A Probabilistic Condensed Representation of Data for Stream Mining

Michael Geilke, Andreas Karwath, and Stefan Kramer

Johannes Gutenberg-Universität Mainz, Staudingerweg 9, 55128 Mainz, Germany
Email: {geilke,karwath,kramer}@informatik.uni-mainz.de

***Abstract*—Data mining and machine learning algorithms usually operate directly on the data. However, if the data is not available at once or consists of billions of instances, these algorithms easily become infeasible with respect to memory and run-time concerns. As a solution to this problem, we propose a framework, called MiDEO (Mining Density Estimates inferred Online), in which algorithms are designed to operate on a condensed representation of the data. In particular, we propose to use density estimates, which are able to represent billions of instances in a compact form and can be updated when new instances arrive. As an example for an algorithm that operates on density estimates, we consider the task of mining association rules, which we consider as a form of simple statements about the data. The algorithm, called POEt (Pattern mining on Online density esTimates), is evaluated on synthetic and real-world data and is compared to state-of-the-art algorithms.**

I. Introduction

Traditional data mining and machine learning algorithms usually operate directly on the data. Whereas batch algorithms may use all the data at every computation step, incremental and online algorithms use only subsets of the data at each step (i.e., single instances, consecutive instances, or a random sample). With increasing amounts of data and a tendency towards streams of data (e.g., click data, sensor data, tweets, ...), the latter algorithms are becoming more and more important for applications. Hence, many traditional tasks now need to be solved in an online setting. Many of the algorithms implicitly assume that collecting the data and performing the task on it is either happening simultaneously or with a small temporal delay, which is not always desired or possible: in several scenarios (see below), separating the collection process and performing the actual task could bring some benefits.

In this paper, we present a new approach to stream mining that is not directly based on raw data in any form (windows, random samples, ...), but on density estimates that were inferred online. The main idea is to decouple the process of density estimation, based on the data stream, and the extraction of human-understandable data mining results (patterns, rules, outliers, clusters, ...) from this density estimate and presentation to the analyst (see Figure 1). The approach embraces two important principles of data mining and knowledge discovery in databases, namely the use of probabilistic methods to model the data and at the same time the value of human-readable patterns and models to gain a deeper understanding of the data. The approach has several benefits:

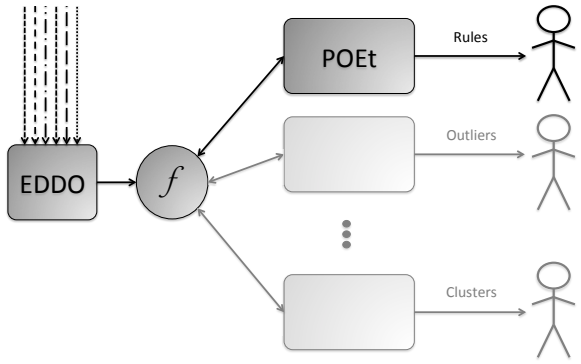

Fig. 1. The Concept of stream mining based on online density estimates MiDEO (Mining Density Estimates inferred Online)

1) It can analyze data of much greater *volume* than could ever fit into main memory. As the density estimates can be updated continuously and the size of the density estimates is a fraction of the size of the data represented by the densities, potential memory limitations do not play a major role anymore.
2) Along the same lines, the *speed of the stream* and the *speed of analysis* need not be coupled anymore (as indicated in Figure 1). While the online density estimator EDDO constantly updates the density estimate f, this density can be accessed by the analysts according to their own needs and analysis requirements.
3) *Unknown task at the time of collecting the data*: If the data mining or machine learning task is not known at the time of collecting the data or several tasks need to be performed, which possibly even depend on each other, a condensed representation of the data can be useful. As density estimates are "universal representations" of data, they can be used to derive

any information needed, be it patterns, rules, outliers, clusters, or the like. To make this possible, inference tasks need to be supported for this specific type of density estimate.

4) *Privacy concerns*: To protect the privacy of the entities of which the data is collected, it could be required that algorithms must not operate on the raw data, but some condensed representation of the data. This could be relevant if contracts between companies prohibit or restrict the access to raw data but allow to perform certain evaluation tasks. Density estimates are suitable for this task, because they keep all the relevant statistical properties of the data, *in principle* without revealing information about individual entities.[1]
5) Basis for *parallelization*: If we had an anytime density estimator instead of an online density estimator, the whole approach could provide a basis for parallelization on many common architectures and platforms. Fractions of the stream could be distributed, results could be gathered given the available time budget and combined (mixed) again to obtain the overall density that is queried by the analysts.

A condensed representation of data that fulfills many of the requirements mentioned above is the online density estimators (EDDO) introduced by Geilke *et al.* [1]. It is able to represent billions of instances in a compact way, can be updated when new instances arrive, and is modifiable and queryable by inference algorithms. Using pattern mining as an example, we will demonstrate that data mining tasks can be directly performed on top of it without access to the raw data.

The main contributions of the paper are as follows. (1) We propose a framework, called MiDEO, using *a condensed representation of data*, which is both *probabilistic* and works in an *online* setting. It offers an universal representation of data on which data mining and machine learning tasks can be performed. (2) As an example, we designed algorithms that perform itemset and association rule mining on this universal representation. In particular, we show that an online density estimate is suitable for this purpose.

The remainder of the paper is organized as follows. In Section II, we briefly discuss related work in the context of itemset and association rule mining. In Section III, we give a short summary of the online density estimator EDDO with the main focus on the aspects that are relevant for this paper. In Section IV, we present algorithms that directly extract itemsets and association rules from online density estimates produced by EDDO. Subsequently, in Section V, the algorithms are compared to state-of-the-art algorithms on synthetic and real-world data. Section VI concludes the paper.

II. Related Work

Itemset mining has been introduced by Agrawal *et al.* [2]. Since then, besides methods to speed up the mining process (e.g., [3], [4]), many other aspects of pattern mining have been studied. For example, uncertainty of itemsets [5], privacy concerns [6], or computing condensed representations of patterns [7], [8]. For the latter, a sufficient subset of all frequent itemsets was often used, from which the other frequent itemsets could be derived. However, these condensed representations were only suitable for the task of mining itemsets or association rules – tasks from other areas of data mining or machine learning were mostly not addressed, although the idea of condensed representations of patterns was originally conceived to be more general than "just" for pattern mining. To the best of our knowledge, no condensed representation *of data*, in particular not one that is (a) *probabilistic* and (b) works in an *online* setting, has been proposed so far.

[1]Of course, a privacy-preserving data mining approach based on this still needs to be worked out in detail.

III. Online Density Estimates

In this paper, we will use the online density estimator introduced by Geilke *et al.* [1], called EDDO, to estimate the discrete joint density of the data in an online fashion. EDDO exploits that every discrete density $f(X_1, \ldots, X_c)$ with variables $X_1, \ldots, X_c$ can be represented as a product

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

EDDO estimates each of the factors by classifiers that provide class probability estimates, thereby yielding a chain of classifiers. For $f_1(X_1)$, it employs a Majority Class classifier, and, for $f_i(X_i \mid X_1, \ldots, X_{i-1})$, $i \in \{2, \ldots, c\}$, it employs Hoeffding trees [9]. The Hoeffding trees model the interdependencies between the variables by selecting variables as tree node based on previous choices above that node. Hence, interdependencies are implicitly covered by the structure of the tree. If an instance needs to be drawn from the density estimate, EDDO simply iterates over the classifiers from $f_1(X_1)$ to $f_c(X_c \mid X_1, \ldots, X_{c-1})$, draws an estimate from each classifier, samples a value based on the distribution obtained, and uses the output as input for the next classifier. In a similar fashion, the probability of a given instance is computed. To increase the robustness of the density estimate, the authors introduced two further variants: $EDDO_{ECC}$ and $EDDO_{EWCC}$. $EDDO_{ECC}$ uses an ensemble of classifier chains instead of a single classifier chain and averages the estimate over all chains. $EDDO_{EWCC}$ also uses an ensemble of classifier chains but additionally weights the classifier chains according to their performance.

These density estimates actually provide a condensed representation that can process thousands of instances per second and enable a number of inference tasks. The possibility to perform inference tasks is probably one of the most important properties, since they are a key component to extracting information from the condensed representation, thereby allowing to perform data mining and machine learning tasks on top of it. In particular, they support the modification of the condensed representation by extracting specific parts of the density and incorporating evidence. Both are important to adapt the representation to the requirements of the task at hand and to perform tasks long after the corresponding data has been collected.

IV. Pattern Mining

In the following, we describe how data mining and machine learning tasks can be directly performed on an estimate of a discrete joint density. As example, we use itemset and

association rule mining and employ the online density estimates produced by EDDO. Although the presented approach is simple, it demonstrates that online density estimates can be used for data mining and machine learning tasks.

In traditional association rule mining, an association rule is an implication of the form $A \Rightarrow B$ with A and B being non-empty subsets of a given set of items $I := \{i_1, i_2 \ldots, i_m\}$ and $A \cap B = \emptyset$. It is obtained from a sequence of transactions $D := \{t_1, t_2, \ldots, t_n\}$, where each transactions is a non-empty subset of I. Transactions are often represented by a Boolean vector of size m, where true means that the item corresponding to that index is contained in the transaction and false means that it is not contained. In order to mine relevant rules, the algorithms require a measure of interestingness, i.e., a minimum support threshold θ and a threshold γ for the confidence of a rule. Then an association rule $(l \Rightarrow r)$ is considered interesting if its support ($supp(l \Rightarrow r)$) exceeds θ and its confidence ($conf(l \Rightarrow r)$) exceeds γ. An alternative to the confidence is the lift, for which the user has to define a threshold δ. Again rules exceeding γ are considered interesting. [2], [10]

In order to directly perform association rule mining on a density estimate of the data, we deviate slightly from the original idea of itemsets. Instead of mining rules where the left- and right-hand side contain items, we mine rules where specific *variable-value combinations* are on both sides (e.g., (X_4, v_3), $(X_9, v_1) \Rightarrow (X_1, v_5)$, where (X_i, v_j) means that attribute X_i takes the value v_j). In this setting, the sequence of transactions are the instances from which the density estimate has been computed.

To evaluate the interestingness of a rule, we will compute the measure of interestingness (e.g., confidence or lift) by deriving the required probabilities from a density estimate f. For the confidence, we compute $f(r \mid l)$ and for the lift we compute $\frac{f(r|l)}{f(r)}$. In contrast to conventional association rule mining, we will not use some threshold to select association rules. Instead, we require the number of desired candidates from the user, sample ten times as many candidates, and select the top 10% with respect to the measure of interestingness.

A. The Algorithms

In this subsection, we propose POEt, an algorithm that performs association rule mining on an online density estimate of the data. As in the case of the Apriori algorithm [10], it first selects possible candidates by generating attribute-value combinations that exceed the probability threshold θ_1. This threshold is not given by the user but is estimated from a sample of instance. Afterwards, it uses these candidates to derive association rules based on the confidence or lift. In order to perform these subtasks, it makes extensive use of inference algorithms as presented by Geilke *et al.* [1]. Similarly to their algorithms, our main focus does not lie on the runtime performance of the algorithm but on the association rules extracted from the online density estimate.

To find an appropriate probability threshold for selecting candidates and to estimate the number of instances to perform certain inference tasks, POEt uses the lower and upper Chernoff bounds [11]. In particular, they provide estimates that hold with a given confidence threshold.

Formally, we have the following setting: There is a stream of instances with variables $X := \{X_1, X_2, \ldots, X_c\}$. Each variable X_i, $1 \leq i \leq c$, may take a predefined set of discrete values $values(X_i)$. A variable-value combination is a set of tuples (X_i, v) with $v \in values(X_i)$ and $X_i \in X$. The density estimate that has been derived from the stream is denoted by $f(X_1, X_2, \ldots, X_c)$. Moreover, POEt requires several parameters that can be set by the user. θ_2 is the confidence level of the lower or upper Chernoff bound, which will be used to estimate the size of samples. λ is another parameter that is required for computing Chernoff bounds. It specifies the deviation from the mean.

Algorithm 1: candidateGeneratingDensity

Input: density estimate f, set of all variables X, set of variables to be marginalized out $Y \subset X$, precision of Chernoff bound $0 < \theta_2 < 1$, $\lambda > 0$ parameter for Chernoff bound
Output: density estimate f'
Require: min. number of training instances m (e.g. 10^5)

```
// Est. required training instances
```
1 $\theta_1 \leftarrow estimateProbabilityThreshold(f, X, Y, \theta_2)$
2 find the minimal n using a binary search starting at $n = 1$ with $n \in \mathbb{N}$, s.t.

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

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

```
// Train estimator
```
3 $f' \leftarrow$ initialize a new density estimator
4 **while** $f'.numProcessedInsts() \leq \max(m, n)$ **do**
5 | $inst \leftarrow$ draw instance from f
6 | **if** $marginalizedProb(f, X, Y, inst) \geq \theta_1$ **then**
7 | | update f' using $inst$
8 | **end**
9 **end**

The first subtask is to select variable-value combinations that exceed a given minimum support threshold θ_1 with high probability. Algorithm 1 derives a density estimate f' from which these candidates can be drawn. As input, it requires the original density estimate f, the set of all variables X, a set of variables Y, and the Chernoff parameters θ_2 and λ. Y contains all those variables that should be ignored when deriving the density estimate. Later on, this will be used to select subsets of variables for which candidates should be generated. Additionally, it requires a global parameter m, which is the minimum number of instances that should be used to train f'. Its purpose is explained later. In the first two lines, the algorithm estimates the number of instances n that are required to obtain a density estimate that contains instances having a probability of θ_1 or higher. Using the upper Chernoff bound, an n is computed that holds with confidence level θ_2, which means that a sample of size n contains instances having a probability of θ_1 or higher with a confidence level of θ_2. In lines 3 through 9, a density estimator f' is trained with up to $\max(m, n)$ instances. The instances are drawn from the original density estimator f and are only passed on to f' if

Algorithm 2: estimateProbabilityThreshold

Input: density estimate f, set of all variables X, set of variables to be marginalized out $Y \subset X$, precision of Chernoff bound $0 < \theta_2 < 1$
Output: probability threshold θ_1
Require: sampleSize (typically 10^4), $topX$ size of subset for computation of probability threshold and confidence (typically 10%)

`// Estimate the lowest probability of the top` $topX\%$ `of the instances`
1 $probs \leftarrow \emptyset$
2 **for** $i = 1$ *to* $sampleSize$ **do**
3 $\quad inst \leftarrow$ draw instance from f
4 $\quad probs \leftarrow probs \cup marginalizedProb(f, X, Y, inst)$
5 **end**
`// Estimate` θ_p `(the minimum probability threshold for the instances)`
6 $\theta_p \leftarrow$ lowest probability of top $topX\%$ in $probs$
7 find the minimal $0 < \lambda_T \leq 1$ using a binary search, s.t.

$$Pr[T < (1 - \lambda_T) \cdot \mu] < e^{-\frac{\mu\lambda_T^2}{2}} < \theta_2 \ ,$$

where $\mu = sampleSize \cdot \theta_p$ and T is the sum of independent Poisson trials with probabilities $Pr(T_i = 0) = 1 - \theta_p$, $Pr(T_i = 1) = \theta_p$.
8 $\theta_1 \leftarrow \theta_p \cdot (1 - \lambda_T)$

the marginalized probability of the instance equals or exceeds θ_1. At this point, the purpose of m becomes obvious. If n is very small, but the user knows that the density estimator typically requires a larger number of instances to provide good estimates, then m ensures that at least that many instances are used for training.

Algorithm 3: marginalizedProb

Input: density estimate f, set of all variables X, set of variables to be marginalized out $Y \subset Y$, instance $inst$
Output: marginalized probability p

1 $p \leftarrow 1$
2 $inst \leftarrow$ remove all variables from $inst$ that are in Y
3 **for** *every tree in* f *where the class attribute* $\notin Y$ **do**
4 $\quad paths \leftarrow$ the set of all paths with $(a_1, v_1, a_2, v_2, \ldots, a_{j-1}, v_{j-1}, a_j)$ in T, where $a_1 = root(T)$, $a_j \in leaves(T)$, a_i are the nodes, v_i are the edge labels, and a_i being an attribute in $inst$ with value v_i, $1 \leq i \leq j$
5 $\quad p_T \leftarrow 0$
6 $\quad$ **for** $path \in paths$ **do**
7 $\quad\quad p_T \leftarrow p_T$ + probability of $path$ in f
8 $\quad$ **end**
9 $\quad p \leftarrow p \cdot p_T$
10 **end**

The procedure $candidateGeneratingDensity$ (Algorithm 1) makes use of two subprocedures: $estimateProbabilityThreshold$ (Algorithm 2) and $marginalizedProb$ (Algorithm 3). Algorithm 2 estimates a probability threshold for the instances, which corresponds to the minimum support threshold in a typical pattern mining setting. Dependent on the dimensionality of the dataset and the variance of the probabilities, the probabilities can be very small or very large. Therefore, the procedure estimates a probability threshold using a sample of instances. As input parameters, Algorithm 2 has the same parameters as Algorithm 1 except for the Chernoff λ. Additionally, there are two global parameters: $sampleSize$ is the size of the sample to determine typical probability values, and $topX$ is the size of the subset that will be used by the algorithm to determine a high probability value. In lines $1 - 5$, Algorithm 2 draws $sampleSize$ many instances from f, computes their marginalized probability (variables from Y are marginalized out), and stores the probabilities in $probs$. Afterwards, an initial probability threshold is chosen based on $probs$. Using the lower Chernoff bound, this threshold is corrected by computing the maximal deviation with respect to the confidence level (line 7 and 8).

Algorithm 4: performPatternMining

Input: density estimate f, set of all variables X, θ_2, λ parameter for Chernoff bound, maximal number of candidates per subset m, maximal number of candidates n
Output: set of association rules R
Require: $topX$ size of subset for computation of probability threshold and confidence

1 $C' \leftarrow \emptyset$, $m' \leftarrow topX \cdot m$, $n' \leftarrow topX \cdot n$
2 $C \leftarrow probableItemsets(f, X, \theta_2, \lambda, m', n')$
`// Select association rules and compute their confidence`
3 **for** $c \in C$ **do**
4 $\quad$ **for** $1 \leq i \leq n$ **do**
5 $\quad\quad r \leftarrow \{(X_i, \text{value of } X_i \text{ in } c)\}$
6 $\quad\quad l \leftarrow \{(X_j, \text{value of } X_j \text{ in } c) \mid X_j \neq X_i\}$
$\quad\quad conf \leftarrow probConfidence(f, X, l \Rightarrow C\ l, \theta_2, \lambda)$
7 $\quad\quad C' \leftarrow C' \cup (l \Rightarrow r, conf)$
8 $\quad$ **end**
9 **end**
`// Select top` $topX\%$ `association rules with respect to the confidence`
10 $\mathcal{R} \leftarrow \emptyset$
11 $b \leftarrow$ smallest confidence value of top $topX\%$ of C'
12 **for** $(l \Rightarrow r, conf) \in C'$ **do**
13 $\quad$ **if** $conf >= b$ **then**
14 $\quad\quad \mathcal{R} \leftarrow \mathcal{R} \cup (l \Rightarrow r)$
15 $\quad$ **end**
16 **end**

Algorithm 3 computes the marginalized probability of instances. It expects the original density estimate f, which is a classifier chain consisting of Hoeffding trees, the set of all variables X, and a set of variables Y, which contains the variables that have to be marginalized out. If $Y = \emptyset$, the probability of an instance is obtained by multiplying the probabilities of this instance for each Hoeffding tree (line 9). The probability of a Hoeffding tree T is simply the class probability stored in a_j, where $a_j \in leaves(T)$, $(a_1, v_1, a_2, v_2, \ldots, a_{j-1}, v_{j-1}, a_j)$ is a path in T, $a_1 = root(T)$, a_i are the nodes (attributes

Algorithm 5: probableItemsets

Input: density estimate f, set of all variables X, precision of Chernoff bound $0 < \theta_2 < 1$, $\lambda > 0$, maximal number of candidates per subset m, maximal number of candidates n
Output: set of candidates C

```
// Choose attribute subsets
```
1 $\mathcal{Y} \leftarrow \emptyset$
2 **for** $i = 0$ *to* $\lfloor \frac{n}{m} \rfloor$ **do**
3 | $\mathcal{Y} \leftarrow \mathcal{Y} \cup Y$, where $Y \subseteq X$, $|Y|$ is chosen according to a geometric distribution with $P(l) = 0.5 \cdot (1 - 0.5)^{l-2}$ for $l = 2, 3, \ldots$ and the elements are chosen uniformly at random
4 **end**
```
// Create a density estimator for each
   attribute subset and draw instances
```
5 $C' \leftarrow \emptyset$
6 **for** $Y \subseteq \mathcal{Y}$ **do**
7 | $f' \leftarrow$ candidateGeneratingDensity(f, X, Y, θ_2, λ)
8 | **for** $i = 1$ *to* m **do**
9 | | $inst \leftarrow$ draw instance from f'
10 | | add $inst$ to C'
11 | **end**
12 **end**
13 $C \subseteq C'$ uniformly at random, such that $|C| = n$

a_i with value v_i), and v_i are the edge labels (values v_i), $1 \leq i \leq j$. If $Y \neq \emptyset$, there are several such paths (line 4) and the probabilities of each path have to be summed up (lines 6 through 8). The variables in Y are ignored by removing them from the instance $inst$ (line 2).

Now that we can generate density estimates from which instances with high probability can be drawn, we can describe the main procedure $performPatternMining$ (Algorithm 4). Besides the parameters f, X, θ_2, and λ, the algorithm has two additional parameters: n is the total number of candidates that are requested by the user, and m is maximal number of instances per subset. In the beginning (lines $1-4$), $performPatternMining$ invokes $probablyItemsets$ with $topX$ as many instances as required by the user (i.e., $m' = topX \cdot m$, $n' = topX \cdot n$) and stores the result in C. Multiplying by $topX$ is necessary, since we will select only those rules that belong to the top $topX\%$ with respect to the confidence or lift. It is basically the same idea we applied earlier. The user cannot know typical values for the confidence or lift without using the algorithm. Therefore, Algorithm 4 selects the association rules relative to the set of candidates (lines 10 through 16). After all, in our pattern mining setting, it is not important which exact confidence or lift values an association rule has, but that it is interesting relative to all the patterns we can find. Using the set of itemsets, Algorithm 4 constructs association rules and computes their confidences. For each itemset, it iterates over the set of attributes. The currently selected attribute is used for the right hand side and the remaining attributes are assigned to the left hand side. From all association rules generated this way, the algorithm picks the top $topX\%$ and returns them to the user.

The procedure $performPatternMining$ (Algorithm 4) makes use of two subprocedures: $probableItemsets$ (Algorithm 5) and $probConfidence$ (Algorithm 6). The input parameters of Algorithm 5 are the same as used before, so we directly start with the explanation. In lines 1 through 4, subsets of X are chosen randomly. Their size is chosen according to a geometric distribution, so that smaller itemsets are preferred over bigger ones. The elements are chosen uniformly at random. As a consequence, the choice of the subsets is biased, but, in our opinion, this is closer to what we have to expect from real-world applications [12]. In the second part of the algorithm (lines 5 through 12), for each subset of variables Y, a density estimate f' is derived where the variables in Y are marginalized out. Then up to m instances are drawn from f' and stored into C'. Since we have selected $\lfloor \frac{n}{m} \rfloor + 1$ subsets in the beginning, $|C'|$ could be larger than n. Therefore, we choose n many itemsets uniformly at random from C'.

Algorithm 6: probConfidence

Input: density estimate f, set of all variables X, $l \Rightarrow r$, θ_2, λ
Output: confidence of $l \Rightarrow r$

1 $Y \leftarrow vars(l)$, $Z \leftarrow vars(r)$
2 $f' \leftarrow deriveConditionalDensity(f, X, Y, Z, \theta_2, \lambda)$
3 **return** $f'(r)$

Algorithm 6 was already described in the beginning of the section (and the lift can be computed similarly). The subprocedure $deriveConditionalDensity$ (Algorithm 7), which is invoked by $probConfidence$, computes a density estimate that is conditioned on some variables $Z \subset X$ and is trained with instances exceeding the probability threshold as computed in Algorithm 1. Again most of the input parameters are the same as before. The set $\{(Z_1, v_1), (Z_2, v_2), \ldots, (Z_k, v_k)\}$ with $Z_j \in X$ for $1 \leq j \leq k$, are the variables and their values on which the resulting density will be conditioned. In the context of the inference algorithms, these tuples are the hard evidence that needs to be considered by the density estimator. Lines 1 and 2 are identical to Algorithm 1. In lines 4 through 8, the algorithm iterates over all Hoeffding trees and fixes the values of the variables $Z_1, Z_2, \ldots, Z_k$. By fixing the values of the corresponding nodes in the Hoeffding tree, we implicitly condition the density represented by the tree on the variables $Z_1, Z_2, \ldots, Z_k$. Lines 9 through 14 are identical to the second part of Algorithm 1 again.

This concludes the description of POEt. Notice that the presented procedures work in an offline fashion. A recomputation of the rules is only necessary if either the estimate is significantly more precise or a concept drift occurred. For the latter, concept drift detection algorithms can be used (e.g., Dries and Rückert [13]). For the former, one could apply the Wilcoxon rank-sum test to log-likelihoods of a sample of instances to decide whether there is a significant difference in the probabilities predicted by the density estimate and an older snapshot.

V. Evaluation

In this section, we compare POEt to Apriori [10], which will provide the ground truth, and Moment [14], which is like

Algorithm 7: deriveConditionalDensity

Input: density estimate f, set of all variables X, set of variables to be marginalized out $Y \subset X$, $\{(Z_1, v_1), (Z_2, v_2), \ldots, (Z_k, v_k)\}$ with $Z_j \in X$ for $1 \leq j \leq k$, precision of Chernoff bound $0 < \theta_2 < 1$, $\lambda > 0$

Output: density estimate f'

Require: min. no. of training instances m (e.g., 10^5)

```
// Est. required training instances
```

$\theta_1 \leftarrow estimateProbabilityThreshold(f, X, Y, \theta_2)$

find the minimal n using a binary search starting at $n = 1$ with $n \in \mathbb{N}$, s.t.

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

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

```
// Train conditional density estimator
```

$f' \leftarrow$ initialize a new density estimator

for *every Hoeffding tree in* f **do**

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

    fix value of node Z_j using v_j

  end

end

while $f'.numProcessedInsts() \leq \max(m, n)$ **do**

  $inst \leftarrow$ draw instance from f

  if $marginalizedProb(f, X, Y, inst) \geq \theta_1$ **then**

    update f' using $inst$

  end

end

POEt intended for stream mining. Although there are many other algorithms that compute exact itemsets and association rules (e.g., variations of Apriori [3], [4], FPGrowth [15], Eclat [16], ...), including more algorithms into the evaluation does not provide any benefits, as the qualitative results are identical. Indeed, Apriori alone would be sufficient to show the effectiveness of the proposed algorithms, since we only aim for a qualitative comparison (i.e., the memory consumption or the run-time is not relevant) and Apriori finds all itemsets and association rules with the desired properties. The reason why we decided to compare to Moment as the only streaming approach is that its implementation (MOA framework [17]) is one of the few available and functioning implementations that computes exact itemsets. Moreover, Moment was one of the first algorithms that performed frequent itemset mining on streams and has already been compared to other approaches (though not with respect to the quality). The Apriori algorithm is available in the SPMF[2] library, and we implemented POEt in the MOA framework (version 2013.08).

For the comparison, we used synthetic and real-world data. As synthetic data, we used data that was generated by the IBM dataset generator [10] (the `ARtool`[3] to be specific) and data that was generated from Bayesian networks. For the IBM dataset generator, we selected several parameter settings: The number of patterns was set to one of the values $\{50, 100, 1000\}$, the transaction length to $\{5, 10, 15\}$, and the average pattern length to $\{2, 4, 6\}$. We generated $100,000$ instance with 100 attributes. For the Bayesian networks, we randomly generated Bayesian networks using the `random.graph` method of the `bnlearn` package [18] of the `R` framework. Its parameter `method` was set to `melancon`, which uses a Markov chain to draw acyclic, directed graphs uniformly at random [19]. We generated 100 Bayesian networks with 10 nodes, where each node had between 4 and 8 values. As real-world dataset, we selected the MovieLens dataset[4], which consists of 23 attributes and $49,282$ instances. Please notice that we had to flatten the MovieLens dataset and the data generated from the Bayesian networks for Apriori and Moment to make them suitable for the algorithms. Hence, the resulting itemsets are no proper itemsets in the original sense, but consist of exclusive attribute-value pairs for each attribute, i.e, (a1, True), (a2, False), ... As performance measure, we computed the percentaged overlap, which is $\frac{|I_1 \cap I_2|}{|I_2|}$, where I_1 are the itemsets produced by POEt and I_2 are the itemsets produced by the method to which we compare.

TABLE I. THE TABLE SHOWS THE PERCENTAGED NUMBER OF ITEMSETS THAT WERE FOUND BY POET IN COMPARISON WITH APRIORI AND MOMENT. THE MINIMUM SUPPORT THRESHOLD WAS SET TO 5%, 10%, AND 25%. THE WINDOW SIZE OF MOMENT WAS SET TO $10,000$. *IBM*, *Bayes*, AND *MovieLens* ARE THE DATASETS FROM THE IBM DATASET GENERATOR, BAYESIAN NETWORKS, AND MOVIELENS, RESPECTIVELY.

Dataset	Algorithm	Support		
		5%	10%	25%
IBM	*Apriori*	0.002	0.002	0.006
	Moment	0.001	0.000	0.001
Bayes	*Apriori*	0.384	0.487	0.524
	Moment	0.101	0.195	0.415
MovieLens	*Apriori*	0.133	0.111	0.333
	Moment	0.143	0.111	0.143

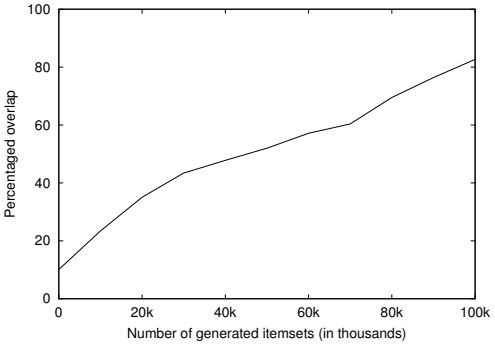

Fig. 2. For one particular dataset generated from a Bayesian network, the figure shows the overlap of itemsets generated by POEt compared to the itemsets generated by Apriori. For the given dataset, POEt generated $100,000$ itemsets in 284 seconds. Apriori produced 525 itemsets. The x-axis represents the number of generated itemsets, and the y-axis represents the overlap.

The results are summarized in Table I. For the data generated by the IBM dataset generator, the overlap between POEt and Apriori or Moment is relatively small (on average it sometimes gets below 0.001). When we inspected the resulting patterns closer, we noticed two things. First, there are datasets where Apriori and Moment find more than $10,000$ itemsets (sometimes even $100,000$ itemsets). Since POEt generated

[2]http://www.philippe-fournier-viger.com/spmf/

[3]http://www.cs.umb.edu/∼laur/ARtool/

[4]http://grouplens.org/datasets/movielens/

TABLE II. THE TABLE SHOWS THE PERCENTAGED NUMBER OF RULES THAT WERE FOUND BY POET IN COMPARISON WITH APRIORI. THE MINIMUM SUPPORT WAS SET TO 5% AND THE MINIMUM CONFIDENCE THRESHOLD WAS SET TO 0% (ALL RULES), 25%, AND 50%. *IBM*, *Bayes*, AND *MovieLens* ARE THE DATASETS FROM THE IBM DATASET GENERATOR, BAYESIAN NETWORKS, AND MOVIELENS, RESPECTIVELY.

Dataset	Confidence		
	0%	25%	50%
IBM	0.000	0.000	0.000
Bayes	0.389	0.345	0.210
MovieLens	0.098	0.093	0.100

only up to $1,000$ in the experiments, it is impossible to reach large overlap values. Second, POEt also produces different kinds of itemsets. Contrary to Apriori and Moment, it not only finds itemsets that describe which items are usually present at the same time, but also which items are not present. For example, whereas Apriori could only provide an itemset representing customers who often watch action and crime movies, POEt could also produce an itemset representing customers who watch action and crime movies but almost never watch comedy – so POEt gives you the additional information that the customer is not interested in comedy movies. On the data generated from Bayesian networks, the results differ from the first experiment. With up to 52 (42%), POEt is able to find many of the itemsets proposed by Apriori (Moment). The same can be observed for the real-world dataset (MovieLens), though the overlap is slightly lower than on the data generated from Bayesian networks. When comparing the probabilities computed by POEt with the confidence values of Apriori and Moment, we noticed that itemsets that have a high probability are not necessarily the itemsets that have a high support count. Hence, there is a certain overlap, but POEt also finds itemsets not presented by Apriori and Moment – something, which we have also seen on the IBM data. To gain further insights into the results, we considered individual datasets and computed the overlap of itemsets generated by POEt compared to the itemsets generated by Apriori with increasing number of sampled instances. One particular example is illustrated in Figure 2. The corresponding dataset has been generated from a Bayesian network and yielded 525 itemsets when running Apriori with a minimum support threshold of 5%. We generated up to $100,000$ instances with POEt (in about 284 seconds) and computed the overlap with the itemsets generated by Apriori every $10,000$ instances. We can see that the overlap increases with increasing numbers of generated itemsets and is getting close to 100% when approaching $100,000$ itemsets.

The results for the association rules are summarized in Table II. Notice that we did not include Moment here, because MOA-Moment only generates itemsets. On the data from the IBM dataset generator, there is basically no overlap with Apriori, which had to be expected, since the rules are generated from the itemsets and there was basically no overlap on itemsets already. The results on the data from Bayesian networks and the MovieLens dataset are comparable to the itemsets. On the Bayesian network data, the overlap decreases with increasing confidence. On the MovieLens dataset, the overlap is more or less stable.

Overall, the experiments demonstrate that although POEt has not yet been optimized in any way, it is able to find a substantial amount of the itemsets and association rules found by either Apriori as well as Moment (in most cases between 9% and 53%). Since the precision of POEt depends on the quality of the initial density estimate and temporarily computed density estimates, improvements in this area should positively influence the overall precision of POEt.

VI. CONCLUSIONS

Using pattern mining as example, we presented algorithms that performed itemset mining and association rule mining on a condensed representation of data, i.e., an online density estimate. In the experiments, we demonstrated that the proposed algorithms produce itemsets similar to the ones generated by traditional pattern mining algorithms as well as itemsets that are completely different and cannot be expressed by them. For this task, the algorithms used a representation of the data that is independent from the task at hand – a condensed representation of the data. With increasing amounts of data and an increasing emphasis on aspects like real-time analysis, privacy, and the like, it will become more important to perform data mining on condensed representations of the data without access to the raw data.

The presented algorithms and experimental results showed the feasibility of such a condensed representation of data for a standard data mining task. In the future, we plan to investigate fast inference algorithms to further speed up the process and other novel algorithms to extract standard types of patterns and models from online density estimates.

REFERENCES

[1] M. Geilke, A. Karwath, E. Frank, and S. Kramer, "Online estimation of discrete densities," in *Proceedings of the 13th IEEE International Conference on Data Mining*, 2013, pp. 191–200.

[2] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in *SIGMOD Conference*, 1993, pp. 207–216.

[3] J. S. Park, M.-S. Chen, and P. S. Yu, "An effective hash based algorithm for mining association rules," in *SIGMOD Conference*, 1995, pp. 175–186.

[4] H. Toivonen, "Sampling large databases for association rules," in *VLDB*, 1996, pp. 134–145.

[5] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Züfle, "Probabilistic frequent itemset mining in uncertain databases," in *KDD*, 2009, pp. 119–128.

[6] Y. Xia, Y. Yang, and Y. Chi, "Mining association rules with non-uniform privacy concerns," in *DMKD*, 2004, pp. 27–34.

[7] N. Pasquier, R. Taouil, Y. Bastide, G. Stumme, and L. Lakhal, "Generating a condensed representation for association rules," *J. Intell. Inf. Syst.*, vol. 24, no. 1, pp. 29–60, 2005.

[8] A. Bykowski and C. Rigotti, "A condensed representation to find frequent patterns," in *PODS*, 2001.

[9] P. Domingos and G. Hulten, "Mining high-speed data streams," in *KDD*, 2000, pp. 71–80.

[10] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB*, 1994, pp. 487–499.

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

[12] Z. Zheng, R. Kohavi, and L. Mason, "Real world performance of association rule algorithms," in *KDD*, 2001, pp. 401–406.

[13] A. Dries and U. Rückert, "Adaptive concept drift detection," *Statistical Analysis and Data Mining*, vol. 2, no. 5-6, pp. 311–327, 2009.

[14] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz, "Moment: Maintaining closed frequent itemsets over a stream sliding window," in *ICDM*, 2004, pp. 59–66.

[15] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *SIGMOD Conference*, 2000, pp. 1–12.

[16] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, 2000.

[17] 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," in *WAPA*, 2010, pp. 44–50.

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

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