Maintaining Nonparametric Estimators over Data Streams

Björn Blohsfeld Christoph Heinz
Bernhard Seeger
Department of Computer Science
Philipps University Marburg
35032 Marburg, Germany
{blohsfel, heinzch, seeger}@mathematik.uni-marburg.de

Abstract

An effective processing and analysis of data streams is of utmost importance for a plethora of emerging applications like network monitoring, traffic management, and financial tickers. In addition to the management of transient and potentially unbounded streams, their analysis with advanced data mining techniques has been identified as a research challenge. A well-established class of mining techniques is based on nonparametric statistics where especially nonparametric density estimation is among the essential building blocks. In this paper, we examine the maintenance of nonparametric estimators over data streams. We present a tailored framework that incrementally maintains a nonparametric estimator over a data stream while consuming only a fixed amount of memory. Our framework is memory-adaptive and therefore, supports a fundamental requirement for an operator within a data stream management system. As an example, we apply our framework to selectivity estimation of range queries, which is a popular use-case for statistical estimators. After providing an analysis of the processing cost, results of experimental comparisons are reported where synthetic data streams as well as real-world ones are considered. Our results demonstrate the accuracy of the results being produced by estimators derived from our framework.

1 Introduction

Situations abound in which data continuously arrives in a transient data stream, e. g. in network traffic, health monitoring, financial applications, and sensor networks. These data-intensive applications generally require an immediate processing of the arriving elements without storing them persistently on disk. Since traditional DBMS are not designed for supporting potentially infinite data streams, data stream management systems (DSMS) have been emerged as a new technology recently. In addition to its push-oriented implementation of underlying operators, a fundamental difference to database technology is the ability of a DSMS to rearrange the query plan at runtime. In particular, the operators of a DSMS should be adaptive with respect to their resource consumption.

An important issue for DSMS are techniques for computing synopses and statistical estimators on unbounded data streams as well as other complex analytical tasks. Though many efficient and accurate synopses have been proposed in the literature, the integration of theses synopses within a DSMS has received only very little attention so far. Several drawbacks of synopses, see also [8], limit their practical application in DSMS. First, with very few exceptions, synopses are tailorcut to a specific task. Second, synopses are typically not memory-adaptive as they do not support reducing or increasing memory at runtime. Third, synopses are directly applicable only to discrete data, while continuous data requires an additional discretization step. Fourth, a synopsis corresponds to a compression of the data stream. If the arriving data only represents a sample of the original stream (as it might occur when load shedding is performed within the DSMS), synopses are accurate with respect to the sample, but not necessarily with respect to the original stream. Fifth, though synopses give excellent asymptotic error guarantees, there might be large constants hidden in the asymptotic costs.

In this paper, we address these issues by proposing a framework for maintaining nonparametric statistical estimators over data streams. It can be specialized in two directions: the estimation technique and the analytical task. First, many techniques known from nonparametric statistics like histograms and ker-

nel methods can be plugged into the framework such that the resulting estimators are similar to their traditional counterparts. Second, the framework supports a large number of analytical tasks like selectivity and density estimation. Many analytical tasks based on density estimation, e.g. density-based clustering [11], can therefore directly be set into the streaming context. One important feature of our framework is that all derived estimators are memory-adaptive, i.e., they allow a dynamic increase or decrease of their memory while still providing valid results. Thus, the framework satisfies an essential requirement of a DSMS. A fundamental difference of our approach to synopses is that the primary goal is to adopt to the underlying data distribution rather than to the elements seen so far.

After showing that our framework is theoretically sound and generally provides accurate estimators, we study a simple but popular analytical task, namely the estimation of the selectivity of range queries over data streams. For that, we consider two elementary selectivity estimators using the empirical cumulative distribution function (ECDF) and kernel-based methods, respectively. The results of our experiments obtained from synthetic and real-world data streams reveal the robustness of ECDF in comparison to kernel-based methods. Moreover, the ECDF estimator derived from our framework offers much higher accuracy (for a given memory size) than a counterpart directly derived from reservoir sampling. In comparison to estimators with infinite storage, our ECDF estimators provide almost the same accuracy while consuming only a small amount of memory. Furthermore, we also consider a popular quantile-based synopsis [20] in our experiments. Our results show that this synopsis is generally inferior to our ECDF estimator.

The remainder of this paper is organized as follows. In Section 2, we briefly discuss related work. Section 3 outlines the requirements for estimators over data streams. In Section 4, we present a reservoir-based approach while our general framework and its parameters are developed in Section 5. In Section 6, we study a concrete use-case of our framework and analyze maintenance cost as well as memory distribution. An experimental evaluation of the presented methods is given in Section 7. Section 8 concludes the paper with a summary of the most important results and provides prospects for future research.

2 Related Work

In the context of data stream processing new challenges and research issues arise as Babcock et al. discussed in their seminal work [6]. Recently, data stream management systems (DSMS), e.g. [21, 3], have been designed and first prototypes have been implemented. In addition to the basic management of data streams, advanced mining methods are of particular interest in a DSMS [10]. For example, the MAIDS project [5] provides a system prototype for online multi-dimensional stream data mining. StatStream [26] adapts different statistical methods for data streams. Gigascope [9] as another DSMS supports specific queries for network applications like monitoring and traffic analysis.

Recently, there has also been interest in adopting common data mining techniques to the data stream model, e.g. clustering on data streams [4].

Our work is based on methods from nonparametric statistics like kernel-based density estimation. These are of particular interest in the area of data mining where density estimation is a core operation for many probabilistic learning methods like density-based clustering [11]. For high-dimensional classification problems, a combination of decision trees and local kernel density estimates is proposed in [12]. Kernel density estimation is also applicable to biased sampling [17] where the local density for a given point is examined. In the context of traditional databases, kernel density estimation can be applied to estimate the selectivity of range queries on metric attributes [7, 19, 16]. For a general exposition of kernel density estimation see [23].

Closely related to our work are techniques for summarizing data streams, also termed sketches and synopses. Accurate multidimensional histograms over data streams are proposed in [15]. The basic idea is to maintain a sketch that allows to create a multidimensional histogram on demand from random mappings. The method gives approximate guarantees about the quality of the estimate. The techniques proposed in [13] also include the construction of a small sketch from the stream that produces approximate histograms. The proposed two-stage algorithm first generates an initial histogram approximation that permits the construction of a histogram with the desired accuracy in the second stage. The computation of approximate aggregate queries is addressed in [14] where a one-pass computation is proposed that employs randomized linear projections of the underlying signal. The approximate computation of quantiles is another problem that can be solved by using synopses [20]. All of these approaches are dedicated to a specific analytical task, whereas we want to establish in this paper a framework that allows to plug in arbitrary nonparametric estimators whose memory consumption in turn can be dynamically rearranged.

An essential aspect of our framework is memory adaptivity that has not been addressed for estimation

techniques and synopses on data streams so far. Memory adaptivity is however a critical requirement for an operator within a DSMS [6] where resources have to be rearranged among the operators at runtime. For traditional DBMS, a related problem is addressed in [18] where a framework is presented for reconciling summary data structures. An optimal combination of these structures is then achieved for a given workload and a limited amount of memory.

3 Preliminaries

In the rest of the paper, we assume a stream to be an infinite sequence of d-dimensional vectors over numerical domains. For example, a traffic sensor might generate a stream that continuously delivers the speed of cars. For the sake of simplicity, we first consider one-dimensional data. The multidimensional case is separately studied in Section 5.6. This model is closely related to the ordered aggregate model [14], but contrary to, the numerical domain is assumed to be continuous. This assumption is generally valid for spatiotemporal databases where sensors deliver the position of moving objects. In order to apply common synopses in this scenario, a discretization of the domain is unavoidable and introduces an additional error.

In [10], the following criteria are proposed for the design of data analysis methods on high-speed, transient, and potentially infinite data streams:

1. The processing time for a single data element is as small as possible and ideally constant.
2. Only a fixed amount of main memory is available, independent from the total number of already seen elements.
3. The methods require only one scan over the data stream.
4. At any time, the methods provide valid results.
5. The produced results are equivalent to those obtained by ordinary data mining methods.
6. Conceptual drifts and changes in the data stream are captured whereas also past data is included that has not become outdated.

Under the given restrictions we are particularly interested in an anytime method with the following two properties: First, the method should return the best answer possible even if it is not allowed to run to completion. Second, the method may improve on the answer if it is allowed to run longer. Data analysis methods, e.g. synopses, that meet these requirements typically deliver approximated results where the accuracy depends on the available memory.

The need for memory-adaptivity is a topic we particularly stress in this paper. Therefore, the following criterion is added to our list:

7. The algorithms dynamically adapt to changing resource capacities, especially memory and CPU.

Memory-adaptive estimators are important for a DSMS due to the following reasons. Queries on streams are typically long-running and therefore, the optimal assignment of memory at the start time of the query could be suboptimal at a later point in time. Consider for example a situation when the system becomes overloaded. Then, the DSMS requires from the operators to reduce their computational costs that generally requires a reduction of their allocated memory. Another example is when a user is not satisfied anymore with the accuracy of the estimation. Then, the DSMS should be in the position to improve accuracy by increasing the available memory for the estimator. In combination with the requirement of delivering valid results anytime, data analysis methods need to adapt the memory consumption at runtime without re-starting the method from scratch. A natural invariant is then to require a relationship between amount of available memory and quality of the result such that the more memory is available the better the results are and vice versa. This is a very strong requirement since most of the mining algorithms over data streams are based on allocating a fixed amount of memory once at initialization time.

Sketch of the problem

The main problem addressed in this paper is the development of a framework for continuously maintaining sound and accurate nonparametric estimators over data streams. The framework in turn could also be a kind of platform for more advanced data mining functionality, i.e. conventional mining methods based on nonparametric estimators might be directly plugged in without taking specific stream requirements into account. The resulting estimators should satisfy all requirements listed above. There, memory-adaptivity is an important aspect for estimators that has not been addressed previously.

4 Reservoir-based Sampling

In this section, we present a first method based on reservoir sampling that only partially fulfills our

requirements mentioned above. An initial approach for continuously maintaining a nonparametric estimator over a data stream is to maintain a subset of the stream, termed reservoir, with a fixed number of elements. By using the elements from the reservoir, an arbitrary estimator can be computed during runtime. To ensure the validity of the estimator, the reservoir has to consist of independent and identically distributed values, also called an *iid*-sample. A well-known approach for maintaining an *iid*-sample over a data stream is reservoir sampling [25] that continuously generates an *iid*-sample of an unbounded data stream without the need of accessing past data.

Figure 1 illustrates the general procedure for building estimators based on reservoir sampling assuming a sample of 5 elements. From this sample, an arbitrary nonparametric estimator $\hat{f}(\cdot)$ can be derived at runtime. Despite its straightforward implementation,

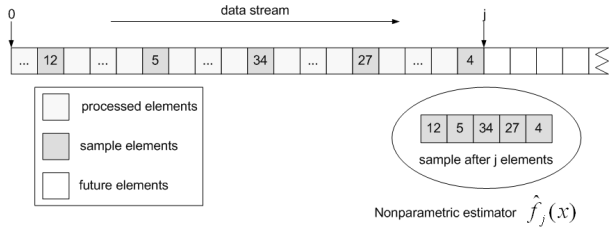

Figure 1. Reservoir-based nonparametric estimator

reservoir sampling offers some significant drawbacks. First, reservoir sampling is not fully memory-adaptive. While the size of the reservoir can be reduced by dropping elements randomly, it is not possible to increase the size of the reservoir at runtime without accessing past data. Second, the estimator only depends on the elements in the reservoir, while all other elements of the stream do not contribute. In particular for small reservoirs, this may cause a poor accuracy of the estimator. Third, an unavoidable drawback of reservoir sampling is that the chance of an element being part of the reservoir linearly decreases with the number of elements seen so far in the stream. Consequently, the reservoir generally consists of out-dated data that might not represent the current status of the stream. This is not desirable in many stream applications where users are primarily interested in recent data.

5 Our Framework

In this section, we present the details of our framework for a memory-adaptive maintenance of arbitrary nonparametric estimators over data streams. In order to deploy our framework, a user has to specify the nonparametric estimation technique $\hat{f}$ that is suitable for supporting the specific analytical task. For example, if a user is interested in density estimation, a kernel-density estimator might be a suitable choice. Moreover, the size of the available memory (M) has to be set in advance. For the sake of simplicity, we first assume that M is a constant and postpone the discussion of a dynamic M. Then, the data stream is partitioned into blocks of fixed size smaller than M. The basic idea is to compute a local estimator for each block. The cumulative estimator corresponds to a convex linear combination of the local estimators derived from already consumed blocks. The actual estimator generally does not fit into memory. Therefore, a lossy compression is performed each time the actual estimator is updated.

In the following subsections, we present the details of our approach. First, we show how the estimation can be computed in a single scan over a stream. As examples, we discuss different estimation techniques that are suitable for being plugged into our framework. Moreover, we introduce different kinds of weighting strategies that allow to put emphasis on recent data. Then, we present the details of our compression technique and discuss the error induced by the compression. Finally, we discuss the suitability and memory-adaptivity of our framework as well as the case of multidimensional data.

5.1 Single-scan computation

In this subsection, we describe the basic computation step of our framework.

Definition 1 *Let us consider a data stream that consists of N elements, $N = j \cdot B,\ j, B \in \mathbb{N}$, where B denotes the block size. Let $\hat{f}_i(x), i = 1, ..., j$ be an estimator for the i-th block of the data stream, then*

$$\hat{g}_j(x) = \sum_{i=1}^{j} \omega_i \hat{f}_i(x) \tag{1}$$

is a **cumulative estimator** *for the first N elements with respect to the weights ω_i, $1 \leq i \leq j$, if*

$$\sum_{i=1}^{j} \omega_i = 1 \quad \textit{and} \quad \omega_i \geq 0 \ \ \forall i = 1, ..., j. \tag{2}$$

A cumulative estimator inherits the basic properties of its block estimators. Let $\hat{f}_i(x),\ i = 1, ..., j$, be block estimators with expectation θ, i.e. $E(\hat{f}_i(x)) = \theta$, where θ denotes the parameter that should be estimated. This means that each of the block estimators is *unbiased*.

Then, the unbiasedness of the cumulative estimator follows from $E(\hat{g}_j(x)) = \sum_{i=1}^{j} \omega_i E(\hat{f}_i(x)) = \theta$.

The method for computing the cumulative estimator as introduced in Definition 11 is not suitable for data streams. Therefore, we present a single-scan algorithm that supports an iterative computation of the cumulative estimator over the data stream. The idea is to define a binary function that takes as input the cumulative estimator of the first i blocks and the block estimator of the $(i+1)$-th block. A linear combination of both inputs then returns the cumulative estimator for the first $i+1$ blocks. The following theorem proves the iterative computability of cumulative estimators.

Theorem 1 *Let $\hat{g}_j(x)$ be a cumulative estimator where j denotes the number of already processed blocks. Then, for all cumulative estimators $\hat{g}_i$ exists a sequence of weights $\tilde{\omega}_i$ with $0 \leq \tilde{\omega}_i \leq 1,\ i = 2, ..., j$, such that*

$$\hat{g}_i(x) = \begin{cases} \tilde{\omega}_1 \hat{f}_1(x), & i = 1 \\ (1-\tilde{\omega}_2)\hat{f}_1(x) + \tilde{\omega}_2 \hat{f}_2(x), & i = 2 \\ (1-\tilde{\omega}_i)\hat{g}_{i-1}(x) + \tilde{\omega}_i \hat{f}_i(x), & 3 \leq i \leq j. \end{cases} \quad (3)$$

Sketch of the proof.
For an arbitrary, but fixed j, we show by induction over i, that there is an equivalent recursive representation of the i-th cumulative estimator $\hat{g}_i(x) = \sum_{l=1}^{i} \omega_l^{(i)} \hat{f}_l(x)$ with $\sum_{l=1}^{i} \omega_l^{(i)} = 1$ and $0 \leq \omega_l^{(i)} \leq 1$. This can be achieved by defining $\omega_l^{(i+1)} = (1-\tilde{\omega}_{i+1})\omega_l^{(i)},\ 1 \leq l \leq i, \omega_1^{(1)} = 1$, and $\omega_{i+1}^{(i+1)} = \tilde{\omega}_{i+1}$. Then, $\sum_{l=1}^{i+1} w_l^{(i+1)} = 1$ follows from our inductive hypothesis ($\sum_{l=1}^{i} w_l^{(i)} = 1$).

5.2 Estimation Techniques

The specific choice of an estimation technique within our framework depends on the considered task. We will exemplarily present two nonparametric estimation techniques. Moreover, we sketch further applications built on top of the estimation techniques and therefore, can also run on data streams.

Important for statistical estimation techniques is to view the data stream as sample of a random variable. The random variable is characterized by its probability density function (pdf) and its cumulative distribution function (cdf) [24]. The cdf can be estimated with the empirical cumulative distribution function.

Definition 2 *Let $(X_1, ..., X_B)$ be the elements of a block (that is assumed to be an iid sample of an unknown distribution). The empirical cumulative distribution function (ECDF) is then defined as*

$$F_B(x) := \frac{1}{B} \sum_{i=1}^{B} I_{(-\infty,x]}(X_i), \quad (4)$$

where

$$I_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases}. \quad (5)$$

The ECDF provides excellent estimates of a random variable's cdf as it is the one with the lowest variance [24] among all unbiased estimators. The only drawback is its staircase shape that prevents its use for density estimation. The ECDF can directly be applied to estimate the selectivity of range queries.

The distribution of a random variable is typically defined in terms of its pdf. A very popular class of estimators for the pdf are built on top of so-called kernels [23]. A kernel-based density estimator (KDE) is essentially the sum over kernel functions that are centered at sample points.

Definition 3 *Let $(X_1, ..., X_B)$ be the elements of a block (that is assumed to be an iid sample of an unknown distribution). The kernel density estimator (KDE) is defined as*

$$\hat{f}(x) := \frac{1}{Bh} \sum_{i=1}^{B} K\left(\frac{x - X_i}{h}\right), \quad (6)$$

where K denotes the kernel function and h denotes the bandwidth.

The KDE inherits some basic properties like smoothness from its kernel function. If the kernel function is continuous, the resulting KDE will also be continuous. The most frequently used kernels are sufficiently smooth, positive, and symmetric. Moreover, kernel functions with a compact support, e. g., the Epanechnikov kernel [23], are particularly advisable for selectivity estimation of range queries since the evaluation of the estimate is inexpensive. The bandwidth h, which controls the influence region of a point, is a crucial parameter for the quality of an estimate. One of the extensively discussed strategies for the bandwidth choice is the normal scale rule [23]. This rule is among the very few where an online computation is possible.

The following list gives a few examples of analytical tasks that can be supported when an accurate density estimate is available: identification of the top-k most frequent intervals, the total frequency of elements in $[a, b]$ and an estimation of the moments. Moreover,

when the kernel function is sufficiently often differentiable, the estimator also guarantees the same degree of differentiability. This allows to estimate the local extremals of the density function of a data stream.

In addition to the presented techniques, there exists a plethora of other nonparametric estimators that can be plugged into our framework, e.g. histograms.

5.3 Weighting Strategies

Our framework supports arbitrary weighting strategies if the corresponding weights only fulfill equation (2). Given an appropriate sequence of weights, the proof of theorem 1 shows how to define the associated weighting sequence. We will present two popular weighting strategies as examples.

Arithmetic weighting: All block estimators $\hat{f}_i(x)$ are weighted equally:

$$\hat{g}_j(x) = \frac{1}{j}\sum_{i=1}^{j}\hat{f}_i(x).$$

The weight sequence $(\tilde{\omega}_i)_{i\geq 1}$ for iterative computations results from

$$\tilde{\omega}_i = \frac{1}{i}.$$

For the cumulative estimator $\hat{g}_i(x),\ i \geq 2$ follows

$$\hat{g}_i(x) = \frac{i-1}{i}\hat{g}_{i-1}(x) + \frac{1}{i}\hat{f}_i(x).$$

Exponential weighting: The new block estimator receives weight α:

$$\hat{g}_i(x) = (1-\alpha)\hat{g}_{i-1}(x) + \alpha\hat{f}_i(x)$$

whereby $\alpha \in (0,1]$ is the smoothing parameter. For j processed blocks, the i-th weight is

$$\omega_i^j = \alpha(1-\alpha)^{j-i},\ i \geq 2.$$

For the estimator after j processed blocks holds

$$\hat{g}_j(x) = \sum_{l=0}^{j-2}\alpha(1-\alpha)^l\hat{f}_{j-l} + (1-\alpha)^{j-1}\hat{f}_1.$$

Exponential weighting emphasizes recent data while decreasing the impact of older data. This technique is also known as exponential smoothing. Exponential weighting also supports a kind of implicit deletion of elements similar to methods that maintain an estimator for a sliding window.

5.4 Compressed-cumulative Estimators

The cumulative estimator as introduced in Definition 1 still does not take the memory limitations of size M into account. Each block estimator requires a certain amount of memory and therefore, the overall memory consumption of the cumulative estimator increases linear in the size of the stream.

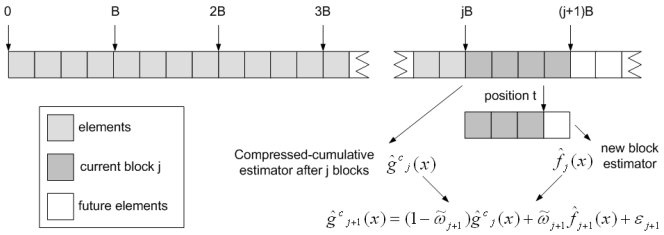

Figure 2. compressed-cumulative nonparametric estimator

In order to limit our memory usage to M, we slightly modify the recursive computation of the cumulative estimator as given in equation (3). After a new cumulative estimator is built, a lossy compression is computed such that its final size is only a fraction of M. Figure 2 illustrates the basic idea that is formally defined in the following.

Definition 4 *Let $\hat{g}_j$ be a cumulative estimator after the j-th block was processed. Then $\hat{g}_j^c$ is termed an* **compressed-cumulative estimator** *of $\hat{g}_j$, if $\hat{g}_i^c$ is a compression of $(1-\tilde{\omega}_i)\hat{g}_{i-1}^c + \tilde{\omega}_i\hat{f}_i$, $2 \leq i \leq j$ and $\hat{g}_2^c$ is a compression of $\hat{g}_2$ such that the occupied memory of $\hat{g}_j^c$, $j \geq 2$ is smaller than a fraction of M.*

There is no limitation on the choice of the specific compression technique as long as it produces high accuracy for a fixed amount of memory. We decided to use cubic splines [22] for compressing the one-dimensional cumulative estimators. Cubic splines are within the class of interpolating polynomials the ones with the least bending. The computation costs for generating a spline with n interpolation points is $O(n)$, while the evaluation cost is constant. Moreover, the memory requirement for a cubic spline is $O(n)$. Cubic splines have proved to be valuable in different application scenarios. Splines were already used for approximating kernel density estimation [19], primarily for reducing the evaluation cost of the estimator. Note that this is different to our approach where splines are used for reducing the size of the estimator.

Since the compression introduces an additional error in each step, the development of the total cumulated

error is of particular interest. Here, the total cumulated error refers to the compression error between the cumulative and the compressed-cumulative estimator.

Theorem 2 *Let ϵ_j be the error term of the j-th compressed estimator, i. e.,*

$$\hat{g}_j^c(x) = \begin{cases} (1-\tilde{\omega}_j)\hat{g}_{j-1}^c(x) + \tilde{\omega}_j \hat{f}_j(x) + \epsilon_j, & j \geq 3 \\ (1-\tilde{\omega}_2)\hat{f}_1(x) + \tilde{\omega}_2 \hat{f}_2(x) + \epsilon_2, & j = 2 \end{cases}. \quad (7)$$

For $j \geq 2$, $\tilde{\omega}_1 = 1$, and $\epsilon_1 = 0$, we obtain

$$\hat{g}_j^c(x) = \sum_{i=1}^{j} \tilde{\omega}_i \hat{f}_i(x) \prod_{l=i+1}^{j} (1-\tilde{\omega}_l) + \sum_{i=1}^{j} \epsilon_i \prod_{l=i+1}^{j} (1-\tilde{\omega}_l). \quad (8)$$

Proof by induction over j.

The overall error after j processed blocks is a discounted sum of the compression errors occurring in each step (see equation (8)). For the weighting strategies presented in Section 5.3, this permits the computation of upper bounds. Let $\epsilon := \max_{k \leq j}\{|\epsilon_k|\}$ be the maximum error. For the arithmetical weighting, it follows

$$\epsilon_j^c = \sum_{i=1}^{j} \epsilon_i \prod_{l=i+1}^{j} (1-\tilde{\omega}_l) \leq \epsilon \sum_{i=1}^{j} \prod_{l=i+1}^{j} (1-\frac{1}{l}) = \frac{j+1}{2}\epsilon.$$

This shows that there is at most a linear dependence between the maximal error and the number of processed blocks. However, this is a rather pessimistic upper bound since we might expect that different error terms annihilate each other. For the exponential weighting with a smoothing parameter $\alpha \in (0,1]$, we obtain the following upper bound:

$$\epsilon_j^c = \sum_{i=1}^{j} \epsilon_i \prod_{l=i+1}^{j} (1-\alpha) \leq \epsilon \sum_{i=1}^{j} (1-\alpha)^{j-i} \leq \frac{1}{\alpha}\epsilon.$$

In this case, the cumulated error is indeed independent from the number of processed blocks. Moreover, it is bounded by the maximum approximation error so far. Upper bounds for other weighting strategies can be obtained in a similar fashion.

In addition to this deterministic way of expressing errors, another approach is to derive probabilistic error bounds. Since each block estimator is a random variable, it follows that the cumulative estimator is also a random variable. The compression is essentially a functional operator that still depends on the sample values. Again, we can model the error as a random variable. Under the assumption of a normal distributed error in each step i, i.e. $\epsilon_i \sim N(\mu_i, \sigma_i)$, we define the actual overall error as

$$\epsilon_j^c := \sum_{i=1}^{j} \epsilon_i \prod_{l=i+1}^{j} (1-\tilde{\omega}_l). \quad (9)$$

In accordance with equation (8), this implies

$$\epsilon_j^c \sim N\left(\sum_{i=1}^{j} \mu_i \prod_{l=i+1}^{j} (1-\tilde{\omega}_l), \sum_{i=1}^{j} \sigma_i^2 \prod_{l=i+1}^{j} (1-\tilde{\omega}_l)^2\right). \quad (10)$$

Thus, the variance of the overall error ϵ_j^c after j processed blocks consists of the weighted error variances of the single blocks whereas later variances carry more weight.

The question about the quality of the resulting estimator arises, i.e. how good are the results we receive from the compressed-cumulative estimator? This essentially depends on the considered task and on the underlying estimation technique. Given an estimation technique with error guarantees, the quality of the compressed-cumulative estimator can be determined starting with its representation in equation (8).

5.5 Suitability and Memory-adaptivity of our Framework

In this section we will discuss the suitability of our framework for the data stream model as introduced in Section 3. Since the stream is processed in units of a block, each element is processed once with little overhead, i.e. requirements 1 and 3 are fulfilled. Note that the processing costs are actually constant with respect to the number of processed elements. A more precise analysis is only possible for a specific use-case as discussed in Section 6. Due to the compression, our estimation technique requires only a fixed amount of memory and thus, requirement 2 is also satisfied. Since each of the data elements has an influence on its according block estimator, it immediately follows that the element has an impact on the compressed-cumulative estimator. This is in contrast to reservoir sampling where an element has either a substantial impact or no impact at all. We also argued that the compression error is expected to be low. Thus, requirements 5 and 6 are also fulfilled. Requirement 4 is also met since the current estimator always provides a valid estimation that constantly improves at runtime (at least up to a certain point in time).

Still an open question is how we are able to be memory-adaptive as postulated in requirement 7. In order to be memory- and CPU-adaptive, we propose to distribute the available memory of size M among

three components: the i-th cumulative-compressed estimator, a full buffer that contains the $(i+1)$-th block and an input buffer that receives the elements of the $(i+2)$-th block. This architecture allows to receive new elements while at the same time the $(i+1)$-st estimator is computed. It might happen that CPU resources are so limited that the input buffer will be filled up before the new estimator is entirely computed. Then, reservoir sampling is simply applied to the input buffer as a fallback strategy.

At each point in time where we have updated the estimator, our method asks the memory manager whether additional memory can be allocated or occupied memory has to be deallocated. Each of the three components can then dynamically adapt to a changing size of the available memory. It is obvious that we can adjust a buffer size when the entire of the current elements are processed. Moreover, the size of the input buffer can also be decreased. An increase of memory is only possible if reservoir sampling is not currently used to fill up the buffer (otherwise the property of a sample is not fulfilled anymore). Concerning the compressed-cumulative estimator, the memory amount of the lossy compression, which in turn controls the compression quality, can be increased or decreased. Note that there is an inherent trade-off in the memory distribution among the three components. The more memory the compressed-cumulative estimator occupies, the better is the resulting compression. However, this is achieved with a smaller memory for the buffers where the quality of the corresponding block estimator decreases with the reduced number of elements. Moreover, a small input buffer also increases the probability for using reservoir sampling as our fallback strategy. Since the quality of all three components relies on their available memory, an adequate memory management is a challenging task within our framework.

5.6 Multidimensional Data Streams

For the sake of simplicity, we have limited our previous presentation to one-dimensional data. Our framework is however sufficiently extensible to cope with multidimensional data as an exploration of the parameters reveals.

At first, our framework allows the usage of multidimensional nonparametric estimators. For one-dimensional estimators like KDE and ECDF, there are many multidimensional counterparts [23]. However, the curse of dimensionality will remain a serious problem in the context of data streams. In order to provide accurate estimators, the sample size must actually increase exponentially in the number of dimensions. Therefore, the memory limitation will become a serious problem for multidimensional data.

The other component of our framework that has to be extended is the compression technique. For the one-dimensional case, we mentioned cubic splines as an adequate compression technique. A multidimensional cubic spline can also be implemented with tensor products of one-dimensional cubic splines [22].

Overall, this shows that our framework is also suitable for generating multidimensional nonparametric estimators.

5.7 Convex Merge Step

We introduced the cumulative estimator as linear combination of nonparametric block estimators. Equation (3) also provides a recursive representation of the cumulative estimator that ensures an iterative computation. If the separate block estimators are real-valued functions like KDE and ECDF, the computation of the new cumulative estimator is a simple convex linear combination of the old cumulative estimator and the new block estimator according to equation (3). In a similar way, two histogram-based estimators can be merged into one. However, a crucial question is how to process the merge step for non-real-valued functions? Such functions occur for example when clustering and decision trees are computed on streams. Then, a fundamentally different convex merge step of two nonparametric estimators is required. A deeper discussion is beyond the scope of this paper and the problem will be addressed in our future research.

6 A Use Case: Selectivity Estimation

In this section, we exemplarily discuss the usage of our framework for a simple but popular use-case, namely the selectivity estimation of range queries over one-dimensional numerical data streams. We discuss an appropriate choice of parameters and the computational complexity of the resulting estimators.

The initial step is to determine the nonparametric estimation technique that provides the functionality for estimating the selectivity of range queries. Therefore, KDE and ECDF as presented in Section 5.2 are well-suited since the selectivity of a range query is the probability of a random variable lying within a certain range. Here, we focus on ECDF. Next, we need an adequate compression technique wherefore we choose cubic splines. As convex merge step, we compute the convex linear combination of two functions. Finally, we use an arithmetic weighting, i.e. we weight each block estimator equally.

We discuss the computational complexity for each block for a fixed memory of size M. For sake of simplicity, the available memory is shared among only two components, the block with B elements and the actual compressed cumulative estimator.
Let us examine the total cost of the j-th step of the estimator where the j-th compressed-cumulative estimator is computed. We assume that each component receives a constant fraction of memory, i. e., $B = \Theta(M)$ and $n = \Theta(M)$. The compression of a function with a spline requires to evaluate the function at $\Theta(n)$ points. For the case of our compressed-cumulative estimator, we evaluate the previous compressed-cumulative estimator in $\Theta(n)$ and the ECDF of the new block in $O(M \log M)$. The latter complexity for the ECDF results from an improved evaluation based on a sort-merge approach. First, the samples of the block estimator are sorted and then all coefficients are iteratively updated by a single scan over the block. The total cost of this approach is then $O(M \log M)$. This proofs the following theorem:

Theorem 3 *Let us consider ECDF as the underlying block estimator in our framework. Furthermore, let the cubic spline and the block estimator receive a constant fraction of main memory M, respectively. In order to compute the corresponding compressed-cumulative estimator on a stream, the amortized cost per element is $O(\log M)$.*

Many other applications are possible within our framework. In order to maintain for example equi-width histograms over data streams, the convex merge step could be the convex linear sum of the frequencies of each bucket. A rearrangement of the buckets number permits a compression. Concerning decision trees, a convex merging of the decision trees based on blocks has to be examined as well as the compression of the tree, i.e. how to generate a compact representation of a tree that consumes only a fixed amount of memory? These use cases clarify that our framework suits as point of origin for an advanced data stream analysis.

7 Experiments

In this section, we report the results of some of our experiments with a special emphasis on KDE and ECDF as estimation techniques.

7.1 Data Sets

In our experiments, we considered synthetic as well as real data sets. In order to express the quality of the different estimation techniques, we used a synthetic data set derived from a known data distribution. We chose a piecewise smooth pdf, termed CP2, which is plotted in Figure 3. Since smooth distributions seldom occur in real applications, we introduced two jumps in CP2. The real-world data set was obtained from the

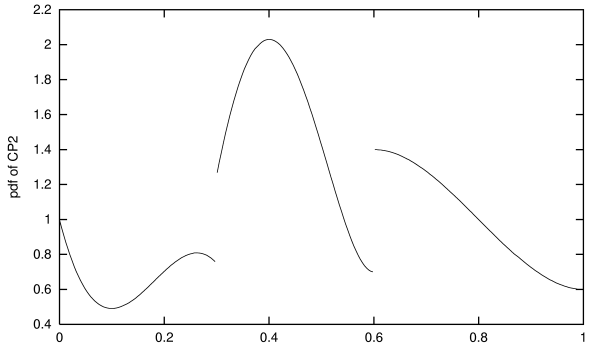

Figure 3. Pdf of the CP2 distribution

Freeway Service Patrol Project that collected traffic flow data from detectors integrated into a 10-mile long section of a highway in California. Each element of those data streams contains the following information on a vehicle: a timestamp, the velocity, and the length. In our queries, we focused on the velocity that seems to be most interesting for realtime analyses, e. g. what is the average speed during rush hour?

7.2 Methods and their Parameter Settings

We applied KDE and ECDF as estimation techniques to initialize our framework (see Section 5.2). While ECDF requires no additional parameters, KDE received the following settings: the Epanechnikow kernel was the kernel function and the normal scale rule was used for computing the bandwidth [23]. Furthermore, we used the arithmetical weighting strategy to combine the block estimators and cubic splines to compress the estimators. The framework itself is fully implemented within XXL, a publicly available Java library for advanced query processing [2].

In order to compare the estimators of our framework with other well-known techniques, we considered two different kinds of methods in our experiments. The one is a synopsis for computing approximate quantiles in a single pass [20], which we used for the following reasons. It provides the typical theoretical error bounds of synopses and its implementation is available in the Colt library [1]. The other refers to estimators derived from reservoir sampling as presented in Section 4.

7.3 Convergence and Memory Adaption

Our first experiment addresses the consistency of an estimator, i. e., how an increasing number of processed elements improves the approximation quality of an estimator. We used a finite data stream consisting of 100000 elements drawn from the CP2 distribution. During the processing of the stream, we continuously maintained a KDE with the parameter settings mentioned above to estimate the density. We continuously examined the current estimator and measured its quality. Therefore, we computed the mean squared error (MSE) on an equidistant grid consisting of 500 points:

$$MSE := \frac{1}{500}\sum_{i=1}^{500}(f(x_i) - \hat{f}(x_i))^2. \qquad (11)$$

In addition, we also examined the effects of memory-adaptivity by continuously increasing the amount of available memory during runtime. We measured memory by charging one memory unit for both a coefficient of a spline and an element of the stream. Overall, we compared three estimation techniques. The first estimator consumed 400 memory units during runtime that were equally distributed among the spline and the new data block. The second estimator also started with 400 units, but received 100 additional units after having processed a block. The third estimator constantly allocated 17600 memory units, which corresponds to the maximum amount of available memory for the second estimator. Figure 4 depicts the results of this experiment. We observe that all estimators improve very

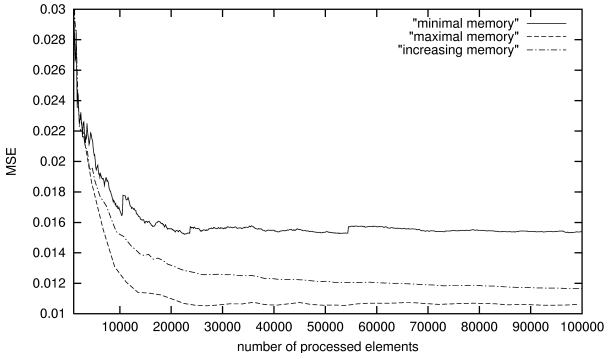

Figure 4. Increasing the memory

fast in the beginning. After 30000 elements, the curves flatten. Overall, the results demonstrate the accuracy of the estimators. Of particular interest are the results of the second estimator. The results show that increasing the amount of available memory during runtime substantially improves the quality of a compressed-cumulative estimator. Therefore, memory-adaptivity really pays off.

7.4 Selectivity Estimation Of Range Queries

This experiment refers to the use case presented in Section 6, namely the estimation of the selectivity of range queries over data streams. We compared the performance of the compressed-cumulative KDE and ECDF for synthetic as well as real data sets. Furthermore, we considered methods based on reservoir sampling.

We again performed experiments with a finite data stream consisting of 100000 elements. While consuming this data stream, we continuously computed the different estimators from the elements seen so far. We set the amount of available memory for all estimators to 5000 memory units.

For each new estimator computed during runtime we performed selectivity estimations of 1000 range queries. Each of them covered 2% of the range of the already processed elements. The distribution of the range queries followed the data distribution of the data stream. In order to compare the results, we used the mean relative error (MRE) as quality measure:

$$MRE := \frac{1}{1000}\sum_{i=1}^{1000}\frac{|\sigma_i - \hat{\sigma}_i|}{\sigma_i}.$$

Here, σ_i denotes the range query selectivity computed by the best possible estimator based on all processed elements of the stream. The estimate of the range query with the current estimator is denoted as $\hat{\sigma}_i$.

First, we discuss the results for the synthetic data set generated from the $CP2$ distribution. Figure 5 depicts the MRE from the estimators as a function of the number of processed elements. As illustrated, the

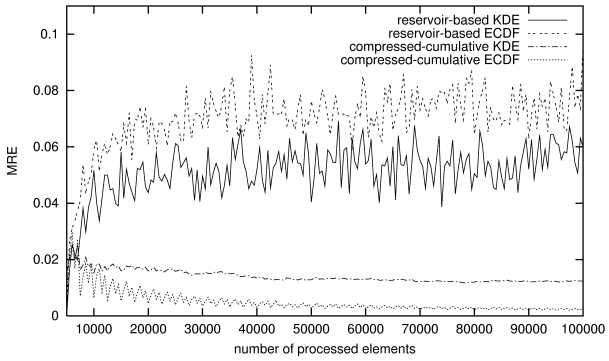

Figure 5. Selectivity estimation of range queries over CP2 data

compressed-cumulative estimators significantly outperform their counterparts based on reservoir sampling. While the compressed-cumulative estimators gives a smooth curve with a continuously decreasing error, the curves of the reservoir-based ones heavily oscillate around 5% and 7%. Their MRE is much higher than the MRE of the compressed-cumulative KDE and the compressed-cumulative ECDF, which is about 1.5% and only 0.3%, respectively.

The case of real-world traffic data reveals different results. Here, we considered all vehicles with velocities between 10 and 30 meters per second. Except the compressed-cumulative ECDF, which had an extremely small MRE of about $0.01\% - 1\%$, the other estimators produced poor results where the MRE was between 20% and 40%. The reason is that the traffic data set consists of a limited number of distinct values and, in a kernel-based approach, the KDE cumulates at these values. In order to overcome this problem, we introduced an artificial noise uniformly distributed in $[0, 5]$. This noisy data delivered much better results. The compressed-cumulative KDE, the reservoir-based ECDF, as well as the reservoir-based KDE produced an error of about 8%, while the compressed-cumulative ECDF again proved to be the best technique with an error of about 0.5%.

In our last experiment, we compared the ECDF with the quantile synopses. Note that quantile synopses are easily applicable to the problem of selectivity estimation of range queries. Since the approximated quantiles presented in [20] allow a one-pass computation, the method is also suitable in our data stream scenario. The experiment is performed on the traffic data set without noise (similar results can be observed for noisy data). We computed the quantiles for different sizes of the data stream and used them to estimate the selectivity of range queries. We measured the quality in terms of MRE by comparing these estimates with the estimates obtained from using the accurate quantiles of the original data set. In order to allow a fair comparison, we set the amount of available memory for the approximate quantiles and the compressed-cumulative ECDF to the same size (38 KB). This results in the following parameter setting for the quantile synopses [1]: $\epsilon = 0,01$ is the maximum approximation error of the quantiles, 10^{-5} is the probability that the approximation error fails to be smaller than ϵ, and the size of the dataset is assumed to be unknown. Note that the amount of memory does not cover the stack size of the programs. Figure 5 depicts the results of this comparison. In the beginning, the compressed-cumulative ECDF is inferior to the quantile synopses. With an increasing number of elements, however, the estimation

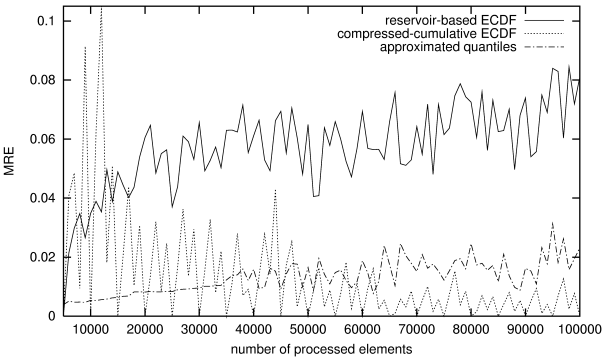

Figure 6. Selectivity estimation of range queries over traffic data

error of the quantile method constantly increases, while the error of the ECDF estimator decreases. Therefore, the compressed-cumulative ECDF provides substantially better results than the quantile synopses at the end. The fact that the error of the synopses may increase is surprising at a first glance, but is in agreement with theoretical findings [20] that the error can be kept under a threshold only if memory slightly increases.

From our experiments we conclude that the estimators built with our framework deliver accurate estimates whose estimation approaches an unknown pdf. This is particularly useful for estimating the selectivity of range queries. It is however remarkable that the most simple nonparametric estimation technique, the ECDF, turns out to be the most robust technique that is generally superior to others.

8 Conclusions

In this work, we presented a tailored framework for the maintenance of arbitrary nonparametric estimators over data streams. This publicly available framework as part of the XXL library is modularly designed and copes with a wide range of applications over data streams that rely on nonparametric estimators. In summary, the estimators process the stream block by block and compute for each of the blocks a separate estimator. A convex linear combination of these estimators delivers an appropriate cumulative estimator. Due to the limited resources, the cumulative estimator is eventually compressed adequately in an iterative manner. In order to illustrate specific applications of our framework, we exemplarily discussed the problems of density estimation over data streams and selectivity estimation of range queries. For the latter problem, we considered two popular nonpara-

metric techniques, ECDF and KDE. In particular, the compressed-cumulative ECDF has shown to be of practical relevance due to its robustness and its generally superiority in comparison to quantile synopses.

Another important and novel aspect guaranteed within our framework is memory-adaptivity, a highly relevant issue for systems that analyze multiple data streams simultaneously and henceforth need the capability of a dynamical adaption of varying system resources at runtime. Our framework allows to control the memory consumption of its components during runtime while still ensuring valid results.

In our future work, we will focus on theoretical quality statements for specific use cases of our framework. Also we want to examine the suitability of other nonparametric estimation techniques within our framework. In addition to these estimation techniques, we will examine new weighting strategies that are controllable during runtime, in order to emphasize interesting regions of the data stream.

Acknowledgements
This work has been supported by the German Research Society (DFG) under grant no. SE 553/4-1.

References

[1] *COLT - Open Source Libraries for High Performance Scientific and Technical Computing in Java*. http://hoschek.home.cern.ch/hoschek/colt.

[2] *XXL - eXtensible and fleXible Library*. Philipps University Marburg: The Database Research Group, http://www.xxl-library.de.

[3] D. Abadi, D. Carney, U. Cetintemel, M. Cherniak, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A New Model and Architecture for Data Stream Management. *VLDB Journal*, 2003.

[4] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A Framework for Clustering Evolving Data Streams. In *Proc. VLDB*, 2003.

[5] L. Auvil, Y. D. Cai, D. Clutter, J. Han, G. Pape, and M. Welge. MAIDS: Mining Alarming Incidents from Data Streams. In *Proc. ACM SIGMOD*, 2004.

[6] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Symp. on Principles of Database Systems*, 2002.

[7] B. Blohsfeld, D. Korus, and B. Seeger. A Comparison of Selectivity Estimators for Range Queries on Metric Attributes. In *Proc. ACM SIGMOD*, 1999.

[8] G. Cormode and S. Muthukrishnan. An improved data stream summary: The Count-Min sketch and its applications. In *LATIN*, 2004.

[9] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A Stream Database for Network Applications. In *Proc. ACM SIGMOD*, 2003.

[10] P. Domingos and G. Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 2003.

[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. ACM SIGKDD*, 1996.

[12] U. Fayyad, A. Gray, and P. Smyth. Retrofitting Decision Tree Classifiers Using Kernel Density Estimation. In *Proc. ICML*, 1995.

[13] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Approximate Histogram Maintenance. In *Symp. on Theory of Computing*, 2002.

[14] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. In *Proc. VLDB*, 2001.

[15] S. Guha, P. Indyk, N. Koudas, and N. Thaper. Dynamic Multidimensional Histograms. In *Proc. ACM SIGMOD*, 2002.

[16] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating Multi-Dimensional Aggregate Range Queries over Real Attributes. In *Proc. ACM SIGMOD*, 2000.

[17] G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold. An efficient approximation scheme for data mining tasks. In *Proc. ICDE*, 2001.

[18] A. König and G. Weikum. A Framework for the Physical Design Problem for Data Synopses. In *Proc. EDBT*, 2002.

[19] F. Korn, T. Johnson, and H. V. Jagadish. Range Selectivity Estimation for Continuous Attributes. In *Statistical and Scientific Database Management*, 1999.

[20] B. Lindsay, G. Manku, and S. Rajagopalan. Approximate Medians and other Quantiles in One Pass and with Limited Memory. In *Proc. ACM SIGMOD*, 1998.

[21] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query Processing, Resource Management, and Approximation in a Data Stream Management System. *Proc. of the Conf. on Innovative Data Systems Research (CIDR)*, 2003.

[22] P. M. Prenter. *Splines and Variational Methods*. John Wiley and Sons, New York, 1974.

[23] D. W. Scott. *Multivariate Density Estimation : Theory, Practice, and Visualization*. John Wiley & Sons, New York, 1992.

[24] J. Shao. *Mathematical Statistics*. Springer Verlag, New York, 1999.

[25] J. S. Vitter. Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 1985.

[26] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proc. VLDB*, 2002.