and a criterion to decide whether or not to accept a given sample [21]. With increasing numbers of samples, the chain that results from this process converges to the true density. The instances generated at the beginning are usually discarded, because it takes a few instances until the true density is approached. In order to determine when this is the case, one can use multiple chains and compare the samples from these chains [22], [21]. To apply the Metropolis-Hastings algorithm, it is sufficient to supply a candidate-generating density, the choice of which will affect the rate of convergence and the number of samples that have to be rejected.

C. 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. Although there are many possibilities to measure performance, we are restricted to algorithms that work incrementally and are able to deal with concept drifts. One possibility is the information loss, which will allow us to measure the performance of classifier chains on specific instances. Using this measure, we assign a weight to each classifier chain, which determines the influence of a classifier chain when drawing instances or computing the probability of an instance.

Algorithm 1 Updating the weights of classifier chains.

Require: Let $cc_1, \ldots, cc_k$ be classifier chains. Further, let $w_1, \ldots, w_k \in \mathbb{R}$ be the corresponding weights and $x := (x_1, \ldots, x_n)$ be an instance.

1: Let p_i be the probability of x w.r.t. cc_i
2: $L := (-\log_2 p_1, \ldots, -\log_2 p_k)$
3: $m := \max\{L_i \mid i \in [1;k]\}$
4: **for** $i = 1, \ldots, k$ **do**
5: $\quad w_i := w_i + (1 - \frac{L_i}{m})$
6: **end for**
7: normalize the vector $(w_1, \ldots, w_n)$

Algorithm 1 is based on a heuristic employing the information loss and describes a procedure for updating the weights when new instances arrive. Let $(x_1, \ldots, x_n)$ be an instance. Algorithm 1 iterates over the classifier chains and computes the probability p_i for each classifier chain cc_i, which is the probability of x with respect to cc_i. Then p_i is used to determine the corresponding information loss $(-\log_2 p_i)$, which is used to update the global weights. The weight update is determined by computing the worst information loss value m, normalizing the information loss values by m, and updating the individual weights by adding $(1 - \frac{L_i}{m})$. Consequently, the weights for classifier chains with a large information loss are increased only slightly, whereas the weights for classifier chains with a small information loss are increased more. In the final step of the algorithm, the weights are renormalized, so that they sum to one.

If we have to compute the probability of an instance, then we simply combine the probabilities of the individual classifier chains with respect to their weights. This probability can also be used to sample instances from the ensemble as described in the previous subsection.

D. 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 Kullback-Leibler divergence (KL-divergence) of the density f and its estimate $\hat{f}$ tends to 0 for increasing numbers of instances. To improve readability, we write $KL(f, \hat{f})$ instead of *the KL-divergence of f and $\hat{f}$*.

Proposition 1: Let f be a discrete joint density with $f(x_1, \ldots, x_n) = f_1(x_1) \cdot \ldots \cdot f_n(x_n \mid x_1, \ldots, x_{n-1})$. Further, let $\hat{f}$ be an estimator employing a single classifier chain, and let $\hat{f}_i$ be the estimate of the ith classifier in the classifier chain. If the number of instances tends to infinity and $KL(f_i, \hat{f}_i) \to 0$, for all $i \in [1;n]$, then $KL(f, \hat{f}) \to 0$.

Proof: For increasing numbers of instances, we prove that $KL(f, \hat{f}) \to 0$, if $KL(f_i, \hat{f}_i) \to 0$ for all $i \in [1;n]$:

$$KL(f, \hat{f}) = \sum_{\vec{x}} \hat{f}(\vec{x}) \cdot \ln \frac{\hat{f}(\vec{x})}{f(\vec{x})} \tag{3}$$

For $g \in \{f, \hat{f}\}$, $\ln g(\vec{x})$ can be written as

$$\ln \left(g_1(x_1) \cdot \prod_{i=2}^{n} g_i(x_i \mid x_1, \ldots, x_{i-1})\right),$$

where g_i corresponds to f_i or $\hat{f}_i$. Since $\ln(a \cdot b) = \ln(a) + \ln(b)$, the KL-divergence of f and $\hat{f}$ can be written as

$$KL(f, \hat{f}) = \sum_{\vec{x}} \hat{f}(\vec{x}) \cdot \ln \frac{\hat{f}_1(x_1)}{f_1(x_1)} + \ldots + \sum_{\vec{x}} \hat{f}(\vec{x}) \cdot \ln \frac{\hat{f}_n(x_n \mid x_1, \ldots, x_{n-1})}{f_n(x_n \mid x_1, \ldots, x_{n-1})}.$$

By definition, $\hat{f}(x) = \hat{f}_1(x_1) \cdot \ldots \cdot \hat{f}_n(x_n \mid x_1, \ldots x_{n-1})$ and, therefore, the following equalities hold for individual parts of

the sum

$$\sum_{\vec{x}} \hat{f}(\vec{x}) \cdot \ln \frac{\hat{f}_i(x_i \mid x_1, \ldots, x_{i-1})}{f_i(x_i \mid x_1, \ldots, x_{i-1})}$$
$$= \sum_{\vec{x}} \prod_{j=1}^{n} \hat{f}_j(x_j \mid x_1, \ldots, x_{j-1}) \cdot \ln \frac{\hat{f}_i(x_i \mid x_1, \ldots, x_{i-1})}{f_i(x_i \mid x_1, \ldots, x_{i-1})}$$
$$= \sum_{\vec{x}} \prod_{j=1, j \neq i}^{n} \hat{f}_j(x_j \mid x_1, \ldots, x_{j-1}) \cdot KL(f_i, \hat{f}_i).$$

Hence, if $KL(f_i, \hat{f}_i) \to 0$ for all $i \in [1; n]$, then $KL(f, \hat{f}) \to 0$. ∎

Hence, 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 density of each partition with increasing numbers of instances. Otherwise, the consistency property is fulfilled if there is an $n_0 \in \mathbb{N}$, such that, for each branch,

- there is no further attribute on which the branch can be split,
- no further pruning takes place, and,
- 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 $f(x_1, \ldots, x_n) = f_1(x_1) \cdot \ldots \cdot f_n(x_n \mid x_1, \ldots, x_{n-1})$. 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 jth classifier in the ith classifier chain. If the number of instances tends to infinity and $KL(f_i, \hat{f}_i) \to 0$ for all $i \in [1; k]$ and $j \in [1; n]$, then $KL(f, \hat{f}) \to 0$.

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. ∎

III. Inference

The estimators presented in Section II can be employed for representing discrete joint densities. With more and more instances, the estimate is either approaching the true density or adapting to concept drifts. However, over time, users are probably interested in different parts of the density, which requires infrastructure to pose queries on the joint density. In this section, we want to demonstrate that the proposed estimators produce estimates that enable such tasks. We consider the following three settings:

1) Let f be a discrete joint density defined on variables $X := \{X_1, \ldots, X_n\}$. Further, let $Y := \{Y_1, \ldots, Y_j\} \subset X$ and $Z := \{Z_1, \ldots, Z_k\} \subset X$ be subsets with $Y \cap Z = \emptyset$. Then we can determine a new density
$$f'(Z_1, Z_2, \ldots, Z_k \mid Y_1 = y_1, \ldots, Y_j = y_j), \quad (4)$$
where y_i is a specific value of Y_i, $i \in [1; k]$.
2) Let f, X, and Z be as in Setting 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_l with probability p_l. Then we determine a new density
$$f'(Z_1, Z_2, \ldots, Z_j \mid Y'), \quad (5)$$
where Y' takes the value y_i with probability p_i, $i \in [1; l]$.
3) Let f be as in Setting 1. and θ be a user-defined value between 0 and 1. Then we determine a new density f', such that, for $T = \sum_{\vec{x}': f(\vec{x}') \geq \theta} f(\vec{x}')$, $f'(\vec{x})$ equals $\frac{f(x)}{T}$ for all instances $\vec{x}$ with $f(\vec{x}) \geq \theta$, and $f'(\vec{x}) = 0$ for all other instances.

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

In the following, we provide algorithms for these settings. However, due to space constraints, we are not able to present our algorithm for Setting 2, which incorporates the given soft evidence into classifier chains employing Hoeffding trees.

A. Hard Evidence

In order to compute the expression (4), there are two main tasks that need to be performed: First, setting the variables Y_i to specific values, and, second, marginalizing out variables that are not contained in Z. In the context of Bayesian networks, both tasks are usually performed by operating on conditional probability tables. Here, we want to perform these tasks on the estimator presented in Section II.

If Hoeffding trees are employed or other classifiers that built on a tree structure, we can simply disable or remove all branches of variable Y_i that correspond to a value that is not equal to y_i, thereby providing a structure that is very similar to the original one. However, when it comes to marginalizing out certain variables, there are only a few special cases in which the current structure can be used. The problem lies with the children of the variable to be marginalized out. For example, if there is a variable $X_i \in X$ with $X_i \notin Y, Z$ where the set of descendants of X_i for its value v_1 is D_1 and the set of descendants 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 we can merge the branches, 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 counters for the classifier's target variable but not the underlying instances.

Algorithm 2 Processes query of Setting 1.

Require: chain cc, sets of variables Y and Z, target performance $avgLL$, step size s

```
 1: initialize result classifier chain result
 2: p := avgLL + 1
 3: while p > avgLL do
 4:    sample := ∅
 5:    counter := 0
 6:    while counter < s do
 7:       inst := sample instance from cc
 8:       if Y_i = y_i for all Y_i in inst, i ∈ [1; k] then
 9:          sample := sample ∪ inst
10:          counter := counter + 1
11:       end if
12:    end while
13:    for inst in sample do
14:       for Z_i ∈ Z do
15:          remove Z_i from inst
16:       end for
17:    end for
18:    result := forward instances from sample to result
19:    p := 0
20:    for inst ∈ sample do
21:       p := p + log(result(inst))
22:    end for
23:    p := p/s
24: end while
25: return result
```

Therefore, we propose an approach that is independent of the underlying classifiers (cp. Algorithm 2): First, we sample s instances from the current estimate, such that all instances match the values of the variables in Y (line 4 through 12). Since we have to skip all sampled instances from cc that do not fulfill this requirement, we use a counter to ensure that we sample s instances as required. Then the algorithm removes all features that are not contained in Z from the instances (line 13 through 17) and forwards them to the target estimator (line 18). Finally, in lines 19 through 23, the average log-likelihood of $sample$ is computed.

The question is: How many instances do we need to obtain a reasonably good estimate? To solve this problem, we measure the performance of the new density estimate every s instances and compare it to $avgLL$. Both are either some user-defined parameters or values that are derived from the current estimate (e.g., to obtain a value for $avgLL$, one could compute the average log-likelihood for the original estimate as starting point and decrease the resulting value according to some heuristics). Each time we forwarded another s instances to the density estimators, we take the next s instances to measure the average log-likelihood. If a user-defined value is underrun, we stop. Otherwise, these instances are forwarded to the density estimator.

Algorithm 2 is easily extended to ensembles of (weighted) classifier chains by adding an outer loop that iterates over all classifier chains.

B. Instances Exceeding a Given Probability

Users may be particularly interested in very likely instances. In this case, we need to infer a distribution from our current estimate that only contains instances exceeding a certain probability θ. Similarly to Algorithm 2, we do the following:

1) Draw instances from the current discrete joint density.
2) Check whether this instance exceeds θ. If yes, use this instance to train the new target distribution. Otherwise, ignore this instance and go to 1.

Again, we have to specify a stopping criterion for this procedure that guarantees a certain quality of the resulting density estimate. As in Setting 1, we can employ the average log-likelihood to measure the performance. Additionally, we can also compute the number of instances that are necessary to include instances of probability $\theta_1 + \epsilon$ with ϵ being a small, positive number close to 0. Given a confidence level $0 < \theta_2 < 1$ and a $\lambda > 0$, we can apply the Chernoff bound [23] and compute the minimal n, such that

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

where $\mu = n \cdot (\theta_1 + \epsilon)$ and and X is the sum of independent Poisson trials with $Pr(X = 0) = 1 - \theta_1 - \epsilon$, $Pr(X = 1) = \theta_1 + \epsilon$.

IV. Experiments

In this section, we evaluate the algorithms presented in Section II on synthetic and real-world data. They have been implemented in the MOA framework (version 20120301) [24]. In order to compare the performance of the online density estimator to existing density estimators, we integrated Bayesian structure learners from the `bnlearn` package [25] into our setting. 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), which are estimated by the `fit` method of the Bayesian network. For the `fit` method two alternatives are provided: maximum likelihood (mle) and Bayesian aposteriori (bayes).

The Bayesian structure learners from the `bnlearn` package are offline algorithms, which makes it difficult to compare them directly with our online density estimator. If the performance differs in favor of the Bayesian structure learners, it is

still possible that the same estimators employing offline base classifier perform better than the Bayesian structure learners. Therefore, we also implemented our density estimator in the WEKA framework (version 3.7.6) [26].

To improve readability, we introduce the following notation: $EDDO_T(L)$ and $EDDB_T$, where $T \in \{CC, ECC, EWCC\}$, and $L \in \{Maj, NB, NBA\}$. $EDDO_T(L)$ represents online density estimators employing HoeffdingTree and $EDDB$, where the *B* stands for batch, represents offline density estimators employing REPTree. 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 (Maj), NaiveBayes (NB), or NaiveBayes adaptive (NBA).

In order to compare the density estimates, we used the KL-divergence for synthetic data and the log-likelihood (LL) for real-world data. 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 estimator. 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 [27]. In case of real-world data, we divided the data set into training and test set, forwarded the instances from the training set to the estimators, and computed the log-likelihood on the test set.

A. Noise-free Synthetic Data

In the first experiment, we randomly generated Bayesian networks 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 [28]. For the experiment, we generated Bayesian networks with between 4 and 8 nodes, for each of which 100 networks were considered. From these discrete joint densities, we drew M instances with M being one of the values of $\{10^3, 10^4, 10^5, 10^6\}$. The instances were forwarded to the Bayesian structure learners and our online and offline density estimators.

In order to add the conditional probability tables to the learned graph structure, all Bayesian structure learners were combined with the options maximum likelihood (mle) and Bayesian aposteriori (bayes), which resulted in 24 estimators. However, for reasons of readability, we only compare our density estimators to the Bayesian structure learner with smallest KL-divergence, which we determined by ranking them according to their KL-divergence for each Bayesian network and each M, computing the total rank sum for each estimator, and choosing the one with smallest value.

For our online and offline density estimators, we considered several variants: one with a single random classifier chain, one with an ensemble of classifier chains, and one with an ensemble of weighted classifier chains. For the ensembles, we chose a number of chains that is linear in the number of features, which in this experiment are as many classifier chains as features. As base classifiers, we considered REPTree and Hoeffding trees with leaf classifiers MajorityClass (Maj), NaiveBayes (NB), and NaiveBayes adaptive (NBAdaptive). In all cases, we considered these classifiers with and without pruning disabled. (Notice that the MOA implementation of Hoeffding trees handles pre-pruning differently. For a split, it simply adds an additional node, which is basically a *null model*, to the list of possible candidates and computes the information gain among them.)

TABLE I. THE TABLE SHOWS THE RANK SUM (LOWER IS BETTER) OF THE BEST BAYESIAN STRUCTURE LEARNER AND THE ESTIMATORS $EDDO$ AND $EDDB$ WHERE PRUNING WAS NOT DISABLED. IT HAS BEEN COMPUTED BY RANKING THE ESTIMATOR FOR EACH BAYESIAN NETWORK AND EACH M AND SUMMING UP THEIR RANKS. THE NUMBER IN PARENTHESES IS THE ESTIMATOR'S RANK WITH RESPECT TO THE GIVEN M. BAYESIAN NETWORKS WERE GENERATED WITH BETWEEN 4 AND 8 NODES, FOR EACH OF WHICH 100 NETWORKS WERE CONSIDERED. M IS THE NUMBER OF INSTANCES THAT WERE DRAWN FROM THE NETWORKS. FOR *ECC* AND *EWCC*, WE USED AS MANY RANDOM CLASSIFIER CHAINS AS NUMBER OF FEATURES. *HC greedy bayes* IS THE HILL-CLIMBING GREEDY SEARCH ALGORITHM EMPLOYING THE *bayes* METHOD TO ESTIMATE THE CONDITIONAL PROBABILITY TABLES.

Estimator	$M = 10^3$	$M = 10^4$	$M = 10^5$	$M = 10^6$
$EDDB_{CC}$	3183 (05)	2519 (04)	2290 (04)	2624 (05)
$EDDB_{ECC}$	1686 (03)	1122 (01)	1234 (01)	1716 (02)
$EDDO_{CC}(Maj)$	4936 (12)	4997 (12)	4504 (11)	4133 (09)
$EDDO_{ECC}(Maj)$	4390 (09)	4806 (11)	4311 (10)	3768 (08)
$EDDO_{EWCC}(Maj)$	3397 (06)	3840 (08)	2942 (05)	2121 (03)
$EDDO_{CC}(NB)$	4477 (10)	4248 (09)	4793 (12)	4791 (11)
$EDDO_{ECC}(NB)$	4010 (08)	3489 (06)	3697 (07)	3702 (07)
$EDDO_{EWCC}(NB)$	1620 (02)	1900 (03)	2027 (03)	2231 (04)
$EDDO_{CC}(NBA)$	3528 (07)	3592 (07)	4167 (08)	4179 (10)
$EDDO_{ECC}(NBA)$	2486 (04)	2706 (05)	3150 (06)	3208 (06)
$EDDO_{EWCC}(NBA)$	740 (01)	1483 (02)	1655 (02)	1652 (01)
HC greedy bayes	4547 (11)	4298 (10)	4230 (09)	4875 (12)

As in the case of the Bayesian structure learners, we compare the density estimators based on their rank sum, which has been computed from the rankings that include the best Bayesian structure learner and our density estimators either with or without pruning disabled. Since disabling pruning had only minor effects on the results, we did not include results with no pruning here. The only noteworthy difference is the performance of the offline density estimators. If pruning is disabled, they receive worse ranks.

The results are summarized in Table I and visualized in Figure 1. Surprisingly, the Bayesian structure learner *HC greedy search with bayes* performs worse than our offline density estimator and most online density estimators. For further investigation, we took a closer look at the individual runs and counted how many times an estimator is ranked best. For $M = 10^3$ and $M = 10^4$, the Bayesian structure learner receives the second largest number, and, for $M = 10^5$ and $M = 10^6$, it received the largest number. This means that *HC greedy search with bayes* performs pretty well on many Bayesian networks, but at the same time poorly on many others. These differences can be reduced by restarting the hill climber several times, thereby decreasing the likelihood of getting stuck in a local optimum. However, with every restart,

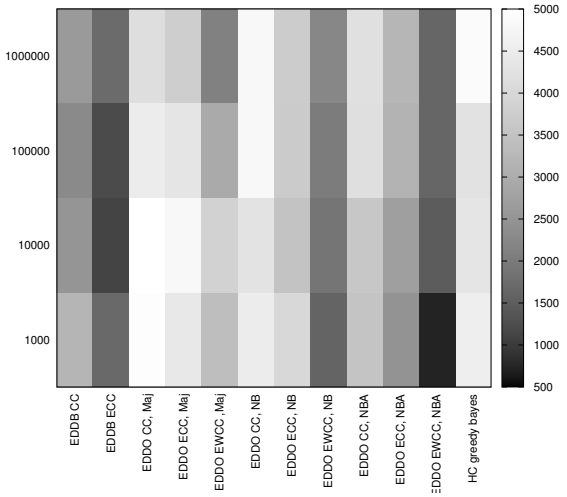

Fig. 1. The figure visualizes the rank sums of Table I by a heat map (lower is better). The estimators are on the x-axis (with slightly different notation), and the number of instances is on the y-axis. Low rank sums are represented by darker colors and high rank sums by lighter colors.

the run-time of the algorithm is also increased.

Next, we compare $EDDB$ to $EDDO(Maj)$. For all M, classifier chains and ensembles of classifier chains employing REPTree have both a lower rank sum than their counterparts employing HoeffdingTree with Maj. Likewise, ensembles of weighted classifier chains cannot match the performance of the offline density estimators. Only in the case $M = 10^6$, the $EDDO_{EWCC}(Maj)$ places itself between $EDDB_{CC}$ and $EDDB_{ECC}$. To get an idea how the KL-divergence values for estimators of type $EDDB$ and $EDDO$ differ in general, we also computed the average KL-divergence for both of them in dependence of the number of instances. For $EDDB$, we observed an average value of 0.094 for 10^3 instances and 0.007 for 10^6 instances. For $EDDO$, we observed 0.463 and 0.059, respectively.

If we compare the different types of estimators, we observe that in all cases $EWCC$ performs better than ECC, and ECC performs better than CC. Considering $EDDO(NB)$ and $EDDO(NBA)$, we also notice remarkable improvements when ensembles of weighted classifier chains are employed. In particular, $EDDO_{EWCC}(NBA)$ has either the lowest or second lowest rank sum among all density estimators. The only estimator that performs better for $M = 10^4$ and $M = 10^5$ is $EDDB_{ECC}$. In order to measure the improvements that are achieved by the individual estimator types, we also computed the average improvement if ECC is used instead of CC and $EWCC$ is used instead of ECC. In all cases, the former yields larger improvements than the latter. Dependent on the underlying base classifiers, we also observe differences in the increase of the improvements. For a node count of $N = 5$, the KL-divergence of $EDDO_{ECC}(Maj)$ is on average 25.5 percent smaller than the KL-divergence of $EDDO_{CC}(Maj)$, and another 27.1 percent smaller if $EWCC$ is used instead of ECC. In contrast to that, employing NBA as base classifier yields an improvement of only 4.6 and 24.8 percent for the corresponding cases. For $N = 6$, we observe almost the converse of that, that is, $EDDO(NBA)$ yields larger improvements than $EDDO(Maj)$.

All in all, this experiment shows that the online density estimators have a performance that is competitive to the offline density estimators and that using ensembles of (weighted) classifier chains is in general beneficial.

B. Run-time

In an online setting where we expect millions of instances each day, it is not only important to provide a good estimate, but that the estimator is able process many instances per second. Therefore, we also measured how many instances can be processed by each estimator per second. We considered three settings where the number of instances to be processed was set to $2 \cdot 10^5$, $2 \cdot 10^6$, and $2 \cdot 10^7$. The instances were generated from 10 randomly generated Bayesian networks with 8 nodes, each of which has a cardinality between 4 and 8. To ensure equal conditions, the experiment has been conducted on a machine with two 3.2 GHz Intel® Xeon® CPUs of type W3565 (4 cores per CPU). For each estimator we restricted the maximum amount of RAM to 2 GB.

TABLE II. THE TABLE SHOWS HOW MANY INSTANCES ARE PROCESSED BY EACH ESTIMATOR PER SECOND. WE CONSIDERED THREE SETTINGS WHERE THE NUMBER OF INSTANCES TO BE PROCESSED, WHICH IS DENOTED BY M, WAS SET TO $2 \cdot 10^5$, $2 \cdot 10^6$, AND $2 \cdot 10^7$. THE VALUES IN THE CELLS ARE THE NUMBER OF INSTANCES THAT HAVE BEEN PROCESSED IN ONE SECOND (THE VALUES ARE ROUNDED), WHERE A '-' MEANS THAT THE ESTIMATOR RAN OUT OF MEMORY.

Estimator	M		
	$2 \cdot 10^5$	$2 \cdot 10^6$	$2 \cdot 10^7$
$EDDO_{CC}$(Maj)	120,894	98,740	48,770
$EDDO_{ECC}$(Maj)	17,703	12,193	5,208
$EDDO_{EWCC}$(Maj)	13,525	8,297	2,523
$EDDO_{CC}$(NB)	120,483	96,455	47,175
$EDDO_{ECC}$(NB)	17,853	12,215	5,216
$EDDO_{EWCC}$(NB)	12,633	8,141	2,485
$EDDO_{CC}$(NBA)	78,196	59,665	33,778
$EDDO_{ECC}$(NBA)	10,667	6,851	3,465
$EDDO_{EWCC}$(NBA)	6,583	4,549	1,862
Chow-Liu mle	71,216	96,422	-
Inter-IAMB mle	35,210	37,184	-
SIHPC mle	37,501	35,884	-

The results are summarized in Table II. For reasons of readability, we only included the best (*Chow-Liu with mle*) and worst Bayesian estimators (*Inter-IAMB with mle* and *SIHPC with mle*). In almost all cases, $EDDO_{CC}$ is faster than the Bayesian estimators (the only exception is $EDDO_{CC}$ with NBA as leaf classifier). Estimators employing an ensemble of (weighted) classifier chains, on the other hand, are substantially slower than the Bayesian estimators if a small amount of instances has to be processed.

The difference between $EDDO_{CC}$ and the ensemble estimators can be explained by the number of classifiers that are involved in the estimate. $EDDO_{ECC}$ and $EDDO_{EWCC}$ use eight times as many classifiers as $EDDO_{CC}$, and since the estimators have not been optimized for a multi-core architecture, the classifiers are mostly updated sequentially. Adapting $EDDO_{ECC}$ and $EDDO_{EWCC}$ to a multi-core architecture should provide some remarkable improvements, since all classifiers can be trained independently. Moreover, $EDDO_{EWCC}$ can further be improved by computing the weights every l instances instead of every instance (see Algorithm 1). The

computation of the weights is the only difference between $EDDO_{ECC}$ and $EDDO_{EWCC}$.

For all chain-based estimators, the number of instances that is processed within one second decreases with increasing numbers of instances. Judging from the memory consumption of the runs, the estimators are still growing after $2 \cdot 10^6$ instances. This decreases the processing speed of the estimator, as the underlying classifiers become more complex, thereby making it more expensive to incorporate new instances. If processing speed is more important than the accuracy of the estimators, the growth of the classifiers could be restricted, leading to a faster and overall more constant processing time.

Considering different leave classifiers, we observe only minor differences between Maj and NB, whereas NBA can only process between 40 % and 70 % of the instances processed by Maj or NB per second.

Summarizing the results from the first and this experiment, we can conclude that $EDDO_{EWCC}$ provides the best estimate at the expense of run-time. Hence, depending on the number of instances that need to be processed in the application domain at hand, the user can choose between a better estimate or a faster run-time.

C. Influence of Chain Orderings

In the first experiment, we observed that employing an ensemble of (weighted) classifier chains yields a better performance than an estimator employing a single classifier chain. To investigate the difference between individual classifier chains, we conducted another experiment: For a node count between 4 and 7 and 10 Bayesian networks for each node count, we created density estimators employing a single classifier chain from all possible feature orderings, forwarded 10^5 instances from the Bayesian networks to these estimators, and computed their KL-divergence. Note that we chose 10^5 instances, since the differences between the classifier chains disappear with increasing numbers of instances (a consequence of the consistency property of the estimator). As in the previous experiment, we ran this experiment with and without pruning disabled. Since we did not observe any major differences, we only discuss the runs where pruning was not disabled.

For each node count N and each Bayesian network, we collected the KL-divergence of all classifier chains, and sorted them with respect to their KL-divergence. Then we determined the smallest and largest KL-divergence that we observed for a particular Bayesian network, and divided the corresponding interval into 5 segments of equal length. Dependent on their KL-divergence value, we assigned each classifier chain to one of five segments, such that the value is between the lower and upper border of the segment.

First of all, we determined the percentage of classifier chains that is contained in the segment with smallest KL-divergence values. For all N and all base classifiers, the percentage is surprisingly large. However, with increasing N, the percentage is gradually getting smaller. For $N = 4$, the smallest value we observe is 33% and, for $N = 7$, the smallest value is 21%. Next, we determined the percentaged difference between the classifier chain with largest KL-divergence and the one with smallest KL-divergence, which is computed with respect to the larger value. On average, the difference between the best and worst classifier chain lies between 54% and 86%, which shows that the choice of the ordering is very important for the performance of the classifier chain. Since finding the best ordering is not an easy task, we proposed ensembles of weighted classifier chain to increase the weight of those chains that performed well on previous instances. However, an ensemble can only use the classifier chains that were chosen randomly at the beginning. So a high likelihood of choosing a classifier chain from the first segment increases the chances of having a good chain in the ensemble of classifier chains, which then enables the ensemble of weighted classifier chains to increase the weight of this good classifier chain. In our experiment, the smallest percentage of such classifier chains was 17% for $N = 6$. Hence, the likelihood of choosing a classifier chain from the first segment is 0.67. The largest percentage is 47% ($N = 4$) resulting in a probability of 0.92. Both values are satisfactory, but experiments with larger node counts are required to make further statements.

D. Noisy Synthetic Data

In the first experiment, the density estimators were directly created from the instances that were generated by the Bayesian networks. However, especially in real-world applications, we have to assume a certain degree of noise in the data, which means that the instances are partly modified before reaching the density estimators. To measure their ability to deal with such data, we repeated the first experiment with different noise levels. The noise level is the probability that the value of an attribute is replaced by another value of this attribute, which we draw uniformly at random. We consider five different noise levels: $0.1, 0.2, 0.3, 0.4, 0.5$. So, if we have a noise level of 0.1 and an instance with 8 attributes, where each attribute has 5 values, then with probability $0.92^8 \approx 0.5132$ the instance remains unchanged.

For reasons of readability, we reduced the number of runs of this experiment and generated 10^6 instances from each density, discarding the options 10^3, 10^4, and 10^5. As in previous subsections, we ran the experiment with and without pruning disabled. However, for every combination of base classifier and noise level, disabling pruning yielded worse results. Therefore, we only included the results where pruning was not disabled. The best Bayesian structure learner was the *HC greedy* algorithm again, but this time the CPTs were estimated by maximum likelihood estimation.

The results are summarized in Table III, which contains the average KL-divergence for each base classifier and each noise level. Density estimators employing classifier chains have an average KL-divergence of between 0.284 and 0.406 for the noise level 0.1, whereas *HC greedy search with mle* already has 0.860. Then, with increasing noise level, the KL-divergence steadily increases and reaches values between 1.072 and 1.150 (noise level 0.5). The KL-divergence of *HC greedy search with mle* increases at a slower pace, but reaches an average KL-divergence of 1.263 in the end. In all cases, there are no remarkable jumps from one noise level to another. The differences between the noise levels appear quite regular.

For real-world applications, noise levels of 0.1 and 0.2 are perhaps more likely than 0.3, 0.4, 0.5, and density estimators

TABLE III. FOR EACH COMBINATION OF BASE CLASSIFIER AND NOISE LEVEL, THE TABLE SHOWS THE AVERAGE KL-DIVERGENCE OVER ALL BAYESIAN NETWORKS. BAYESIAN NETWORKS WERE GENERATED WITH BETWEEN 4 AND 8 NODES, FOR EACH OF WHICH 100 NETWORKS WERE CONSIDERED. M IS THE NUMBER OF INSTANCES THAT WERE DRAWN FROM THE NETWORKS. FOR *ECC* AND *EWCC*, WE USED AS MANY RANDOM CLASSIFIER CHAINS AS NUMBER OF FEATURES. *HC greedy search mle* IS THE HILL-CLIMBING GREEDY SEARCH ALGORITHM EMPLOYING THE MAXIMUM LIKELIHOOD METHOD TO ESTIMATE THE CPTS.

Estimator	noise level				
	0.1	0.2	0.3	0.4	0.5
$EDDB_{CC}$	0.309	0.535	0.758	0.965	1.150
$EDDB_{ECC}$	0.284	0.512	0.737	0.945	1.130
$EDDO_{CC}$(Maj)	0.406	0.607	0.784	0.952	1.112
$EDDO_{ECC}$(Maj)	0.382	0.591	0.771	0.943	1.105
$EDDO_{EWCC}$(Maj)	0.334	0.554	0.748	0.928	1.097
$EDDO_{CC}$(NB)	0.341	0.527	0.731	0.926	1.111
$EDDO_{ECC}$(NB)	0.291	0.488	0.695	0.896	1.082
$EDDO_{EWCC}$(NB)	0.260	0.464	0.678	0.883	1.072
$EDDO_{CC}$(NBA)	0.346	0.541	0.748	0.941	1.116
$EDDO_{ECC}$(NBA)	0.307	0.513	0.722	0.919	1.095
$EDDO_{EWCC}$(NBA)	0.274	0.486	0.702	0.904	1.086
HC greedy mle	0.860	0.925	1.037	1.147	1.263

employing classifier chains show a significantly better performance than *HC greedy search with mle* for these noise levels. In particular, $EDDO_{EWCC}(NB)$ has average KL-divergence values that are less than half of the corresponding values of *HC greedy search with mle*.

E. Concept Drift

Another important aspect of density estimation are streams of drifting distributions. Usually, the instance stream does not come from a stationary distribution but changes over time. The MOA framework provides several classifiers that are able to deal with concept drifts. One such classifier is `HoeffdingTree` employing the leaf classifier `NaiveBayes Adaptive`, which is also the classifier we considered for this experiment. The instance stream is constructed by MOA's `ConceptDriftStream` class. It accepts two streams and some parameters specifying a window in which it slowly changes from the first stream to the second one. The window of the `ConceptDriftStream` was set to $25,000$, $50,000$, and $75,000$ instances and the central position of this window to $250,000$. To simulate a concept drift, we randomly picked two Bayesian networks having the same number of nodes, where each node has cardinality 5. Afterwards, we forwarded the instances to our density estimator, measured the KL-divergence before the probability of switching to the second Bayesian network is larger than or equal to 0.1, and counted how many instances are needed until the same KL-divergence is reached again. Since the rate of convergence decreases when approaching 0, we also stopped when the KL-divergence dropped below 0.1. The KL-divergence was measured every $100,000$ instances.

The results are summarized in Table IV. The differences between the given window sizes are more prominent for $EDDO_{CC}(NBA)$ and $EDDO_{ECC}(NBA)$, where they vary from $29,600$ to $185,200$ instances. In both cases, the minimal number of instances is needed for the window size $50,000$. For $EDDO_{EWCC}(NBA)$, on the other hand, the differences

TABLE IV. FOR EACH COMBINATION OF DENSITY ESTIMATOR AND WINDOW SIZE, THE TABLE SHOWS THE AVERAGE NUMBER OF INSTANCE THAT WERE NEEDED TO TRAIN THE ESTIMATOR WITH INSTANCES FROM THE FIRST NETWORK AND THEN REACHING THE SAME KL-DIVERGENCE BEFORE THE PROBABILITY OF A CONCEPT DRIFT EXCEEDED 0.1 OR THE KL-DIVERGENCE DROPPED BELOW 0.1. BAYESIAN NETWORKS WERE GENERATED WITH BETWEEN 4 AND 8 NODES, FOR EACH OF WHICH 100 NETWORKS WERE CONSIDERED. M IS THE NUMBER OF INSTANCES THAT WERE DRAWN FROM THE NETWORKS. FOR *ECC* AND *EWCC*, WE USED AS MANY RANDOM CLASSIFIER CHAINS AS NUMBER OF FEATURES.

Estimator	window size		
	$25,000$	$50,000$	$75,000$
$EDDO_{CC}(NBA)$	$992,400$	$807,200$	$836,800$
$EDDO_{ECC}(NBA)$	$920,000$	$769,600$	$799,800$
$EDDO_{EWCC}(NBA)$	$944,000$	$928,800$	$924,000$

are lower and is decreasing with increasing window sizes. In general, ECC performs best and $EWCC$ worst.

For all density estimators, less than 10^6 instances are required until the KL-divergence dropped below 0.1 or the KL-divergence of the first Bayesian network. However, for a few pairs of Bayesian networks, the density estimator needs more than 10^7 instances. This usually happens if the first network is easier to estimate than the second one. In one particular example, there are two Bayesian networks each of which exhibiting 8 nodes. After $250,000$ instances, the density estimator reaches a KL-divergence of 0.210 for the first network and a KL-divergence of 0.567 for the second one. In our experiment, the density estimator requires $10,700,000$ instances for the same pair of Bayesian networks. But after $1,321,281$ instances it reached a KL-divergence of 0.491, which is already less than the KL-divergence we computed for the second network after $250,000$ instances.

In summary, the density estimators require on average between two and three times as many instances as they would when learning a stationary distribution. This also includes cases where the second network is harder to estimate, which has a larger influence on the number of instances than having a pair of networks where the first one is harder to estimate.

F. Real-World Data

Finally, we tested the online density estimators on two real-world data sets from the UCI repository: the US census data set[1], which is relatively large with $2,458,285$ instances and 68 categorical attributes, and the Covertype data set[2], which has $581,012$ instances and 54 attributes. Since the Covertype data set contains 10 integer valued attributes, we discretized the data as follows: attributes measured in meters are divided in segments of 500 meters, attributes measured in degrees are divided in segments of 60 degrees, and attributes measuring the *Hillshade index* are divided in segments of size 32.

For comparison, we considered the Bayesian structure learners from the first experiment. Since some of them either required more than 15 hours of computing time or were not able to represent the Bayesian network due to large CPTs, we did not include these Bayesian structure learner in the results. Among the remaining Bayesian structure learners, we

[1] `http://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990)`

[2] `http://archive.ics.uci.edu/ml/datasets/Covertype`

selected the best one for each data set. The performance of the estimators was measured with the average log-likelihood. The first half of the instances were forwarded to the density estimators, and the remaining instances were used to compute the average log-likelihood. Contrary to the first experiment, we only allowed 10 classifier chains for each ensemble. Using the number of attributes instead, the ensembles would consist of 68 and 54 classifier chains respectively, which would increase the memory consumption significantly (almost a factor of 7).

TABLE V. THE TABLE SHOWS THE AVERAGE LOG-LIKELIHOOD OF THE DENSITY ESTIMATORS FOR TWO REAL-WORLD DATA SETS: US CENSUS AND COVERTYPE. FOR *ECC* AND *EWCC*, WE USED 10 RANDOM CLASSIFIER CHAINS. *mle* IS THE MAXIMUM LIKELIHOOD ESTIMATION METHOD, WHICH IS USED TO ESTIMATE THE CPTS.

Estimator	US Census	Covertype
$EDDO_{CC}(Maj)$	-23.5	-13.2
$EDDO_{ECC}(Maj)$	-23.0	-12.9
$EDDO_{EWCC}(Maj)$	-23.0	-12.9
$EDDO_{CC}(NB)$	-28.6	-12.4
$EDDO_{ECC}(NB)$	-25.1	-11.8
$EDDO_{EWCC}(NB)$	-25.1	-11.8
$EDDO_{CC}(NBA)$	-23.8	-12.4
$EDDO_{ECC}(NBA)$	-22.9	-11.9
$EDDO_{EWCC}(NBA)$	-22.9	-12.0
TABU mle	-45.3	-114.3
RSMAX2 mle	-55.5	-43.5

The results are summarized in Table V. On the US Census data set, the Bayesian structure learner *TABU with mle* has the smallest average log-likelihood among all density estimators, and, on the Covertype data set, *RSMAX2 with mle* has the smallest value, which in both cases is substantially smaller than the values of the online density estimators. The Covertype data set seems to be particularly difficult to estimate. The Bayesian networks that were learned for this data set contain several CPTs in which many rows contain almost the same probability (e.g., five times 0.2 or eight times 0.125). Moreover, the runtime of the Bayesian structure learners depends heavily on the data set from which the instances are drawn. Whereas most learners were able to process the US Census data set within minutes, on the Covertype data set, some of them needed many hours or were not able to process the data set at all.

For the online density estimators, we observe only small differences between the various base classifiers. NB seems to yield the best performance and NBA is either on the same level as Maj or slightly better. Similarly, differences between the ensemble methods are also rather small. As in the first experiment, the ensemble method still performs better than using a single classifier chain. However, there is very little difference between ECC and $EWCC$.

V. CONCLUSIONS

In this paper, we proposed three online algorithms to estimate discrete joint densities: 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 on the estimates. The results of the experiments showed that their performance on synthetic data and real-world data is competitive to offline density estimators, that the estimators are able to deal with noisy data, and that the density estimates are able to adapt to concept drifts.

In future work, we would like to work on faster and more sophisticated inference algorithms that solve a broader range of inference tasks and would also like to extend the work towards continuous joint densities and conditional densities.

REFERENCES

[1] P. Ram and A. G. Gray, "Density estimation trees," in *Knowledge Discovery and Data Mining*, 2011, pp. 627–635.

[2] S. Davies and A. W. Moore, "Interpolating conditional density trees," in *Uncertainty in Artificial Intelligence*, 2002, pp. 119–127.

[3] V. Vapnik and S. Mukherjee, "Support vector method for multivariate density estimation," in *Neural Information Processing Systems*, 1999, pp. 659–665.

[4] M. P. Holmes, A. G. Gray, and C. L. I. Jr., "Fast nonparametric conditional density estimation," *CoRR*, vol. abs/1206.5278, 2012.

[5] E. Frank and R. R. Bouckaert, "Conditional density estimation with class probability estimators," in *ACML*, 2009, pp. 65–81.

[6] F. Buchwald, T. Girschick, E. Frank, and S. Kramer, "Fast conditional density estimation for quantitative structure-activity relationships," in *AAAI*, 2010.

[7] J.-N. Hwang, S.-R. Lay, and A. Lippman, "Nonparametric multivariate density estimation: a comparative study," *IEEE Transactions on Signal Processing*, vol. 42, no. 10, pp. 2795–2810, 1994.

[8] D. W. Scott and S. R. Sain, *Multi-Dimensional Density Estimation*. Amsterdam: Elsevier, 2004, pp. 229–263.

[9] M. Kristan and A. Leonardis, "Online discriminative kernel density estimation," in *International Conference on Pattern Recognition*, 2010, pp. 581–584.

[10] M. Kristan, A. Leonardis, and D. Skocaj, "Multivariate online kernel density estimation with gaussian kernels," *Pattern Recognition*, vol. 44, no. 10-11, pp. 2630–2642, 2011.

[11] C. G. Lambert, S. E. Harrington, C. R. Harvey, and A. Glodjo, "Efficient on-line nonparametric kernel density estimation." *Algorithmica*, vol. 25, no. 1, pp. 37–57, 1999.

[12] A. Zhou, Z. Cai, L. Wei, and W. Qian, "M-kernel merging: Towards density estimation over data streams," in *In Proc. of DASFAA*, 2003, pp. 285–292.

[13] A. Elgammal, R. Duraiswami, and L. S. Davis, "Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 1499–1504, 2003.

[14] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine Learning*, vol. 85, no. 3, pp. 333–359, 2011.

[15] K. Dembczynski, W. Cheng, and E. Hüllermeier, "Bayes optimal multilabel classification via probabilistic classifier chains," in *International Conference on Machine Learning*, 2010, pp. 279–286.

[16] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Knowledge Discovery and Data Mining*, 2000, pp. 71–80.

[17] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Knowledge Discovery and Data Mining*, 2001, pp. 97–106.

[18] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine Learning*, vol. 36, no. 1-2, pp. 105–139, 1999.

[19] E. Frank and S. Kramer, "Ensembles of nested dichotomies for multi-class problems," in *ICML*, 2004.

[20] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[21] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The American Statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[22] A. Gelman and D. Rubin, "Inference from iterative simulation using multiple sequences," *Statistical Science*, vol. 7, pp. 457–511, 1992.

[23] R. Motwani and P. Raghavan, *Randomized algorithms*. New York, NY, USA: Cambridge University Press, 1995.

[24] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl, "Moa: Massive online analysis, a framework for stream classification and clustering," *Journal of Machine Learning Research - Proceedings Track*, vol. 11, pp. 44–50, 2010.

[25] M. Scutari, "Learning Bayesian networks with the bnlearn R package," *Journal of Statistical Software*, vol. 35, no. 3, pp. 1–22, 2010.

[26] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[27] T. P. Mann, "Numerically stable hidden Markov model implementation," *An HMM scaling tutorial*, pp. 1–8, 2006.

[28] G. Melançon and F. Philippe, "Generating connected acyclic digraphs uniformly at random," *Inf. Process. Lett.*, vol. 90, no. 4, pp. 209–213, 2004.