Towards Kernel Density Estimation over Streaming Data

Christoph Heinz, Bernhard Seeger

Department of Mathematics and Computer Science
University of Marburg
Germany
{heinzch, seeger}@mathematik.uni-marburg.de

Abstract

A variety of real-world applications heavily relies on the analysis of transient data streams. Due to the rigid processing requirements of data streams, common analysis techniques as known from data mining are not applicable. A fundamental building block of many data mining and analysis approaches is density estimation. It provides a well-defined estimation of a continuous data distribution, a fact which makes its adaptation to data streams desirable. A convenient method for density estimation utilizes kernels. However, its computational complexity collides with the processing requirements of data streams. In this work, we present a new approach to this problem that combines linear processing cost with a constant amount of allocated memory. We even support a dynamic memory adaptation to changing system resources. Our kernel density estimators over streaming data are related to M-Kernels, a previously proposed technique, but substantially improve them in terms of accuracy as well as processing time. The results of an experimental study with synthetic and real-world data streams substantiate the efficiency and effectiveness of our approach as well as its superiority to M-Kernels with respect to estimation quality and processing time.

1 Introduction

A variety of heterogeneous real-world applications depends on an adequate online analysis of massive data streams. In order to gain insight into the characteristics of those streams, one could apply data mining and analysis techniques. They provide for instance functionality to reveal interesting patterns, detect outliers, or classify the data. The constantly growing volume of data streams as well as their volatile nature, however, render the direct application of common 'offline' mining techniques difficult; the data arrives faster than it can be analyzed. In fact, to be applicable to data streams, an analysis technique has to meet stringent processing requirements like processing the stream in one pass [8].

Taking those requirements into account, we address in this paper the adaptation of a fundamental building block of many analysis techniques, namely density estimation, to the data stream scenario. The main objective of density estimation is to reveal the unknown probability density function of a distribution, given solely a representative sample of values. A density estimator is a comprehensive statistical model of the process described by the sample values. With a well-defined density estimator at hand, a variety of data analysis issues can be addressed: *"In general, density estimation provides a classical basis across statistics for virtually any kind of data analysis in principle, including clustering, classification, regression, time series analysis, active learning, and so on..."* [11]. We aim to provide a foundation for the application of these analysis tasks to data streams by adapting density estimation in compliance with the aforementioned processing requirements.

In most real-world applications over streams, we have no a priori knowledge about the stream. For that reason, the class of *nonparametric* density estimation approaches is very appealing as they make no assumptions on the unknown density; *"the data speak strictly for themselves"* [21]. Since data streams often deliver observations of continuous, real-valued distributions, e.g. temperature, heart rate, we confine the subsequent considerations to continuous density estimators. A theoretically well-founded and also practically approved approach for the nonparametric estimation of continuous distributions utilizes kernels [20], [21]. Kernel-based density estimators can approximate *any* distribution arbitrarily good (in probabilistic terms), provided that the associated sample is sufficiently large [21]. Hence, an adaptation of kernel density estimation to data streams would be highly promising. However, the heavy computational cost of kernel density estimators is a severe obstacle; their memory allocation is linear in the sample size, accompanied by linear evaluation cost. As these facts violate the aforementioned processing requirements, we cannot directly build kernel density estimators over data streams.

International Conference on Management of Data
COMAD 2006, Delhi, India, December 14–16, 2006

1.1 Our Contributions

In this work, we tackle the problem of resource-aware kernel density estimation over streaming data. In [13], we will present a short version of this work. While [13] only sketches the basic idea, this work presents its detailed discussion. Generally, our main contributions are:

- We redesign and extend the general idea of M-Kernels [5], a previously proposed approach for kernel density estimation over data streams. In fact, our approach solves major drawbacks of M-Kernels and provides sophisticated extensions.
- Our approach complies with the rigid processing requirements of data streams: We process each element once in constant time while building an estimator online. The estimator allocates a constant amount of memory and dynamically adapts to changing system resources. It can keep pace with evolving streams by emphasizing recent data with the help of exponential smoothing.
- For the core operation of the algorithm, the merge of two adjacent kernels, we introduce a new cost measure. It guarantees optimum merge kernels and is inexpensive to compute.
- We complement our analytical results by a discussion of efficient implementation techniques. A tree-based implementation ensures that the processing cost is logarithmic in the size of the allocated memory; it also allows an efficient evaluation of an estimator.
- A thorough experimental study confirms that our estimators perform well for synthetic and real-world data streams and that they are superior to M-Kernels.

The outline of this paper is as follows. In Section 2, we discuss related work. We prepare the ground for our approach by introducing our data stream model, its processing requirements, kernel density estimation, and M-Kernels in Section 3. In Section 4, we discuss the details of our approach. Section 5 presents the results of an experimental study. Finally, we conclude with a summary and an outlook on our future work in Section 6.

2 Related Work

The analysis and mining of transient data streams has come to the fore in recent years. [9] gives a comprehensive overview of arising questions, challenges, and associated techniques for mining data streams. For an adaptation to data streams, mining algorithms have to meet specific processing requirements [8]. Up to now, several core algorithms of data mining were successfully adapted to data streams, e.g. classification [2].

In [1], the authors address the problem of clustering evolving data streams. They use so-called microclusters to periodically store local and temporal summary statistics of the current clusters. The kernel entries we later present also store local statistics to summarize elements, but their application and the underlying summary statistics distinguish both techniques from each other.

Another important aspect is the incorporation of different mining techniques into a stream mining system. StatStream [22] for instance is a tool kit for the simultaneous analysis of multiple time series. Gigascope [7] as another example is a stream database providing mining and analysis techniques for network traffic.

In this work, we specifically tackle the adaptation of kernel density estimation to data streams. A short version of this work sketching the basic idea will be presented in [13]. In comparison to, this work not only thoroughly discusses all details, but also presents concrete implementation concepts and evaluation strategies, complemented by an extensive experimental study.

Generally, density estimation as research topic in mathematical statistics is thoroughly discussed in [20, 21]. In data mining and analysis, it serves as building block for a plethora of probabilistic learning methods [11]. Biased sampling for instance exploits kernel density estimates to maintain its samples [15]. Database research also reaps the benefits of kernel density estimation, e.g. selectivity estimation of range queries [4], [16].

The computational complexity of kernel density estimation renders its application to massive data sets difficult. In order to reduce this complexity, [11] provides an approximate solution for offline data sets. The basic idea is to establish a dual-tree structure: one tree partitions a given set of training data and the other one the query set. A set of query points can be efficiently evaluated with a simultaneous traversal of both trees. Another approximate solution for multidimensional data relies on a multi-pole based algorithm [18]. This technique provides online computable kernel density estimators by maintaining a multivariate Taylor series expansion of the estimator. Concerning their applicability to data streams, both approaches did not address a variable, limited amount of available memory as well as the tracking of evolving streams (see also Section 3.1).

There are also initial approaches for kernel density estimation over data streams. [19] presents a kernel-based solution for the selectivity estimation of range queries over multidimensional spatial streams. Its basic idea is to update local variances with a kd-tree-like structure on top of a continuously maintained sample. This approach has problems with capturing evolving streams. As already mentioned in Section 1, M-Kernels are another technique for computing kernel density estimators over one-dimensional data streams. Due to their relevance for this work, we take a closer look at them in Section 3.5.

3 Preliminaries

In this section, we first present our underlying data stream model and its processing requirements. Then, we give a brief introduction to kernel density estimation. We point out the problems of its adaptation to data streams, followed by a detailed discussion of M-Kernels.

3.1 Data Stream Model

A one-dimensional data stream consists of an unbounded sequence $X_1, X_2, ...$ of numbers with $X_i \in \mathbb{R}$ for $i \in \mathbb{N}$. Except otherwise stated, we assume the stream to be at each time instant a sample with independent and identically distributed (iid) observations of an unknown continuous random variable X. The premise of independence of two arbitrary stream elements is reasonable as data sources often send their elements autonomously, e.g. traffic sensors. The premise of an identical distribution also holds for most applications. The case of a stream whose distribution changes will be separately discussed in Section 4.5.

In general, these assumptions allow us to apply a variety of statistical analysis techniques to determine meaningful models of a stream, e.g. cluster, decision trees.

3.2 Processing Requirements

To keep pace with transient data streams, an online analysis technique has to meet the following stringent processing requirements [8]:

1. Each element is processed only once.
2. The per-element processing time is constant.
3. The amount of allocated memory is constant.
4. A valid model is available anytime.
5. The models incorporate changes in the data stream.
6. The provided models should be equivalent to their offline counterparts.

As the practical applicability of an analysis technique also depends on whether it can be integrated into complex systems, we add another processing requirement:

7. A model can adapt its allocated memory to changing system resources anytime.

With respect to these requirements, we specifically aim to provide kernel density estimators over data streams.

3.3 Kernel Density Estimation

One of the core concepts in mathematical statistics is the **probability density function** (pdf). Essentially, each continuous random variable X has a unique pdf f. A pdf is a positive, real-valued function which integrates to one. It provides a comprehensive summary of X; we know the probability of each possible outcome of X. Hence, the knowledge of f is crucial to the analysis of X. Let for instance X describe the heart rate of a patient. Granted that the associated pdf is known, we can compute amongst other characteristics mean and variance or determine the probability that the heart rate lies above a certain threshold.

In real-world scenarios, however, neither X nor its pdf is known. Typically, we only have observations of X in form of a sample $X_1, ..., X_n$ with $X_i \in \mathbb{R}$ for $i = 1, ..., n$. Density estimation, a core topic in mathematical statistics, attends to this problem and provides suitable methods to estimate a pdf with the help of a representative sample. Parametric approaches assume that f falls in a specific parametric family, e.g. Gaussian densities. In contrast to, nonparametric approaches do not assume any specific form of f. Those approaches are very appealing as they solely base on the sample. Hence, they avoid specifying the wrong parametric family as it can occur with parametric approaches.

A theoretically well-founded and practically approved nonparametric approach is kernel density estimation [21], [20]. A **kernel density estimator** (KDE) with **kernel function** K and **bandwidth** $h^{(n)}$ is defined as

$$\hat{f}^{(n)}(x) := \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h^{(n)}} K\left(\frac{x - X_i}{h^{(n)}}\right), x \in \mathbb{R} \quad (1)$$

for independent and identically distributed observations $X_1, ..., X_n$ drawn from a continuous distribution with unknown pdf f. Essentially, a KDE is the overall sum of 'bumps' centered at each observation X_i. While the bandwidth $h^{(n)}$ determines the width of each bump, the kernel function determines its shape. In the following, we refer to those bumps as **kernels**. Figure 1 displays two KDEs based on different bandwidths for a sample consisting of 7 observations and with the Gaussian kernel as kernel function.

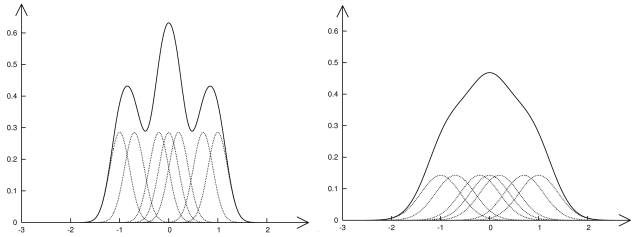

Figure 1: Kernel density estimators and underlying kernels with (left) $h^{(n)} = 0.2$ and (right) $h^{(n)} = 0.4$

The KDEs in this figure indicate that the bandwidth significantly affects the shape of a KDE. In fact, an adequate setting of the bandwidth is of utmost importance for the quality of a KDE and considerable research effort has been spent for the development of appropriate bandwidth strategies [20]. Figure 1 provides a grasp of the effects that arise from varying the bandwidth. If the bandwidth is chosen too low, the KDE is undersmoothed and introduces spurious details. If the bandwidth is chosen too high, the KDE is oversmoothed and hides important details. To guarantee probabilistic convergence, the bandwidth has to decrease with the sample size [21]. For sample sizes converging to infinity, the bandwidths tend to zero. Then, the KDE is a sum of Dirac delta functions at the sample points [21].

In contrast to the bandwidth, the setting of the kernel function is minor [20]. It mostly suffices to choose a kernel function that is a density itself, e.g. Gaussian kernel, Uniform kernel, Epanechnikow kernel. Note that a KDE as sum of kernels inherits among other properties continuity and differentiability from its kernel function. From a

practical point of view, it is advisable to choose a bounded kernel function as it reduces for a given point the number of kernels to evaluate. If the kernel function has an unbounded support, each kernel must be evaluated.

As already mentioned in Section 2, kernel density estimation has become highly relevant in various application scenarios. *"Apart from the histogram, the kernel estimator is probably the most commonly used estimator and is certainly the most studied mathematically."* [21]. Its broad acceptance results from the combination of simplicity with convenient mathematical properties [20]. First, a KDE strictly relies on the sample without a priori distribution assumptions, e.g., membership of a family of standard distributions. Second, a KDE is asymptotically unbiased. Third, a KDE is consistent in terms of the mean integrated squared error, i.e., the more sample points, the better the estimation quality. Fourth, compared to histograms, the common setting in database systems, KDEs have a higher rate of convergence, can produce smooth estimates, and do not need to know the range of the support in advance.

Finally, let us mention that kernel density estimation was also discussed for the case of dependent data [12], but in the context of stochastic processes.

3.4 Kernel Density Estimation over Streaming Data

The aforementioned benefits of kernel density estimation recommend its adaptation to data streams as we could gain meaningful insights into the characteristics of the stream. A suitable adaptation can also serve as foundation for the application of common offline, density-based mining techniques to streams.

Since we assume a data stream to be an iid sample of an unknown continuous distribution (see Section 3.1), the adaptation of kernel density estimation seems straightforward. But the computational cost of KDEs collides with the processing requirements presented in Section 3.2: According to (1), a KDE requests memory linear in the sample size, i.e. in the size of the stream. Even if large amounts of data could be stored, the use of KDEs will become unfeasible due to high evaluation cost. Furthermore, common bandwidth strategies require access to the complete sample, i.e., we must store each processed element. Hence, kernel density estimation in its original form can not be directly applied to data streams.

A naive approach for an adaptation is to continuously maintain a constant-size sample of the already processed elements. Then, one can build a KDE anytime on top of the current sample elements. However, the estimation quality will not significantly improve anymore after the sample has been initialized since the consistency of KDEs presupposes an increasing sample size.

3.5 M-Kernel Approach

In [5], the authors proposed M-Kernels for computing KDEs over one-dimensional data streams. This technique only partially complies with the processing requirements presented in Section 3.1. Essentially, an **M-Kernel** is a kernel with **mean** $X_i^{(n)}$ and bandwidth $h_i^{(n)}$ that is additionally weighted with $c_i^{(n)}$. The overall sum of M-Kernels constitutes the current KDE after n processed stream elements:

$$\hat{f}^{(n)}(x) := \frac{1}{n}\sum_{i=1}^{m}\frac{c_i^{(n)}}{h_i^{(n)}}K\left(\frac{x-X_i^{(n)}}{h_i^{(n)}}\right) \qquad (2)$$

with $\sum_{i=1}^{m} c_i^{(n)} = n$. While n, the number of processed stream elements, continuously increases, the maximum number of M-Kernels is restricted to m. To keep the number of M-Kernels and therefore the amount of allocated memory constant, M-Kernels can be merged. The accuracy loss of a merge is measured with **merge costs** $L1costs_i^{(n)}$. The entirety of M-Kernels $\langle X_i^{(n)}, h_i^{(n)}, c_i^{(n)}, L1costs_i^{(n)}\rangle, i = 1, ..., m$ is organized in a list sorted by $X_i^{(n)}$.

3.5.1 Parameter Settings

The Gaussian kernel is used as underlying kernel function in (2). The bandwidth as second parameter is initially set to 1 for a new M-Kernel, but, as we will see, it changes in case of a merge.

3.5.2 Processing of M-Kernels

Generally, for each incoming element X_{n+1}, a new M-Kernel $< X_{n+1}, 1, 1, L1costs^{(n+1)} >$ is inserted, provided that X_{n+1} is not equal to the mean of an existing M-Kernel. If the latter is the case, i.e., $\exists i \in \{1, ..., m\} : X_i^{(n)} = X_{n+1}$, the associated weight $c_i^{(n)}$ is incremented.

If the total number of M-Kernels exceeds the maximum number m after an insertion, the two M-Kernels $\langle X_i^{(n)}, h_i^{(n)}, c_i^{(n)}, L1costs_i^{(n)}\rangle$ and $\langle X_j^{(n)}, h_j^{(n)}, c_j^{(n)}, L1costs_j^{(n)}\rangle$ *closest to each other* are substituted by their **merge kernel** $\langle X^*, h^*, c_i^{(n)} + c_j^{(n)}, L1costs^{(n)}\rangle$. The mean X^* and the bandwidth h^* of this merge kernel minimize the **merge costs function** $L1costs(X, h)$ which measures the mean absolute deviation between two M-Kernels:

$$L1costs(X,h) := \int_{-\infty}^{\infty}\left|\frac{c_i^{(n)}}{h_i^{(n)}}K\Big(\frac{x-X_i^{(n)}}{h_i^{(n)}}\Big) + \frac{c_j^{(n)}}{h_j^{(n)}}K\Big(\frac{x-X_j^{(n)}}{h_j^{(n)}}\Big) - \frac{c_i^{(n)}+c_j^{(n)}}{h}K\Big(\frac{x-X}{h}\Big)\right|dx. \qquad (3)$$

As it suffices to consider merges between adjacent M-Kernels, an M-Kernel only stores $L1costs_i^{(n)} := L1costs(X^*, h^*)$ which denotes the merge costs with its successor. In case of a merge, the M-Kernel with overall minimum merge costs is merged with its list successor. After a merge, the merge costs of the merge kernel as well as of its left neighbor are updated.

The lack of a closed formula for $L1costs(X, h)$ renders the computation of its minima difficult. The authors propose to overcome this problem with numerical approximations.

3.5.3 Drawbacks of M-Kernels

The following drawbacks of M-Kernels severely limit their applicability:

- The unbounded support of the Gaussian kernel, which is used as underlying kernel function, exacerbates an efficient evaluation of M-Kernels.
- There is no theoretical foundation for the proposed bandwidth computation. It is not ensured that the bandwidth decreases with the sample size; but this is a fundamental prerequisite for the consistency of KDEs [21].
- The numerical approximation of the minima in (3) causes additional computational effort and leads to less accurate values.

4 Our Approach

In the following, we propose our redesign of the M-Kernel approach as well as sophisticated extensions. Particularly with regard to the above drawbacks, we present suitable parameter settings and improvements of essential processing steps. Additionally, we discuss extensions in form of efficient algorithms and a strategy to cope with evolving data streams.

4.1 Kernel Entries and their Processing

Instead of M-Kernels, we use kernel entries for KDEs over streaming data. An M-Kernel - see Section 3.5 - is characterized by $\langle X_i^{(n)}, h_i^{(n)}, c_i^{(n)}, L1costs_i^{(n)} \rangle$. On the contrary, a **kernel entry** is characterized by $\langle X_i^{(n)}, c_i^{(n)}, min_i^{(n)}, max_i^{(n)}, L2costs_i^{(n)} \rangle$. Like M-Kernels, a kernel entry stores mean $X_i^{(n)}$ and weight $c_i^{(n)}$. Additionally, they are equipped with minimum $min_i^{(n)}$ and maximum $max_i^{(n)}$. In comparison to M-Kernels, we use a global bandwidth for all kernel entries and a new definition of merge costs $L2costs_i^{(n)}$.

For an incoming element X_{n+1}, we establish a new kernel entry $\langle X_{n+1}, 1, X_{n+1}, X_{n+1}, L2costs_i^{(n+1)} \rangle$, provided X_{n+1} is not equal to the mean of an existing kernel entry. If this is the case, we only increment the associated weight of this kernel entry.

In case the total number of kernel entries exceeds the current maximum number m, we perform analogous to M-Kernels a merge step. More precisely, we determine and substitute the pair of adjacent kernel entries $\langle X_i^{(n)}, c_i^{(n)}, min_i^{(n)}, max_i^{(n)}, L2costs_i^{(n)} \rangle$ and $\langle X_j^{(n)}, c_j^{(n)}, min_j^{(n)}, max_j^{(n)}, L2costs_j^{(n)} \rangle$ with overall minimum merge costs by their **merge kernel**. The merge kernel is defined as $\langle X^*, c_i^{(n)} + c_j^{(n)}, min\{min_i^{(n)}, min_j^{(n)}\}, max\{max_i^{(n)}, max_j^{(n)}\}, L2costs^{(n)} \rangle$.

Before we discuss the computation of X^*, we present the parameter settings of a KDE based on kernel entries.

4.2 Parameter Settings

Contrary to M-Kernels, which rely on the Gaussian Kernel, we exploit a kernel function with bounded support. Kernel functions with unbounded support lead to high evaluation cost as every kernel contributes to the result (see equation (1)). In this work, we decided to use the **Epanechnikow kernel**:

$$K(x) := 0.75 \cdot (1 - x^2) \cdot 1_{[-1,1]}(x), x \in \mathbb{R}. \quad (4)$$

Not only has this kernel function a simple form, its efficiency is also asymptotically optimal among all kernels [11]. As we will see, the computation of the merge costs as well as the evaluation of a KDE strongly rely on the simple form of this kernel.

As already mentioned in Section 3.3, the bandwidth as second parameter of a KDE is vital to the estimation quality. Theoretically well-founded and practically approved bandwidth strategies [20] often assign a global bandwidth to all kernels. However, these strategies depend on the complete sample, which corresponds in our scenario to all processed stream elements. Thus, the one-pass paradigm is violated. We overcome this problem with an approximate solution that complies with the processing requirements of data streams. We concretely consider a simple but convenient bandwidth strategy, namely the **normal scale rule** [21]. For a sample with n elements and standard deviation $\sigma^{(n)}$, this rule defines the bandwidth as

$$h^{(n)} := 1.06 \cdot \sigma^{(n)} \cdot n^{-\frac{1}{5}}. \quad (5)$$

For the sake of an online computation of $h^{(n)}$, we have to estimate the standard deviation $\sigma^{(n)}$ of the data stream in an online fashion in amortized constant time. A suitable estimate of $\sigma^{(n)}$ is the sample standard deviation, which itself can be estimated in one pass with a numerically stable algorithm presented in [6]. Given this estimate $\hat{\sigma}^{(n)}$, we can compute an approximate global bandwidth $\hat{h}^{(n)}$ while consuming the data stream:

$$h^{(n)} = 1.06 \cdot \sigma^{(n)} \cdot n^{-\frac{1}{5}} \approx \hat{h}^{(n)} := 1.06 \cdot \hat{\sigma}^{(n)} \cdot n^{-\frac{1}{5}}. \quad (6)$$

4.3 Evaluation Strategies

The entirety of kernel entries combined with the upper parameter settings allows us to establish a KDE anytime.

Analogous to the M-Kernel approach, we define the current KDE after n processed elements as

$$\hat{f}^{(n)}(x) := \frac{1}{n} \sum_{i=1}^{m} \frac{c_i^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x - X_i^{(n)}}{\hat{h}^{(n)}}\right). \quad (7)$$

In the following, we refer to this evaluation strategy as **one-value-evaluation**. For infinite data streams, this strategy is not suitable as the resulting KDE will become a sum of m Dirac delta functions, i.e., it will consist of m singularities $X_i^{(n)}, i = 1, ..., m$. This is the consequence of the bandwidth tending to zero for increasing sample sizes.

We circumvent this problem with our **min-max-evaluation** strategy. Remember that each kernel entry stores a counter $c_i^{(n)}$ for the number of 'incorporated' elements and, additionally, the elements' minimum $min_j^{(n)}$ and maximum $max_j^{(n)}$. We gain advantage of these local statistics by evaluating $c_i^{(n)}$ elements equidistantly distributed over $[min_i^{(n)}, max_i^{(n)}]$ for each kernel entry:

$$\hat{f}^{(n)}(x) = \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{c_i^{(n)}} \frac{1}{\hat{h}^{(n)}} K\Big(\frac{1}{\hat{h}^{(n)}}\Big(x - min_i^{(n)} - (j-1)\frac{max_i^{(n)} - min_i^{(n)}}{c_i^{(n)} - 1}\Big)\Big). \quad (8)$$

This strategy ensures that the complete data range will be covered with elements, i.e., it has an inherent smoothing. Due to that smoothing, the problem of singularities for infinite data streams can not occur anymore. A necessary requirement for the application of this strategy is that the inner sum in (8) can be converted into a closed formula; otherwise, the overall evaluation cost would be $O(n)$. The Epanechnikow kernel allows us to determine a closed formula with rather simple algebraic conversions. Due to space constraints, we have to omit a detailed discussion.

Besides the evaluation, the Epanechnikow Kernel also plays an important role in the computation of the merge costs.

4.4 Merge Costs Computation

Our definition of the merge costs between two kernel entries also bases on the inherent objective in (2): Consider the sum of two kernels weighted with $c_i^{(n)}$ and $c_j^{(n)}$ over means $X_i^{(n)}$ and $X_j^{(n)}$ respectively. Their merge kernel is the kernel with weight $c_i^{(n)} + c_j^{(n)}$, whose suitably chosen mean ensures an optimum approximation of the sum. Thus, the means and the weights are the crucial factors during merge. For the sake of completeness, let us mention that, given the min-max-evaluation strategy, we could additionally incorporate the generated points in the merge costs computation; but this would complicate the subsequent computations massively.

In the following, we examine the mean squared deviation, a common measure for the similarity of real-valued functions, to quantify the accuracy loss induced by the merge of two kernel entries:

$$\begin{aligned} L2costs(X) &:= \int_{-\infty}^{\infty} \Big(\frac{c_i^{(n)}}{\hat{h}^{(n)}} K\Big(\frac{x - X_i^{(n)}}{\hat{h}^{(n)}}\Big) + \frac{c_j^{(n)}}{\hat{h}^{(n)}} K\Big(\frac{x - X_j^{(n)}}{\hat{h}^{(n)}}\Big) - \frac{c_i^{(n)} + c_j^{(n)}}{\hat{h}^{(n)}} K\Big(\frac{x - X}{\hat{h}^{(n)}}\Big)\Big)^2 dx \\ &= \frac{0.75}{\hat{h}^{(n)}} \int_{-\infty}^{\infty} \Big(c_i^{(n)}\Big(1 - \Big(\frac{x - X_i^{(n)}}{\hat{h}^{(n)}}\Big)^2\Big) 1_{[-1,1]}\Big(\frac{x - X_i^{(n)}}{\hat{h}^{(n)}}\Big) \\ &\quad + c_j^{(n)}\Big(1 - \Big(\frac{x - X_j^{(n)}}{\hat{h}^{(n)}}\Big)^2\Big) 1_{[-1,1]}\Big(\frac{x - X_j^{(n)}}{\hat{h}^{(n)}}\Big) \\ &\quad - (c_i^{(n)} + c_j^{(n)})\Big(1 - \Big(\frac{x - X}{\hat{h}^{(n)}}\Big)^2\Big) 1_{[-1,1]}\Big(\frac{x - X}{\hat{h}^{(n)}}\Big)\Big)^2 dx \end{aligned} \quad (9)$$

with X as the *only* variable. The M-Kernel approach, in contrast, includes the bandwidth as second variable at the expense of a more complicated (and inaccurate) computation of the minimum. As mean of the merge kernel, we set the minimum X^* of $L2costs(X)$. Thus, we minimize the accuracy loss of the merge: the merge is optimum with respect to the mean squared deviation. It is important to note that the closer two kernels are with respect to their means, the smaller are their merge costs. Hence, it suffices to consider only the merge of adjacent kernel entries. For that reason, we define $L2costs_i^{(n)} := L2costs(X^*)$ as cost of merging the i-th and the $(i+1)$-th kernel entry.

For illustrative purposes, we present in Figure 2 the shape of $L2costs(X)$ for two kernel entries. The left y-axis describes the weighted kernels and the right one $L2costs(X)$, while the x-axis describes the support of the kernel functions as well as the possible means of the merge kernel.

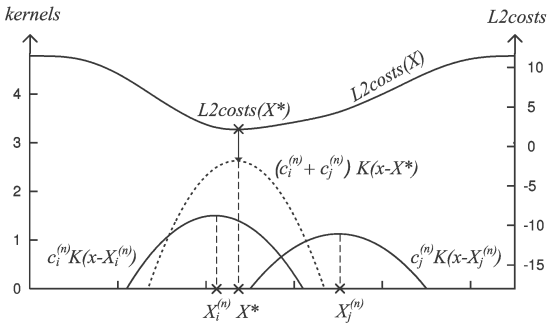

Figure 2: Merge of kernel entries with $\hat{h}^{(n)} = 1$

Concerning the existence and the computation of the minimum of $L2costs(X)$, the following theorem holds:

THEOREM 1. *For two arbitrary kernel entries, the minimum of $L2costs(X)$ exists and can be computed in constant time.*

In the following proof, we only present the essential steps because the computation of the minimum is rather technical. In the proof, we benefit again from the simple form of the Epanechnikow kernel.

PROOF. We consider two arbitrary kernel entries with means X_i and X_j. In case of $X_i = X_j$, the minimum of $L2costs(X)$ equals X_i. In the following, we assume $X_i < X_j$ without loss of generality.

The minimum of $L2costs(X)$ corresponds to the roots of its first derivative. To compute the first derivative, we have to transform (9) into a closed formula via integration. We start with converting the integral of the squared sum in (9) into a sum of integrals over separate products. We refer to the resulting integrals as **summands**. Their integrands are products of two kernels with means $X, X_i^{(n)}$, or $X_j^{(n)}$ and supports $[X - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$, $[X_i^{(n)} - \hat{h}^{(n)}, X_i^{(n)} + \hat{h}^{(n)}]$, or $[X_j^{(n)} - \hat{h}^{(n)}, X_j^{(n)} + \hat{h}^{(n)}]$ respectively. Hence, the intersection of the associated pair of supports determines the integration borders of a summand. However, each summand that incorporates X has integration borders varying in X. Thus, the integration of the summands is not straightforward due to the dependency on the variable X. We overcome this problem by partitioning the support of $L2costs(X)$ appropriately, so that the integration borders of each summand are uniquely defined for an arbitrary X.

For this partitioning, we examine the relative position of the kernel supports $[X_i^{(n)} - \hat{h}^{(n)}, X_i^{(n)} + \hat{h}^{(n)}]$ and $[X_j^{(n)} - \hat{h}^{(n)}, X_j^{(n)} + \hat{h}^{(n)}]$ to each other. Given $[X - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$, we distinguish whether this support intersects both other supports. We compute the largest k with $X_j^{(n)} - X_i^{(n)} \geq k\hat{h}^{(n)}$, i.e. $k = \lfloor (X_j^{(n)} - X_i^{(n)})/\hat{h}^{(n)} \rfloor$. If $k \in \{0, 1, 2, 3\}$, the supports intersect. Otherwise, they do not.

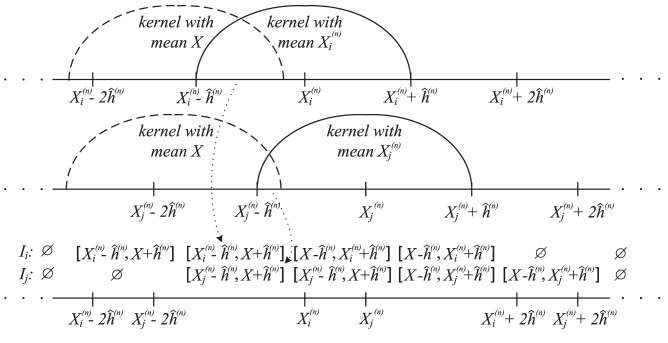

Figure 3: Integration borders for $k = 0$ based on the support partitioning for $L2costs(X)$

In case of $k \notin \{0, 1, 2, 3\}$, the mean of the merge kernel is the mean of the kernel with higher weight. If $c_i^{(n)} = c_j^{(n)}$, $X_i^{(n)}$ and $X_j^{(n)}$ both minimize (9), i.e., we can set the mean either to $X_i^{(n)}$ or $X_j^{(n)}$.

In case of $k \in \{0, 1, 2, 3\}$, let I_i be the integration interval of a summand whose integrand is the product of kernels over X and $X_i^{(n)}$ (I_j analogous with X and $X_j^{(n)}$). In order to determine I_i and I_j, we slide a kernel with continuously varying support $[X - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$ over the x-axis and examine simultaneously the effects on I_i and I_j by evaluating the intersection of the supports, i.e. $[X - \hat{h}^{(n)}, X + \hat{h}^{(n)}] \cap [X_i^{(n)} - \hat{h}^{(n)}, X_i^{(n)} + \hat{h}^{(n)}]$ and $[X - \hat{h}^{(n)}, X + \hat{h}^{(n)}] \cap [X_j^{(n)} - \hat{h}^{(n)}, X_j^{(n)} + \hat{h}^{(n)}]$. For $k = 0$, Figure 3 displays the different cases of intersections, the resulting support partitioning for $L2costs(X)$, as well as the integration intervals I_i and I_j within each support partition. For $X \in [X_j^{(n)} - 2\hat{h}^{(n)}, X_i^{(n)}]$, the computation of I_i and I_j is illustrated as an example. The support partitions for $k = 1, 2, 3$ can be derived analogously.

Eventually, the support partitioning delivers a piecewise definition of $L2costs(X)$, so that the integration interval of each summand in (9) is uniquely defined. Given the primitive of the Epanechnikow kernel, we integrate $L2costs(X)$ on each support partition and finally receive a closed formula. Actually, it is a polynomial of degree 5 in each partition. We determine the minima of those polynomials by calculating the roots of their derivatives, which requires to solve quartic equations. Then we compare the 'local' minima in each partition and get the uniquely defined overall minimum of $L2costs(X)$.

Overall, the support partitioning as well as the subsequent computation of the minimum requires constant time. □

Now that we presented the main components of our KDEs over data streams, we will discuss a convenient extension that allows us to keep pace with evolving data streams.

4.5 Capturing evolving Streams

An inherent difficulty for stream analysis techniques is that the characteristics of a data stream can vary over time. Financial data, for instance, has typically a more volatile nature rather than being stable over time. In the following, we show how an optional extension of our technique allows us to focus on recent trends of an evolving stream.

Let us examine an evolving stream from a formal point of view. Remember that we consider a data stream as sample of an unknown random variable (see Section 3.1). Up to now, we assumed the stream to be stable, i.e., all stream elements follow the same distribution. In case of evolving streams, their underlying distribution will change over time and, as a consequence, also the underlying pdf we want to estimate. In order to emphasize current trends in the density estimate, we couple our online KDEs with **exponential smoothing** [10], a weighting scheme from the area of time series analysis and forecasting.

The basic idea of this coupling is to give older data less weight in the evaluation of an online KDE. For reasons of simplicity, we limit the following considerations to the one-value-evaluation strategy. Let us consider the current

KDE after an insertion of a new element X_n and before the merge step is performed (the element shall not be a duplicate):

$$\hat{f}^{(n)}(x) = \frac{1}{n}\sum_{i=1}^{m} c_i^{(n-1)} \frac{1}{\hat{h}^{(n)}} K\left(\frac{x - X_i^{(n-1)}}{\hat{h}^{(n)}}\right) + \frac{1}{n} \cdot \frac{1}{\hat{h}^{(n)}} K\left(\frac{x - X_n}{\hat{h}^{(n)}}\right) \quad (10)$$

for $n \geq 2$ and $\hat{f}^{(1)}(x) = \frac{1}{\hat{h}^{(1)}} K\left(\frac{x - X_1}{\hat{h}^{(1)}}\right)$. Hence, each kernel entry is equally weighted with $1/n$. With exponential smoothing, these equal weights are substituted by exponentially decreasing ones. Concretely, given $\alpha \in (0, 1)$, each new element receives weight α and triggers a re-scaling of older weights by a factor $(1 - \alpha)$:

$$\hat{f}_\alpha^{(n)}(x) = (1-\alpha)\hat{f}_\alpha^{(n-1)}(x) + \alpha \frac{1}{\hat{h}^{(n)}} K\left(\frac{x - X_n}{\hat{h}^{(n)}}\right) \quad (11)$$

for $n \geq 2$ and $\hat{f}_\alpha^{(1)}(x) = \frac{1}{\hat{h}^{(1)}} K\left(\frac{x - X_1}{\hat{h}^{(1)}}\right)$.

For illustration purposes, we assume the maximum number m of kernel entries to be unbounded and determine the resulting weighting sequence. For $n \geq 2$ holds

$$\hat{f}_\alpha^{(n)}(x) = (1-\alpha)^{n-1} \cdot \frac{1}{\hat{h}^{(n)}} K\left(\frac{x - X_1}{\hat{h}^{(n)}}\right) + \sum_{i=2}^{n-1} (1-\alpha)^{n-2}\alpha \cdot \frac{1}{\hat{h}^{(n)}} K\left(\frac{x - X_i}{\hat{h}^{(n)}}\right) + \alpha \cdot \frac{1}{\hat{h}^{(n)}} K\left(\frac{x - X_n}{\hat{h}^{(n)}}\right). \quad (12)$$

Instead of equal weights $\frac{1}{n}$ for each kernel, we now have 'discounted' weights $(1-\alpha)^{n-2}\alpha$ and $(1-\alpha)^{n-1}$ for older kernels and α for the new kernel. Generally, the weighting sequence sums to 1 for each $n \in \mathbb{N}$. This is a necessary prerequisite for each KDE as otherwise the integration of the pdf to 1, a fundamental property, is violated. M-Kernels, for example, also support a weighting by means of a fade-out function [5], but the resulting weighting scheme violates this prerequisite.

An aspect not yet discussed is the setting of α. With α, we can control the impact of old and new data respectively. The higher α is set, the higher recent data is weighted in the current KDE. The lower it is set, the higher older data is weighted.

Let us mention that this weighting scheme models a kind of 'smooth' sliding window. Sliding windows are a popular technique in data stream processing [3] where queries are often answered with respect to recent data as older data typically can not be stored due to limited system resources. In contrast to common sliding windows, where older elements are 'abruptly' discarded, we smoothly fade them out.

4.6 Resource-awareness

To run within a system with limited resources, the resource-awareness of an analysis technique is a crucial factor. Complex systems for the analysis of multiple data streams will run many analysis tasks simultaneously, given only a limited amount of computational resources. Hence, the techniques that address these tasks must be able to adapt to changing resources. Our technique meets this requirement due to its inherent mechanisms for a seamless resource adaptation. Since evaluation as well as storage cost are defined in terms of kernel entries, we only discuss the adaptation to a changing amount of available memory. If the available amount of memory increases, we establish new kernel entries for new elements - except duplicates - without merging until the new maximum capacity is reached. If the memory decreases, we simply perform the merge step sufficiently often to reduce the total number of kernel entries until they fit in the available memory.

4.7 Implementation Aspects

For practical purposes, we present two suitable implementations of our approach. While one implementation bases on a sorted list, the other one bases on trees. Besides the underlying data structures, their policies for the update of the merge costs distinguish the implementations from each other.

4.7.1 List-based Implementation

Similar to M-Kernels, we can organize the entirety of kernel entries $\langle X_i^{(n)}, c_i^{(n)}, min_i^{(n)}, max_i^{(n)}, L2costs_i^{(n)} \rangle, i = 1, ..., m$, in a linked list sorted by mean $X_i^{(n)}$.

Insertion Step

If a new arriving element equals the mean of an existing kernel entry, we increment its weight. If not, we insert a new kernel entry in compliance with the ordering by mean.

A new element also affects our bandwidth setting due to the dependency on the number of processed elements. As the bandwidth is part of our merge costs function - see (9) - the merge costs of all kernel entries are also affected. Consequently, we have to update all merge costs for each new element. Wile performing this update, we can determine the current kernel entry with overall minimum merge costs to simplify the merge step.

Merge Step

In case of a merge due to an exceeded maximum capacity m, we substitute the adjacent kernel entries with overall minimum merge costs by their merge kernel. The merge kernel is located between its associated kernel entries, i.e., the merge step does not violate the list ordering. After the merge, we update the merge costs between the merge kernel and its left neighbor (if existent) and that between the merge kernel and its right neighbor (if existent).

Algorithm Analysis

The insertion of a new element has complexity $O(m)$ due to the update of all merge costs. If the kernel entry with

minimum merge costs is determined, the merge step has $O(1)$. Overall, the complexity of the list-based implementation is $O(m)$.

4.7.2 Tree-based Implementation

The tree-based implementation has a substantially lower complexity compared to the list-based one. This implementation is approximate because we do not recompute the merge costs of all kernel entries after an update of the bandwidth; we only recompute the merge costs of those entries that are 'locally' affected by an insertion or a merge.

Let us examine the requirements for an efficient processing of a set of kernel entries. On the one hand, an ordering by mean would facilitate the efficient insertion and search of kernel entries. On the other hand, an ordering by merge costs would facilitate the efficient detection of the kernel entry with minimum merge costs. We satisfy both requirements simultaneously with a data structure consisting of a binary search tree and a priority search tree. We store the kernel entries in a binary search tree (**mean tree**) with the mean $X_i^{(n)}$ as underlying ordering criterion. Additionally, we maintain a priority search tree termed **merge costs tree** with entries $\langle L2costs_i^{(n)}, X_i^{(n)} \rangle$ and the merge costs as ordering criterion.

Insertion Step

For a new element, either an already inserted kernel entry will be updated or a new entry is inserted into the mean tree. In both cases, we recompute the merge costs between the associated kernel entry and its predecessor (if existent) and that between this entry and its successor (if existent). In order to keep both trees consistent, we remove the associated 'old' merge costs from the merge costs tree and insert the new merge costs.

Merge Step

If the overall number of kernel entries exceeds m after an insertion, we merge the adjacent kernel entries with minimum merge costs. We remove the minimum merge costs from the merge costs tree and determine the associated kernel entry in the mean tree. In compliance with the mean tree ordering, we substitute this kernel entry by its merge kernel and remove its successor. Finally, we update the merge costs between the merge kernel and its predecessor (if existent) and that between the merge kernel and its new successor (if existent). While doing so, we keep the merge costs tree consistent by removing and inserting the associated merge costs.

Algorithm Analysis

The insertion of a new element as well as the merge step both have complexity $O(log\, m)$. Hence, the overall performance of the tree-based implementation is $O(log\ m)$ per stream element, compared to $O(m)$ of the list-based one.

Comparison with list-based Implementation

Due to the upper procedure, only kernel entries locally affected by an insertion or a merge receive an update of their merge costs with respect to the current bandwidth. For that reason, the tree-based implementation gives an approximate solution. On the contrary, the list-based implementation recomputes all merge costs in case of an updated bandwidth at the expense of higher processing costs. However, merges are most likely to occur in dense data regions where the probability for new elements will be higher than in sparse regions. For the tree-based implementation follows that the merge costs in these regions are with a high probability up-to-date, i.e., this implementation is virtually self-adaptive. The results of our experimental study showed that the loss in accuracy by this approximation only had minor effects on the overall quality of the resulting KDEs whereas the processing time substantially improved compared to the list-based implementation.

4.7.3 Implementation of Exponential Smoothing

To implement the exponential smoothing, we have to weight a new kernel entry with weight α and rescale the other weights with $(1 - \alpha)$ (see also equation (11)). Thus, smoothed KDEs have insertion cost of $O(m)$ for the list-based and the tree-based implementation. This deteriorates the processing cost of tree-based KDEs from $O(log\, m)$ to $O(m)$.

5 Experimental Evaluation

We scrutinized our approach in a thorough experimental study whose core results are presented in the following. With the experiments, we primarily addressed the following questions: How do our KDEs perform for different real-world data streams? How is their runtime behavior in terms of processing time? How do they react to sudden changes of their available amount of memory?

5.1 Settings

5.1.1 Techniques

According to Section 4, we can construct different variants of KDEs with our approach. On the one hand, we can use list-based KDEs, and, on the other hand, tree-based ones. For both, we can apply the one-value-evaluation or the min-max-evaluation strategy. As list-based KDEs with min-max-evaluation did not significantly differ from the other techniques, we do not present their results for the sake of clarity. In order to get an impression of the performance of our KDEs, we included M-Kernels as competitive technique in our experiments. In the subsequent charts, we associated each technique with a specific line type as displayed in Figure 4. All techniques were implemented with PIPES [17], our Java library for advanced data stream processing and analysis.

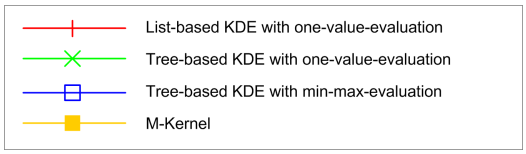

Figure 4: Line types of the techniques

5.1.2 Data Sets

In order to assess these techniques, we considered synthetic as well as real-world data streams. We chose a set of heterogeneous real-world data streams from the time series archive of UC Riverside [14]: BURSTIN, NETWORK, FLUID_DYNAMICS, PACKET, and POWER_DATA. Those data streams originate from diverse fields like facility monitoring or networking and exhibit different characteristics, e.g. noisy/smooth, stationary/non-stationary. We additionally included a synthetic data set, called Claw, whose underlying density is a mixture of Gaussian densities.

5.1.3 Quality Measure

While processing a data stream, we continuously evaluated the quality of the current KDE $\hat{f}^{(n)}$ by comparing it with the best offline KDE $\hat{f}_{opt}$. For the real-world streams, the best offline KDE uses the Epanechnikow kernel and the normal scale rule and relies on the complete stream. For the synthetic stream, the best offline KDE refers to the true density. We measured the quality of $\hat{f}^{(n)}$ with respect to $\hat{f}_{opt}$ in terms of the mean squared error:

$$MSE(n) := \frac{1}{500} \sum_{i=1}^{500} \left(\hat{f}_{opt}(x_i) - \hat{f}^{(n)}(x_i) \right)^2 \quad (13)$$

where $x_1, ..., x_{500}$ is an equidistant partition of the support of $\hat{f}_{opt}$.

5.2 Estimation Quality

An important question is whether our KDEs 'converge' in terms of a decreasing MSE for an increasing number of processed elements, i.e., the better the quality the more elements are processed? In order to answer this question, we continuously compared $\hat{f}^{(n)}$ and $\hat{f}_{opt}$ by evaluating the current MSE always after 500 elements had been processed. Generally, the techniques were allowed to store 100 M-Kernels and 100 kernel entries respectively. Figure 5 displays the experimental results for the different data streams. It is worth mentioning that for the case of CLAW, where we compared with the *true* density, the same trends hold. Specifically, we observed the following trends:

5.2.1 Performance of our KDEs

The results indicate that our KDEs are very robust since the MSE decreased for an increasing number of processed elements. In a few cases, the quality temporarily worsened, indicated by an increased MSE. This can be explained with the temporary emphasis on features that receive less weight in the best offline KDE. Overall, our KDEs achieved excellent rates of convergence and succeeded in estimating the densities underlying the examined data streams.

5.2.2 Performance of M-Kernels

M-Kernels were clearly inferior to our KDEs; they mostly failed to capture the unknown density. M-Kernels exhibited a mostly constant MSE with high absolute values, i.e., they did not improve anymore. A closer examination revealed that they basically suffered from an inappropriately chosen bandwidth. This mostly induced an oversmoothed estimation that hid important details.

5.2.3 List- vs. tree-based KDEs

While list-based KDEs ensure that the kernel entries are always up-to-date with respect to the bandwidth, tree-based KDEs only update locally affected kernel entries. However, the differences in quality between tree- and list-based KDEs were only marginal; their performance was almost identical. If we take the higher processing costs of list-based KDEs into account, we can state that tree-based KDEs are the better choice for practical purposes.

For tree-based KDEs, we additionally examined the min-max-evaluation strategy. For BURSTIN, this strategy was superior to one-value-evaluation. We traced this effect back to the smoothing of this strategy in sparse data regions.

5.3 Processing Time

An aspect of utmost importance is the processing time of an online technique. We provide a notion of the computational complexity of the examined techniques by comparing the time they required for processing a complete data stream. We set the parameters as in the last experiment and measured the time in seconds while the stream was processed. Figure 6 displays the results. They indicate that M-Kernels had the longest processing time due to the computational effort for the numerical approximation of the mean of the merge kernel. Our list-based KDEs were faster than M-Kernels. However, both were clearly inferior to tree-based KDEs. This effect results from the logarithmic cost for an insertion whereas linear costs arise for list-based KDEs.

Another aspect we examined is the influence of the evaluation strategies - see Section 4.3 - on the processing time. There, the differences between tree-based KDEs with one-value-evaluation and with min-max-evaluation were marginal.

5.4 Resource-awareness

We emphasized in this work the necessity of resource-awareness which is a fundamental prerequisite for the use of an online analysis technique within a complex system.

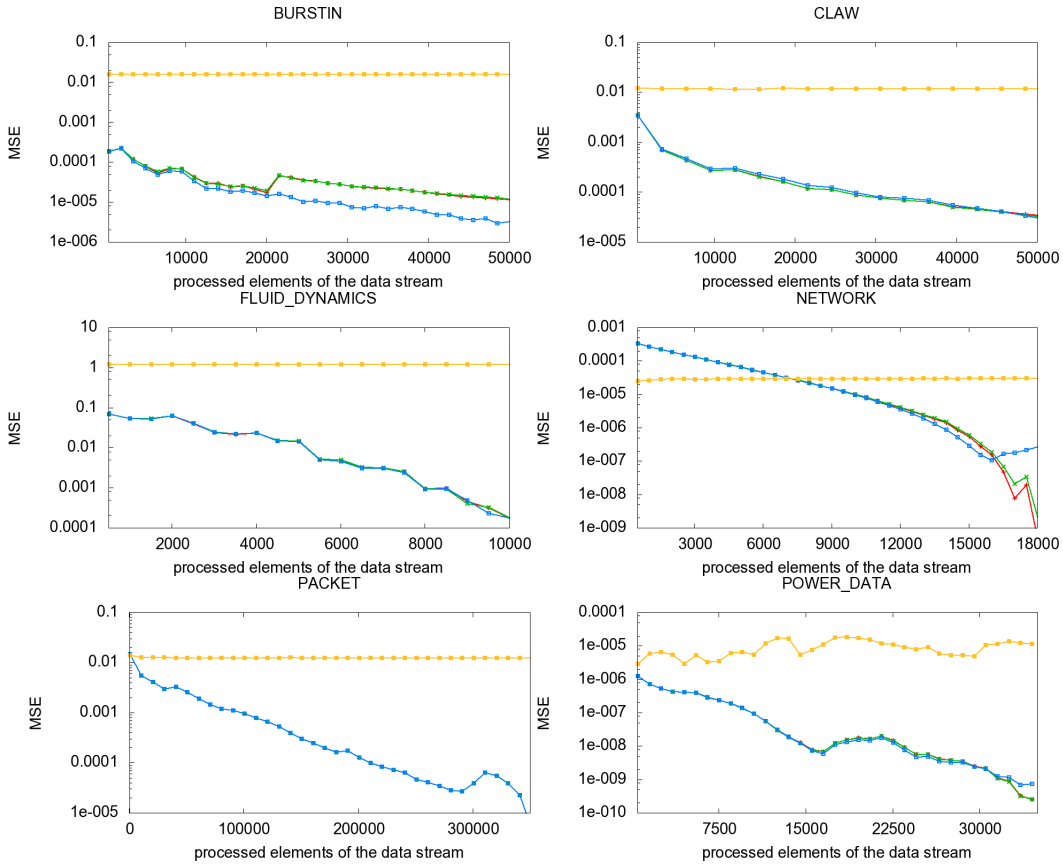

Figure 5: Logarithmically scaled MSE for different data streams

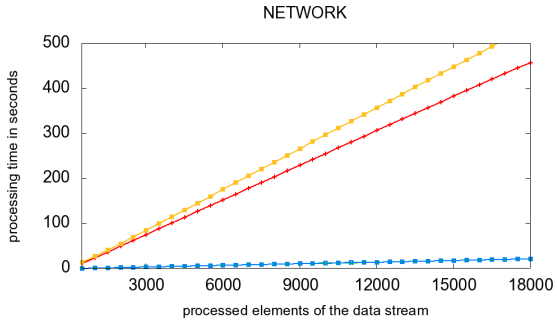

Figure 6: Processing time in seconds

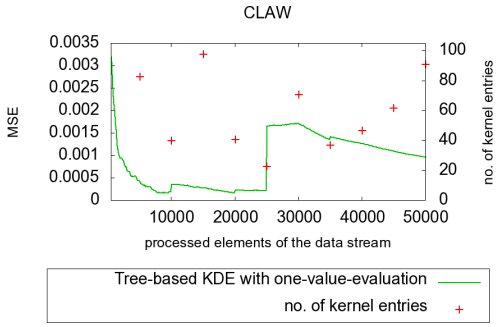

Figure 7: MSE of tree-based KDE for a changing number of kernel entries

For that reason, we examined how our KDEs react to sudden changes of their available amount of memory. We studied the arising effects for tree-based KDEs with one-value-evaluation over a stream of Claw data. While processing the stream, we randomly varied the maximum number of kernel entries from minimum 10 to maximum 100 each 5000 processed elements. By examining the continuously computed MSE, we can study the impact of those memory modifications on the quality of the KDEs.

Figure 7 summarizes the results of this experiment. While the x-axis displays the number of processed elements, the left y-axis displays the MSE and the right y-axis the number of kernel entries. The curve plots the MSE and the crosses depict the current number of kernel entries. We observe that our KDEs react very flexible to changes of the maximum number of kernel entries. Note that even significant decreases of the kernel entry number only caused a momentary loss in accuracy; afterward the KDEs 'recovered' again, indicated by a henceforth decreasing MSE.

6 Conclusions

In this work, we tackled the adaptation of kernel-based density estimation to the data stream scenario in compliance with rigid processing requirements. Kernel density estimation is among the most appealing nonparametric estimation

techniques in statistics and its adaptation to data streams provides a sophisticated base for further stream analysis.

We proposed a new solution whose basic idea is to summarize processed elements with simple statistics. These statistics are stored in kernel entries which are the essential building blocks of an estimator. An intelligent merge scheme for those kernel entries allows us to adapt to changing system resources. To emphasize recent trends and drifts in the stream, we presented an optional weighting strategy to fade out the weight of older data. Besides these basic principles of our technique, we discussed suitable implementations. While the list-based estimator has the highest accuracy at the expense of linear processing cost, the tree-based estimator has logarithmic processing cost at the expense of a slight inaccuracy. Our experimental results for real-world streams indicate that both methods provide a high degree of accuracy which improves constantly the more elements they process. In comparison to M-Kernels, a previous kernel method for data streams, our tree-based estimators were superior as they combined much higher accuracy (on average two orders of magnitude) with substantially lower processing cost (also roughly two orders of magnitude).

In our future work, we will generalize our approach to multidimensional data streams. This requires to develop new data structures which support an efficient storage of kernel entries as well as a fast evaluation of the estimator. Another aspect we will address is the coupling of our technique with change point detection methods as known from stochastic process theory in order to locate and react to concept drifts in the stream.

Acknowledgements
This work has been supported by the German Research Society (DFG) under grant no. SE 553/4-3.

References

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

[2] C. Aggarwal, J. Han, J. Wang, and P. Yu. On Demand Classification of Data Streams. In *Proc. KDD*, 2004.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *PODS*, 2002.

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

[5] Z. Cai, W. Qian, L. Wei, and A. Zhou. M-Kernel Merging: Towards Density Estimation over Data Streams. In *Proc. of DASFAA*, 2003.

[6] T. F. Chan, G. Golub, and R. LeVeque. Algorithms for Computing the Sample Variance: Analysis and Recommendations. *The American Statistician*, 37, 1983.

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

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

[9] M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *SIGMOD Record*, 34(2), 2005.

[10] I. Gijbels, A. Pope, and M. Wand. Automatic forecasting via exponential smoothing: Asymptotic properties, 1997.

[11] A. Gray and A. W. Moore. Nonparametric Density Estimation: Toward Computational Tractability. In *Proc. of ICDM*, 2003.

[12] P. Hall, S. N. Lahiri, and Y. K. Truong. On bandwidth choice for density estimation with dependant data. *Annals of Statistics*, 23, 1995.

[13] C. Heinz and B. Seeger. Resource-Aware Kernel Density Estimators over Streaming Data. In *Proc. of CIKM (to appear in)*, 2006.

[14] E. Keogh and T. Folias. The UCR Time Series Data Mining Archive. `www.cs.ucr.edu/~eamonn/TSDMA`, 2002.

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

[16] F. Korn, T. Johnson, and H. V. Jagadish. Range Selectivity Estimation for Continuous Attributes. In *Proc. of SSDBM*, 1999.

[17] J. Krämer and B. Seeger. PIPES - A Public Infrastructure for Processing and Exploring Data Streams. In *Proc. of ACM SIGMOD*, 2004.

[18] C. Lambert, S. Harrington, C. Harvey, and A. Glodjo. Efficient on-line nonparametric kernel density estimation. *Algorithmica*, 25(1), 1999.

[19] C. M. Procopiuc and O. Procopiuc. Density Estimation for Spatial Data Streams. In *Proc. of SSTD*, 2005.

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

[21] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

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