Density Estimation over Data Streams

Dissertation
zur
Erlangung des Doktorgrades
der Naturwissenschaften
(Dr. rer. nat.)

dem
Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
vorgelegt von

Christoph Heinz
aus Weilburg

Marburg/Lahn 2007

Vom Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
als Dissertation am 12. April 2007 angenommen.
Erstgutachter: Prof. Dr. Bernhard Seeger
Zweitgutachter: Prof. Dr. Jörg Sander
Tag der mündlichen Prüfung am 18. April 2007

Abstract

A growing number of real-world applications share the property that they have to deal with transient data arriving in massive volumes, so-called data streams. The characteristics of these data streams render their analysis by means of conventional techniques extremely difficult, in the majority of cases even impossible. In fact, to be applicable to data streams, a technique has to meet rigid processing requirements such as a single pass over the stream and strict limitations on the available memory. With respect to these requirements, an adequate real-time mining and analysis of a data stream has turned out to be a highly demanding task, which can only be tackled by developing tailored techniques. In this work, we delve more deeply into an important building block of many data mining and analysis approaches, namely density estimation, with the objective to adapt it to the data stream scenario. Density estimation, a vital research topic in mathematical statistics, aims to capture the unknown distribution of a real-valued data set by estimating its underlying probability density function. A suitable density estimate can be exploited to gain insight into the main features of the data; it can also prepare the ground for a variety of further analysis tasks. Due to the importance of density estimation, various methods have been developed in recent decades. Among the most promising methods are the nonparametric ones. They stand out from the parametric approaches as they only make very weak assumptions on the data; they let the data speak for themselves. Two important members of the class of nonparametric methods are based on kernels and wavelets. Kernel as well as wavelet density estimators are theoretically founded and practically approved, but their computational burden severely impedes their applicability. In particular, neither kernel nor wavelet density estimators can be computed over real-valued data streams because their computational complexity collides with the processing requirements for streams. However, as density estimation can be a crucial component of data stream analysis, we devoted ourselves to the adaptation of kernel and wavelet density estimators to data streams in compliance with the processing requirements for streams. Our solution for wavelet density estimators, termed compressed-cumulative WDEs, relies on processing blocks of data of the stream, where each data block is associated with a separate estimator. While processing the stream, the block estimators are successively merged into one estimator. Our solution for kernel density estimators utilizes specific building blocks, termed Cluster Kernels, to derive an estimator for the stream on demand. These Cluster Kernels summarize already processed elements in terms of local statistics, which are used for an approximate resampling of elements. Both solutions exhibit a generic design to facilitate the integration of new strategies for their main processing steps. In order to assess them, we conducted an extensive experimental study, including a thorough comparison with competitive techniques. As the results of this study reveal, our techniques outperform all competitors. Overall, they succeed in estimating the unknown density of a data stream.

Zusammenfassung

Eine stetig wachsende Zahl von Anwendungen sieht sich mit der Verarbeitung flüchtiger, in hohen Volumina eintreffender Daten konfrontiert. Aufgrund der Charakteristika dieser sogenannten Datenströme stellt sich deren Analyse mittels konventioneller Techniken als schwierig, wenn nicht gar unmöglich, heraus. Um Datenströme analysieren zu können, muss eine entsprechende Technik rigiden Verarbeitungsanforderungen gerecht werden, zu denen etwa der nur einmalig mögliche Zugriff auf jedes Element als auch die strikte Beschränkung des verfügbaren Speichers zählen. Aufgrund dieser Anforderungen ist die adäquate Verarbeitung und Analyse eines Datenstroms eine äußerst anspruchsvolle Aufgabe, welche die Entwicklung neuer Techniken erforderlich macht. Das Ziel dieser Arbeit ist die Adaption eines Bausteins vieler Mining- und Analyse-Ansätze, nämlich der Dichteschätzung, für den Datenstromkontext. Dichteschätzung selbst ist eine Fragestellung der mathematischen Statistik, die sich der Schätzung der Wahrscheinlichkeitsdichte eines reellwertigen Datensatzes widmet, um dessen unbekannte Verteilung und Charakteristika zu erfassen. Ein geeigneter Dichteschätzer kann überdies als Grundlage für weiterführende Analysen dienen. Der Wichtigkeit der Dichteschätzung wurde in den vergangenen Jahrzehnten mit der Entwicklung entsprechender Schätzverfahren Rechnung getragen. Insbesondere nichtparametrische Verfahren haben sich dabei als von besonderer Bedeutung herausgestellt, da sie, im Vergleich zu parametrischen Verfahren, nahezu keine Annahmen bezüglich der Daten machen. Kernel- und Wavelet-Dichteschätzer zählen zu den nichtparametrischen Verfahren. Beide sind sowohl theoretisch fundiert als auch praktisch bestätigt. Aufgrund ihrer Komplexität sind sie jedoch nicht ohne weiteres auf Datenströmen berechenbar. Da die Dichteschätzung aber eine wichtige Komponente bei der Analyse eines Datenstroms darstellt, widmet sich diese Arbeit speziell der Adaption von Kernel- und Wavelet-Dichteschätzern auf Datenströme unter Berücksichtigung der strikten Verarbeitungsanforderungen von Strömen. Die für Wavelet-Dichteschätzer entwickelte Lösung, genannt compressed-cumulative WDEs, basiert auf einer blockweisen Verarbeitung des Datenstroms, wobei jeder Datenblock mit einem separaten Schätzer assoziiert ist. Diese werden während der Verarbeitung des Datenstroms sukzessive vereinigt. Die für Kernel-Dichteschätzer entwickelte Lösung verwendet spezielle Bausteine, sogenannte Cluster Kernels, um einen Schätzer bei Bedarf zu konstruieren. Cluster Kernels fassen bereits verarbeitete Elemente mittels lokaler Statistiken zusammen, welche wiederum ein approximatives Resampling der Elemente ermöglichen. Beide Techniken sind generisch aufgebaut, um eine einfache Integration neuer Strategien für die wesentlichen Verarbeitungsschritte zu ermöglichen. Im Rahmen einer umfassenden experimentellen Studie wurden beide eingehend untersucht und insbesondere auch mit ähnlichen Techniken verglichen, welche jedoch allesamt unterlegen waren. In der Summe belegen die Ergebnisse dieser Studie die Fähigkeit der vorgestellten Techniken, die unbekannte Dichte eines Datenstroms adäquat zu schätzen.

Contents

Chapter 1

Introduction

In this chapter, we give an introduction to the main topic of this thesis, namely density estimation over data streams. We start in Section 1.1 with a description of the data stream phenomenon. Then we outline how the mining and analysis of data streams can gain advantage from density estimation. In this work, we present sophisticated solutions for density estimation over data streams. Section 1.2 provides a grasp of their general idea and summarizes the main contributions of this thesis. Section 1.3 concludes this chapter with an overview of the organization of this thesis.

1.1 Data Streams and Density Estimation

The objective of this section is to convey a feeling for data streams in general and for the problems an adequate stream analysis faces. In particular, this section sketches the important role density estimation can play in data stream mining and analysis.

1.1.1 Data Stream Phenomenon

A steadily increasing number of real-world applications share the property that they have to handle transient data that arrives at very high rates. These applications span a wide range of fields, including for example stock market analysis, network monitoring, web applications, satellite observation, traffic management [29]. All these applications face the problem that they continuously receive data in massive volumes. The resulting amounts of data they have to process may easily total millions of records per day. For example, in a single day AT&T produces 300 million call records, Google runs more than 200 million searches, and EBay processes around 10 million bids. Not only business applications generate huge amounts of data, but also scientific applications. The data volumes collected by satellites are in the category of gigabytes per day and more. Another example from science is the Large Hadron Collider [4], a particle accelerator and collider currently under construction at CERN, which is expected to generate a data volume of 500 gigabits per second.

The general tendency is that the amount of data produced will continue to increase in the future due to advances in hardware technology and software development. Consider for instance the emerging field of (wireless) sensor networks [143], which monitor physical entities by continuously measuring their current state with sensors. Due to rapidly decreasing

production cost for sensors, their large-scale deployment, accompanied by the installation of the corresponding sensor networks, is becoming more affordable, e.g., use of RFID in supply chain processes.

Overall, a variety of mechanisms and techniques to generate data in an automatic manner have been developed. As the generated data often arrives continuously and at a high rate, it constitutes a *data stream* [16, 71, 118, 30]. A data stream typically exhibits a transient and volatile nature because its elements arrive unpredictably in large volumes, accompanied by sudden changes of their characteristics.

Keeping these characteristics in mind, let us turn our attention to the applications based on data streams and their requirements. At the first sight, these applications make the same demands as those based on "traditional" data: They want to store the data in order to be able to revisit it. They want to execute queries over the stored data, which deliver the requested information about the data. They want to perform mining and analysis tasks, which extract hidden knowledge structures from the data. Hence, traditional database management systems (DBMS) suggest themselves as vehicle to fulfil these needs. A DBMS provides convenient methods to store and retrieve data, complemented by well-defined query facilities to compute exact query answers. Mining and analysis tasks can be placed on top of the DBMS functionality or even be incorporated into it.

However, as a closer look reveals, this approach is unfeasible for massive data streams. To cite [16]: *"Traditional DBMS's are not designed for rapid and continuous loading of individual data items,..."* The major bottlenecks are the large volume of data produced by a data stream as well as the continuous arrival of its elements; the data may accumulate faster than it can be processed. As a consequence, the objective of computing exact query answers with respect to the complete stream typically cannot be accomplished. Instead, solutions that compute approximate query answers in an online fashion have come to the fore recently, e.g. [31]. A large body of work exists on the topic of tailored data stream management systems (DSMS) [103, 13, 36, 5, 39, 127]. A DSMS is designed to cope with massive data streams and to execute the corresponding continuous queries.

Analogous to DBMS related techniques, traditional data mining and analysis techniques fail to deal with data streams in the majority of cases [55, 60, 118]. Their computational complexity collides with the rapid nature of data streams; it is not possible to recompute a model of the complete data stream, e.g. a decision tree, for each new stream element. As time advances, the recomputation of the model would become so expensive that each technique would inevitably lose track of the data stream. Ergo, the characteristics of data streams also necessitate the development of new mining and analysis techniques, which provide their incrementally computed results in an online fashion.

Due to its importance for stream-based applications, let us take a closer look at stream mining and analysis and its specific needs.

1.1.2 Mining and Analysis of Data Streams

To give a feeling for the general idea of data mining and analysis, we start with some quotations. According to [60], *"Data mining is that interdisciplinary field of study that can extract models and patterns from large amounts of information stored in data repositories..."* An alternative definition of data mining is given in [81]: *"Simply stated, data mining refers to extracting or 'mining' knowledge from large amounts of data."* The same source states:

"The major reason that data mining has attracted a great deal of attention in the information industry in recent years is due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge." In fact, data mining and analysis has become a vital and active research field, indicated by the vast number of techniques developed for mining and analysis purposes.

However, as already sketched above, most of these solutions are not applicable to data streams due to their computational complexity. In fact, to be applicable to data streams, a technique has to meet stringent processing requirements [55, 88, 22]. For example, only a single pass over the data is permitted, given only a limited amount of resources. Only techniques that meet these requirements are able to keep pace with a data stream and its abrupt changes. The necessity of keeping pace with the stream, a quasi real-time processing, is of utmost importance since most applications based on data streams must fulfil time-critical needs. Consider for example the monitoring of credit card transaction data. The case of a fraud should be instantly detected by classifying it as an unusual pattern in the transaction data stream. Another example is the monitoring of vital signs of patients in medical environments. In case of an anomalous activity, which may induce a life-threatening condition of the patient, an alert should be immediately triggered. While these two examples refer to online pattern detection, the online maintenance of models of the stream is also of high relevance in real-world applications. In stock market analysis, continuously updated correlation and regression models of financial data streams can help a trader to identify arbitrage opportunities. To react to the current demands of customers in a timely manner, streams of shopping transaction data can be continuously analyzed by means of tree-based classification rules.

As these examples illustrate, a large number of real-world applications can reap the benefits of suitable stream mining solutions. This fact combined with the outlook of even larger data volumes to be processed in future has drawn the attention of the database and data mining research community [60, 118, 64, 8]. Even though stream mining is currently still at an early stage, a set of convenient techniques is already available for the data stream scenario, including for example clustering [78], change detection [18], classification [10], frequent itemsets [43]. In [60], the authors give a comprehensive overview of the current state of stream mining, complemented by a discussion of its major research issues. To get an idea of these research issues, let us list a few of them in accordance with [60]:

- *Handling the continuous flow of data streams:* The question is how non-ending streams of data can be efficiently stored, indexed, and queried.
- *Required result accuracy:* Even though stream mining solutions can only provide approximate results, these results should still be of high quality in terms of low approximation errors.
- *Modeling changes of mining results over time:* To cope with the unsteadiness of data streams, changes in the stream must be detected and adequately incorporated into the mining results.
- *Minimizing energy consumption of mobile devices:* As streams often originate in resource-constrained environments, an analysis technique must take the limited resources of the corresponding devices into account.

- *Integration between DSMS and ubiquitous stream mining approaches:* Stream mining techniques should be coupled with the core functionality of a DSMS in order to unify storage, querying, mining, and reasoning over streaming data within one system.

As these issues underline, stream mining and analysis is a highly challenging field of study and a lot of interesting work remains to be done in future.

In this work, we delve more deeply into one important stream mining and analysis task, namely density estimation over data streams.

1.1.3 Density Estimation over Data Streams

Before we consider density estimation in the data stream scenario, we give an informal motivation for density estimation. The starting point for density estimation is a given set of real-valued data, whose main features and characteristics are to be investigated. Let us assume that the data is associated with a so-called probability density function, which uniquely describes its distribution. The probability density function can be used to extract all characteristics of the data. However, it is typically unknown; only the data is available. To overcome this problem, density estimation methods use the data to construct a suitable estimate of the underlying probability density function. The resulting density estimate is a comprehensive statistical model of the distribution described by the data. The viability of a density estimate results from the fact that it can be utilized for various exploratory analysis tasks, including for example

- *Data visualization:* A visual exploration of a density estimate is far more convenient than the visual exploration of the raw data.
- *Computation of summary measures:* A density estimate can be exploited to compute important summary measures such as mean, variance, standard deviation, and entropy.
- *Feature detection:* A visual and analytic examination of a density estimate can reveal important features of the underlying distribution, e.g. multimodality, skewness.
- *Detection of sparse regions and hot spots:* A density estimate shows the data regions that comprise many elements or only a few elements.
- *Rank of a point:* A density estimate gives the rank of an arbitrary point p, i.e., the percentage of elements dominated by p.

A statistically reliable density estimate can also serve as a building block in data mining and analysis approaches, e.g. biased sampling [99], discriminant analysis [130]. To cite [73]: *"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..."*

The theory of density estimation is a field of active research in mathematical statistics and many convenient techniques have been developed in the last decades. Among the most useful techniques are the nonparametric ones [128, 130, 137, 57]. The term nonparametric refers in this context to the fact that these approaches make only very weak assumptions about the distribution of the data. In essence, they let the data speak for themselves. Research in nonparametric density estimation has produced a large number of density estimators,

including histograms [128], kernel density estimators [130], wavelet density estimators [57]. In contrast to nonparametric approaches, the parametric ones assume that the data belongs to a known parametric family. However, the necessity of knowing the right family of distributions in advance may be a severe obstacle in many scenarios. For that reason, nonparametric techniques are better suited in most cases.

The theoretical foundation of density estimation combined with the numerous opportunities it opens for analysis purposes render its application to real-valued data streams a highly promising approach. It is worth mentioning that the distribution underlying the data stream must be continuous to apply the aforementioned density estimation approaches. Hence, the stream elements must be real-valued. As we will see in Chapter 2, many approaches for stream analysis assume that the elements are from a finite universe, which massively facilitates a lot of computational tasks. However, real-world streams are often per se real-valued, e.g. spatial, temporal, or multimedia streams. Analyzing these methods with discrete methods would require an initial discretization step, which comes at the expense of an additional discretization error. For the case of real-valued streams, a density estimator, continuously computed over a stream, can make a significant contribution to the mining and analysis of the stream; it can serve multiple analysis purposes according to the previous considerations. However, a closer look at common density estimation techniques reveals that their computational complexity collides with the processing requirements for streams. As a consequence, we cannot compute them in their original form over real-valued data streams.

In order to overcome this problem, we addressed in the course of this work the following question: *How can we compute suitable density estimates over real-valued data streams in compliance with the rigid processing requirements for streams?* Since we have no a priori knowledge of the stream characteristics in the majority of cases, we concentrated on the class of nonparametric density estimation methods. More precisely, among the different nonparametric methods, we decided for kernel density estimation and wavelet density estimation; both combine a solid theoretical foundation with high practical relevance.

Example In order to convey an impression of how we can exploit our density estimators for complex stream analysis tasks, let us consider the concrete application scenario we examined in [87]. As data stream within this scenario, we have used a time series from the Times Series Data Mining Archive of UC Riverside [98]. The time series we chose describes the monitoring of vital signs of sleeping patients in a sleep laboratory. We particularly concentrated on the analysis of the heart rate measurements.

The typical method to gain insight into the sleeping disorders a patient exhibits is to analyze the recorded measurements manually in a post-processing phase. As an improvement, we propose to analyze the streams online to get insights while the patient is sleeping. For this purpose, we can utilize our online density estimators. More precisely, we can use them to describe the long-term sleeping behavior of the patient since they rely on the entirety of already processed measurements. To describe the short-term sleeping behavior, we can additionally maintain a density estimate over a sliding time window.

In [87], we presented a prototypical implementation of a heart rate monitor, which makes use of the analysis facilities of our estimators. Let us briefly describe the components of this monitor with the screenshot given in Figure 1.1. To refer to the components, we marked them with numbers. The component marked with (1) displays the incoming heart rate mea-

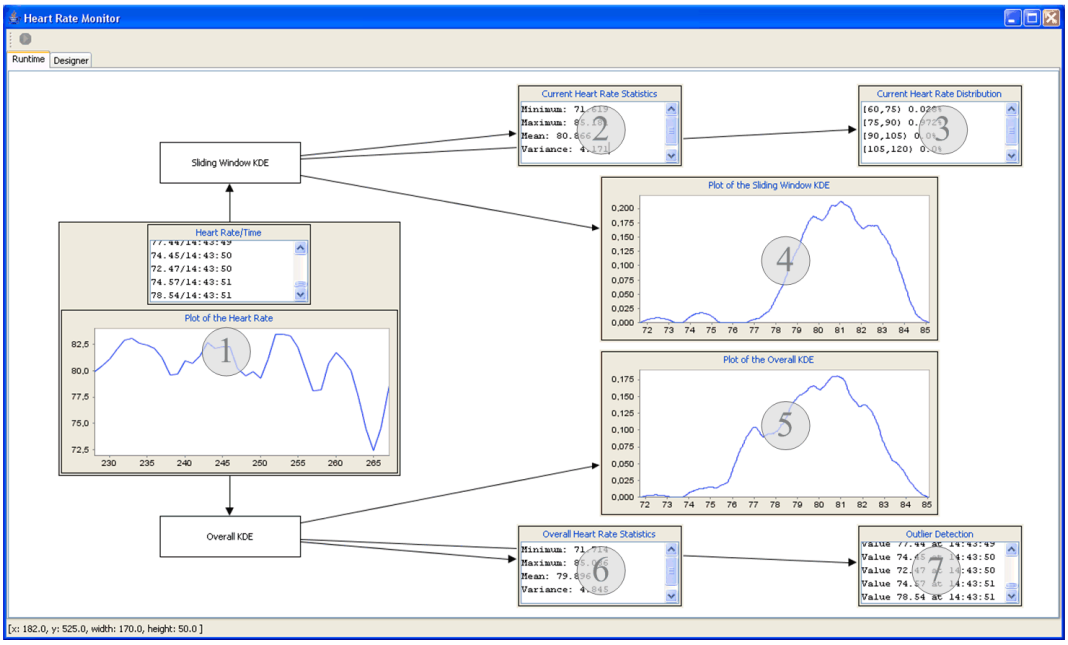


Figure 1.1: Heart Rate Monitor

surements. Based on these measurements, we continuously maintain two density estimators: one to analyze the short-term sleeping behavior and the other one to analyze the long-term sleeping behavior.

The density estimator for the short-term behavior is based on a sliding time window over the last 10 minutes. By means of this density estimator, we compute summary statistics (2) for the current time window. Additionally, we plot the current density estimate (4). In another component (3), we observe how deeply the patient is sleeping: We partition the value range equidistantly and continuously compute the probability for each partition. This shows whether the heart rate was high or low in the last 10 minutes.

To analyze the long-term sleeping behavior, we use our online computable density estimators. Again, we plot the current density estimate (5) and compute the according summary statistics (6). Additionally, we display the measurements labelled as outliers (7) with respect to the current density estimate. In the same context [86], we defined an outlier as a low-density point, i.e., a point is labelled as an outlier if the region it falls into has a probability below a preset threshold.

By means of the heart rate monitor, a medical examiner can get a rough picture of a patient's sleep as well as more detailed insights into his/her current condition. Overall, we see that we can utilize online density estimators to tackle different stream analysis tasks in an online fashion.

1.2 Contributions

The main objective of this work is to provide a convenient answer to the previously posed question, namely how to compute density estimators over data streams. Before we go into

the details of how we answered this question, let us briefly summarize the main contributions of this work.

Wavelet Density Estimators over Data Streams We present a novel approach to adaptive wavelet density estimation over data streams. Not only do we meet the rigid processing requirements for data streams, we also ensure an adaptive allocation of changing system resources. To the best of our knowledge, this is the first adaptation of wavelet density estimators to the data stream scenario.

Our wavelet density estimators are derived from a framework we developed for maintaining arbitrary nonparametric estimators over data streams. The basic idea of the framework is to process the stream in a blockwise manner, where each data block is associated with a separate estimator. While processing the stream, we iteratively compute the convex linear combination of those block estimators, which gives us an overall estimator for the already processed stream. To meet the limitations on the resource allocation, the current overall estimator is additionally compressed. With respect to this procedure, the following questions must be answered to plug an estimation technique into our framework: How is an estimator for a data block defined and which weight does it receive? How can two estimators be merged? How can the overall estimator be suitably compressed so that it fits into the available memory?

We address these questions for wavelet density estimation as the technique that is to be adapted to data streams. To determine a block estimator, we can either use linear or thresholded wavelet density estimators. While the linear ones are particularly suited for the estimation of smooth densities, the thresholded ones are suited for densities exhibiting discontinuities and irregularities. In case a new block estimator has been computed, it is merged with the current overall estimator, which delivers the new overall estimator. Concerning this merge, we present an efficient merge procedure, which makes use of the representation of a wavelet density estimator in terms of specific coefficients. To compress the current overall estimator, we also reap the benefits of the coefficient-based representation and discard those coefficients that represent the least significant details of the estimator. Due to the generic design of our approach, other types of wavelet density estimators can be adapted to data streams in the same manner.

Kernel Density Estimators over Data Streams We introduce Cluster Kernels, our approach to kernel density estimation over streaming data. The design of Cluster Kernels inherently ensures that the processing requirements for data streams are met.

The basic idea of our approach is to maintain a constant number of so-called Cluster Kernels while processing the stream. These Cluster Kernels serve as essential building blocks for the construction of a kernel density estimator for the already processed stream. Each Cluster Kernel represents a kind of local cluster of processed stream elements. More precisely, as the limited resources prohibit the storage of all elements, each Cluster Kernel maintains local statistics which describe the behavior of the data stream in different data localities. In the construction of a kernel density estimator, we use the local statistics of each Cluster Kernel to approximately resample the already processed elements. With regard to the time-critical nature of data stream applications, we present resampling strategies that ensure an evaluation of the kernel density estimator in constant time.

A crucial requirement for Cluster Kernels is that their number remains constant. On

account of this requirement, we provide a sophisticated merge scheme based on a suitable notion of closeness between Cluster Kernels. A vital feature of this merge scheme is that it minimizes the loss of accuracy. With the help of the merge scheme, we can also adapt the amount of allocated memory flexibly to the current system resources.

A major advantage of Cluster Kernels is their flexible and modular design, which facilitates the substitution of essential algorithm components. Hence, we can utilize the Cluster Kernel approach to compute other types of kernel density estimators over data streams. Besides discussing the general approach, we also present concrete strategies and solutions for Cluster Kernels, resulting in ready-to-use kernel density estimators for data streams.

Implementation of Density Estimators over Data Streams We complement the discussion of the principles of our kernel and wavelet density estimators with the presentation of the algorithms we developed for their construction and use. Additionally, we describe suitable data structures which allow us to implement these algorithms efficiently. To provide an easy access to our techniques, we outline how we implemented them in our library XXL.

XXL is a Java-based library which provides complex mechanisms for advanced query processing [28, 136]. An integral part of XXL is PIPES [103], a subpackage which gathers the functionality for processing and exploring data streams. PIPES features an easy integration of new, user-defined functionality, but also offers convenient ready-to-use solutions. It provides the essential components to build and execute multiple continuous queries over data streams. A continuous query is implemented as a directed, acyclic graph, whose nodes are the physical operators of the query. We implemented kernel-based and wavelet-based density estimators as operators within PIPES, i.e., a density estimator is a special operator that receives elements from its data source, processes them, and delivers the current density estimate as result.

These estimates can serve as building blocks for further stream mining and analysis tasks. In case the corresponding analysis technique is also implemented as an operator, we simply connect it with the density estimator operator and from then on, it continuously receives the current density estimate, which can be used for analysis purposes. The fundamental advantage of this method is that a single density estimator can serve multiple analysis techniques for a data stream simultaneously. Consequently, we can unify multiple density-based analysis tasks over a data stream in one operator graph with a density estimator as central node.

Comprehensive Experimental Evaluation To scrutinize our density estimators over data streams, we conducted a broad experimental study. In the experiments, we examined the performance they achieved for a variety of data streams, including real-world streams from heterogeneous application scenarios as well as synthetic streams. The experiments were designed to assess our techniques with respect to the key ingredients of an adequate stream analysis, namely high estimation quality, low processing time, and small memory footprint.

The evaluation of our experimental study yields many interesting results for the different variants of our density estimators. The analysis of these results allows us to draw conclusions for the choice of the right variant for a given application scenario. The analysis also substantiates the robustness of our estimators; they achieved high estimation accuracy for each stream we examined. In order to get an impression of their relative performance, we compare them with a set of competitive techniques, each of them capable of estimating the distribution of a data stream in an online fashion.

Publications The main results of this thesis as well as some other results of our data stream research have been published in several international conferences and workshops.

- Michael Cammert, Christoph Heinz, Jürgen Krämer, and Bernhard Seeger. A Status Report on XXL - a Software Infrastructure for Efficient Query Processing. *IEEE Data Engineering Bulletin*, 26(2), 2003.
- Michael Cammert, Christoph Heinz, Jürgen Krämer, and Bernhard Seeger. Datenströme im Kontext des Verkehrsmanagements. *GI-Workshop on "Mobilität und Informationssysteme"*, Zurich, Suisse, 2003.
- Michael Cammert, Christoph Heinz, Jürgen Krämer, and Bernhard Seeger. Anfrageverarbeitung auf Datenströmen. *Datenbankspektrum*, 11, 2004.
- Michael Cammert, Christoph Heinz, Jürgen Krämer, and Bernhard Seeger. Sortierbasierte Joins über Datenströmen. *GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW)*, Karlsruhe, Germany, 2005.
- Björn Blohsfeld, Christoph Heinz, and Bernhard Seeger. Maintaining Nonparametric Estimators over Data Streams. *GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW)*, Karlsruhe, Germany, 2005.
- Christoph Heinz and Bernhard Seeger. Wavelet Density Estimators over Data Streams. *ACM Symposium on Applied Computing (SAC)*, Santa Fe, New Mexico, USA, 2005.
- Michael Cammert, Christoph Heinz, Jürgen Krämer, Tobias Riemenschneider, Maxim Schwarzkopf, Bernhard Seeger, and Alexander Zeiss. Stream Processing in Production-to-Business Software. *International Conference on Data Engineering (ICDE)*, Atlanta, Georgia, USA, 2006.
- Christoph Heinz and Bernhard Seeger. Statistical Modeling of Sensor Data and its Application to Outlier Detection. *GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze"*, Stuttgart, Germany, 2006.
- Christoph Heinz and Bernhard Seeger. Exploring Data Streams with Nonparametric Estimators. *International Conference on Scientific and Statistical Database Management (SSDBM)*, Vienna, Austria, 2006.
- Christoph Heinz and Bernhard Seeger. Resource-Aware Kernel Density Estimators over Streaming Data. *International Conference on Information and Knowledge Management (CIKM)*, Arlington, Virginia, USA, 2006.
- Christoph Heinz and Bernhard Seeger. Stream Mining via Density Estimators: A concrete Application. *International Conference on Management of Data (COMAD)*, Delhi, India, 2006.
- Christoph Heinz and Bernhard Seeger. Towards Kernel Density Estimation over Streaming Data. *International Conference on Management of Data (COMAD)*, Delhi, India, 2006.

- Michael Cammert, Christoph Heinz, Jürgen Krämer, Bernhard Seeger, Sonny Vaupel, and Udo Wolske. Flexible multi-threaded scheduling for continuous queries over data streams. *International Workshop on Scalable Stream Processing Systems*, Istanbul, Turkey, 2007.

- Christoph Heinz and Bernhard Seeger. Adaptive Wavelet Density Estimators over Data Streams. *International Conference on Scientific and Statistical Database Management (SSDBM)*, Banff, Canada, 2007.

1.3 Outline of the Thesis

The main topics of the previous section are also reflected in the organization of this thesis.

Chapter 2 discusses related work. We present the main approaches of the different topics we touch on in this work and take a closer look at approaches similar to our work.

Chapter 3 introduces the data stream model and mathematical preliminaries. We first present a formal model of data streams, accompanied by a discussion of the processing requirements for streams. Then we give an introduction to the theory of density estimation due to the important role it plays in this work. This overview primarily discusses kernel-based and wavelet-based density estimation as our objective is to adapt them to data streams.

Chapter 4 gives an overview of XXL and PIPES. As we implemented our techniques in XXL, this overview prepares the ground for the understanding of their implementations. We describe the main concepts of XXL and give a feeling for its vast functionality. Additionally, we sketch the essential building blocks of its subpackage PIPES.

Chapter 5 thoroughly discusses compressed-cumulative WDEs, our approach to wavelet density estimation over data streams. We start with the discussion of our framework for maintaining nonparametric estimators over data streams. We describe its main components and their interplay. Then we show the application of the framework for the case of wavelet density estimators, i.e., we show how to realize each of the framework components for them.

Chapter 6 goes into the details of our Cluster Kernel approach, which provides kernel density estimators over data streams. We first present the general idea of the Cluster Kernel approach as well as its main components. Then we discuss M-Kernels [26], a competitive density estimation technique that is compatible with our approach. Afterward, we elaborate concrete strategies and settings for the main components of Cluster Kernels.

Chapter 7 presents the results of our experimental study, which scrutinizes the performance and accuracy of our online density estimators. Besides assessing our techniques, we also compared them with competitive techniques.

Chapter 8 concludes this thesis with a summary of its salient results. Additionally, we sketch future research directions that use the presented techniques as starting point.

Chapter 2

Related Work

This chapter intends to give an overview of approaches related to the main topics of this thesis. Section 2.1 covers approaches for processing and querying data streams. In Section 2.2, we examine stream mining and analysis solutions. Section 2.3 surveys kernel- and wavelet-based density estimation methods as well as their adaptation to data streams.

Due to the growing number of applications based on data streams, a vast number of techniques for processing, querying, and analyzing streams have been proposed in recent years. The importance of density estimation for data analysis also has lead to a large number of density estimation techniques. As an exhaustive discussion of the entirety of techniques available for data streams and density estimation is beyond the scope of this work, we primarily concentrate on those techniques that touch on one of the topics of this work in terms of comparable objectives or methods applied. However, we also attempt to provide relevant sources for further investigations in related topics.

2.1 Processing and Querying of Data Streams

Many approaches for the processing and querying of data streams have been developed in recent years. Babcock et al. examine in their seminal work [16] models and issues for data streams, accompanied by the discussion of how to address them adequately. The authors particularly outline the difficulties of query processing for data streams and motivate the need for approximate solutions based on data summarization techniques like sliding windows and synopses. Another overview of the main issues in data stream management and the corresponding requirements for stream processing is given in [71]. With respect to these requirements, this work discusses data models and query languages, the implementation of the corresponding stream operators as well as the principal questions of continuous query processing and optimization. In [118], the author formally introduces data stream models and examines the mathematical and algorithmic principles that are relevant to the data stream scenario. A comprehensive overview of recent data stream research is given in [8]. Though the focus of this book is on stream mining and analysis, the author also addresses approaches related to processing and querying streams.

DSMS Approaches Several projects have attended to the development of a general-purpose data stream management system (DSMS), upon which the following ones are the

largest:

- STREAM [13, 117], the Stanford Stream Data Manager, is a DSMS approach, which is based on the relational model. It offers to run continuous queries over multiple continuous data streams and stored relations. The STREAM system is equipped with a declarative query language called CQL to describe the corresponding continuous queries in an SQL-like manner. In case of high query load and high data rates, STREAM adapts to the current system resources by providing approximate query answers.

- Aurora is designed to deal with applications that monitor continuous data streams [34, 5]. Aurora is essentially a data-flow system. It provides the user a set of specific operators, so called boxes, which can be connected in order to build data-flow graphs. The outputs of those graphs are delivered to the monitoring applications. Each output can be equipped with a quality of service (QoS) specification. A QoS monitor continually monitors the system performance and performs load shedding [132] in case of a resource overload.

- TelegraphCQ [104, 36] is a system for continuous data-flow processing, which can handle large streams of continuous queries over massive data streams. It comprises an extensible set of data-flow modules (or operators), which can be composed into complex data-flows. The main objectives of TelegraphCQ are an adaptive query processing as well as the shared execution of multiple continuous queries. TelegraphCQ uses the open source database PostGreSQL as starting point for its implementation.

- PIPES [103, 30] is an infrastructure which provides the main components to build and execute multiple continuous queries over multiple data streams. PIPES is an essential part of XXL [136, 28], a library offering advanced data processing facilities. As we embedded the techniques developed in the course of this work into XXL and PIPES, we discuss both of them in more detail in Chapter 4.

Due to the different problems and requirements a DSMS faces in comparison to a DBMS, the main concepts and components of a DBMS must be fundamentally reconsidered and reengineered to be applicable in a DSMS. Amongst others topics, this relates to the underlying query language, the storage and buffer management, the continuous query optimization, the implementation of stream operators, and the user and application interfaces. These topics have been successfully addressed in recent years, resulting in a variety of convenient solutions. As these solutions are only marginally relevant to this work, we do not go into their details. For an overview, we refer to [101].

Approximate Query Answering Approximate query answering is a key concept in a DSMS. Since the unbounded size of a data stream in combination with the limited resources often render the computation of an exact query answer impossible, high-quality approximate query answers are computed instead. An approximate query answer is typically computed with respect to a summary, a so-called synopsis, of the data stream. As density estimators also provide a kind of data summary and can be used to compute approximate query answers, we take a closer look at the common data reduction and summarization approaches [16, 60]: random sampling, load shedding, sketches, histograms, quantiles, wavelets.

The idea of using random sampling is to choose probabilistically a subset of the stream, which reflects the stream's main characteristics. For each stream element, a probabilistic selection scheme decides whether it enters the sample or not. To compute samples of a data stream, reservoir sampling [139] can be applied; it makes a single pass over the data and always provides an *iid* sample of the already processed elements. As reservoir sampling is not designed to focus on recent data, [7] provides a method to bias a reservoir-based sample by means of temporal bias functions. A more general approach for sampling over data streams is presented in [95]. The authors abstract in this work from concrete sampling algorithms and introduce a generic stream sample operator which can be used to implement various sampling algorithms and sampling-based aggregation techniques for data streams. Generally, sampling is a suitable method to reduce the stream size, but it suffers from the problem that important data like outliers or anomalies are ignored if they do not enter the sample.

Load shedding is similar to random sampling as it reduces the stream size by dropping elements. Its main objective is to reduce the stream rate in case a DSMS has a high system load. [132] discusses where to shed load in a DSMS and how much. The elements to be dropped are either chosen probabilistically or with respect to the importance of their content. [134] models quality-driven load shedding in a DSMS as a feedback control problem and proposes to solve it with a specific controller for load shedding decisions. Similar to sampling, load shedding runs the risk of dropping important data.

Sketches were introduced in [12]; they are random projections of a given data vector, which can be utilized to approximate the frequency moments of the data. In [69, 70], the authors use sketches to estimate point and range queries on a signal described by a data stream. In this context, a signal is a function that maps values of a finite domain to non-negative integers and a stream describes this signal.[1] The idea is to maintain a sketch of the signal while processing the stream. With the help of the sketch, linear projections of the underlying signal can be estimated. This particularly includes the estimation of the wavelet coefficients of the signal, which can be used to estimate point or range queries. Circumventing some drawbacks of this approach, [42] proposes the Group-Count Sketch, which provides space- and time-efficient tracking of approximate wavelet summaries for one-dimensional and multi-dimensional data streams. This sketch-based structure features the recovery of the most important wavelet coefficients with guaranteed accuracy. In [44], count-min sketches are proposed, which support simple as well as more complex queries over a data stream. These sketches improve the time and space bounds of previous approaches. For more details on how to use sketches for summarizing data streams, we refer to [118, 16, 44, 64, 8].

Histograms are summary structures that describe the distribution of values in a data set. More formally, a histogram is an estimate of the density of the distribution underlying the data. [128] thoroughly discusses the notion of histograms from a mathematical point of view. Due to their simplicity, histograms have been extensively used in database research. A comprehensive overview of the different types of histograms and their use in databases is given in [123]. In [77], an online computable $(1 + \epsilon)$-approximation to the V-optimal histogram[2] is presented. This approximation can be computed in linear time and requires polylogarithmic space. [68] makes use of the aforementioned sketches to derive approximate

[1] For example, the signal describes the number of outgoing call minutes for each telephone number. A corresponding stream element comprises a telephone number and the minutes per call.

[2] A histogram is V-optimal if the sum of the bucket variances is minimum.

histograms for data streams. The proposed approach determines in a first step a robust histogram, from which the histogram of desired accuracy is derived in a second step. [76] also uses continuously maintained sketches to compute approximate multi-dimensional histograms over a data stream on demand. By iterating over possible histograms in a greedy algorithm, a near-optimal histogram, whose sketch is close to the sketch of the data distribution, is successively built.

Similar to histograms, quantiles give insights into the distribution of a data set. The algorithms in [110] provide suitable approximations of the quantiles of a data set, which can be computed in a single pass with limited memory. One of the algorithms gives a deterministic guarantee on the accuracy, but presupposes the knowledge of the size of the data set. Another algorithm does not need to know the size, but can only give probabilistic guarantees on the accuracy. As we use this technique in our experiments, we take a closer look at its principles in Section 7.1.3. [74] presents an algorithm for approximating quantiles in a single pass over the data. This approach is independent of the data set size and gives deterministic guarantees on the accuracy. Using this technique as an ingredient, [109] maintains approximate quantiles of the N most recent elements of a data stream.

Wavelets provide convenient mechanisms to support an approximate query answering. Generally, wavelets can be used to hierarchically decompose a set of values into wavelet coefficients. Given these coefficients, their number can be reduced by omitting some of them with respect to a thresholding scheme and a given error metric that is to be minimized. A common approach is to use the L_2-metric, which is minimized if the largest absolute wavelet coefficients are retained. The remaining coefficients provide a summary of the original values, which is typically denoted as wavelet synopsis. A wavelet synopsis can be used to reconstruct the original values approximately; it also supports the computation of approximate query answers. As we also use wavelets in this work, but in a different way then wavelet synopses do and with different objectives, let us consider recently proposed wavelet synopses for static as well as for streaming data. Note that the above approaches [70, 42] using sketches to estimate wavelet coefficients in an online manner also in essence deliver wavelet synopses. In [65], the authors introduce probabilistic wavelet synopses, which provide unbiased approximate query answers with error guarantees on the accuracy of individual answers. They deterministically keep the most important wavelet coefficients, while the remaining ones are randomly rounded either to zero or to a larger value. [66] discusses wavelet synopses for general error metrics. This work presents deterministic wavelet thresholding schemes for minimizing general, non-L_2 error metrics required in approximate query answering approaches. The authors present an optimal algorithm for one-dimensional wavelet thresholding as well as efficient approximation schemes for deterministic wavelet thresholding for multi-dimensional data. [115] presents linear-time, I/O optimal algorithms to determine weighted, Haar-based wavelet synopses that are optimal with respect to a workload of point queries. In [97], the authors attend to the one-pass computation of a wavelet synopsis under maximum-error metrics. They introduce greedy wavelet thresholding algorithms that achieve near-optimal accuracy with respect to maximum-error metrics for the static case; the algorithms run in near-linear time and need only small storage space. Additionally, the authors extend their techniques to the streaming case, where only one pass over the data and constant storage space is allowed. A similar topic is discussed in [75]. This work presents algorithms to build wavelet synopsis over data streams with respect to several non-Euclidean error measures. For the case of multiple

time series streams, [93] presents wavelet synopses that support efficient range-constrained similarity search. With the help of a separate wavelet synopsis for each stream, the distance between two streams as well as the k nearest "stream neighbors" of a given stream can be determined with respect to a given range of the streams. Generally, the majority of wavelet synopsis approaches uses Haar wavelets for the sake of simplicity. However, Haar wavelets are the least smooth wavelet family; using smoother wavelet families typically improves the quality of wavelet representations.

Besides reducing the data or computing data summaries, an alternative, intuitive method for computing approximate query answers is to refer the answer to a sliding window of elements. In many applications, queries with respect to sliding windows are even explicitly desired as part of the query facilities since the user is often only interested in the most recent elements.[3] The notion of sliding window queries is strongly coupled with the definition of the associated query semantics, i.e., how are sliding windows and the associated queries over them defined? To answer this question, several approaches define a semantics for sliding window queries, e.g., [102], [14]. Other operations over data streams can also be computed with respect to sliding windows such as stream statistics [47] and aggregates [107].

Now that we have discussed the common approaches for data reduction and summarization, let us note that many of them, in particular all sketch-based ones, assume that the value of each stream element is from a finite universe. As a consequence, their application to real-valued data streams, whose universe is infinite, requires an initial discretization. But this discretization introduces an additional error, which must be taken into account besides the error of the approximate query answering. As the size of the finite universe typically adversely affects the algorithmic complexity, a trade-off between accuracy and computational complexity is the consequence. The finer the discretization and therefore the better the accuracy is, the larger becomes the universe and the computational complexity. It is worth mentioning that our methods to analyze real-valued data streams do not need an initial discretization step.

Besides approaches for processing and querying streams, there also exists a multitude of approaches for stream mining and analysis.

2.2 Mining and Analysis of Data Streams

The emerging demand for stream mining and analysis solutions has initiated the development of techniques that can extract knowledge of a data stream in form of patterns and models in an online fashion. Many of the above presented techniques for approximate query answering can also be employed for stream mining and analysis purposes. The aforecited work [118], which discusses data stream models and corresponding algorithmic principles, is also relevant to stream analysis. In [55], the authors describe some general research issues arising in stream mining, complemented by a discussion of the processing requirements an analysis technique has to meet to be applicable to streams. [64] provides a summary of different techniques for querying and mining streams. A comprehensive review of stream mining is given in [60]. The authors review in this work the different solutions recently proposed in the literature, including some of the approximate query processing approaches we presented in the previous section. They particularly examine the stream solutions proposed for clustering, classification,

[3] SQL-99 introduces an explicit window clause.

frequency counting, time series analysis, and stream mining systems. Besides the discussion of already proposed solutions, this work also summarizes the arising research issues in this field of study. In Section 1.1.2, we listed a few of them for illustrative purposes. The aforecited book [8] thoroughly discusses very recent stream mining and analysis approaches. This book gives a comprehensive overview of the key stream mining problems and of the arising challenges research academia is confronted with. To give a feeling for the range of topics covered in this book, let us list some of them: clustering, classification, pattern mining, change detection, load shedding, stream synopsis, join processing, stream indexing, distributed mining, and mining in sensor networks.

Let us now examine the different approaches for the main stream mining and analysis tasks.

Stream Mining Systems We start with approaches to unify stream mining and analysis techniques within a stream mining system:

- MAIDS [15] is a prototype of a stream mining system, whose objective is to mine dynamics and alarming incidents in multi-dimensional stream data. MAIDS features a tilted time window framework and multi-resolution model as well as a stream datacube for multi-dimensional analysis. It provides separate modules for classification, frequent pattern mining, and clustering of data streams, accompanied by sophisticated facilities for stream mining visualization.

- Gigascope [45, 46] is a lightweight stream database, which is specialized for network monitoring applications like traffic analysis or intrusion detection. To pose queries, Gigascope provides GSQL, which is a restricted SQL version plus some stream database extensions. To overcome the problem of blocking operators, ordering properties of the stream attributes are used for non-blocking operator implementations. For the sake of flexibility, GSQL also supports user-defined functions and operators.

- StatStream [145] is a data stream system which can monitor descriptive statistics of multiple data streams in real time. Besides simple statistics like average and standard deviation, the system particularly maintains the correlations of all pairs of streams with respect to sliding windows. In order to compute these statistics in an online fashion, correlations are approximated with the help of the discrete Fourier transform. A grid structure facilitates the detection of likely correlated pairs.

- VEDAS [96] is a mobile and distributed data stream mining system which supports the health monitoring of vehicles and the characterization of their driver's behavior in real-time. VEDAS is equipped with a module that detects unusual driving and vehicle behavior pattern by continuously monitoring and analyzing the information streams generated by the vehicle. The associated analysis tasks are computed with lightweight onboard devices like PDAs. VEDAS also takes care of the limited computational and power resources of the mobile devices.

Clustering Several approaches for clustering data streams have been recently proposed:

In [17], the authors describe how to maintain variance and k-medians of sliding windows over data streams, using memory sublinear in the window size. Their algorithm for k-median

clustering over sliding windows uses a previously proposed algorithm as starting point, with a focus on expanding the applicability of a data structure called exponential histogram. The algorithm uses continuously maintained exponential histograms to estimate the value of the k-median objective function.

In [38], another k-median algorithm for data streams is proposed. It uses sublinear space to produce a constant factor approximation in one pass. The algorithm is randomized and works with high probability. This work additionally addresses the notion of outliers in the streaming model and presents algorithms for outlier clustering, which exclude a fraction of points, the outliers, from the clustering.

[54] presents a general method for scaling up machine learning algorithms and illustrates this approach for k-means clustering. To expedite machine learning algorithms, the starting point is to determine an upper bound on the learner's loss as function of the number of used examples in each algorithm step. While preserving the bound, the number of examples required in each processing step can be minimized. The authors make use of this principle to derive VFKM, a faster version of the k-means algorithm, which determines the number of examples per step with the help of the Hoeffding bound.

Another algorithm for the k-median problem for data streams is proposed in [120]. This algorithm processes the stream in chunks of data. For each data chunk, a weighted representation is determined in case it comprises (with high probability) more than k distinct values. Then a fast k-median algorithm called LOCALSEARCH, which uses local search techniques, is applied to the chunk, resulting in k weighted centers. To obtain cluster centers for the complete stream, LOCALSEARCH is applied to the weighted centers of the already processed data chunks.

[9] introduces CluStream, a framework for clustering evolving data streams, which allows the user to trace the evolution of clusters over different time periods. CluStream separates the clustering process into an online microclustering component and an offline macroclustering component. The microclustering component maintains simple temporal and value-based statistics of the stream elements at particular moments in the stream, resulting in snapshots of the stream.[4] With respect to a pyramidal time frame, the snapshots are stored at differing levels of time granularity. The macroclustering component uses these snapshot statistics to derive approximate clusters for a user-specified time horizon. In a subsequent work [11], the same authors propose a method for the projected clustering of high-dimensional streams, which determines clusters for particular groups of dimensions in an online manner.

In [19], clustering of data streams is examined from a different point of view. The proposed algorithm clusters parallel streams in order to reveal similar behavior of different data streams over time, which is indicated by their membership of the same cluster. The algorithm utilizes a fast version of k-means clustering, which exploits an online transformation of the original data.

Classification The classification of data streams has been addressed in the following approaches:

VDFT [91] is a decision tree learning system based on a novel decision tree learning method called Hoeffding trees. As Hoeffding trees process each example only once in constant

[4] Our Cluster Kernel approach - see Chapter 6 - uses the same value-based statistics in two of its resampling strategies.

time, they can be computed in an online manner. The decision how many examples are to be used at each tree node relies on the Hoeffding bound. In a follow-up work [92], the authors discuss how to update the decision tree with respect to time-changing streams.

In [63], the authors consider mining of data streams under block evolution. Block evolution refers to the periodical insertion or deletion of blocks of data. The authors present GEMM, a generic algorithm for data mining model maintenance that transforms an algorithm for incremental model maintenance into an algorithm, which allows to restrict the model on temporal subsets of the data blocks. To model the time-dependent selection of these subsets, the authors introduce a data span, which either refers to all data blocks or to a sliding window of blocks, and a block selection sequence, which either includes or excludes blocks from the model. The authors describe how to maintain a model for data blocks selected with respect to a given data span and a block selection sequence. The basic idea is to maintain the required partial models for all future windows overlapping with the current time. The authors illustrate their algorithm for the case of frequent itemset models and decision tree models. In a previous work [62], they give a comprehensive discussion of GEMM. In [63], they also propose FOCUS, a framework for change detection, which uses the differences between mining models as measure of change.

A general framework for mining concept-drifting data streams with weighted ensemble classifiers is introduced in [141]. The idea of this work is to process the data streams in sequential chunks and to train a separate classification model for each chunk. The ensemble of resulting classifiers is used to derive an overall classifier. In order to capture trends in the stream, each of the ensemble classifiers is weighted with respect to its expected classification accuracy on the current test examples.

In [10], an approach for on demand classification of data streams is developed, which particularly takes care of changes in the streams. The authors propose to examine training and testing streams simultaneously to ensure a dynamic classification. Similar to [9], they use so-called supervised microclusters to summarize the temporal information and the values of the training stream elements in the form of simple statistics. To build a classifier with respect to different time horizons, the authors introduce a geometric time frame for the computation of the supervised microclusters.

Change Detection As data streams often exhibit an evolving nature, different techniques to detect changes in a stream have been recently proposed.

In [56], the authors discuss general problems and challenges of mining changes from data streams. They also present preliminary results they achieved.

To incorporate changes of a data stream in decision trees, [58] presents a new concept of demand-driven active data mining, which addresses the problem that the true class labels of a data stream are typically not known. The authors modify the traditional classification tree algorithm so that it does not use true class labels or, in case of a high suspected error, only those of a few examples chosen from the recent stream.

A general framework for diagnosing changes in evolving data streams is presented in [6]. The authors introduce the concept of velocity density estimation, which can determine and visualize evolving trends in a data stream. This concept essentially relies on kernel density estimators built over spatial vectors and time. The velocity density is defined as the scaled difference of kernel density estimators computed over different time slices; its value at a

point describes the local rate of change of the density within an associated time horizon. To trace the evolution of a data stream, a temporal and a spatial velocity profile are computed at periodic time instants. These profiles can be used to diagnose regions of dissolution, coagulation, and shift in the data. For the case of high-dimensional data, this work proposes batch-processing methods to determine the combination of dimensions which exhibit the greatest change.

The work in [18] examines changes in a data stream from a statistical point of view, with the objective to develop a statistical notion of change in a stream. The authors formalize the detection and quantification of change in a stream under the assumption that the stream elements are drawn from an underlying probability density function. The proposed algorithm continuously examines pairs of windows. Each pair comprises a sliding window over the stream and a reference window, which is updated in case of a change. By means of a nonparametric test, the presented algorithm determines whether the elements in the two windows follow the same distribution. The test is designed to keep the number of false negatives and false positives small. Additionally, it provides a description of the change it has detected. A crucial component of the test is a suitable distance measure between distributions, which requires only small sample sizes to find changes with strong statistical guarantees.

Frequent Items For a discussion of approaches to determine frequent itemsets over data streams, we refer to the stream mining overview given in [60]. Besides the approaches listed there, the following ones have been recently proposed. In [94], a hash-based algorithm to determine a list of the most frequent items over a data stream is introduced. The algorithm supports insertion and deletion of transactions and does not need to know the range of the stream. [112] proposes an algorithm that maintains frequency counts for items occurring frequently in the union of multiple distributed data streams. Taking the limited resources into account, the algorithm aims at minimizing the resulting communication load. The approach for finding frequent items in data streams presented in [37] introduces a sketch-based data structure that provides estimates of the frequencies of the most common items.

Before we turn our attention to density estimation over data streams, let us note that, besides the presented data mining tasks, other mining tasks have also been successfully adapted to the data stream scenario. However, a multitude of open research questions still remains to be answered.

2.3 Density Estimation over Data Streams

The objective of this work is to estimate the probability density function of a data stream by means of kernel and wavelet density estimators. For that reason, let us examine related approaches in this field of study. It is worth mentioning that Chapter 3 discusses the theory of density estimation in more detail and particularly outlines how kernels and wavelets can be employed in this context.

Kernel Density Estimation A comprehensive overview of the theory of kernel density estimation is given in [128, 130]. Due to the viability of kernel density estimation, it found its way into many different research topics, including several database and data mining related ones. We consider some of these approaches to convey a feeling of the wide scope of

tasks kernel density estimators can tackle. In the previous section, we already mentioned an approach that uses kernel density estimators to determine changes in a data stream [6].

Kernel density estimation has been successfully used to estimate the selectivity of range queries over attributes, whose domain is the real numbers:

[23] thoroughly compares different selectivity estimators for range queries on real-valued attributes. This work particularly examines histograms and kernel density estimators as well as a novel hybrid approach. This hybrid approach partitions the data with respect to its change points, which must be computed beforehand, into bins. Each bin is associated with a separate kernel density estimator; the entirety of kernel density estimators constitutes the estimator for the complete data set.

[100] proposes two approaches to estimate the range selectivity for real-valued attributes. One approach, called KernelSpline, determines a kernel density estimator and compresses it afterward with a cubic spline. More precisely, the kernel density estimator is evaluated at a given grid of points and a cubic spline is computed with respect to these grid values. The construction of KernelSplines is discussed for one- and two-dimensional data. The second approach, called OptimalSpline, uses B-spline basis functions and maximum likelihood estimation to approximate the density.

[53, 79] examine different selectivity estimators for multi-dimensional range queries over real-valued attributes. The authors propose a new selectivity estimator termed GENHIST. GENHIST is essentially a multi-dimensional histogram; its buckets are of variable size and they are allowed to overlap. Besides GENHIST, the authors also consider multivariate kernel density estimators. Both approaches are compared with a set of competitive approaches within an extensive experimental study.

Besides range selectivity estimation, kernel density estimation also supports other data analysis approaches:

In [99], the authors utilize kernel density estimators to determine biased samples of large data sets. A biased sample is a random sample of a data set, which is biased in the sense that elements from regions of interest have a higher probability to enter the sample. To achieve this goal, the authors compute a kernel density estimator for the data set and afterward, they determine for each element the local density with respect to this estimator. The probability for an element to enter the sample is a function of its local density. If this function emphasizes regions with a small local density, i.e. sparse regions, the sample will be biased toward outliers. If the function emphasizes regions with a high local density, i.e. dense regions, the sample will be biased toward clusters. The computation of a biased sample requires to process the complete data set and to decide for each element, depending on its probability, whether it enters the sample or not. The authors illustrate the application of biased sampling for cluster and outlier detection.

Kernel density estimators can also be used in high-dimensional classification problems as [59] shows. This work introduces a hybrid approach combining decision trees and kernel density estimators. Given a classification problem and a corresponding decision tree, the posterior class probabilities of each node are estimated with kernel density estimators. To reduce the number of variables the kernel density estimator is based on, the decision tree structure is used to identify the discriminative dimensions.

[144] presents an approach to cluster a set of multi-dimensional, concept-drifting data streams by means of kernel density estimators. The distribution of each stream is estimated

with a kernel density estimator. To expedite the evaluation of a kernel density estimator, its function values on a regular grid are continuously maintained. The nearest grid point is used to evaluate the kernel density estimator at a given point. However, the consequence of this approximation is that the estimator will be piecewise constant between the grid points. Thus, one of the main advantages of kernel density estimators, namely their smoothness, is lost. As high input rates may render the maintenance of these function values at the grid points difficult, the authors propose a load-skimming approach which adapts the number of grid points to the current workload. As a stream may exhibit concept-drifts, the authors propose to decay the values at the grid points or to additionally maintain a density estimator over a sliding window, which can substitute the current estimator in case of significant changes. Given the continuously maintained density estimators for the separate streams, the similarity between two streams can be expressed in terms of the similarity between their densities. This approach can be used to apply an arbitrary cluster algorithm in order to find clusters of the streams.

[131] investigates the detection of outliers in sensor data by using kernel density estimators. The authors present a framework that utilizes kernel density estimators to approximate multi-dimensional data distributions of sensor data in a distributed manner. Each sensor maintains a local kernel density estimator over a sliding window of measurements. Due to a hierarchical organization of the sensor nodes, the estimators of the sensors can be successively composed, resulting in an estimator for the data distribution of the complete sensor network. The distributed computation of the estimators keeps the communication overhead between the sensor nodes small. The framework can be exploited to detect distance- or density-based outliers in sensor streams in a distributed fashion.

Since the computational complexity of kernel density estimation often renders its application difficult, [73] provides a faster solution for a static set of multi-dimensional training and query data. The basic idea is to establish a dual-tree structure: one tree partitions a given set of training data and the other one a given set of query data. As tree structure, either multiresolution *kd*-trees[5] or ball trees are used, depending on the number of dimensions. For each query point, the tree for the training data is traversed. At each node of this tree, lower and upper bounds on the mass contribution of the data in the node can be computed; these bounds can be used to prune the node and its children from the search. The authors extend this idea to evaluate chunks of query points by traversing both trees simultaneously.

In [105], the authors also aim to expedite the evaluation of kernel density estimators. The algorithm they propose makes use of the multivariate Taylor series expansion to derive a so-called multipole expansion. This approach requires the underlying kernel to be differentiable over the whole computational region.

In [124], the authors present an approach to maintain specific kernel density estimators termed Local Kernels over a multi-dimensional data stream. They discuss their approach against the background of estimating the selectivity of range queries over spatial data streams. While processing the stream, a sample of the already processed elements is continuously maintained with the help of reservoir sampling [139]. On top of the sample is a *kd*-tree-like structure which partitions the data space. Each leaf of the tree is associated with one sample element and a cell of the resulting partitioning. A cell maintains local statistics to summarize the stream elements that fell into it. These local statistics comprise the number of associated

[5]Multiresolution *kd*-trees additionally store local statistics in each node.

elements as well as their standard deviation along each dimension. While processing the stream, the tree as well as its local cell statistics are continuously updated. If the current stream element does not enter the sample, the local statistics of the cell the element falls into are updated. If it enters the sample, a leaf is deleted from the tree and its local statistics are redistributed. Afterward, a new leaf with the new sample point is added. To determine a kernel density estimator for the already processed stream, a local kernel density estimator is built for each cell; the entirety of local estimators delivers an overall kernel density estimator. The bandwidths of the Local Kernels are computed with the local statistics. To approximate the kernel density estimator over all elements of a cell, each Local Kernel uses the associated sample point of its cell as kernel center, which receives as weight the number of cell elements. In case the elements in the cell have a high standard deviation, this approximation will deliver poor results because it concentrates the Local Kernel around the sample point without taking care of the spread in the cell. Generally, the overall kernel density estimator is built as sum of S local bumps with S the size of the sample. As the local bandwidths determine the width of each bump and due to the fact that the bandwidths decrease with the number of cell elements, the overall estimator will consist of S peaks for extremely large streams. Let us note that we address a similar problem in the discussion of Cluster Kernels, our approach to kernel density estimation over data streams. The described effect corresponds in our terminology to a problem occurring with the one-value-resampling strategy. For more details about this problem and the way we overcome it, we refer to Section 6.3. Another drawback of the Local Kernel approach is the adaptation to the current system resources. In case the available memory is reduced, a sufficient number of sample elements must be deleted from the tree, accompanied by the redistribution of their local statistics. In case the available memory is increased, the sample must be increased, which is per se not supported by reservoir sampling as it would require to access all processed elements.

M-Kernels as proposed in [26] are kernel-based density estimators over one-dimensional data streams. M-Kernels can be expressed as one specialization of our Cluster Kernel approach. We use them as competitor in our experimental evaluation. For these reasons, we discuss them in more detail in Section 6.2.

Wavelets and Wavelet Density Estimation Basically, wavelets are mathematical objects that can describe many different functions. To gain deeper insights into the theory of wavelets, we refer to [48, 82]. Due to the convenient features of wavelets, they have been successfully applied in a wide range of applications such as signal processing, turbulence analysis, image processing. We focus in the following on wavelet-based methods for data mining and analysis purposes as well as on the use of wavelets in statistical applications. In Section 2.1, we already discussed several approaches that utilize wavelet-based synopses for approximate query processing. Some of these approaches particularly examine how to compute wavelet synopses over data streams.

Concerning data mining, [108] gives a comprehensive survey of the different wavelet-based techniques proposed in this context. To provide a systematic access to these techniques, the authors segment the process of data mining into an iterative sequence of the following subsequent steps: data management, data preprocessing, algorithms for data mining tasks, and data postprocessing. Each of these steps can gain advantage from the proper application of wavelets. As a consequence, a plethora of wavelet-based solutions has been developed for

each step. As [108] summarizes these solutions in a convenient manner, we direct the reader to this work for a more detailed review. To give a feeling for the different tasks wavelets can tackle in data mining, let us briefly sketch some of them.

Data management refers to suitable mechanisms for accessing, storing, and managing data. These mechanisms can be efficiently supported by wavelets since they offer a well-defined hierarchical decomposition of multi-dimensional data.

Data preprocessing describes the process of cleaning and correcting real-world data, which is often high-dimensional and additionally subject to noise, loss, and inconsistencies. In order to denoise data, wavelet theory provides many sophisticated thresholding policies, which reveal an underlying true function by removing the noise it is contaminated with. To cope with high-dimensional data, the wavelet decomposition is a helpful tool. The decomposition of the data delivers a set of wavelet coefficients which can be reduced with respect to a thresholding scheme, resulting in a compact summary of the original data. This method is also the basic idea of the previously discussed wavelet synopses.

Algorithms for the common data mining tasks often use wavelets as an important building block. Clustering for example can detect clusters at different scales, using the multiresolution view on the data provided by the wavelet decomposition. The decomposition can also be exploited for classification purposes. Several approaches for similarity search, another mining task, apply wavelets to detect similar patterns in different data sets.

From a mathematical point of view, wavelets can be utilized to address a variety of statistical applications. [82] particularly discusses the role they play in the statistical estimation of functions, with a focus on nonparametric density estimation, nonparametric regression, and wavelet thresholding policies. This work also gives an introduction to the theory of wavelets and to their use in function approximation. [122] outlines the application of wavelets for data analysis purposes and the different statistical applications based on wavelets. The statistical applications discussed also refer to density estimation, regression, and thresholding policies. A comprehensive survey of the facilities wavelets offer for statistical modeling is given in the monograph of Vidakovic [138]. Besides the aforementioned statistical applications based on wavelets, the author also addresses Bayesian methods in wavelets, wavelets and random processes, and wavelet-based random variables and densities. The local nature of wavelets can also be exploited to detect jumps and sharp cusps in functions contaminated with noise [142]. The detection relies on the examination of large absolute wavelet coefficients across fine scale levels.

For the case of wavelet density estimation, one of the main topics of this work, [137] presents an overview of the different approaches proposed in the literature. The important class of thresholded wavelet density estimators is introduced in the seminal work of Donoho et al. [57]. Let us emphasize that, to the best of our knowledge, wavelet density estimators have yet not been explored in the context of databases or data streams. The solution developed in the course of this work is the first adaptation of wavelet density estimators to the data stream scenario.

To prepare the ground for the discussion of our solutions, we introduce in the next chapter our data stream model and give an overview of the theory of kernel and wavelet density estimation.

Chapter 3

Data Stream Model and Mathematical Preliminaries

In this chapter, we introduce the data stream model and the mathematical preliminaries of this work. We describe in Section 3.1 the underlying data stream model and the processing requirements that must be met within the data stream scenario. As the focus of this work is density estimation over data streams, Section 3.2 gives a brief overview of density estimation in general and Section 3.3 presents a naive density estimation approach. Then we summarize the essentials of the two density estimation approaches we particularly concentrate on: Section 3.4 summarizes kernel density estimation and Section 3.5 wavelet density estimation. In Section 3.6, we compare density estimation methods. Section 3.7 concludes this chapter with a naive, sample-based density estimation approach for data streams.

Let us note that in the following sections, we will keep the mathematical treatment of the density estimation methods at a summary level. We will confine their discussion to the essential facts required to understand their principles. A more rigorous mathematical treatment of all facets of density estimation is beyond the scope of this work. Instead, we refer the interested reader to the vast literature on this topic.

3.1 Data Stream Model and Processing Requirements

Let us start with the description of our data stream model. Then we discuss the processing requirements a technique must meet to be applicable within the data stream scenario.

3.1.1 Data Stream Model

A data stream consists of a potentially unbounded sequence $X_1, X_2, ...$ of d-dimensional, real-valued numbers, i.e. $X_i \in \mathbb{R}^d$ for $i \in \mathbb{N}$. The elements are indexed with respect to the order in which they arrive. Except where otherwise stated, once an element has been processed, it cannot be deleted subsequently from the stream. However, we will provide extensions of our techniques to at least approximately model deletions in the stream.

We assume each stream element X_i to be generated by a continuous random variable X and to be independent of its previous elements $X_j, j < i$. Thus, the stream elements are at each time instant outcomes of independent and identically distributed (iid) random variables;

the stream constitutes a continuously increasing *iid* sample of X. Furthermore, we assume that X has a probability density function f which is square-integrable, i.e. $f \in L_2(\mathbb{R})$.

The assumption of independence between two stream elements is reasonable for many real-world applications. Typically, the underlying data sources will autonomously send their elements, i.e., previous elements do not affect the current element.

The assumption of an identical distribution may not always hold due to streams evolving over time. More formally, an index $k \in \mathbb{N}$ exists so that $X_1, ..., X_{k-1}$ are outcomes of X and $X_k, X_{k+1}, ...$ are outcomes of a different random variable Y. The problem of detecting those changes in a data stream is addressed in [18]. Throughout this work, we assume that the stream elements are identically distributed. In Section 8.2, we will outline how to extend our techniques with regard to the detection of changes in evolving streams.

Even though the assumptions on a data stream are relatively weak, they must be strictly obeyed as otherwise our techniques can not produce statistically reliable results. For example, if the assumption of a continuous random variable is violated and the random variable is discrete instead, i.e., its density is also discrete, our techniques nevertheless produce continuous density estimates. For that reason, an application of our techniques has to start with the question whether all assumptions are fulfilled.

Overall, the assumption that a stream constitutes an *iid* sample of an unknown continuous distribution X allows us to exploit a variety of statistical methods for an exploratory data analysis. It is worth mentioning that this is the only assumption we require to develop our techniques. We need no other information about the stream, e.g. value range, size.

Generally, we expect a data stream to deliver its elements at a very high rate. Combined with its potentially infinite number of elements, its mining and analysis becomes a difficult task [55]. In fact, we cannot make use of traditional techniques for database mining and analysis since their computational complexity typically renders an application to data streams impossible. On account of this problem, research academia is developing specific stream analysis techniques that comply with the rigid processing requirements for streams.

3.1.2 Processing Requirements

In order to keep pace with transient and volatile data streams, an analysis technique has to meet the following stringent processing requirements in accordance with [55]:

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 of the stream is available anytime.
5. The models incorporate changes in the data stream.
6. The provided models should be equivalent to their offline counterparts.

For the practical applicability of an analysis technique, resource-awareness is a crucial factor. In order to take care of resource-awareness, we add another processing requirement:

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

Given the data stream model as well as these processing requirements, we aim to estimate the density of a data stream. Therefore, we describe in the following section the general problem of density estimation and how to deal with it.

3.2 Density Estimation

One of the core concepts in statistics is the *probability density function* (pdf). The pdf f of a continuous random variable X is unique. It is a positive, real-valued function which integrates to one. The pdf completely describes X and its characteristics. Except where otherwise stated, we concentrate on univariate, continuous random variables. The importance of f for X relies on the following fundamental relation

$$\forall a, b \in \mathbb{R} : \; P(X \in [a, b]) = \int_a^b f(x)dx. \tag{3.1}$$

The probability of all possible outcomes of X can be determined by integrating f. The pdf can be exploited to determine important characteristics of X like mean, variance, quantiles, Fourier transform. It also can be used to determine the pdf of transformations $h(X)$ of X for a given function h.

Strongly connected with the pdf of X is the *cumulative distribution function* (cdf), which is defined as

$$\forall a \in \mathbb{R} : \; F(x) = P(X \leq a) = \int_{-\infty}^a f(x)dx. \tag{3.2}$$

Note that pdf and cdf are analogously defined for the case of multivariate random variables. For a more detailed discussion of random variables and their properties, we refer to standard textbooks on probability theory, e.g. [20].

In many real-world scenarios, neither X nor its pdf is known. Typically, we only have observations $X_1, ..., X_n$ of X in the form of a sample. Throughout this chapter, we assume that the sample $X_1, ..., X_n$ consists of *iid* observations of a continuous random variable X with a pdf $f \in L_2(\mathbb{R})$. Except where otherwise stated, $\int$ refers to the integral over $\mathbb{R}$.

To estimate f under this assumption, mathematical statistics offers a plethora of convenient methods. Density estimation methods can be categorized into parametric and nonparametric approaches [129]. Some authors also consider semi-parametric approaches as a third category [73].

Parametric Density Estimation Parametric density estimation approaches assume that the density f, from which the sample $X_1, ..., X_n$ is drawn, belongs to a known family of distributions $f(\cdot, \theta)$ indexed by a parameter $\theta \in \Theta$ with $\Theta \subset \mathbb{R}^p$ and p a fixed positive integer.[1] To determine a density estimate, a natural approach is to estimate θ with the help of the sample. For example, let the family be normal distributions parameterized by mean μ and variance σ^2, i.e., $\theta = (\mu, \sigma^2)$ and $\Theta \subset \mathbb{R} \times \mathbb{R}_+$. In order to obtain a density estimate, it only remains to estimate mean and variance, which can be done by computing the sample

[1] See also [129], pp. 64.

mean and the sample variance. In case f belongs to the family of distributions $f(\cdot, \theta)$, the resulting density estimate is statistically reliable and can be computed very efficiently.

However, parametric density estimation has severe drawbacks. Either the family of distributions must be known in advance or the unknown distribution must be sufficiently close to this family for the estimate to be accurate. If both is not the case, parametric density estimation will deliver poor results; the estimate will be unreliable.

Semi-parametric Density Estimation Semi-parametric approaches are more flexible than parametric ones since their parameter space can be larger. A common semi-parametric approach are mixture models, which define a weighted sum of distributions [114]. The parameters within this model are the number of incorporated distributions and their parameters.

The computation of the best fit is typically accompanied by a significant computational effort, e.g., the EM algorithm [116] performs several iterations of its basic processing steps. According to [73]: *"Semi-parametric models (such as mixtures of simpler distributions) are more flexible and more forgiving the user's lack of the true model, but usually require significant computation in order to fit the resulting non-linear models (such as the EM iterative re-estimation method)."*

Nonparametric Density Estimation Nonparametric density estimation approaches, as the name indicates, do not assume that f belongs to a known parametric family of distributions. The nonparametric family to which f belongs may be arbitrarily large, e.g., the space of all square-integrable functions. Nonparametric approaches let the data speak for themselves since they base solely on the sample. *"Nonparametric methods make minimal or no distribution assumption and can be shown to achieve asymptotic estimation optimality for ANY distribution under them."* [73] In comparison to parametric approaches, they inherently avoid specifying the wrong parametric family. In comparison to semi-parametric approaches, they are more flexible and computationally less expensive. Due to the fact that they are almost assumption-free, nonparametric density estimation approaches are of particularly interest for real-world applications, which typically have no additional knowledge about their underlying data. For that reason, we focus on nonparametric estimation techniques in this work.

Even though nonparametric approaches are very convenient, they also have a few drawbacks, which, however, do not carry that much weight in the data stream scenario. First, the rate of convergence of nonparametric approaches can be slower compared to parametric approaches where the underlying parametric model is correctly chosen. In the majority of real-world applications, a slower rate of convergence is preferable to running the risk of getting poor results due to a misspecified model. Second, to produce statistically meaningful density estimates, large sample sizes are required. As data streams per se deliver large amounts of data, this requirement is inherently met. Third, nonparametric approaches have a high computational complexity. We present in this work suitable approximate solutions that fully comply with the aforementioned processing requirements.

A plethora of nonparametric density estimation techniques exists, including histograms, orthogonal series estimators, kernel density estimators, and wavelet density estimators. For a detailed discussion of those methods, we refer to the large body of existing work on density estimation, e.g. [130], [128]. In this work, we concentrate on kernel density estimation and

wavelet density estimation. Both are theoretically well-founded and have proved their viability in practice. As we will see in the following sections, kernel and wavelet density estimation have valuable mathematical properties, while still being simple in essence. Prior to their discussion, let us consider a naive density estimator.

3.3 Naive Density Estimation

The following considerations are based on [129] and [130]. Given an *iid* sample $X_1, ..., X_n$ of X, a naive density estimator can be determined by means of the cumulative distribution function of X. The cdf can be efficiently estimated by the *empirical cumulative distribution function* (ECDF), which is defined as

$$\hat{F}^{(n)}(x) = \frac{1}{n} \sum_{i=1}^{n} I_{(-\infty,x]}(X_i), x \in \mathbb{R}. \tag{3.3}$$

Hence, the evaluation of $\hat{F}^{(n)}$ at a given point x returns the percentage of sample points below or equal to x.

Since F is the antiderivative of f, a natural approach to estimate f is to use

$$\hat{f}^{(n)}(x) = \frac{\hat{F}^{(n)}(x+h) - \hat{F}^{(n)}(x-h)}{2h}$$

with a small h. For the evaluation of this *naive estimator*, it follows

$$\hat{f}^{(n)}(x) = \frac{1}{2hn} \#\{X_i \in (x-h, x+h]\}.$$

A disadvantage of the naive estimator is its non-smooth shape. Essentially, the naive estimator is a histogram.

To express the naive estimator more formally, a weight function K is introduced

$$K(x) = \begin{cases} \frac{1}{2}, & x \in [-1, 1) \\ 0, & \text{else.} \end{cases}$$

With the weight function, the naive estimator can be expressed as

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

As we will see in the next section, this equation describes the naive estimator as specific kernel density estimator.

3.4 Kernel Density Estimation

In this section, we give a brief overview of kernel density estimators and their most important statistical properties. For a detailed discussion, we refer to the monographs of Silverman [130] and Scott [128] on kernel density estimation. The subsequent overview basically relies on these

monographs. Generally, we will concentrate on kernel density estimators over univariate data as those over multivariate data are generalizations of the univariate case.

A *kernel density estimator* (KDE) with *kernel function* K and *bandwidth* h is defined as

$$\hat{f}^{(n)}(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K\left(\frac{x - X_i}{h}\right), x \in \mathbb{R} \tag{3.4}$$

for an *iid* sample $X_1, ..., X_n$ with $X_i \in \mathbb{R}$.[2] Due to the dependency on $X_1, ..., X_n$, $\hat{f}^{(n)}(x)$ is a random variable itself.

Basically, a KDE is the overall sum of "bumps" centered at each observation X_i. While the bandwidth h determines the width of each bump, the kernel function determines its shape. Henceforth, we refer to those bumps as *kernels*. Figure 3.1 displays two KDEs based on different bandwidths for a sample consisting of 7 observations and with the Gaussian kernel as kernel function.

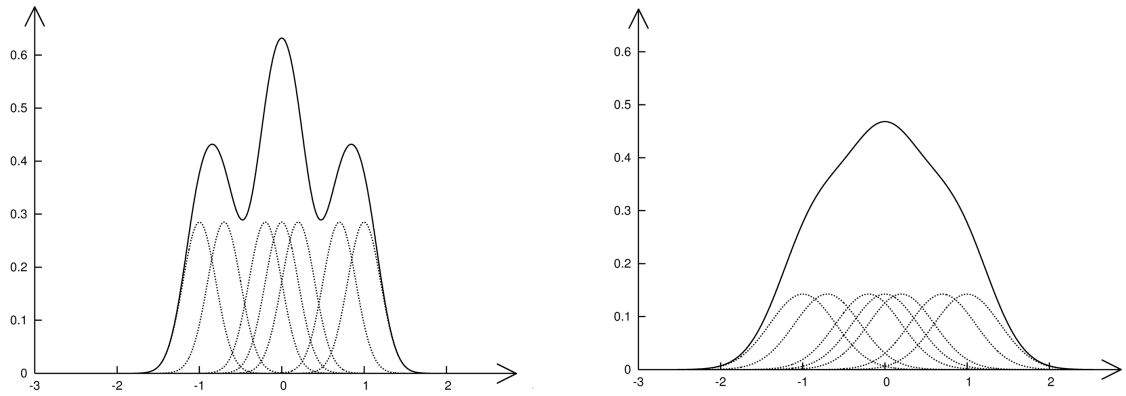


Figure 3.1: Kernel density estimates and underlying kernels with (left) bandwith=0.2 and (right) bandwidth=0.4

The KDEs in this figure indicate that the bandwidth significantly affects the shape of a KDE. The figure provides a good example of the effects that arise from varying the bandwidth. Changing the bandwidth from 0.2 to 0.4 transforms a trimodal KDE into a unimodal one. Generally, if the bandwidth is chosen too high, the KDE is oversmoothed and hides important details. If the bandwidth is chosen too low, the KDE is undersmoothed and introduces spurious details. The case of the bandwidth tending to zero leads to a sum of spikes at the sample points.[3]

In contrast to the bandwidth setting, the setting of the kernel function is of minor concern as we will later see. To ensure that $\hat{f}^{(n)}$ is a density, the kernel function itself must be a pdf. As a KDE is the sum of kernels, it inherits continuity and differentiability from its underlying kernel function.

[2] Kernel density estimation was also discussed for the case of dependent data in the context of stochastic processes [80].

[3] See also [130], p. 15.

3.4.1 Measure of Discrepancy

To measure the distance between the density estimate $\hat{f}^{(n)}$ and the true density f, a common choice is the *mean integrated squared error*

$$MISE(\hat{f}^{(n)}) = E \int \left(\hat{f}^{(n)}(x) - f(x)\right)^2 dx. \tag{3.5}$$

With the help of elementary conversions, an alternative form of the MISE can be determined:

$$\begin{aligned} MISE(\hat{f}^{(n)}) &= E \int \left(\hat{f}^{(n)}(x) - f(x)\right)^2 dx \\ &= \int bias(\hat{f}^{(n)}(x))^2 dx + \int var(\hat{f}^{(n)}(x)) dx. \end{aligned} \tag{3.6}$$

Hence, the MISE is the sum of the integrated squared bias and the integrated variance.

The complicated form of the integrated squared bias and the integrated variance exacerbates the examination of the MISE. Because of this fact, both components are approximated, which gives also an approximation of the MISE. Prerequisites for these approximations are the following assumptions on the kernel function:

$$K(x) = K(-x), \tag{3.7}$$

$$\int K(x)dx = 1, \tag{3.8}$$

$$\int xK(x)dx = 0, \tag{3.9}$$

$$\int x^2 K(x)dx = \sigma_K^2 \neq 0. \tag{3.10}$$

The unknown density f must have continuous derivatives of all orders required.

Let $R(g) = \int g(x)^2 dx$ be the *roughness* of a function g. By means of a Taylor expansion of f, the following results can be derived[4]

$$bias(\hat{f}^{(n)}(x)) = \frac{1}{2}\sigma_K^2 h^2 f''(x) + O(h^4) \quad \Rightarrow \quad \int bias(\hat{f}^{(n)}(x))^2 dx \approx \frac{1}{4}\sigma_K^4 h^4 R(f''),$$

$$var(\hat{f}^{(n)}(x)) = \frac{f(x)}{nh}R(K) - \frac{f(x)^2}{n} + O\left(\frac{h}{n}\right) \quad \Rightarrow \quad \int var(\hat{f}^{(n)}(x))dx \approx \frac{R(K)}{nh}.$$

For the *asymptotic mean integrated squared error* AMISE, it follows with these approximations

$$AMISE(\hat{f}^{(n)}) = \frac{1}{4}h^4\sigma_K^4 R(f'') + \frac{R(K)}{nh}. \tag{3.11}$$

Equation (3.11) exhibits a problem that often appears in similar forms in density estimation methods. A small value of the smoothing parameter h reduces the bias, but increases the variance. A large value of h reduces the variance, but increases the bias. These effects are apparent in Figure 3.1.

[4] For more details, we refer to Section 6.2.1 in [128] and Section 3.3 in [130].

By means of algebraic conversions, the bandwidth h_{opt} that minimizes the AMISE can be determined:

$$h_{opt} = \left(\frac{R(K)}{\sigma_K^4 R(f'')} \right)^{\frac{1}{5}} n^{-\frac{1}{5}}. \tag{3.12}$$

The optimal bandwidth slowly converges to zero with respect to the sample size. For the AMISE, it follows with h_{opt}

$$AMISE(\hat{f}^{(n)}) = \frac{5}{4} R(f'')^{\frac{1}{5}} \left(\sigma_K R(K)\right)^{\frac{4}{5}} n^{-\frac{4}{5}}. \tag{3.13}$$

For practical purposes, h_{opt} cannot be used because it depends on the roughness of the unknown density f. This dependence has led to several approaches being proposed to estimate the optimal bandwidth with the help of the sample $X_1, ..., X_n$ only.

3.4.2 Bandwidth Strategies

As the bandwidth is crucial to the quality of a KDE, a large body of work exists on this topic. For a detailed comparison of different bandwidth strategies, we refer to [130, 128] and, in particular, to [135].

The starting point of a bandwidth strategy is typically the optimal bandwidth described in (3.12). It is optimal in terms of a minimum AMISE. Assuming that we have decided on a kernel function, the only unknown component of h_{opt} is $R(f'')$. Thus, the objective is to determine a suitable estimate of $R(f'')$. For the sake of completion, let us mention that this is one way to determine a suitable bandwidth for a KDE. Other approaches have been developed that utilize the integrated squared error (ISE) as error criterion instead of the mean integrated squared error [135].

A natural approach is to estimate $R(f'')$ by using a reference density instead of f. For example, the *normal scale rule* takes the normal distribution as reference density and the Gaussian kernel as kernel function, which delivers

$$h_{opt} \approx 1.06 \sigma n^{-\frac{1}{5}} \tag{3.14}$$

as estimate. Therefore, it only remains to estimate σ, which can be done with the sample variance $\hat{\sigma}^2$. A modification of this rule that is less sensitive to outliers includes the interquartile range R as measure of spread:

$$h_{opt} \approx 1.06 \, min\left(\sigma, \frac{R}{1.34}\right) n^{-\frac{1}{5}}. \tag{3.15}$$

The *maximum smoothing principle* relies on a lower bound for $R(f'')$, which in turn gives an upper bound on h_{opt}. Using this upper bound as estimate and the sample variance as measure of spread delivers

$$h_{opt} \approx 3\hat{\sigma} \left(\frac{R(K)}{35\sigma_K^4} \right)^{\frac{1}{5}} n^{-\frac{1}{5}}. \tag{3.16}$$

As these bandwidth strategies tend to oversmooth the data for multimodal densities, more complex strategies have been developed. However, they come along with a higher computational complexity which renders their application in the data stream scenario difficult.

Kernel function	*K(x)*
Uniform	$K(x) = \begin{cases} \frac{1}{2}, & x \in [-1,1] \\ 0, & \text{else} \end{cases}$
Triangle	$K(x) = \begin{cases} 1-\lvert x\rvert, & x \in [-1,1] \\ 0, & \text{else} \end{cases}$
Epanechnikov	$K(x) = \begin{cases} \frac{3}{4}(1-x^2), & x \in [-1,1] \\ 0, & \text{else} \end{cases}$
Quartic	$K(x) = \begin{cases} \frac{15}{16}(1-x^2)^2, & x \in [-1,1] \\ 0, & \text{else} \end{cases}$
Triweight	$K(x) = \begin{cases} \frac{35}{32}(1-x^2)^3, & x \in [-1,1] \\ 0, & \text{else} \end{cases}$
Gaussian	$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$

Table 3.1: Common kernel functions

An interesting class of strategies relies on cross-validation. For example, least-squares cross-validation estimates $R(\hat{f}^{(n)})$ with the help of the sample and minimizes this estimate with respect to h. The corresponding h that minimizes the estimate is used as bandwidth estimate. Another interesting class of bandwidth strategies are plug-in methods. For example, the Park and Marron plug-in method proposes to estimate $R(f'')$ with the help of the roughness of the second derivative of $\hat{f}^{(n)}$.

3.4.3 Kernel Function

In comparison to the bandwidth, the setting of the kernel function is minor. In the previous considerations, the kernel function only had to meet the requirements (3.7) to (3.10). In Table 3.1, we list some kernel functions that meet those requirements [135].

If we take a closer look at equation (3.13), we see that one component of the AMISE is $R(K)$, the roughness of the kernel function. In an attempt to make the AMISE small, one way is to minimize $R(K)$ while ensuring that (3.7) to (3.10) are met. The solution to this problem is the *Epanechnikov kernel*, which can be found in Table 3.1. Besides the Epanechnikov kernel, the Gaussian kernel also plays an important role due to its unlimited differentiability.

While the kernel functions in Table 3.1 are pdfs themselves, higher order kernels which are no longer pdfs are also relevant. Higher order kernels take positive and negative values. They can be exploited to reduce the bias, which in turn improves the MISE.[5]

In real-world applications, the unknown density f may have discontinuities, e.g. at the boundaries. In these cases the estimation accuracy deteriorates in the vicinity of these discontinuities. To overcome this problem, specific boundary kernels can be utilized.[6]

For practical purposes, it is advisable to use kernel functions with bounded support as they facilitate the evaluation of a KDE at a point x. If the kernel function has an unbounded support, all kernels must be evaluated. If the kernel function has a bounded support, only the kernels that cover x must be evaluated. We address this aspect in Section 3.4.6.

[5] See Section 6.2.3.1 in [128].
[6] See Section 6.2.3.5 in [128].

3.4.4 Asymptotic Properties

Against the background of data streams, it is particularly interesting how KDEs behave for increasing sample sizes. We expect an estimator to improve in accuracy the larger the sample is, i.e., it has to be consistent. The following results will show that KDEs are consistent estimates of the true density under relatively mild conditions on bandwidth and kernel function. To emphasize the dependence of the bandwidth on the sample size, we index h with the sample size n.

For the consistency of $\hat{f}^{(n)}$ at a point $x \in \mathbb{R}$, the following conditions must be fulfilled. The kernel function must satisfy

$$\int |K(x)|dx < \infty, \tag{3.17}$$

$$\int K(x)dx = 1, \tag{3.18}$$

$$\lim_{|x|\to\infty} |xK(x)| = 0 \tag{3.19}$$

and the bandwidth must satisfy

$$\lim_{n\to\infty} h^{(n)} = 0, \tag{3.20}$$

$$\lim_{n\to\infty} nh^{(n)} = \infty. \tag{3.21}$$

Under these conditions on bandwidth and kernel function, the following statement[7] holds for f continuous at x

$$\hat{f}^{(n)}(x) \to f(x) \text{ in probability as } n \to \infty. \tag{3.22}$$

It is worth mentioning that most conceivable kernel functions satisfy the conditions (3.17) to (3.19).

3.4.5 Multivariate Kernel Density Estimation

Multivariate KDEs are essentially generalizations of univariate KDEs. Let us briefly discuss multivariate KDEs based on product kernels as described in [128]. A multivariate KDE based on product kernels for a given sample $(X_{11}, ..., X_{1d}), ..., (X_{n1}, ..., X_{nd})$ with d-dimensional values is defined as

$$\hat{f}^{(n)}(x) = \frac{1}{nh_1 \cdot ... \cdot h_d} \sum_{i=1}^{n} \prod_{j=1}^{d} K\left(\frac{x_j - X_{ij}}{h_j}\right), x \in \mathbb{R}^d. \tag{3.23}$$

For the computation, the same kernel function is used in each dimension, but each dimension has its own bandwidth h_i. Note that multivariate KDEs can also be more generally defined.[8]

Analogous to the univariate case, the MISE serves as measure of discrepancy. Again, an approximation of the MISE can be derived by truncating the Taylor expansion of variance and bias. The resulting approximation, again termed AMISE, is of order $O(n^{-\frac{4}{4+d}})$.

[7] See Section 3.7.1 in [130].
[8] See also [128], Section 6.3.2.

For the bandwidth, no general closed-form expression for the optimal solution, which minimizes the AMISE, exists. However, the solution can be derived for special cases. For practical purposes, one can use approximations of the bandwidth, e.g. the multivariate analogue of the normal scale rule.

An important aspect to take into account is the relationship between sample size, dimension, and accuracy. As the order of the AMISE shows, the higher the dimension is, the higher the sample size must be to achieve a preset accuracy. This so-called curse of dimensionality may become a severe obstacle for the use of KDEs over high-dimensional data.

3.4.6 Practical Considerations

After discussing the theoretical aspects of KDEs, let us briefly consider some practical aspects which facilitate the application of KDEs in the univariate case. More precisely, we summarize the essential steps required for the computation of a KDE and examine its efficient evaluation.

Computation of KDE We assume that the kernel function K has a compact support $[a, b]$. The sample $X_1, ..., X_n$ shall be ordered. According to equation (3.4), the corresponding KDE can be computed in a straightforward manner. The chief step is to compute the bandwidth h with respect to a preset bandwidth strategy. Given the bandwidth, the kernel function, and the sample, we already have the KDE at hand.

Evaluation of KDE Given a point x to evaluate, we confine the evaluation of the kernels to those that deliver a non-zero value. We have to determine the lowest X_i such that $K(\frac{x-X_i}{h}) \neq 0$. As the support of $K(\frac{x-X_i}{h})$ is $[X_i + ah, X_i + bh]$, this corresponds to finding the lowest X_i with $X_i + bh \geq x$. Due to the ordering of the sample values, this kernel can be efficiently detected in $O(n \log n)$ by means of a binary search. Then we evaluate all kernels with $X_j, j \geq i$ as long as their support covers x. Hence, we stop at the first X_j with $X_j + ah \geq x$.

3.4.7 Viability of Kernel Density Estimation

Now that we gave an overview of KDEs, let us recapitulate their most important properties:

- KDEs are simple to compute and deliver smooth estimates. Their parameter settings are based on theoretically well-founded results.
- KDEs are data-driven; they only rely on the sample without restrictive a priori distribution assumptions. Neither do they need to know the data range in advance, nor do they assume any membership of a family of standard distributions.
- KDEs have powerful asymptotic properties. They provide consistent estimates of the true densities under very mild conditions..

Due to this combination of simplicity with meaningful mathematical properties, 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.*"[9] In Section 2.3, we already conveyed an impression of the range of applications that use kernel density estimators as a main building block.

[9] See [130], p. 17.

3.4.8 Adaptation to Data Streams

The viability of kernel density estimators highly recommends their adaptation to data streams; we can exploit them to attain meaningful insights into the characteristics of streams. A suitable adaptation can also prepare the ground for the application of offline mining techniques based on KDEs to data streams.

Since we assume a data stream to be an increasing *iid* sample of a continuous distribution with an unknown pdf (see Section 3.1.1), the prerequisites for an application of KDEs to streams are satisfied. An initial approach would be to maintain a KDE which is updated for each new stream element. However, a closer look at the definition of KDEs and their parameter settings reveals that this approach violates the processing requirements for streams described in Section 3.1.2. More precisely, requirements 1,3, and 7 are violated. The reason is that the KDE requires each element for its evaluation as well as for the computation of the bandwidth. This necessity leads to multiple passes over the stream and to a continuously increasing amount of occupied memory. Even if large amounts of data could be stored, the use of KDEs would become unfeasible due to high evaluation cost.

We conclude that KDEs in their original form can not directly be applied to data streams. To meet the rigid processing requirements for streams, we must develop approximate solutions for KDEs over streams instead.

3.5 Wavelet Density Estimation

In this section, we give an overview of wavelets with a particular focus on wavelet-based density estimation. Wavelets are specific mathematical objects that have been incorporated into a plethora of scientific fields, ranging from signal processing over data compression and image analysis to statistical function estimation. As already mentioned in Chapter 2, a vast literature on wavelets and their various applications exists. For deeper insight into the theory of wavelets, we direct the reader to the monograph of Ingrid Daubechies [48]. For a description of statistical modeling by wavelets, we refer to [138, 122]. The following overview relies primarily on [122], [138], [137]. It concentrates on the essentials of wavelets that are necessary for the construction of wavelet density estimators.

3.5.1 An Introduction to Wavelets

In simple terms, wavelets are wavy functions equipped with convenient mathematical properties. Wavelets constitute bases for a plethora of function spaces and therefore can be exploited to describe and decompose a function.

Except where otherwise stated, we confine the subsequent discussion to $L_2(\mathbb{R})$, the space of all square-integrable functions on $\mathbb{R}$. The inner product of this space is defined as

$$\langle f, g \rangle = \int f(x)g(x)\ dx. \tag{3.24}$$

A *wavelet* is a function ψ defined on $\mathbb{R}$, whose dyadic dilated and translated versions $\psi_{j,k}, j, k \in \mathbb{Z}$ constitute an orthonormal basis of $L_2(\mathbb{R})$. A dyadic dilated and translated wavelet is defined as

$$\psi_{j,k}(x) = 2^{\frac{j}{2}}\psi(2^j x - k),\ j, k \in \mathbb{Z} \tag{3.25}$$

where j denotes the *dilation* index (or *resolution*) and k the *translation* index. The dilation squeezes or expands the wavelet while the translation shifts it along the x-axis. A wavelet is closely connected with its associated *scaling function* ϕ. The dilated and translated versions of ϕ are analogously defined. As we will see, the scaling function describes coarse, global features of a function, while the wavelet describes local details.

3.5.1.1 Multiresolution Analysis

An essential concept of wavelets is the *multiresolution analysis* (MRA). Informally, the MRA offers a "zoom-in, zoom-out" into a function and its features. Formally, an MRA is a nested sequence of closed subspaces $V_j, j \in \mathbb{Z}$ in $\mathbb{R}$ with the following properties

$$\ldots \subset V_{j-1} \subset V_j \subset V_{j+1} \subset \ldots, \tag{3.26}$$

$$\overline{\bigcup_{j=-\infty}^{+\infty} V_j} = L_2(\mathbb{R}), \bigcap_{j=-\infty}^{+\infty} V_j = \{0\}, \tag{3.27}$$

$$f(\cdot) \in V_0 \Rightarrow f(\cdot - k) \in V_0 \, \forall k \in \mathbb{Z}, \tag{3.28}$$

$$f(\cdot) \in V_j \Leftrightarrow f(2\cdot) \in V_{j+1}, \tag{3.29}$$

$$\exists \phi \in V_0 : \{\phi(\cdot - k) : k \in \mathbb{Z}\} \text{ is an orthonormal base for } V_0. \tag{3.30}$$

One can show that $\{\phi_{j,k} : k \in \mathbb{Z}\}$ constitutes an orthonormal base for V_j.

With an MRA, a base for $L_2(\mathbb{R})$ can be derived. Let the so-called *detail space* W_j be the orthogonal complement of V_j in V_{j+1}, i.e. $V_j \oplus W_j = V_{j+1}$. With equation (3.27), it follows in an iterative decomposition

$$L_2(\mathbb{R}) = \bigoplus_{j \in \mathbb{Z}} W_j = V_{j_0} \oplus \bigoplus_{j \geq j_0} W_j. \tag{3.31}$$

Each detail space W_j is generated by wavelet versions. More precisely, $\{\psi_{j,k} : k \in \mathbb{Z}\}$ is an orthonormal base for W_j. Overall, $\{\psi_{j,k} : j, k \in \mathbb{Z}\}$ is an orthonormal base for $L_2(\mathbb{R})$.

Due to $V_0 \subset V_1$, the scaling function can be expressed as a linear combination in V_1. This leads to the *two-scale equation*

$$\phi(x) = \sum_{k \in \mathbb{Z}} h_k \phi_{1,k}(x) = \sum_{k \in \mathbb{Z}} h_k \sqrt{2} \phi(2x - k). \tag{3.32}$$

This equation is fundamental for the construction of wavelets as the h_k's define the corresponding wavelet:

$$\psi(x) = \sum_{k \in \mathbb{Z}} (-1)^k h_{-k+1} \phi_{1,k}(x). \tag{3.33}$$

As the h_k's bridge the gap from MRA to signal processing, they are often also called *filter*.

3.5.1.2 Wavelet Series Expansion

With the decomposition of $L_2(\mathbb{R})$ in equation (3.31), a function $f \in L_2(\mathbb{R})$ can be represented in its *wavelets series expansion*

$$f(x) = \sum_{j,k\in\mathbb{Z}} d_{j,k}\psi_{j,k}(x) \tag{3.34}$$

$$= \sum_{k\in\mathbb{Z}} c_{j_0,k}\phi_{j_0,k}(x) + \sum_{j=j_0}^{+\infty}\sum_{k\in\mathbb{Z}} d_{j,k}\psi_{j,k}(x). \tag{3.35}$$

We refer to the coefficients $c_{j_0,k} = \langle f, \phi_{j_0,k}\rangle, k \in \mathbb{Z}$ as *scaling coefficients* and to the $d_{j,k} = \langle f, \psi_{j,k}\rangle, j, k \in \mathbb{Z}$ as *wavelet coefficients*. Equation (3.35) shows that a function can be described as an approximation at scale j_0 plus detail information at finer scales $j \geq j_0$. Hence, the scaling functions represent coarse, global features of f whereas the wavelets represent local details.

3.5.1.3 Discrete Wavelet Transformation

For practical purposes, the *discrete wavelet transformation* (DWT) is of utmost importance. The DWT offers a hierarchical decomposition of a function.

This decomposition concretely refers to the scaling coefficients. More precisely, for the relation between scaling and wavelet coefficients, it can be shown that

$$c_{j,k} = \sum_{l\in\mathbb{Z}} h_{l-2k}c_{j+1,l},\ d_{j,k} = \sum_{l\in\mathbb{Z}}(-1)^l h_{-l+2k+1}c_{j+1,l}. \tag{3.36}$$

Hence, a set of scaling coefficients at a given level can be decomposed into scaling and wavelet coefficients of lower resolutions. This process can also be reversed. Given the scaling and wavelet coefficients at resolution $j-1$, the scaling coefficients of V_j can be computed via

$$c_{j,k} = \sum_{l\in\mathbb{Z}} h_{k-2l}c_{j-1,l} + (-1)^k h_{2l-k+1}d_{j-1,l}. \tag{3.37}$$

The decomposition algorithm exhibits an inherent down-sampling. In case of a finite number of h_k's, the decomposition delivers approximately half as many scaling and wavelet coefficients in the next lower level.[10]

By means of the DWT, a function f can be compressed in an intuitive manner by omitting local details. As an example, let us consider a function f completely represented in V_{j_0}, i. e.

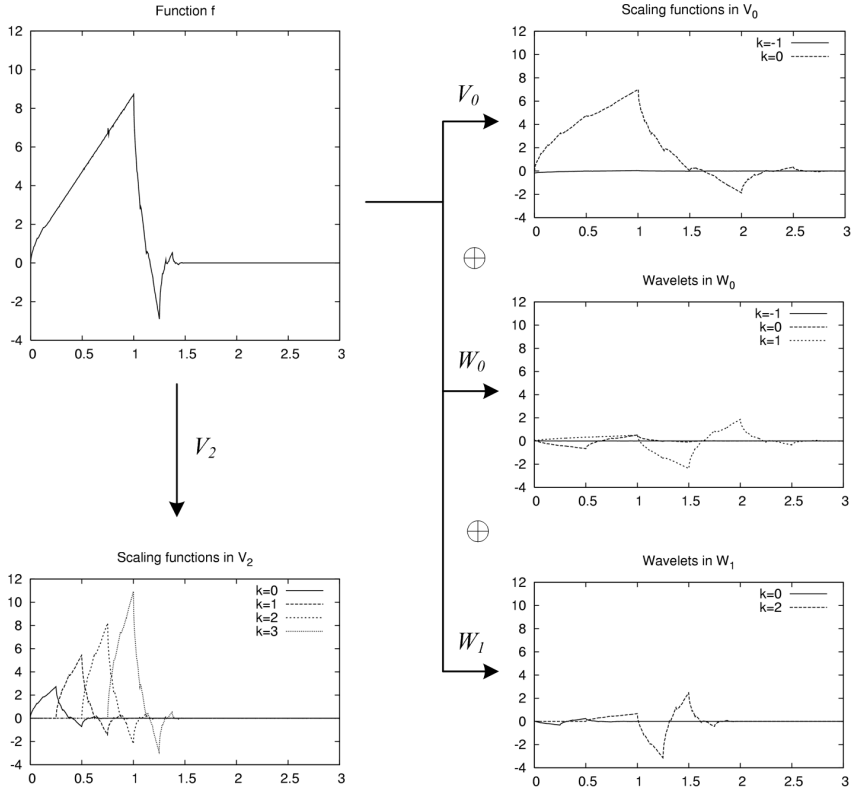
$$f(x) = \sum_{k\in\mathbb{Z}} c_{j_0,k}\phi_{j_0,k}(x). \tag{3.38}$$

For this representation of f, only the scaling coefficients $\{c_{j_0,k} : k \in \mathbb{Z}\}$ at resolution j_0 are required. In order to reduce their number, a natural approach is to apply the DWT to decompose them into scaling and wavelet coefficients of lower resolutions. Since the wavelet coefficients describe local details, we omit those with lowest absolute value to compress the function. This procedure prunes the least significant details of a function while still maintaining its global features. In general, this procedure for reducing the coefficient number is optimal in terms of a minimum approximation error [52].

[10] See Section 6.2 in [122].

Example Let us give a concrete example to convey a notion of the interplay between MRA, wavelet series expansion, and DWT. Figure 3.2 displays a function $f \in V_2$ and two ways of representing it. As underlying wavelet family, we utilize Daubechies2 wavelets, which we explain subsequently.

The chart in the lower left shows the scaling functions in V_2 that build f. Alternatively, we can apply the DWT and derive the coefficients of the scaling functions and wavelets at lower resolutions. Then, f can be described as the sum of scaling functions in V_0 plus wavelets in W_0 and W_1. For the sake of clarity, we do not display scaling functions or wavelets whose coefficients are below 0.001.



$$
\begin{aligned}
f(x) &= 1\cdot\varphi_{2,0}(x) + 2\cdot\varphi_{2,1}(x) + 3\cdot\varphi_{2,2}(x) + 4\cdot\varphi_{2,3}(x) && (V_2)\\
&= -0.13\cdot\varphi_{0,-1}(x) + 5.14\cdot\varphi_{0,0}(x) && (V_0)\\
&\quad + 0.02\cdot\psi_{0,-1}(x) - 0.38\cdot\psi_{0,0}(x) - 1.37\cdot\psi_{0,1}(x) && (W_0)\\
&\quad - 0.13\cdot\psi_{1,0}(x) - 1.28\cdot\psi_{1,2}(x) && (W_1)
\end{aligned}
$$

Figure 3.2: Representation of a function on different resolutions with the Daubechies2 family

3.5.1.4 Daubechies Wavelets

So far, we have not discussed the concrete forms of scaling function and wavelet. Both rely on the filter coefficients $\{h_k : k \in \mathbb{Z}\}$. In case the number of coefficients is finite, the wavelet

as well as the scaling function have a compact support. To achieve good approximation properties in the wavelet series expansion, the wavelets have to meet specific requirements, e.g. vanishing moments [82].

The most simple wavelet family are Haar wavelets. The corresponding scaling function and wavelet are displayed in Figure 3.3. As they are not smooth, their practical relevance is limited. However, it is interesting to note that many wavelet-based analysis approaches, e.g. [69, 25], still use Haar wavelets for the sake of simplicity, even though smoother wavelet bases exist.

In practice, Daubechies wavelets have become widely accepted. They are very convenient as they combine orthogonality with a compact support, different degrees of smoothness, and vanishing moments [48]. Figure 3.3 depicts some members of the Daubechies family. Daubechies wavelets are indexed by $N \geq 1$, which indicates their number of vanishing moments. The higher N, the smoother are the scaling function and the wavelet. Note that Haar wavelets are Daubechies wavelets with $N = 1$. A drawback of Daubechies wavelets is that they have no closed formula for their evaluation, except for the Haar wavelets. But their values can be arbitrarily closely approximated by means of the Daubechies-Lagarias algorithm [138].

3.5.2 Wavelet Density Estimation

Now that we have been introduced to wavelets, let us investigate wavelet density estimation [57, 82, 137, 138]. The theory of wavelet density estimation is elaborate as it covers a wide range of function spaces and error measures. Concrete results for wavelet density estimators, particularly those concerning their asymptotic properties, are typically very technical and require a lot of mathematical background on advanced calculus and algebra. Therefore, a detailed discussion of wavelet density estimation is beyond the scope of this work. For a detailed discussion, we refer the interested reader instead to [82, 138]. Our introduction to wavelet density estimation relies on less technical overviews given in [137] and [122].

Let $X_1, ..., X_n$ be an *iid* sample drawn from a random variable with unknown density $f \in L_2(\mathbb{R})$.[11] We assume that an orthonormal base consisting of compactly supported wavelets is given for $L_2(\mathbb{R})$.

The basic idea of wavelet density estimation is to exploit the wavelet series expansion (3.35) of the unknown density f. The idea is to estimate its unknown scaling and wavelet coefficients with the help of the sample. The fact that f is a density yields (analogous for $c_{j,k}$)

$$d_{j,k} = \langle f, \psi_{j,k} \rangle = \int f(x)\psi_{j,k}(x)dx = E(\psi_{j,k}(X)). \tag{3.39}$$

An unbiased estimate of the coefficients results from

$$\hat{c}_{j,k} = \frac{1}{n}\sum_{i=1}^{n} \phi_{j,k}(X_i), \;\; \hat{d}_{j,k} = \frac{1}{n}\sum_{i=1}^{n} \psi_{j,k}(X_i). \tag{3.40}$$

Combining these *empirical coefficients* with the wavelet series expansion of f delivers a naive

[11] Initial wavelet density estimation approaches for strictly stationary stochastic processes also exist, e.g. [106].

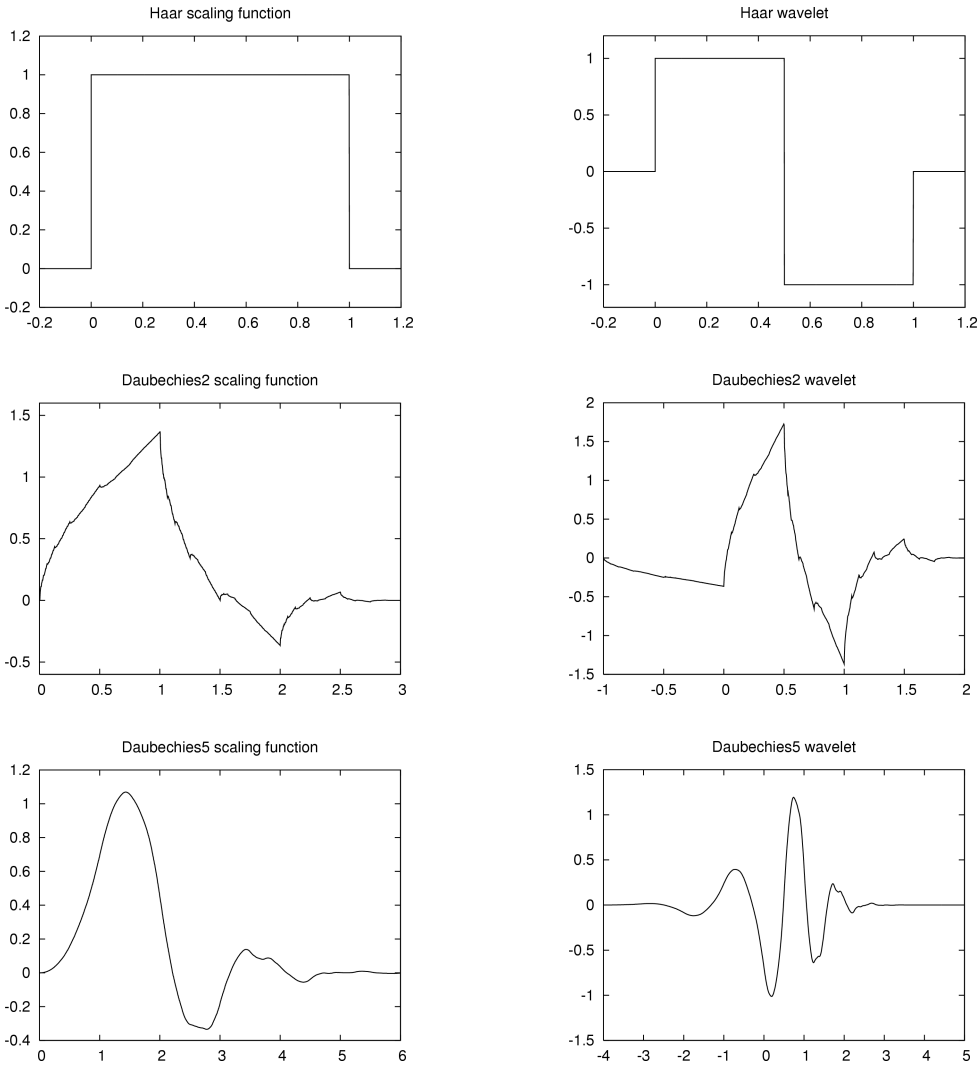


Figure 3.3: Examples for scaling function and wavelet pairs

wavelet density estimator (WDE)

$$\hat{f}(x) = \sum_{k\in\mathbb{Z}} \hat{c}_{j_0,k}\phi_{j_0,k}(x) + \sum_{j=j_0}^{\infty}\sum_{k\in\mathbb{Z}} \hat{d}_{j,k}\psi_{j,k}(x). \tag{3.41}$$

Except for the Haar family, WDEs may take on negative values in sparse regions. This problem can be addressed in various ways [138, 137]; we do not deal with this topic in this work.

Concerning the number of empirical coefficients in (3.41), the compact support of the wavelets leads to a finite number of empirical coefficients for each resolution as we will see in Section 3.5.2.4. But the unlimited number of resolutions induces an infinite overall number of empirical coefficients to compute for the naive WDE. Despite not being computable, a WDE with an infinite number of resolutions also has an infinite variance and is not consistent

with respect to the integrated squared error.[12] Thus, the main problem of wavelet density estimation is to choose a finite number of empirical coefficients. In the following section, we focus particularly on two well-established wavelet density estimators using a finite number of coefficients: linear and thresholded WDEs. For an overview of other approaches, we refer to [137].

3.5.2.1 Linear Wavelet Density Estimation

One way to achieve a finite number of empirical coefficients is to introduce a maximum resolution j_1^{lin}. The resulting *linear WDE* is defined as

$$\hat{f}(x) = \sum_{k \in \mathbb{Z}} \hat{c}_{j_1^{lin},k} \phi_{j_1^{lin},k}(x). \tag{3.42}$$

The crucial parameter of a linear WDE is j_1^{lin}. Note that the problem of its adequate choice is similar to the bandwidth problem of KDEs.

To determine the quality of linear WDEs, the MISE is a suitable measure of discrepancy. The MISE can be split into bias and variance with both components depending on the resolution j_1^{lin}. As for KDEs, a trade-off between bias and variance of the estimator occurs; decreasing one component comes at the expense of increasing the other. But one can determine the resolution j_1^{lin} that minimizes this sum.

To get an idea of the rate of convergence of linear WDEs in terms of the MISE, let us give a concrete result from [82] (Theorem 10.1) without going into details. Let f belong to the following Sobolev class of functions: $W_L^m = \{f : ||f^{(m)}||_2 \leq L, f \text{ is a pdf}\}$ with $m \in \mathbb{N} \setminus \{1\}$ and $L > 0$. The number m denotes the regularity of f.[13] Under suitable assumptions on the wavelet family and given the minimizing $j_1^{lin} = j_1^{lin}(n)$, one can show

$$E||\hat{f}_{j_1^{lin}(n)} - f||_2^2 \leq C n^{-\frac{2m}{2m+1}} \text{ for some } C > 0. \tag{3.43}$$

Hence, the smoother the unknown density is, i.e. the higher m, the better is the estimation quality of linear WDEs. Similar results can be obtained for more general function spaces, so called Besov spaces.[14]

For practical purposes, the optimal j_1^{lin} suffers from the drawback that it depends on the unknown regularity properties of f. To overcome this drawback, [137] proposes to use

$$j_1^{lin} = \frac{\log_2 n}{3} - 2 - \log_2 \sigma \tag{3.44}$$

as an upper bound in case Daubechies wavelets are used. The unknown standard deviation σ can be estimated by the root of the sample variance. This setting is appropriate for linear WDEs based on Haar wavelets. As Haar wavelets are the least smooth Daubechies wavelets, the use of this setting as an upper bound is reasonable. Concerning more advanced strategies for the setting of j_1^{lin}, their computational complexity renders their application in the data stream scenario difficult.

In general, linear WDEs are a convenient technique for the estimation of smooth densities. However, for the case of densities with heterogeneous smoothness properties, thresholded wavelet density estimators are preferable.

[12] See also [138] p. 219.
[13] See [82], page 128 for more details.
[14] See [138], Theorem 7.2.6.

3.5.2.2 Thresholded Wavelet Density Estimation

The basic idea of thresholded wavelet density estimators is to perform an adaptive fit to the local smoothness of the density. In their seminal work [57], Donoho et al. introduced a nonlinear WDE which applies a thresholding procedure to its empirical wavelet coefficients. This delivers a *thresholded WDE*

$$\hat{f}(x) = \sum_{k\in\mathbb{Z}} \hat{c}_{j_0^{thr},k}\phi_{j_0^{thr},k}(x) + \sum_{j=j_0^{thr}}^{j_1^{thr}} \sum_{k\in\mathbb{Z}} \tilde{d}_{j,k}\psi_{j,k}(x) \tag{3.45}$$

with $\tilde{d}_{j,k}$ denoting a thresholded wavelet coefficient. By a careful selection of empirical wavelet coefficients, local features of f, such as discontinuities or sharp cusps, shall be detected without introducing spurious local artifacts.

In [57], two thresholding strategies were proposed: soft and hard thresholding. *Hard thresholding*

$$\tilde{d}_{j,k} = \begin{cases} \hat{d}_{j,k}, & |\hat{d}_{j,k}| > \lambda \\ 0, & \text{else} \end{cases} \tag{3.46}$$

sets coefficients with absolute value smaller than λ to zero. *Soft thresholding* shrinks large coefficients:

$$\tilde{d}_{j,k} = \begin{cases} \hat{d}_{j,k} - \lambda, & \hat{d}_{j,k} > \lambda \\ \hat{d}_{j,k} + \lambda, & \hat{d}_{j,k} < -\lambda \\ 0, & \text{else.} \end{cases} \tag{3.47}$$

Hence, a thresholded WDE has the following parameters: the resolutions j_0^{thr}, j_1^{thr} and a threshold λ. An appropriate setting of the threshold is crucial since it dictates which local details are kept.

Concerning the asymptotic properties of thresholded WDEs, Donoho et al. provide in [57] a remarkable result: *"Our main point is that the same form of estimator, based on simple thresholding of the wavelet coefficients, achieves nearly optimal performance, in terms of rates of convergence over a variety of function spaces. Here, near optimality means that the rates are best possible except possibly for terms logarithmic in sample size."* The variety of function spaces refers to specific Besov spaces. Besov spaces are very general function spaces, which contain most other common function spaces as special cases. As the results given in [57] are very technical, we do not go into their details.

For the parameters $j_0^{thr}, j_1^{thr}, \lambda$ the following settings are reasonable according to [138]:[15]

$$n^{\frac{1}{2r-1}} \leq 2^j \leq \frac{n}{\log n} \text{ where } r \text{ is the regularity of the wavelet,} \tag{3.48}$$

$$\lambda_j = K\sqrt{\frac{j}{n}} \text{ for a suitably chosen constant } K > 0. \tag{3.49}$$

For the resolutions, it follows

$$j_0^{thr} = \left\lceil \frac{\log_2 n}{2r-1} \right\rceil, j_1^{thr} = \lfloor \log_2 n - \log_2(\log n) \rfloor . \tag{3.50}$$

[15] See Section 7.3 in [138].

Note that the threshold λ_j depends on j; it is level-dependent. Concerning the concrete setting of the constant K for a given sample, neither [57] nor [138] give an explicit answer.

[82] attends to this question and discusses some alternatives for a level-dependent setting of the threshold. One approach is to use multiples of $\max_{k \in \mathbb{Z}} |\hat{d}_{j,k}|$ for each resolution.[16] Due to its simplicity and its low computational cost, we adopt this strategy throughout this work. [82] discusses in Section 11.2 other, more complex strategies for a data-dependent setting of the threshold. Instead of local thresholding for each wavelet coefficient, one can also apply global thresholding for a complete level of coefficients or block thresholding for blocks of coefficients. For more details, we refer to [82].

Overall, thresholded WDEs provide an automatic adaptation to the unknown smoothness of the density f. In case f is non-smooth, they are superior to linear WDEs.[17]

3.5.2.3 Multivariate Wavelet Density Estimation

Up to this point, we have discussed the case of univariate wavelet density estimators. Their multivariate counterparts are generalizations of the univariate case [138]. Their development starts with a multi-dimensional MRA. The MRA for two dimensions is discussed in [138] and in [122]. Given a multivariate MRA, a wavelet density estimator can be defined in the same manner as in equation (3.41). As in the univariate case, the essential step is to estimate the scaling and wavelet coefficients and to limit their overall number.

For the case of linear multivariate WDEs, Tribouley introduced in [133] a strategy for choosing j_1^{lin}, which bases on a cross validation procedure.

For thresholded WDEs, strategies for the choice of confining resolutions as well as suitable thresholding procedures must be developed.

3.5.2.4 Practical Considerations

Now that we have discussed concrete WDEs, let us examine some practical aspects for univariate WDEs. Specifically, we recapitulate the essential steps required for the computation of a WDE. Moreover, we examine which empirical coefficients must be computed at a given resolution. As we will see, a scaling function with compact support induces a finite number of empirical coefficients. This is an important result as it induces a finite overall number of coefficients if the resolutions are limited (which is the case for linear and thresholded WDEs). In addition to the computation of a WDE, we also discuss its efficient evaluation.

Computation of WDE Let $[a, b]$ be the support of the scaling function ϕ. The support of a translated and dilated version $\phi_{j,k}$ is $[2^{-j}(a+k), 2^{-j}(b+k)]$. Additionally, we assume the sample $X_1, ..., X_n$ to be ordered as it facilitates the subsequent computation of the empirical coefficients.

The first step for the computation of linear as well as thresholded WDEs is to determine an initial resolution j_1^{lin} and j_1^{thr} respectively. Given this resolution, which we abbreviate to j_1, the next step is to compute the empirical scaling coefficients at this resolution according to equation (3.40). Due to the compact support of ϕ, the number of non-zero empirical coefficients is finite. To show this fact, let $k_{min}, k_{max} \in \mathbb{Z}$ be indexes so that $\hat{c}_{j_1,k} = 0$ for

[16] See Section 10.3 in [82].
[17] See Section 10.4 in [82].

$k \notin \{k_{min}, ..., k_{max}\}$. As X_1 and X_n are minimum and maximum of the sample respectively, k_{min} is the smallest index with $\phi_{j_1,k}(X_1) \neq 0$ and k_{max} the largest with $\phi_{j_1,k}(X_n) \neq 0$. It follows that

$$k_{min} = \min\{k \in \mathbb{Z} : 2^{-j_1}(b+k) \geq X_1\} \text{ and } k_{max} = \max\{k \in \mathbb{Z} : 2^{-j_1}(a+k) \leq X_n\}. \quad (3.51)$$

Therefore, they can be determined via

$$k_{min} = \left\lceil 2^{j_1} X_1 - b \right\rceil, k_{max} = \left\lfloor 2^{j_1} X_n - a \right\rfloor. \quad (3.52)$$

The next step is to compute for all $k \in \{k_{min}, ..., k_{max}\}$ the corresponding empirical scaling coefficients $\hat{c}_{j_1,k}$. For the computation, we can exploit the ordering of the sample $X_1, ..., X_n$. More precisely, for $\hat{c}_{j_1,k}$ with $k \in \{k_{min}, ..., k_{max}\}$, we only evaluate those X_i with $\phi_{j_1,k}(X_i) \neq 0$. By means of a binary search on $X_1, ..., X_n$, we can efficiently determine the first X_i with $\phi_{j_1,k}(X_i) \neq 0$. Then we sum up all non-zero $\phi_{j_1,k}(X_j), j \geq i$.

For thresholded WDEs, an additional thresholding procedure is performed on the empirical wavelet coefficients at the resolutions $j_0^{thr}, ..., j_1^{thr}$. It is important to note that this does not require the computation of the according wavelet coefficients at these resolutions. Instead, it suffices to compute the empirical scaling coefficients for j_1^{thr}. With the help of the DWT, the empirical wavelet coefficients of lower resolutions can be determined.

Concerning the computational complexity of the considered WDE types, we expect thresholded WDEs to be more demanding than linear WDEs. First, thresholded WDEs perform an additional thresholding procedure, which necessitates a DWT beforehand, the computation of the level-dependent thresholds, and the application of the thresholding procedure to the empirical wavelet coefficients. Second, their initial resolution will typically be larger than that of linear WDEs. In fact, a comparison of (3.50) and (3.44) shows that j_1^{thr} will be larger than j_1^{lin}. According to (3.52), these resolutions are crucial for the number of empirical coefficients that must be computed.

To give a feeling for the effects, let us consider a concrete example. One of the real data streams we examined in our experiments, termed Tide [98], consists of 8746 observations; its minimum is -38 and its maximum 79. For the linear WDE, we computed $j_1^{lin} = -1$ as initial resolution as well as $k_{min} = -24$ and $k_{max} = 39$. In comparison to the linear WDE, the thresholded WDE had higher resolutions of $j_1^{thr} = 11$ and $j_0^{thr} = 6$; its index range was $k_{min} = -76760$ and $k_{max} = 162675$. We observe that the thresholded WDE requires significantly more empirical coefficients which results in higher computational cost.

Evaluation of WDE To evaluate a WDE at a given point x, the same considerations as above allow us to confine the evaluation of the wavelet and scaling functions at the different resolutions to the "non-zero ones". More precisely, let a WDE be described in terms of its empirical scaling and wavelet coefficients. For the sake of simplicity, we consider the evaluation of the scaling functions at level j. The evaluation of the wavelets at different resolutions can be performed analogously.

Let $[a, b]$ be the compact support of the scaling function ϕ. Thus, $\phi_{j,k}$ has support $[2^{-j}(a+k), 2^{-j}(b+k)]$. Again, we determine indexes k_{min}, k_{max} so that $\phi_{j,k}(x) = 0 \ \forall k \notin \{k_{min}, ..., k_{max}\}$. It follows that

$$k_{min} = \left\lceil 2^j x - b \right\rceil, \ k_{max} = \left\lfloor 2^j x - a \right\rfloor. \quad (3.53)$$

All that remains to be determined is which of the empirical coefficients of the WDE at resolution j fall into the range $k_{min}, ..., k_{max}$.

3.5.3 Viability of Wavelet Density Estimation

After this overview of WDEs, we summarize their essential properties:

- The basic form of WDEs is simple to compute. As they require no a priori knowledge of the unknown density, WDEs are completely data-driven.
- Due to their localization properties, WDEs can adapt automatically to the unknown smoothness of the density under estimation. While linear WDEs are suitable for estimating smooth densities, thresholded WDEs are suitable for estimating densities with local irregularities.
- WDEs exhibit remarkable asymptotic properties for a wide range of densities. In particular, they achieve near optimal performance for a variety of function spaces.

Due to those properties, wavelet-based density estimation is a highly promising approach. However, its application to database related topics has not been thoroughly investigated yet.

3.5.4 Adaptation to Data Streams

Against the background of data stream analysis, wavelet density estimation can be exploited to gain deeper insights into the characteristics of a stream. The fact that specific types of WDEs can cope with smooth as well as with non-smooth densities is important in this context.

As we assume that a data stream generates a continuously increasing *iid* sample of a random variable X with unknown density f, the requirements for the application of wavelet density estimation are met. However, as a closer look clarifies, neither linear WDEs nor thresholded WDEs can be directly applied to data streams since their computation violates the processing requirements for streams listed in Section 3.1.2. More precisely, their computation does not satisfy requirements 1, 3, and 7 for the following reasons. First, the resolutions of linear and thresholded WDEs, i.e. $j_0^{thr}, j_1^{thr}, j_1^{lin}$, depend on the sample size. As the data stream - which constitutes the sample - continuously increases, the resolutions will also increase. According to (3.40), the computation of the empirical coefficients depends on the resolution. As a consequence, a change of the resolution implies that all empirical coefficients need to be recomputed. This, however, requires the storage of all elements of the stream, a fact that violates processing requirement 1. The second problem is the limitation of a constant amount of memory that can be allocated. As (3.52) shows, the number of empirical coefficients to compute can not be a priori determined since it depends on the extrema of the sample and on the sample size. Hence, we can neither confine the number of empirical coefficients, nor can we adjust it. These facts violate processing requirements 3 and 7.

Overall, we conclude that WDEs are not directly applicable to data streams. Instead of an exact adaptation, an approximate solution must be found that fully complies with the processing requirements for data streams.

3.6 Comparison of Density Estimation Methods

After we have discussed different methods for nonparametric density estimation, we briefly compare them. So far, besides a naive density estimator, we have discussed kernel density estimators and wavelet density estimators in depth.

A common density estimation technique not yet discussed is the histogram [128]. Due to its simplicity, the histogram is often used for practical purposes. For an overview of histograms, we direct the reader to [128]. As discussed in Section 2.1, initial approaches to adapt histograms to the data stream scenario have already been developed.

A variety of histogram approaches exist. They all share the property that they put the sample points into bins. Each bin corresponds to a rectangle. The width of this rectangle is equal to the width of the bin and the height is proportional to the number of sample points falling into the bin. To evaluate a histogram at a given point x, one has to determine the bin x falls into. The value of the histogram at x is the height of the corresponding rectangle.

Even though they are widely used, histograms have severe drawbacks. First, they produce non-smooth estimates. Second, an optimal binning requires the knowledge of the data range in advance. Third and most important, their convergence rate with respect to the MISE is $O(n^{-2/3})$. In comparison to this rate, the corresponding convergence rates of kernel and wavelet density estimators are better. Due to these reasons, we do not consider histograms in this work.

For the comparison of wavelet and kernel density estimators, let us cite [82], p. 180: *"For data analytic purposes with small to moderate data size a kernel estimate may be preferred for its simplicity and wide distribution. For finer local analysis and good asymptotic properties the wavelet estimator is certainly the method to be chosen."* Let us emphasize that the simplicity of kernel density estimators is a strong point as it facilitates the development of a suitable adaptation to data streams. For wavelet and kernel density estimators, it can be stated that both combine a comparably simple form with powerful mathematical properties. They provide consistent estimates of the unknown density. Due to their nonparametric nature, they are particularly relevant for practical applications.

3.7 A naive Density Estimator over Data Streams

Subsequent to the presentation of different statistical methods for estimating an unknown density, we will present a naive approach to adapt these methods to the data stream scenario.

The basic idea is to continuously maintain a constant-size sample of the already processed elements of the stream. Given the current sample, one can, at anytime, build arbitrary density estimators, e.g. KDEs, WDEs, histograms, with respect to the sample elements. In order to produce valid estimators, a necessary prerequisite is that the sample is always representative, i.e., it has to constitute an *iid* sample of the already processed elements. In this context, reservoir sampling [139] is a suitable technique to maintain this sample while processing the stream. Figure 3.4 illustrates the basic idea of this approach.

However, as a closer look reveals, this naive sampling-based approach suffers from severe drawbacks. More precisely, it does not fulfil all processing requirements for streams. First, the adaptation to the available system resources is only partially satisfied. While the size of the sample can be reduced by simply dropping elements randomly, it is not possible to increase

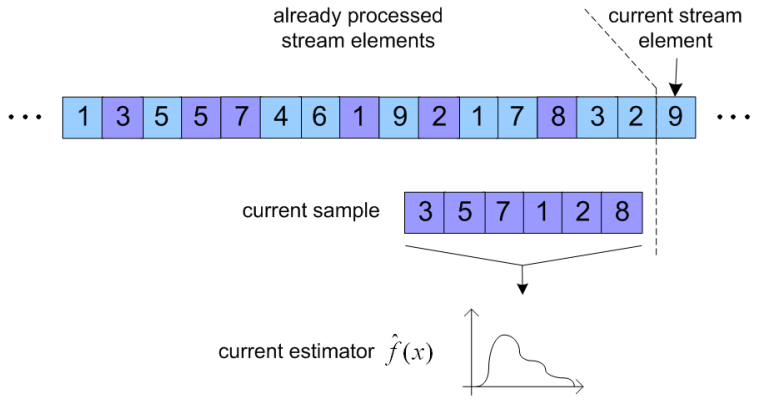


Figure 3.4: Naive density estimation approach for data streams

the size of the sample at runtime without accessing past data. Second, the estimator only depends on the current sample elements, i.e., all other processed elements do not contribute. But density estimators require increasing sample sizes to improve their quality. Overall, this approach is inadequate even though it is simple and can be applied with arbitrary estimation techniques. Hence, we need more sophisticated techniques to provide suitable density estimators over data streams.

In the course of this work, we tackled this task and developed convenient wavelet and kernel density estimators over data streams, which we will present in the following chapters. Besides discussing their details, we will also address their implementation concepts as this is a crucial prerequisite for their application. Since we embedded all implementations in our library XXL, the next chapter gives an overview of XXL and particularly of its subpackage PIPES. The concepts discussed in this overview are necessary for understanding the later presented implementations of our techniques.

Chapter 4

An Overview of XXL and PIPES

In this chapter, we present an overview of our library XXL and of its subpackage PIPES. Before we discuss the salient features of XXL, we start in Section 4.1 with an introduction to its general philosophy. As XXL makes extensive use of functional programming principles, Section 4.2 introduces our function concept. One of the core features of XXL is its cursor package, which is sketched in Section 4.3. The stream processing facilities of XXL are gathered in the package PIPES, whose essential building blocks we present in Section 4.4. Section 4.5 describes the implementations of mathematical objects in XXL. Section 4.6 concludes this chapter with an outlook on other packages of XXL and their functionality.

4.1 XXL

An aspect often neglected in the development of a new technique is the provision of a comprehensible implementation. However, this aspect is crucial for the applicability and comparability of a new technique. A common approach is to use rudimentary implementations for the single purpose of running a set of specific experiments. As a consequence, those implementations often suffer from a poor design and limited applicability, not to mention that they are not freely available in the majority of cases. In contrast, we strive to provide flexible and comprehensible implementations of density estimators over data streams. These implementation libraries can be easily used as building blocks for more complex stream mining techniques. To accomplish this objective, we incorporated all implementations into our library XXL.

XXL (eXtensible and fleXible Library) is a platform-independent Java library combining generic frameworks as well as concrete toolboxes for the implementation of advanced query processing functionality [136, 28]. The decision for the development of a library was based on two reasons. First, as monolithic DBMSs are often too inflexible to integrate new, user-defined functionality, a library would facilitate this process. Second, a library can also serve as a repository for new algorithms and use-cases, particularly with regard to the fair comparison of different techniques. The development of XXL is characterized by strict guidelines in order to achieve the following objectives:

- *Extensibility:* Functionality in XXL is typically modeled with different degrees of abstraction which range from interfaces as most abstract structure over preimplementations of necessary functionality to ready-to-use classes implementing the full function-

ality. To integrate new functionality into XXL, one can start at one of these stages, depending on how much flexibility is required. Another important concept in this context are so-called wrapper classes, which bridge the gap from new functionality to XXL by wrapping new objects into XXL objects.

- *Flexibility:* The abstraction hierarchy in the implementation of functionality also ensures a high degree of flexibility. In general, the flexibility reflects in highly parameterized implementations. Their modular design and adaptable parameter settings facilitate the easy adaptation to the user's requirements.

- *Applicability:* Even though the classes in XXL are very generically designed, which typically comes along with a higher complexity, they still offer a user an intuitive access to their functionality. This is accomplished by means of the detailed documentation and the illustrative use-cases each class is equipped with. Default implementations and preimplementations of common functionality also facilitate the use of a class.

- *Reusability:* The generic implementations render an easy reuse of existing functionality in different contexts possible. For example, XXL comprises implementations of commonly used data structures like heaps and binary search trees, which can process arbitrary objects provided that they are suitably parameterized. To simplify their use, many classes rely on well-established design patterns [61].

- *Connectivity:* To widen the scope of XXL and its applications, it was coupled with several external Java libraries; this coupling created synergy effects on both sides. These libraries augment the functionality of XXL with regard to: XML storage, support of spatial predicates and functions, matrix operations, to list just a few. Specific wrapper classes play an important role in this coupling; they wrap the objects of an external library to XXL objects and vice versa.

The members of our database research group continuously extend and maintain XXL. XXL is freely available under the terms of the GNU LGPL license. Its current version consists of more than 1200 Java files and 3200 classes, including anonymous classes, resulting in circa 320.000 lines of code. The latest version requires Java 1.5. It is worth mentioning that large parts of XXL already support generics, i.e., the classes ensure a type-safe use by incorporating types as parameters. For the current version of XXL, its documentation as well as illustrative web demos, we direct the reader to our homepage [121].

XXL covers a wide range of functionality related to advanced query processing. This includes for example index structures like B-trees and R-trees, raw access to the file system, spatial joins, an SQL parser, and much more. For the sake of clarity, XXL is divided into several subpackages with each one gathering specific functionality, e.g., the package `predicates` consists of all predicates. Let us briefly discuss those parts of XXL that were particularly relevant for the implementation of our density estimators over streams.

4.2 Functions and Aggregation Functions

Functions An essential concept of XXL is to encapsulate abstractions with functions, which complies with the functional programming paradigm. As the Java SDK from Sun provides no

equivalent for a function, XXL provides an abstract class `Function` that closes this gap. The implementation of a new function requires overwriting one of its abstract `invoke` methods. The new functionality is encapsulated within this method. A crucial advantage of `Function` is that it can be exploited to implement higher-order functions, i.e., functions with other functions as input or output. More precisely, by means of the method `compose`, a new function can be created as a composition of existing functions. For practical purposes, functions are often implemented with an internally stored state. Another practical aspect is to keep the overall number of explicit classes extending `Function` small. To achieve this goal, it is advisable to implement a function as an anonymous class if it is only used at a single point in the implementation. An anonymous class is a local, nameless class that is defined and used at the same place. XXL makes extensive use of the function concept. For example, the parameters of many implementations are functions since this offers an intuitive way to keep an implementation flexible and extensible.

To convey an impression of how to use a function, let us give an example:

```
Function<Double, Double> div = new Function<Double, Double>() {
    public Double invoke(Double arg1, Double arg2) {
        return arg1 / arg2;
    }
};
```

This function implements in the corresponding `invoke` method the division of two `Double` objects. The code `<Double, Double>` refers to the aforementioned generics; it indicates that the function maps `Double` objects to `Double` objects. In the same manner, we can implement two functions `sin` and `cos` which compute sine and cosine for a given `Double` object in their `invoke` method. Given `div`, `sin`, and `cosine`, the tangent can be easily implemented as their composition:

```
Function<Double, Double> tan = div.compose(sin, cos);
```

Generally, a Function is parameterized with a type; its invoke methods are called with objects of this type or extensions of it.

A function often used in database systems is the predicate, which either returns true or false for a given object. For performance reasons and the sake of semantic clarity, a predicate is not implemented as `Function`, but as a separate abstract class `Predicate`.

Aggregation Functions An aggregation function is a binary function that is particularly useful for implementing the iterative computation of an aggregate. More precisely, an aggregation function $f(\cdot,\cdot)$ uses the current aggregate agg_n and a new element x_{n+1} to compute the new aggregate via $agg_{n+1} = f(agg_n, x_{n+1})$.

This concept is of utmost importance for this work as an aggregate can also be defined in a broader sense than the common descriptive statistics like minimum and maximum which it is typically associated with. To cite [125]: *"An aggregation process is a collection of activities that produce a statistical object starting from a given set of raw data."* A statistical object may also be a histogram or the previously discussed kernel and wavelet density estimators. Hence, the idea is to encapsulate the computation of density estimators within specific aggregation functions.

Within XXL, the abstract class `AggregationFunction` models a general aggregation function. A class extending `AggregationFunction` must overwrite the binary `invoke` method. The reason for implementing aggregation functions in a separate class and not as extensions of `Function` is the inherently ensured type safety. An aggregation function is invoked with two different types: the type of the aggregate and the type of the next element. A Function only supports one type and extensions of it.

As an example, let us implement the iterative computation of the minimum. The `invoke` method of the corresponding extension of `AggregationFunction` is implemented as

```
public Double invoke(Double aggregate, Double value) {
    return Math.min(aggregate, value);
}
```

provided that the current minimum is stored outside this aggregation function.

To implement an iterative computation of an aggregate, one has to define the according aggregation function and couple it with an `Aggregator`. An `Aggregator` implements an operator that receives an element, updates the current aggregate with respect to its aggregation function, and delivers the new aggregate as result. As an aggregator already delivers partial results, e.g. for the minimum of a data set, it follows the idea of online aggregation as proposed in [90]. XXL provides two `Aggregator`, one in the package `cursors` and one in the package `pipes`.

4.3 Cursors

In order to process queries, a fundamental prerequisite is the availability of suitable mechanisms to access and process the underlying elements. Depending on how the elements are accessed, we distinguish between demand-driven and data-driven processing. In the latter case, the elements arrive autonomously and must be directly processed. The necessary functionality for this kind of processing is given in the package `pipes`. In the case of demand-driven processing, the elements do not arrive autonomously; they are accessed on demand. The package `cursors` unifies the according functionality for this case.

With the `Iterator` interface, Java already provides a mechanism to iterate over the elements of a collection. However, to implement database operators by means of the ONC concept [72], XXL provides with the `Cursor` interface a more appropriate solution. `Cursor` extends `Iterator` by adding methods to support for example an explicit open and close phase.

Based on cursors, XXL provides an elaborate demand-driven operator algebra as one of its core features. This algebra consists of a set of operators which receive cursors as input, modify the elements of the cursors, and provide the output as cursors. Essentially, they are physical operators that implement the operations defined in the extended relational algebra. For example, a `Filter` implements a selection by filtering out those elements that do not satisfy a user-defined predicate. Another operator is the aforementioned `Aggregator`, which computes aggregates in an iterative manner.

The operators of this cursor algebra are complemented by input cursors which wrap specific data sources, e.g. `java.sql.ResultSet`, into a cursor. The entirety of these cursors can be exploited to implement the operator trees corresponding to a query.

For a more detailed discussion of this demand-driven cursor algebra, we refer to [136, 28]. For this work, the data-driven processing facilities of the package `pipes` are more relevant. It is worth mentioning that the functionality of `cursors` and `pipes` can also be coupled by means of specific wrapper classes.

4.4 PIPES

In recent years, the XXL development team has primarily focused on the PIPES project. PIPES is an infrastructure that consists of the essential building blocks necessary to implement a DSMS. It offers the functionality to express, implement, and execute continuous queries over data streams. PIPES is seamlessly integrated into XXL. Its functionality is gathered in the subpackage `pipes`. In [103] and [27], the rich functionality of PIPES was demonstrated. As we will see, its facilities can also be exploited to implement stream mining algorithms. We demonstrated this approach in [84] and [87] against the background of using density estimators as stream mining tools.

Query Graphs and Operators As continuous queries are typically long-running and may share subqueries, they are managed in a directed, acyclic query graph. Note that traditional queries in a DBMS are managed in operator trees. Formally, one has to distinguish between the logical and the physical query graph with nodes corresponding to the logical operations of an algebra and to their physical implementations respectively. Concerning an algebra for continuous queries, [102] presents a sophisticated approach which relies on attributing elements with a validity in the form of a time interval. By means of this validity interval, the operations of the extended relational algebra, except sorting, can be temporally defined with respect to sliding windows and, most important, implemented in a non-blocking manner.

We concentrate on physical query graphs, whose nodes are physical operators. We distinguish between three types of nodes in a query graph:

- *Source:* A source transfers its elements to all subscribed sinks.
- *Sink:* A sink receives the elements of all sources it is subscribed to.
- *Pipe:* A pipe as an intermediate node is both, source and sink. It receives elements from its sources, processes them, and transfers the results to its sinks.

In order to build a query graph, the corresponding nodes can be connected by means of a flexible publish/subscribe mechanism [103]. This mechanism ensures that as long as a node is subscribed to another node, it receives all elements published by this node. A vital feature of the mechanism is the direct interoperability [32], which refers to a direct communication of connected nodes without intermediate queues. However, it is also possible to place specific queues between operators.

To facilitate the development process, PIPES provides abstract preimplementations for each node type, which already implement basic functionality. This includes amongst others the publish/subscribe mechanism. Consequently, a new operator only has to implement its specific functionality and inherits the processing logic within the graph from the preimplementations. As an example, let us implement an operator that filters out all odd elements of an `Integer` stream:

```
AbstractPipe<Integer, Integer> filter = new AbstractPipe<Integer, Integer> {
    public Integer process(Integer i) {
        if(i%2 == 0)
            super.transfer(i);
    }
};
```

`AbstractPipe` is the abstract preimplementation of an operator; it only requires to implement the `process` method. PIPES also offers preimplementations as well as concrete implementations of sources and sinks, e.g. random number sources, sinks for plotting real-valued functions.

Implementation of Analysis Tasks over Data Streams The facilities PIPES provides for an easy implementation of new operators and for combining them in complex operator graphs give us a natural starting point for implementing density estimators over streams. The main premise in this context is that we implement these techniques as operators by extending `AbstractPipe`. A crucial benefit of doing so is that we can unify several analysis tasks in one graph and additionally gain advantage from subquery sharing, i.e., one analysis technique can serve multiple other analysis techniques simultaneously. For example, we want to plot the density of a data stream and compute its summary statistics. Instead of implementing these tasks separately, we build an operator graph with the following nodes. The first node corresponds to the data source. The second node is connected to the first one and computes online density estimates. The third and fourth node are connected to the second node; one computes the summary statistics with respect to the current density estimate and the other one plots the estimate. Another, more complex example is the heart rate monitor presented in Chapter 1, which we also implemented as a query graph in PIPES.

The ability to combine several analysis tasks in one query graph also renders the easy integration of new analysis techniques based on density estimators possible. Recall from Chapter 1 and Chapter 2 that density estimators are used as ingredients in further mining and analysis tasks. In order to make this functionality available in the data stream context, the density estimators we developed in the course of this work can be used as essential building blocks. In order to implement new functionality on top of our density estimators, corresponding operators must be developed, which are subscribed to a density estimator operator. Due to the subscription, the new operator continuously receives the current density estimates and can exploit them to tackle its analysis tasks.

Runtime Environment Besides functionality for constructing query graphs, PIPES also offers separate frameworks for the essential runtime components of a DSMS. Following the general philosophy of XXL, the frameworks are modularly designed and highly parameterized in order to keep them flexible and extensible.

The *scheduler* is responsible for running the query graph, i.e., it decides which components of the query graph are currently processed by the CPU. The scheduling framework of PIPES ensures a high degree of flexibility by means of the flexible three-layer architecture presented in [32]. This architecture circumvents the drawbacks of assigning a separate thread to each operator or assigning one thread to the complete graph. Instead, it follows a hybrid

approach which assigns multiple threads to disjoined subgraphs, complemented by suitable graph partitioning strategies.

As the available memory is a limiting factor in a DSMS, PIPES provides a *memory manager*. Its objective is to distribute the available memory dynamically among the stateful operators, e.g. aggregation and join. This guarantees that the DSMS can cope with a volatile system load induced by queries entering or leaving the system and varying stream rates.

A main question for a given set of continuous queries is how to combine them optimally within a query graph. PIPES tackles this problem within its optimization framework. Using the cost model proposed in [33], the *optimizer* chooses the best query graph, i.e., the one with minimum cost, from a set of equivalent query graphs, which were determined with the help of suitable heuristics. As the set of queries underlying a query graph changes during runtime, the optimizer must also dynamically re-adjust the graph by integrating new queries or discarding old ones. The implementation of a dynamic reoptimization is part of the ongoing and future work in PIPES.

The decisions of scheduler, memory manager, and optimizer depend crucially on the collection of metadata about the characteristics of the query graph and the current system state. The *query monitor* of PIPES is an instance that can access the metadata of an operator and trigger the metadata computation. As the computational cost of monitoring metadata is not negligible, the monitor supports on demand monitoring of metadata instead of monitoring all available metadata for all operators in the graph. To provide an on demand monitoring, the query monitor uses a secondary publish/subscribe mechanism to access the relevant metadata of an operator only if necessary.

For illustrative purposes, Figure 4.1 gives an overview of the main components of PIPES.

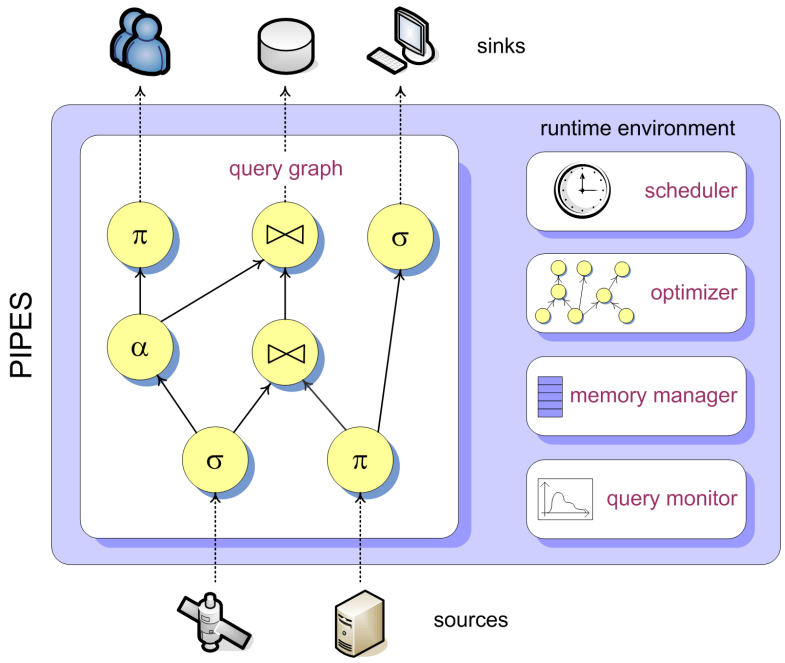


Figure 4.1: Main components of PIPES

4.5 Math

The package `math` comprises a variety of implementations of mathematical objects with a particular focus on statistics. Its main functionality was developed in [21]. The majority of classes developed for implementing our density estimators over data streams will be incorporated into this package.

The subpackage `functions` provides several classes that implement functionality related to real-valued functions. Its core interfaces are `RealFunction`, `Differentiable`, and `Integrable`. While the first one models a one-dimensional real-valued function $f : \mathbb{R} \to \mathbb{R}$, the second and the third one have methods that deliver derivative and antiderivative of a real-valued function. Additionally, this package provides several wrapper classes, e.g. to wrap a `Function` into a `RealFunction`. Note that the previously discussed `AggregationFunction` is also contained in this package.

The subpackage `numerics` comprises several implementations of numeric methods. In addition to methods for the numerical integration of a function, it also includes methods for the computation of cubic Beziér splines.

The subpackage `statistics` is the largest subpackage of `math`. The majority of its classes are specific aggregation functions, implementing the computation of diverse aggregates, e.g. simple descriptive statistics like count, min, max, average, and variance. Most aggregation functions share the property that they internally store a state required for their computations. For example, to compute the count aggregate, the current number of already processed elements must be stored as state.

An important aggregation function is `ReservoirSample`, which implements the reservoir sampling method proposed in [139]. This method is particularly suited for online processing as it maintains an *iid* sample of the already processed elements and can be implemented as a single scan over the data. Hence, it can also be applied in the data stream context. While processing, `ReservoirSample` stores the current reservoir elements internally as state.

The one pass implementation of reservoir sampling can be exploited to implement the naive approach for online density estimation we sketched in Section 3.7. Its basic idea is to compute a density estimator with respect to the current sample. To convey an impression of the ease with which such complex functionality can be implemented in XXL, let us briefly discuss the according implementation:

```
AggregationFunction<EpanechnikovNKDE,Double> sampleKDE =
    new AggregationFunction<EpanechnikovNKDE,Double>() {

    Number[] reservoir = new Number[100];

    ReservoirSample reservoirSample = new ReservoirSample(reservoir,
        new ReservoirSample.XType(100));

    public EpanechnikovNKDE invoke(EpanechnikovNKDE old, Double next) {
        reservoir = reservoirSample.invoke(null, (Number) next);
        return new EpanechnikovNKDE(reservoir);
    }
};
```

We see that `sampleKDE` stores internally the current reservoir, whose maximum size is limited to 100 elements, as its state. In the `invoke` method, `sampleKDE` uses the aggregation function `reservoirSample` to update the reservoir for each new element. As result of the method call, an instance of `EpanechnikovNKDE` is returned, which uses the reservoir elements to compute the bandwidth (per default by the normal scale rule) and to build the according KDE with respect to the Epanechnikov kernel.

Concerning KDEs, the subpackage `kernels` comprises basic functionality for their construction. This includes for instance the implementation of different kernel functions and bandwidth strategies.

4.6 Other Packages

Due to the large functionality of XXL, we confined this overview to the features we require to discuss the implementation of density estimators over streams. To get an idea of the other features of XXL, let us briefly list some of them.

The package `binarySearchTrees` provides common search tree implementations like AVL trees or red-black trees. The `collections` package consists of several data structures for dealing with collections, e.g. heaps, queues. The package `indexStructures` offers several index structures like R-trees, M-trees, B-trees. The basic functionality to access external resources is provided by the package `io`. The package `spatial` addresses specific join algorithms that support complex spatial predicates. Functionality to process and query XML data is given in the package `xml`.

Overall, we completed with this overview of XXL and PIPES the discussion of the preliminaries required for this work. Having these preliminaries in mind, we present in the next chapter the first density estimation technique for streams developed in the course of this work, namely compressed-cumulative WDEs.

Chapter 5

Compressed-cumulative WDEs: Wavelet Density Estimators over Data Streams

In this chapter, we present compressed-cumulative WDEs, our approach to wavelet density estimation over data streams. As this approach is founded on a convenient framework for maintaining statistical estimators over data streams, we start in Section 5.1 with a description of the framework and its components. Afterward, we examine in Section 5.2 how this framework can be exploited to build wavelet density estimators over data streams.

5.1 A Framework for Statistical Estimators over Data Streams

We start with the discussion of a framework for maintaining complex statistical estimators over data streams. The major advantage of this framework is that the estimators it delivers inherently meet the rigid processing requirements for streams.[1] Due to its flexible and modular design, a variety of estimation techniques can be plugged into the framework. As we will see in Section 5.2, wavelet density estimators are among those techniques. From a practical point of view, the application of the framework turns out to be comparably simple as most of its functionality is already available. In order to adapt a concrete estimation technique to streams, one only has to develop some clearly specified processing steps within the framework.

The concepts and techniques underlying the framework were developed by Björn Blohsfeld. In [21], he thoroughly discussed the essence of the framework, accompanied by an extensive experimental study for several estimators. This work basically investigated the adaptation of KDEs to streams. Using [21] as starting point, we developed in [22] natural generalizations and extensions, resulting in a sophisticated framework for maintaining complex statistical estimators over streams. For more details about the concepts underlying the framework, we refer to [21].

Let us provide a grasp of the framework's general idea. As the previous chapters have clarified, neither WDEs nor KDEs can be directly applied to data streams. The same is

[1] See Section 3.1.2.

true for other complex statistical estimators. On account of this problem, we propose an approximate solution that relies on building separate estimators for subsets of the stream. More precisely, the stream is partitioned into blocks of data and each block is associated with a separate estimator. To obtain an overall estimator for the stream, the weighted linear combination of all block estimators is iteratively computed, followed by its suitable compression to fit into the available memory.

We confine the following presentation of the framework to its main components, their settings, and their interaction. With regard to its application, we also discuss how its components and their interaction are implemented in XXL.

5.1.1 Cumulative Estimators

The starting point is a statistical estimation technique that is to be adapted to data streams. For the sake of simplicity, let us assume that this estimation technique delivers real-valued functions such as KDEs or WDEs; we weaken this assumption later. We also assume the available amount of memory to be sufficiently large in the beginning.

The following definition formalizes the basic idea of the framework:

Definition 5.1. *Let the data stream consist of N elements, with N a multiple of the block size b, i.e. $\exists j \in \mathbb{N} : N = j \cdot b$. Let $\hat{f}_i(x), i = 1, ..., j$ be an estimator for the i-th block of the data stream. The cumulative estimator for the complete stream with respect to the weights ω_i, $1 \leq i \leq j$, is defined as*

$$\hat{g}_j(x) = \sum_{i=1}^{j} \omega_i \hat{f}_i(x), \text{ where } \sum_{i=1}^{j} \omega_i = 1 \text{ and } \omega_i \geq 0 \ \forall i = 1, ..., j. \tag{5.1}$$

Note that the index j denotes the overall number of blocks of the stream. We also use j as resolution index for wavelets. However, the context in which j is concretely used in this chapter will clearly determine its meaning.

Figure 5.1 gives a concrete example for a cumulative estimator over a finite stream consisting of 16 elements. Each block estimator relies on a block of 4 elements.

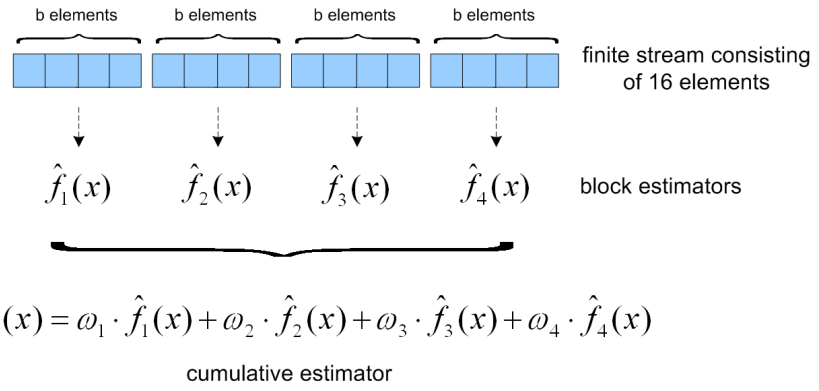


Figure 5.1: Cumulative estimator for a finite data stream

Definition 5.1 defines the cumulative estimator as the convex linear combination of block estimators. However, the cumulative estimator relies in its original form on the complete data

stream whereas we want an online computation. To circumvent this drawback, we present a one-pass algorithm that supports an iterative computation of the cumulative estimator while processing 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. It delivers as output the cumulative estimator for the first $i+1$ blocks, which is computed as convex linear combination of the two input estimators.[2] This iterative computation is founded on the following theorem.

Theorem 5.1. *Let $\hat{g}_j(x)$ be a cumulative estimator for a data stream consisting of j blocks of size b. A sequence of weights $\tilde{\omega}_i$ exists with $0 \leq \tilde{\omega}_i \leq 1,\ i = 1, ..., j$, $\tilde{\omega}_1 = 1$ so that*

$$\hat{g}_i(x) = \begin{cases} \hat{f}_1(x), & i = 1 \\ (1-\tilde{\omega}_i)\hat{g}_{i-1}(x) + \tilde{\omega}_i \hat{f}_i(x), & 2 \leq i \leq j \end{cases} \tag{5.2}$$

holds. For all $i = 1, ..., j$, $\hat{g}_i(x)$ is a cumulative estimator for the first i blocks of the stream.

Figure 5.2 illustrates the iterative computation of the cumulative estimator.[3]

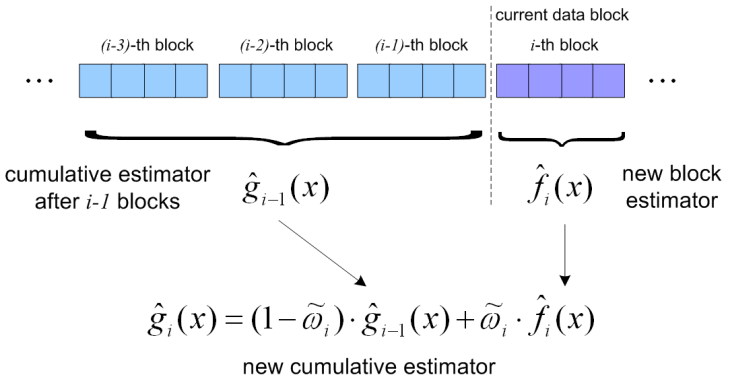


Figure 5.2: Online computation of the cumulative estimator

Proof. We consider $\hat{g}_j(x) = \sum_{i=1}^{j} \omega_i \hat{f}_i(x)$ with $j \in \mathbb{N}$ arbitrary, but fixed and $\omega_i \geq 0$, $\sum_{i=1}^{j} \omega_i = 1$. Without loss of generality, let $\omega_1 > 0$.

We recursively decompose g_j according to (5.2)

$$\begin{aligned} \hat{g}_j(x) &= (1-\tilde{\omega}_j)\hat{g}_{j-1}(x) + \tilde{\omega}_j \hat{f}_j(x) \\ &= (1-\tilde{\omega}_j) \cdot (1-\tilde{\omega}_{j-1})\hat{g}_{j-2}(x) + (1-\tilde{\omega}_j) \cdot \tilde{\omega}_{j-1}\hat{f}_{j-1}(x) + \tilde{\omega}_j \hat{f}_j(x) \\ &\vdots \\ &= \sum_{i=1}^{j-1} \prod_{l=i+1}^{j} (1-\tilde{\omega}_l)\tilde{\omega}_i \hat{f}_i(x) + \tilde{\omega}_j \hat{f}_j(x). \end{aligned}$$

[2] Recall the aggregation function concept described in Section 4.2.
[3] This figure bases on [21], p. 84.

As $\hat{g}_j(x)$ is a cumulative estimator, the next step is to determine weights $\tilde{\omega}_i$ so that $\omega_i = \prod_{l=i+1}^{j} (1-\tilde{\omega}_l)\tilde{\omega}_i$ for $i = 1, ..., j-1$ and $\tilde{\omega}_j = \omega_j$.

We define the weights $\tilde{\omega}_i$ as

$$\tilde{\omega}_i = \frac{\omega_i}{\sum_{k=1}^{i} \omega_k} \text{ for } i = 1, ..., j. \tag{5.3}$$

These weights fulfil the above requirements:

For $i = j$, we find that $\tilde{\omega}_j = \frac{\omega_j}{\sum_{k=1}^{j} \omega_k} = \omega_j$.

For $i = 1, ..., j-1$,

$$\begin{aligned}
\prod_{l=i+1}^{j} (1-\tilde{\omega}_l)\tilde{\omega}_i &= \prod_{l=i+1}^{j} \left(1 - \frac{\omega_l}{\sum_{k=1}^{l} \omega_k}\right) \cdot \frac{\omega_i}{\sum_{k=1}^{i} \omega_k} \\
&= \prod_{l=i+1}^{j} \frac{\sum_{k=1}^{l-1} \omega_k}{\sum_{k=1}^{l} \omega_k} \cdot \frac{\omega_i}{\sum_{k=1}^{i} \omega_k} \\
&= \frac{\sum_{k=1}^{j-1} \omega_k}{\sum_{k=1}^{j} \omega_k} \cdot \frac{\sum_{k=1}^{j-2} \omega_k}{\sum_{k=1}^{j-1} \omega_k} \cdot \ldots \cdot \frac{\sum_{k=1}^{i} \omega_k}{\sum_{k=1}^{i+1} \omega_k} \cdot \frac{\omega_i}{\sum_{k=1}^{i} \omega_k} \\
&= \frac{\omega_i}{\sum_{k=1}^{j} \omega_k} \\
&= \omega_i.
\end{aligned}$$

From the definition of the $\tilde{\omega}_j$, it follows $\tilde{\omega}_1 = 1$ and $0 \leq \tilde{\omega}_i \leq 1$ for $i = 2, ..., j$. □

This theorem is of utmost importance as it allows us to compute cumulative estimators in an online fashion. For a new block estimator, it is not necessary to recompute the weights of all previous block estimators; they are automatically updated.

5.1.2 Weighting Strategies

A necessary premise on a weighting strategy for a cumulative estimator is that its weights ω_i meet the requirements of equation (5.1). Provided they are fulfilled, we can exploit the formula in equation (5.3) to compute the weights $\tilde{\omega}_i$ for the online computation of the cumulative estimator. We can also define the weights for the online computation first and then derive the associated weights of the cumulative estimator for the complete stream as described in the proof of Theorem 5.1. It is worth mentioning that in case the block sizes are not equal, the weights must be suitably adjusted.

For practical purposes, the following two weighting strategies can be employed; both can cope with data streams of arbitrary size.[4]

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

$$\hat{g}_j(x) = \frac{1}{j} \sum_{i=1}^{j} \hat{f}_i(x). \tag{5.4}$$

For the corresponding set of weights for the online computation, it follows

$$\tilde{\omega}_i = \begin{cases} 1, & i = 1 \\ \frac{1}{i}, & 2 \leq i \leq j. \end{cases} \tag{5.5}$$

The cumulative estimator can be computed online via

$$\hat{g}_i(x) = \begin{cases} \hat{f}_1(x), & i = 1 \\ \frac{i-1}{i}\hat{g}_{i-1}(x) + \frac{1}{i}\hat{f}_i(x), & 2 \leq i \leq j. \end{cases} \tag{5.6}$$

Exponential Weighting The block estimators are weighted with $\alpha \in (0, 1)$ as follows

$$\hat{g}_j(x) = (1-\alpha)^{j-1}\hat{f}_1(x) + \sum_{i=2}^{j} \alpha(1-\alpha)^{j-i}\hat{f}_i(x). \tag{5.7}$$

The corresponding weights for the online computation have a surprisingly simple form

$$\tilde{\omega}_i = \begin{cases} 1, & i = 1 \\ \alpha, & 2 \leq i \leq j. \end{cases} \tag{5.8}$$

Thus, for the online computation of the cumulative estimator, it follows

$$\hat{g}_i(x) = \begin{cases} \hat{f}_1(x), & i = 1 \\ (1-\alpha)\hat{g}_{i-1}(x) + \alpha\hat{f}_i(x), & 2 \leq i \leq j. \end{cases} \tag{5.9}$$

Exponential weighting is based on exponential smoothing [67], a convenient weighting scheme from the area of time series analysis. The parameter α, the so-called smoothing parameter, adjusts the relation between the weight of the new block estimator and the weight of the current cumulative estimator. With α, we can control the impact of old and new data respectively. The higher α is set, the higher recent data is weighted. Contrary to, 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 common technique in data stream processing [16] where queries are often answered with respect to recent data since older data cannot be stored due to limited system resources. Contrary to common sliding windows, where older elements are abruptly discarded, we smoothly fade them out.

[4] Without loss of generality, let the stream size be a multiple of the block size b.

5.1.3 Compressed-cumulative Estimators

An aspect not yet investigated refers to the memory requirements of the cumulative estimator. Even though it is computable in an online manner, it still depends on all block estimators. As each block estimator will require a certain amount of memory, the memory allocated by the cumulative estimator increases linearly in the number of blocks. However, the processing requirements for streams only allow us to allocate a constant amount of memory.

To circumvent this shortcoming and to ensure that the cumulative estimator fits into the available memory, we slightly modify its recursive computation. After a new cumulative estimator was built, it is compressed. The parameter settings of the compression technique must guarantee that the compressed cumulative estimator fits into the available memory. Formally, this leads to the following definition.

Definition 5.2. *Let $\hat{g}_j$ be the cumulative estimator after the j-th block was processed. The compressed-cumulative estimator $\hat{g}_j^c$ is a compression of $\hat{g}_j$ that allocates a preset amount of memory.*

The setting of the compression technique depends on the estimation technique underlying the cumulative estimator. An inevitable drawback of the compression is that it introduces an additional compression error in each step, except for the rare case of a loss-free compression. For the case of arithmetic as well as exponential weighting, [21] gives upper bounds on the development of the compression error.

5.1.4 Application of the Framework

Let us now recapitulate the essential steps that are necessary to apply the presented framework. The starting point is a statistical estimation technique to be adapted to data streams. To plug this technique into the framework, one has to define the following steps:

- *Computation of a block estimator:* For each processed data block, a separate block estimator must be computed.
- *Setting of a weighting strategy:* Either a predefined weighting strategy is applied or a new weighting strategy must be developed.
- *Merge step:* One has to define how the current compressed-cumulative estimator and the new estimator are merged with respect to the given weights.
- *Compression step:* A suitable compression strategy must be developed which produces compressions that fit into a preset amount of memory.

In [21], Blohsfeld defined these steps for the adaptation of KDEs to streams. In accordance with our considerations in Section 3.4.6, the KDEs for the separate blocks can be computed in a straightforward manner. To compress the current estimator, cubic Beziér splines were used. As they are real-valued functions and the block KDEs as well, their merge is simply the convex linear combination of real-valued functions.

Overall, this framework inherently ensures that the processing requirements given in 3.1.2 are met. Each element is processed only once when the estimator for the block it belongs to is built. The complexity of this computation depends on the block size, which is constant.

The block estimator is merged with the current compressed-cumulative estimator. As the resulting estimator is compressed afterward, the element will not be required anymore, i.e., requirement 1 and 2 are met. The compression ensures that only a constant amount of memory is allocated, which corresponds to requirement 3. As each block estimator contributes to the compressed-cumulative estimator, each element has an impact on the estimator; requirement 6 is satisfied. At each point in time, the current compressed-cumulative estimator is available, which corresponds to requirement 4. To capture changes in the stream according to requirement 5, we can exploit the exponential weighting strategy to emphasize recent data. The flexible adaptation of the allocated memory to changing system resources as demanded by requirement 7 can be accomplished by adjusting the block size or the compression ratio.

Before we elaborate the application of the framework to adapt wavelet density estimators to streams, we briefly describe its implementation in XXL.

5.1.5 Implementation in XXL

The presented framework is fully implemented in XXL. For a demand-driven processing, Blohsfeld implemented its basic functionality by means of specific cursors and functions. For a detailed description of this implementation, we refer to [21]. In the course of this work, we implemented the framework functionality for a data-driven processing. In order to achieve the development objectives of XXL[5], we strived for a modular implementation and made extensive use of functional modeling. As described above, the integration of a new estimation technique requires implementing specific processing steps of the framework. In the implementation, we encapsulate these steps into specific functions. The application developer only has to implement these functions since their interaction is already available. Generally, the use of the framework is facilitated by preimplementations and illustrative use-cases for estimation techniques already plugged into the framework.

The implementation of the framework basically relies on two operators. The first operator, termed `BlockBasedBufferPipe`, receives the elements of the data stream, buffers them, and, in case a new data block is available, transfers this block to the second operator. The second operator, termed `BlockBasedMergeAggregator`, builds a new block estimator based on the block elements, merges it with the current compressed-cumulative estimator, and compresses the new estimator afterward. To gain advantage of the processing facilities of PIPES, both operators are implemented as extensions of `AbstractPipe`.[6] Since the framework must ensure resource-awareness, both operators implement the interface `MemoryManageable`. An operator implementing this interface can report its current amount of allocated memory and, most important, can be advised to reduce or increase this amount.

BlockBasedBufferPipe `AbstractBufferPipe` is an abstract class that models a buffer operator, i.e., it receives incoming elements and inserts them into an internal queue. Generally, `AbstractBufferPipe` runs in an own thread, which periodically calls the abstract `execute` method. `BlockBasedBufferPipe` is an extension of `AbstractBufferPipe`. In its implementation of the `execute` method, a preset number of elements is taken from the internal queue and transferred as a block to all subscribed sinks. `BufferPipe` as another extension of `AbstractBufferPipe` transfers each element separately in the `execute` method.

[5] See also Section 4.1.
[6] See also Section 4.4.

The allocated memory of a `BlockBasedBufferPipe` is determined by the current number of elements in its internal queue. To reduce this number, one can decrease the block size. Then a new data block will be filled faster and, as a consequence, also transferred earlier. On the contrary, we can also increase the block size in case the available amount of memory has been increased. Throughout this work, we use the setting of the block size as strategy to adjust the storage space of the buffer.

In situations of high system load, the elements can arrive much faster than the compressed-cumulative estimator can be computed; the allocated memory will rapidly increase even though the block size may be small. To cope with those situations, we must drop elements from the internal queue. For the choice of the elements to drop, we can incorporate one of the load shedding techniques proposed in the literature, e.g. [132], [134].

BlockBasedMergeAggregator The `BlockBasedMergeAggregator` is the central operator of the framework; it unifies the functionality for the online computation of compressed-cumulative estimators. The main components of a `BlockBasedMergeAggregator` are the `BlockBasedBufferPipe` it is subscribed to and a function termed `MergeAggregateFunction`, which encapsulates the essential processing steps of the framework. During runtime, the `BlockBasedMergeAggregator` receives a block of elements and invokes with this block the `MergeAggregateFunction`. This function returns the new compressed-cumulative estimator, which the `BlockBasedMergeAggregator` transfers afterward to its sinks. Due to the important role the `MergeAggregateFunction` plays in this concept, let us take a closer look at it.

`MergeAggregateFunction` is an extension of `Function`. It consists of three essential functions: an aggregation function, a weighting function, and a merge and compress function. Those functions are consecutively called, with each one preparing the ground for the application of its successor.

The aggregation function must be an extension of `AggregationFunction`, i.e., it is a binary function. Its `invoke` method takes the current compressed-cumulative estimator and the new data block as input and delivers the corresponding block estimator as result. It is worth mentioning that in most cases the computation of the block estimator does not require the knowledge of the current compressed-cumulative estimator. However, we included it to render the integration of arbitrary techniques possible.

The associated weight of the new block estimator is computed by the weighting function. More precisely, the weighting function must implement the `RealFunction` interface; it delivers for i, the current number of processed blocks, the corresponding weight $\tilde{\omega}_i$.

The function that implements the merge step and the subsequent compression step must extend `Function`; its functionality is implemented in the `invoke` method. This method is called with the following parameters: the new block estimator, the current compressed-cumulative estimator, the current weight $\tilde{\omega}_i$, and the amount of available memory. Those parameters are processed as follows. As described in equation (5.2), the new block estimator and the current compressed-cumulative estimator are merged with respect to the weights $1-\tilde{\omega}_i$ and $\tilde{\omega}_i$. The resulting estimator is compressed so that it fits into the available memory. Eventually, the compressed estimator is returned as output of the `invoke` method.

Figure 5.3 gives an overview of the interplay between `BlockBasedMergeAggregator`, `BlockBasedBufferPipe`, and `MergeAggregateFunction`.

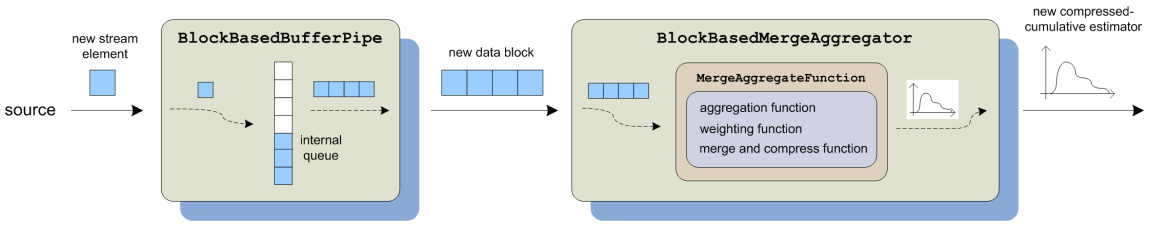


Figure 5.3: Main components of the framework implementation in XXL

Memory Management After this summary on the parameters and the processing logic of `BlockBasedBufferPipe`, `BlockBasedMergeAggregator`, and `MergeAggregateFunction`, let us briefly describe their memory management. All of them implement the `MemoryManageable` interface, i.e., they can dynamically adjust the amount of memory they allocate.

As described in Section 4.4, the memory manager of PIPES is responsible for distributing the available memory among the stateful operators in the current query graph. In our setting, only the `BlockBasedMergeAggregator` is subscribed to the memory manager. It distributes its assigned memory among the `BlockBasedBufferPipe` and the `MergeAggregateFunction`, i.e., between buffer and current compressed-cumulative estimator. To produce meaningful estimators, it is crucial to find a well-balanced distribution of the available memory. This results from the fact that the block size for the buffer and the compression ratio for the compressed-cumulative estimator both affect the estimation quality. To be consistent, a premise of all statistical estimation techniques is that the underlying sample size increases. Hence, a prerequisite for a good block estimator is a large block size. But we also want to keep the compression error low, which necessitates reserving enough memory for the compression. Thus, we observe an inherent trade-off. Producing better block estimator comes at the expense of reducing the compression quality and vice versa.

Generally, the strategy for setting these two contradictory parameters depends on the examined estimation technique.

5.2 Compressed-cumulative WDEs over Data Streams

In this section, we investigate the adaptation of wavelet density estimators to data streams with the help of the framework described in the last section. We presented the basic idea of this approach in [83] and discussed it in more detail in [89]. According to Section 5.1.4, we must develop the following components to apply the framework: computation of a block WDE, merge step, compression step, and weighting strategy. As Section 5.1.2 already presented concrete weighting strategies, we will not address this topic.

5.2.1 Computation of a Block WDE

In Section 3.5.2.4, we discussed practical aspects of computing a WDE. For that reason, we only sketch the main steps. Except where otherwise stated, we consider univariate streams.

The first step is to choose the underlying wavelet family. Due to their convenient properties, Daubechies wavelets are a suitable choice. The second step is to decide which type of WDE shall be computed. We focus on linear and thresholded WDEs, which were discussed

in Section 3.5.2.1 and 3.5.2.2 respectively. Other types of WDEs can be adapted in the same manner. Provided that the wavelet family and the type of WDE are chosen, the next step is to sort the block elements $X_1, ..., X_b$, which facilitates the computation of the empirical coefficients. Depending on whether linear or thresholded WDEs are used, we compute their resolutions j_1^{lin} and j_0^{thr}, j_1^{thr} respectively. For the corresponding initial resolution j_1^{lin} or j_1^{thr}, we compute the empirical scaling coefficients as described in equation (3.40). By means of the DWT, they are decomposed into scaling and wavelet coefficients of lower resolutions. For the case of thresholded WDEs, these resolutions are $j_0^{thr}, ..., j_1^{thr} - 1$. For thresholded WDEs, we additionally apply a level-dependent thresholding procedure to the empirical wavelet coefficients. Eventually, we have a set of non-zero empirical scaling and wavelet coefficients that define the block WDE.

5.2.2 Merge Step

A vital feature of compressed-cumulative estimators is that they can be iteratively computed by merging the current compressed-cumulative estimator and the new block estimator (see equation (5.2)). To define this merge step for the case of WDEs, we exploit the wavelet series expansion of a WDE.

Without loss of generality, we consider the convex merge of the first two block WDEs, i.e. $\hat{g}_2(x) = (1-\tilde{\omega}_2)\hat{f}_1(x) + \tilde{\omega}_2\hat{f}_2(x)$. As we will see, a necessary prerequisite for the merge is that both WDEs have the same scaling resolution. Hence, we use the DWT to transform the scaling coefficients of both WDEs down to the minimum j_0 of their current scaling resolutions. Let $\{\hat{c}_{j_0,k}^{(1)} : k \in \mathbb{Z}\}$ and $\{\hat{c}_{j_0,k}^{(2)} : k \in \mathbb{Z}\}$ be the empirical scaling coefficients of the wavelet series expansions of $\hat{f}_1$ and $\hat{f}_2$ respectively. Let those sets also comprise the coefficients with value zero as it facilitates the introduction of the merge step. In the same manner, we denote the empirical wavelet coefficients of $\hat{f}_1$ and $\hat{f}_2$ as $\{\hat{d}_{j,k}^{(1)} : j,k \in \mathbb{Z}, j \geq j_0\}$ and $\{\hat{d}_{j,k}^{(2)} : j,k \in \mathbb{Z}, j \geq j_0\}$. By means of the wavelet series expansion of $\hat{f}_1$ and $\hat{f}_2$, we can decompose $\hat{g}_2$ as follows

$$\begin{aligned}
\hat{g}_2(x) &= (1-\tilde{\omega}_2)\hat{f}_1(x) + \tilde{\omega}_2\hat{f}_2(x) \\
&= (1-\tilde{\omega}_2)\sum_{k\in\mathbb{Z}} \hat{c}_{j_0,k}^{(1)}\phi_{j_0,k}(x) + (1-\tilde{\omega}_2)\sum_{j\geq j_0}\sum_{k\in\mathbb{Z}} \hat{d}_{j,k}^{(1)}\psi_{j,k}(x) \\
&\quad +\tilde{\omega}_2\sum_{k\in\mathbb{Z}} \hat{c}_{j_0,k}^{(2)}\phi_{j_0,k}(x) + \tilde{\omega}_2\sum_{j\geq j_0}\sum_{k\in\mathbb{Z}} \hat{d}_{j,k}^{(2)}\psi_{j,k}(x) \\
&= \sum_{k\in\mathbb{Z}} \hat{c}_{j_0,k}^{(3)}\phi_{j_0,k}(x) + \sum_{j\geq j_0}\sum_{k\in\mathbb{Z}} \hat{d}_{j,k}^{(3)}\psi_{j,k}(x) \qquad (5.10)
\end{aligned}$$

with $\hat{c}_{j_0,k}^{(3)} = (1-\tilde{\omega}_2)\hat{c}_{j_0,k}^{(1)} + \tilde{\omega}_2\hat{c}_{j_0,k}^{(2)}$ and $\hat{d}_{j,k}^{(3)} = (1-\tilde{\omega}_2)\hat{d}_{j,k}^{(1)} + \tilde{\omega}_2\hat{d}_{j,k}^{(2)}$. Hence, the convex merge step for two block WDEs is simply the convex merge of their empirical coefficients. This is a remarkable result as we can merge two functions without even evaluating them. An important aspect in this context is that the result of the merge, the new compressed-cumulative WDE, is also given in its wavelet series expansion. Thereby, we can merge the current compressed-cumulative WDE and a new block WDE in the same manner, i.e., this procedure is a valid realization of the merge step for WDEs. In Section 5.2.6, we will present an efficient implementation of this merge step, which only considers the non-zero coefficients.

Having this merge procedure in mind, the prerequisite of the same scaling resolution for both WDEs becomes clear. In case they would differ, the resulting estimator would include scaling coefficients of different resolutions, which no more complies with the definition of the wavelet series expansion in equation (3.41).

A valuable side effect of the merge procedure is that we expect the resulting overall number of empirical coefficients after the merge to be smaller than the sum of the contributing coefficients. As all block WDEs follow the same distribution, the occurrence of similar features is likely. As these features are described by the same empirical coefficients, we expect a high number of "merge partners". The merge of these coefficients in turn reduces the overall number of coefficients.

Figure 5.4 depicts an example for the convex merge of two functions given in their wavelet representations. Both are represented with respect to Haar wavelets. The weight ω for the merge is set to 0.25.

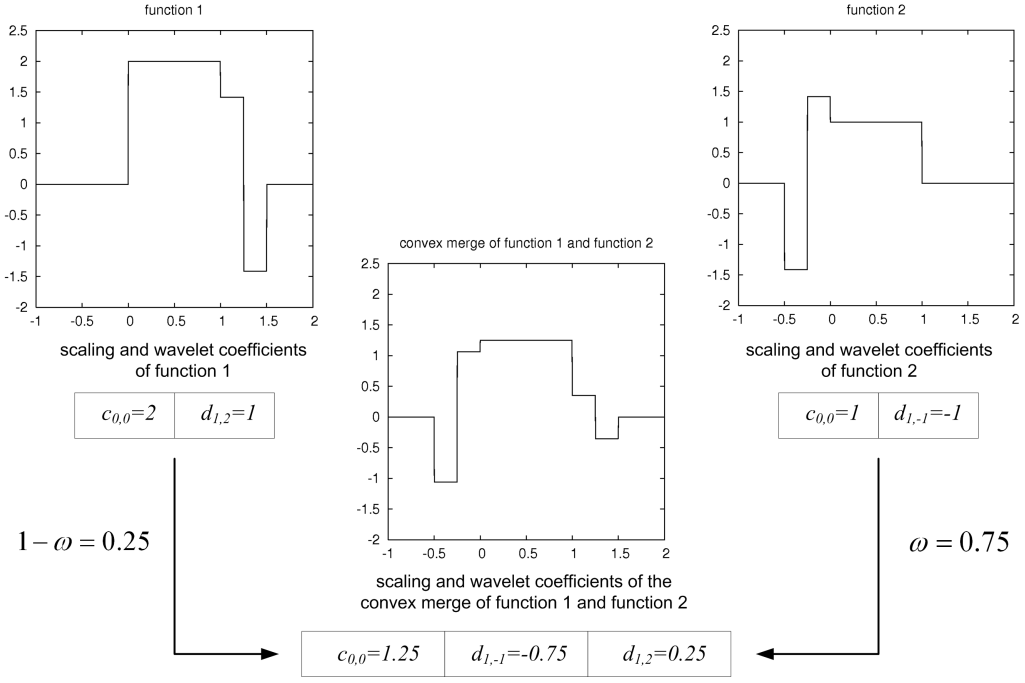


Figure 5.4: Convex merge of two functions given in their wavelet series expansion

5.2.3 Compression Step

The compression step is subsequent to the merge step. In case the new compressed-cumulative WDE does not fit into the available memory, we must perform a suitable compression. As the memory allocation of a WDE is basically determined by its empirical coefficients, we compute the maximum number of coefficients M with respect to the available amount of memory. Thus, the task we have to tackle is how to trim a WDE's coefficient number down to M in a suitable manner.

In Section 3.5.1.3, we already introduced a convenient strategy for compressing a function given in its wavelet series expansion. This strategy discards the least significant local details.

In the wavelet terminology, these details correspond to the wavelet coefficients with lowest absolute value. Therefore, we must sort the entirety of empirical wavelet coefficients by their absolute values and discard the required number of coefficients. Let us emphasize that this method should not be applied to scaling coefficients. If we would deal analogously with them, we would delete global features of the function and obscure its shape.

It may occur that all wavelet coefficients were discarded and the remaining scaling coefficients still allocate too much memory, i.e., their number exceeds M. In that case, the DWT gives us a simple solution. We decompose the scaling coefficients into scaling and wavelet coefficients of the next-lower resolution. Important in this context is that this gives us roughly half as many scaling coefficients.[7] Again, we discard the resulting wavelet coefficients with lowest absolute value. The repeated application of this procedure ensures that we eventually get a set of empirical coefficients, whose number is less or equal to M.

However, discarding wavelet coefficients always introduces a compression error because we loose local details. We introduce a measure for the relative error resulting from a compression step. Let $\hat{g}$ be the current compressed-cumulative WDE before and $\hat{g}^c$ the one after the compression. Let $I \subset \mathbb{Z} \times \mathbb{Z}$ be the set of indexes (j,k) of discarded empirical wavelet coefficients $\hat{d}_{j,k}$. For the L_2-norm of the compression error $e(x) = |\hat{g}(x) - \hat{g}^c(x)|$, it follows

$$\begin{aligned} ||e||_2^2 &= || \sum_{(j,k)\in I} \hat{d}_{j,k}\psi_{j,k}(x)||_2^2 \\ &= \sum_{l,m\in\mathbb{Z}} |\langle \sum_{(j,k)\in I} \hat{d}_{j,k}\psi_{j,k}(x), \psi_{l,m}\rangle|^2 \\ &= \sum_{(j,k)\in I} \hat{d}_{j,k}^2. \end{aligned} \tag{5.11}$$

The first equation relies on Parseval's identity [82] and the second one on the orthonormality of wavelets. With respect to this identity, we define the *relative compression error* as

$$err = \frac{||e||_2^2}{||\hat{g}||_2^2} = \frac{\sum_{(j,k)\in I} \hat{d}_{j,k}^2}{\sum_{j,k\in\mathbb{Z}} \hat{d}_{j,k}^2}. \tag{5.12}$$

The sum of squared wavelet coefficients is often defined as *energy*.[8] Hence, the relative compression error quantifies the percentage of lost energy. Note that the relative compression error corresponds to the value of the Lorentz curve[9] at the point p, with p the percentage of discarded wavelet coefficients.

In order to convey an impression of the effects induced by discarding wavelet coefficients according to the above strategy, let us examine the example given in Figure 5.5. This figure displays in its center a WDE based on soft thresholding over a data set termed Earthquake[10]. We compressed this WDE with different compression ratios to get a feeling for the loss in

[7] See Section 3.5.1.3.

[8] See [138], p. 10.

[9] See also [138], p. 10.

[10] For more details about this data set, which is part of our experimental evaluation, we refer to Section 7.1.2.

accuracy. We observe that neither compression ratios of 25% or 50% nor ratios of 75% and 85% change the shape of the WDE significantly. Only small local details vanish after the compression. As expected, the higher the compression ratio, the more local details vanish. But the general shape of the WDE does not change significantly. Those facts also reflect in the energy and the relative compression errors additionally provided within this figure.

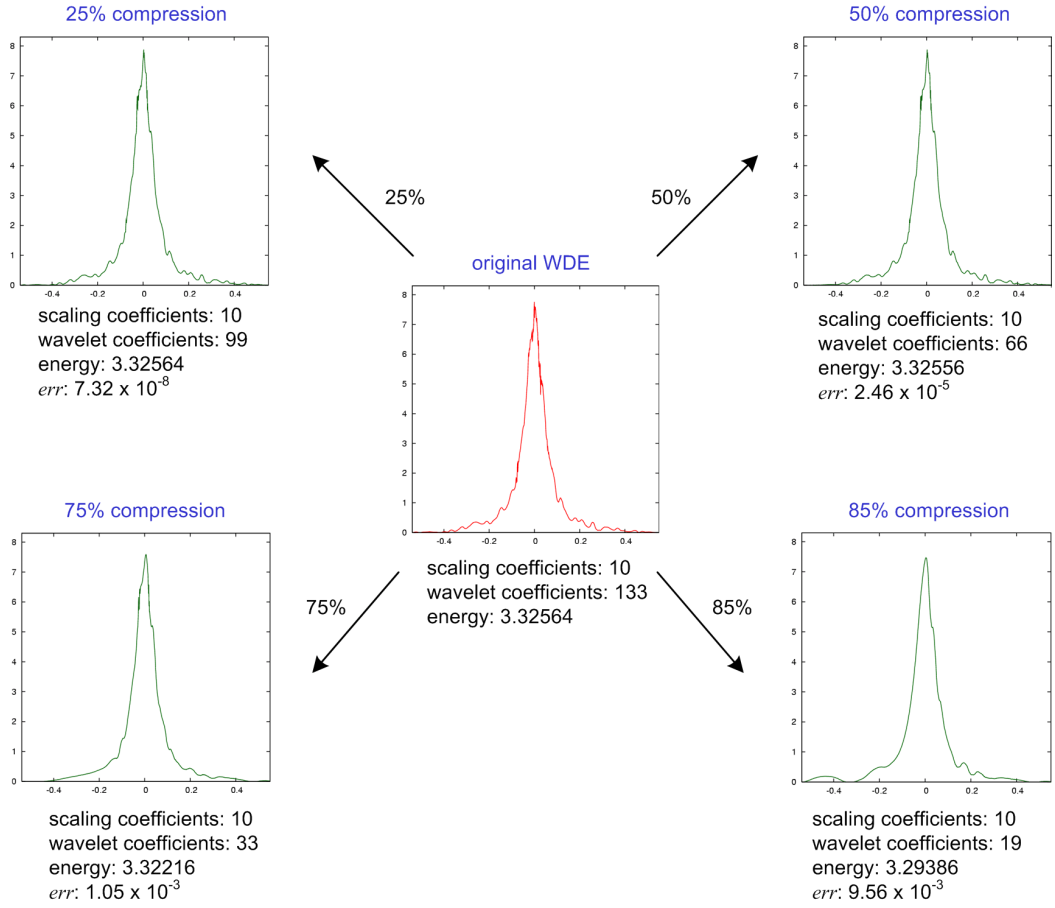


Figure 5.5: Effects of different compression ratios on a WDE

Overall, with the compression step, we developed all processing steps that are required for using the framework. We discussed how to compute block WDEs, how to merge them in an online fashion, and how to compress them appropriately. Algorithm 1 summarizes the general steps for computing compressed-cumulative WDEs over data streams. Before we turn our attention to an efficient implementation of this algorithm and an analysis of its computational complexity, we address an important aspect concerning the applicability of our technique, namely the memory management for compressed-cumulative WDEs.

5.2.4 Memory Management

In order to comply with the processing requirements for streams, compressed-cumulative WDEs are only allowed to allocate a constant amount of memory, which even may change

Algorithm 1 Compressed-cumulative WDEs over a Data Stream

for each incoming element **do**
  insert element into buffer;
  if buffer size $\geq b$ **then**
    remove first b elements from the buffer;
    increment block counter i;
    compute WDE $\hat{f}_i$ for these elements;
    if $i = 1$ **then**
      $\hat{g}_i = \hat{f}_i$;
    else
      compute weight $\tilde{\omega}_i$;
      decompose empirical scaling coefficients of $\hat{g}_i$ and $\hat{f}_i$ down to the minimum of their scaling resolutions;
      $\hat{g}_i$ = convex merge of $\hat{g}_{i-1}$ and $\hat{f}_i$ with weight $\tilde{\omega}_i$;
    end if
    set M_s as number of empirical scaling coefficients of $\hat{g}_i$;
    set M_w as number of empirical wavelet coefficients of $\hat{g}_i$;
    while $M_s + M_w > M$ **do**
      if $M_s > M$ **then**
        discard all empirical wavelet coefficients;
        decompose empirical scaling coefficients down to next lower resolution;
        update M_s and M_w;
      else
        $p = 1 - \frac{M - M_s}{M_w}$;
        sort empirical wavelet coefficients by absolute value in descending order;
        discard the last $p \cdot M_w$ empirical wavelet coefficients;
      end if
    end while
    transfer $\hat{g}_i$;
  end if
end for

during runtime. For that reason, we must efficiently manage the available memory and also provide mechanisms to adapt to abrupt changes. As already addressed in Section 5.1.5, the available memory is distributed among the buffer and the compressed-cumulative estimator. Thus, to adapt the amount of allocated memory, we can adjust the block size or the number of coefficients of the compressed-cumulative WDE.

Memory Distribution Let us first consider the distribution of the memory among buffer and compressed-cumulative WDE. The more memory the buffer has at its disposal, the larger the block size can be set. A larger block size in turn results in better block WDEs as the quality of a WDE is also determined by the size of its underlying sample. But larger block sizes lead to a higher number of empirical coefficients, i.e., the block WDEs and consequently the compressed-cumulative WDEs will allocate more memory.

A convenient approach for the memory distribution is to estimate the coefficient number of block WDEs based on different block sizes. For this estimation, we have to take a closer look at the computation of a WDE. According to Section 3.5.2.4, the number of empirical scaling coefficients before the DWT is applied is $k_{max} - k_{min} + 1$. After the DWT, the overall number of empirical wavelet and scaling coefficients will be roughly the same. Hence, we can use $k_{max} - k_{min} + 1$ as an upper bound for the coefficient number. The coefficient number of thresholded WDEs may be significantly lower due to the additional thresholding procedure. To compute the upper bound, we have to estimate k_{min} and k_{max}. According to equation (3.52), they depend on the initial resolution, the extrema of the underlying sample, and the support of the scaling function. While the support of the scaling function is known, the extrema can be estimated by the extrema of previous blocks. The initial resolution depends on the type of WDE. The initial resolution for thresholded WDEs is defined in equation (3.50); it is solely determined by the sample size. The initial resolution of linear WDEs is defined in equation (3.44); it additionally depends on the standard deviation, which can be estimated with the root of the sample variance of previous blocks.

Given the estimated coefficient numbers for different block sizes, we can choose the largest block size so that the corresponding coefficient number would fit into half of the remaining memory. The remaining memory refers to the available memory minus a certain amount reserved for the buffer. The other half of the remaining memory is reserved for the current compressed-cumulative WDE.

Additionally, we can examine the energy distribution of previous block WDEs. In case most energy is concentrated in a few coefficients, it may be advisable to use larger block sizes, even though the compressed-cumulative WDEs must be compressed due to block WDEs with many coefficients. However, the compression error will be small, but the larger block size will induce a better estimation quality of the block WDEs.

Memory Adaptation Let us now consider the adaptation to changing system resources. We have to distinguish between two cases: an increase or decrease of the available amount of memory.

In case the available memory for the compressed-cumulative WDE is increased, we simply determine a new maximum number of empirical coefficients. In case it is decreased, we can use the same mechanism as in the compression step to reduce the coefficient number. We successively discard the empirical wavelet coefficients with the lowest absolute value. In

this context, the relative compression error can be used to quantify the loss in accuracy for different compression ratios before the actual compression is carried out. This information can support the decision making for the setting of the new amount of available memory.

The memory adaptation for the buffer is simple. We either increase or decrease the block size, resulting in larger or smaller blocks of elements to transfer.

5.2.5 Evaluation, Integration, and Summary Measures

While the previous sections described the construction of compressed-cumulative WDEs, this section examines their concrete use. This refers to their evaluation, their integration, as well as to the computation of summary measures. As a compressed-cumulative WDE is given in its wavelet series expansion, the following considerations are also valid for WDEs in general.

Throughout this work, we represent WDEs with respect to Daubechies wavelets due to the convenient mathematical properties of this wavelet family. However, a severe drawback is that they do not provide a closed formula for their evaluation, except for Haar wavelets. The same problem occurs if we want to evaluate their antiderivatives. Therefore, we must compute a suitable approximation, a problem we address later in this chapter. Let us assume that we have such an approximation at our disposal.

Section 3.5.2.4 already discussed the evaluation of a WDE at a given point x. For each resolution j, we can confine the evaluation of the versions $\phi_{j,k}$ and $\psi_{j,k}$ to those whose support covers x. This corresponds to computing an index range for k as described in equation (3.52). In a similar manner, we can evaluate the antiderivative of a WDE.

For analysis purposes, summary measures like mean and variance are often exploited. The computation of mean and variance requires the integration of $x\phi_{j,k}(x)$ and $x^2\phi_{j,k}(x)$. Again, the lack of a closed formula for the evaluation of Daubechies wavelets renders an exact computation impossible. Instead of, we must use numerical approximation techniques to compute an approximate solution for these integrals. Given an approximation for the variance, we can approximate the standard deviation by extracting the square root.

Besides mean, variance, and standard deviation, we can also approximate other summary measures with the help of compressed-cumulative WDEs. Let us briefly describe some of them.

To determine the value range of all processed elements, we can approximate minimum and maximum of all processed stream elements. We approximate them by examining the left-most and the right-most empirical scaling and wavelet coefficient of each resolution. The comparison of minimum and maximum of the corresponding scaling and wavelet versions delivers the overall minimum and maximum.

The entropy of a continuous random variable is an information measure defined as $H(X) = \int_{-\infty}^{+\infty} -x \log f(x) dx$. A natural approach to approximate the entropy is to use the compressed-cumulative WDE instead of f and to approximate the corresponding integral numerically.

Compressed-cumulative WDEs can also be exploited to approximate the percentage of already processed stream elements that fell into an arbitrary interval $[a, b]$ with $a, b \in \mathbb{R}$. To serve this purpose, we only have to compute the integral over the interval $[a, b]$.

5.2.6 Implementation and Algorithm Analysis

For the sake of applicability, we discuss how compressed-cumulative WDEs can be efficiently implemented. This implementation must provide the essential processing steps of Algorithm 1 as well as an efficient evaluation of the resulting estimator.

As the previous considerations have clarified, a WDE is defined by its empirical coefficients and the underlying wavelet family. For the wavelet family, i.e., scaling functions and wavelets, we only need their basic form as we can derive the dilated and translated versions from them. Thus, the chief question is how to organize the empirical coefficients within a suitable data structure.

An empirical coefficient consists of three parameters: resolution j, translation index k, and the value of the coefficient. As we will see, empirical coefficients are typically accessed by their resolution and their translation index. To facilitate the management of the coefficients, we partition them into sets of coefficients with the same resolution. We use a combined data structure for this partitioning. This data structure consists of a sorted list. Each list entry is composed of a resolution index and a height-balanced binary search tree. While the list entries are ordered by the resolution index, the search trees are ordered by the translation index. The search tree of a list node stores all empirical coefficients with the list entry's resolution. This data structure allows us to access and modify coefficients in logarithmic time. Figure 5.6 illustrates its basic idea. For the sake of simplicity, we term this data structure *coefficient set*. Generally, we organize the empirical scaling coefficients and the empirical wavelet coefficients of a WDE in two separate coefficient sets.

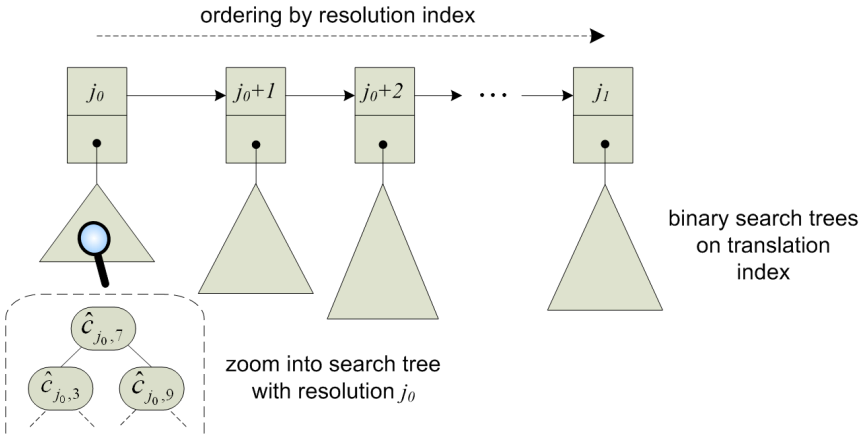


Figure 5.6: Data structure for storing a set of empirical coefficients

With the coefficient sets at hand, let us now examine how the processing steps of Algorithm 1 as well as the evaluation of a compressed-cumulative WDE can be implemented. We will complement this examination with an analysis of the computational complexity.

DWT The application of the DWT is a main component of Algorithm 1. Equation (3.36) and (3.37) describe how the DWT is applied to a set of coefficients with respect to a given filter. As the coefficients are only accessed by their resolution and translation index in these equations, the DWT can be efficiently implemented with coefficient sets.

According to the two-scale equation (3.32), the filter coefficients are specific scaling coefficients. For that reason, we also organize them in a coefficient set.

The DWT can be implemented in linear time[11], provided that each coefficient can be accessed in constant time. As a coefficient in a coefficient set is accessed in logarithmic time, our implementation has complexity $O(n \log n)$ with n the number of top-level scaling coefficients.

Computation of Block WDE To compute a block WDE, we must implement the following steps. First, we compute the resolution j_1 and the index range $k_{min}, ..., k_{max}$ of the empirical scaling coefficients of resolution j_1. Given this range, the second step is to compute the according empirical coefficients and to insert them into a new scaling coefficient set. To accelerate the computation, we sort the elements of the block beforehand. As described in Section 3.5.2.4, this allows us to evaluate the scaling function only for the block elements that actually contribute to an empirical scaling coefficient. For the case of thresholded WDEs, the next step is to decompose the scaling coefficients down to j_0^{thr} and to apply the thresholding procedure to the empirical wavelet coefficients afterward. Note that the wavelet coefficients are stored in their own wavelet coefficient set. In the implementation of the thresholding procedure, we iterate over all nodes of the list of this set. For each node, we iterate over the coefficients of its tree and apply the thresholding procedure with respect to the corresponding level threshold. Eventually, we obtain a WDE described by its scaling coefficient set and its wavelet coefficient set.

The complexity of computing a block WDE is determined by the computation of the empirical scaling coefficients and the subsequent DWT. According to equations (3.40) and (3.52), the computation of the empirical scaling coefficients has complexity $O((k_{max} - k_{min} + 1)b)$ with b the block size. Combining equation (3.52) with the initial resolutions of linear and thresholded WDEs (see equation (3.44) and (3.50)), we see that k_{min} and k_{max} both depend linear on the block size. Thereby, the computation of all empirical scaling coefficients has complexity $O(b^2)$ and their insertion into the scaling coefficient set $O(b \log b)$. As the DWT has complexity $O(b \log b)$ and the thresholding procedure for the case of thresholded WDEs $O(b)$, the computation of a block WDE has an overall complexity of $O(b^2)$.

Merge Step Equation (5.10) describes the convex merge of two functions in their wavelet series expansion. To implement the convex merge of two coefficient sets set_1, set_2 for a given weight $\tilde{\omega}$, we successively insert the elements of set_2 into set_1. First of all, we iterate over all elements of set_1 and scale them with $1 - \tilde{\omega}$. Then we iterate over all elements of set_2. Each element is scaled with $\tilde{\omega}$ and then probed against set_1. In case a "join partner" exists, the element of set_1 is substituted by the sum of both elements. If not, the element of set_2 is inserted into set_1. Due to the ordering by j, k, those steps can be efficiently implemented with coefficients sets.

The computational complexity of the merge step is as follows. The current compressed-cumulative WDE stores $O(M)$ empirical coefficients in its coefficient sets; M is their maximum number. The block WDE stores $O(b)$ empirical coefficients. Scaling all coefficients of the compressed-cumulative WDE has complexity $O(M)$. The subsequent insertion of the

[11] See [122], Section 6.2.

coefficients of the block WDE into the coefficient sets of the compressed-cumulative WDE has complexity $O(b \log M)$. We conclude that the merge step has complexity $O(M + b \log M)$.

Compression Step The compression step is the only step coefficient sets cannot efficiently support. This results from the fact that the compression step requires an ordering of all empirical wavelet coefficients by their absolute values.

To implement the compression step, we iterate over all elements of the wavelet coefficient set and insert them into a heap which uses their descending absolute values as ordering criterion. Then we dequeue as many elements as the available memory allows and re-insert them into the wavelet coefficient set, which was reseted before.

The complexity of the compression step depends on the number of coefficients of the compressed-cumulative WDE. After the merge step, this number is $O(M + b)$. The insertion of all empirical wavelet coefficients into the heap has complexity $O((M + b) \log(M + b))$. Dequeuing M coefficients from the heap and re-inserting them into the wavelet coefficient set is done in $O(M \log M)$. Overall, the compression step has complexity $O((M + b) \log(M + b))$.

Algorithm Analysis Let us summarize the computational complexity for processing a data block and updating the compressed-cumulative estimator. The computation of a block WDE has complexity $O(b^2)$, the subsequent merge step $O(M + b \log M)$, and the compression step $O((M + b) \log(M + b))$. This results in an overall complexity of $O(b^2 + (M + b) \log(M + b))$ per data block, i.e., $O(b + \frac{1}{b}(M + b) \log(M + b))$ per element.

Evaluation of WDE In the implementation of the evaluation, we iterate over the nodes of the wavelet coefficient set and determine for each one the index range of those empirical wavelet coefficients, whose wavelet version must be evaluated. We obtain these coefficients by posing a range query with this range over the associated tree. We proceed in the same manner with the scaling coefficient set. Note that the list of the scaling coefficient set consists of a single node since all empirical scaling coefficients have the same resolution.

Generally, the computational complexity of the evaluation of a WDE is $O(M)$. In case only a constant number of scaling function or wavelet versions contributes to the evaluation at each resolution, our implementation reduces the complexity to $O(\log M)$ for each resolution.

5.2.7 Implementation in XXL

To facilitate the application of compressed-cumulative WDEs, let us discuss their concrete implementation in XXL. Section 5.1.5 already presented the implementation of the underlying framework in XXL. Its functionality relies on the `BlockBasedBufferPipe`, which buffers and transfers blocks of elements, and the `BlockBasedMergeAggregator`, which successively computes and merges block estimators, accompanied by a subsequent compression. A `BlockBasedMergeAggregator` is associated with a `MergeAggregateFunction`, which encapsulates the basic processing steps in its functions. To implement compressed-cumulative WDEs, we solely have to implement these functions properly.

Implementation of WDEs A prerequisite for the implementation of these functions is the implementation of WDEs. According to equation (3.41), we consider WDEs in their

wavelet series expansion, which comprises the scaling and wavelet coefficients as well as the corresponding scaling and wavelet functions. Therefore, we first consider how to model a wavelet series expansion. For the organization of the coefficients, we can use coefficient sets.

`ListTreeSet` is our implementation of the coefficient set. For the implementation of the data structures that constitute a coefficient set, we exploit the Java SDK from Sun [3]. Specifically, we use `java.util.ArrayList`, an array-based implementation of a sorted list, and `java.util.TreeMap`, an implementation of red-black trees. Based on the combination of these data structures, `ListTreeSet` provides a plethora of methods to access and modify coefficients. Among the most important are the following ones:

- `add(int j, int k, double c)`: inserts a coefficient with resolution j, translation index k, and value c
- `get(int j, int k)`: returns 0 if the coefficient with resolution j and translation index k is not in the coefficient set and the value of the coefficient otherwise
- `rangeQueryCoeffs(int j, int kmin, int kmax)`: returns an iterator over all coefficients of resolution j, whose translation index is between k_{min} and k_{max}
- `mergeSet(ListTreeSet set, double weight1, double weight2)`: merges the two sets with respect to the given weights
- `truncateSet(double p)`: removes the p percent of the coefficients with lowest absolute value

With an instance of `ListTreeSet`, we can efficiently manage a set of scaling or wavelet coefficients.

In the wavelet series expansion, each coefficient is associated with a dilated and translated version of the scaling function or wavelet. It suffices to store the basic form of scaling function and wavelet as their versions can be derived from the basic forms. To apply Daubechies wavelets, a necessary step is to approximate their function values. We computed them on a very fine grid and used those values to compute a piecewise linear approximation. Besides in the evaluation, we also gain advantage from the piecewise linear representation in case we have to integrate a Daubechies wavelet.

The abstract class `FWT` unifies the components of the wavelet series expansion of a function. A wavelet family is implemented as an extension of `FWT`; this extension defines the underlying scaling function and wavelet as well as the corresponding filter. `FWT` uses two separate instances of `ListTreeSet` to store the scaling and wavelet coefficients respectively. Among the most frequently used features of this class is an implementation of the DWT. For practical purposes, `FWT` provides a so-called factory. Given an identifier for one of the implemented wavelet families, this factory returns a new instance of the class that implements this family. Generally, `FWT` prepares the ground for the implementation of WDEs.

The abstract class `AbstractWDE` is a preimplementation of WDEs; it provides common functionality of most WDE types as well as mechanisms to adjust the allocated memory. Internally, it uses the class `FWT` to store the wavelet series expansion of the WDE. `AbstractWDE` implements the computation of the empirical scaling coefficients, the evaluation and integration of the WDE, and the merge of two WDEs. In order to establish this functionality, `AbstractWDE` implements the following interfaces: `RealFunction`, `Integrable`,

`SummaryMeasures`, and `MemoryManageable`. While `RealFunction` and `Integrable` model the evaluation and integration respectively, `SummaryMeasure` models the computation of descriptive statistics like mean and variance. `MemoryManageable` models the adaptation to a changing amount of available memory. We implemented the methods of these interfaces with respect to the strategies and results of the previous sections. Due to the preimplemented functionality, new WDE types can be easily implemented as extensions of `AbstractWDE`. We already provide implementations of linear and thresholded WDEs in the classes `LinearWDE` and `ThresholdWDE`. Both classes provide specific factories which facilitate the computation of WDEs for a given sample.

Implementation of Compressed-cumulative WDEs Now that we presented the concepts underlying the implementation of WDEs, the remaining step is to implement the functions of `MergeAggregateFunction`. Different implementations of the weighting function already exist. Hence, the aggregation function and the merge and compress function remain to be implemented. While the class `BlockWDEAggregateFunction` implements the first function, the second one is implemented in the class `ConvexSetMerger`. `BlockWDEAggregateFunction` is an extension of `AggregationFunction`. It uses the aforementioned factories to compute the WDEs for data blocks with respect to a preset wavelet family. `ConvexSetMerger` is an extension of `Function`. In its `invoke` method, it merges the scaling and wavelet coefficient sets of two WDEs and compresses the resulting WDE afterward.

Storage Requirements Finally, we describe the storage requirements of compressed-cumulative WDEs. As they are stored in their wavelet series expansion, their storage space is mainly dictated by the number of coefficients in the coefficient sets. Each coefficient in turn comprises three components: resolution, translation index, and value.

Besides the estimator and its components, the second component that allocates memory is the buffer. Its storage space depends on the number of stream elements it stores in its internal queue.

Overall, the presented implementation concepts completed our discussion of compressed-cumulative WDEs over univariate data streams. Keeping the underlying framework and particularly Algorithm 1 in mind, let us briefly discuss some extensions.

5.2.8 Extensions

Due to the generality of the underlying framework and Algorithm 1, we can widen the scope of compressed-cumulative WDEs by means of suitable extensions. These extensions cover other WDE types and compression strategies, multivariate WDEs as well as deletions in the stream. A more detailed and elaborate discussion of these extensions remains to be addressed in future work.

5.2.8.1 Other WDE Types and Compression Strategies

Up to this point, we have discussed our approach with respect to linear and thresholded WDEs that are to be computed over univariate streams. However, using Algorithm 1 as starting point, we can easily plug-in other WDE types as well as different thresholding strategies.

In Section 3.5, we mentioned that other WDE types besides the linear and the thresholded ones exist. For a detailed discussion of other types, we refer to [138, 137] and the references these sources give. As an example for a different WDE type, let us consider how to provide *bona fide* estimates, i.e., non-negative estimates that integrate to 1. One way to obtain bona fide estimates is to estimate the square root of a density. A detailed discussion of this approach is given in [138], Section 7.4.1. Relevant in our context is that this approach also starts with estimating the empirical coefficients. They are used to build an estimate of the square root of the unknown density, which is squared afterward to obtain the intended density estimate. We can use the essential mechanisms of Algorithm 1 to compute those square root estimates in an online fashion: we build block estimates and successively merge them by merging their empirical coefficients; to reduce their storage space, we discard the least relevant empirical wavelet coefficients. These mechanisms can also be utilized for other WDE types as long as they are defined in terms of empirical scaling and wavelet coefficients. We conclude that the crucial step for adapting other WDE types is to examine their computation in order to establish the corresponding block WDEs.

Besides other WDE types, we can also plug-in other thresholding policies to select the empirical wavelet coefficients of a block WDE. [82] discusses in Chapter 11 various approaches for wavelet thresholding and adaptation. Again, the application of these policies only takes place in the computation of the block WDE. Therefore, they do not interfere with the other processing steps of Algorithm 1.

Another interesting aspect to examine is the compression of the current compressed-cumulative WDE. The main question is which of the empirical coefficients that define the current compressed-cumulative WDE shall be discarded so that the remaining ones fit into the available memory. As it ensures a minimum compression error, we concentrated on the strategy presented in Section 5.2.3, namely discarding the empirical wavelet coefficients with lowest absolute value. In recent years, a set of other strategies for discarding wavelet coefficients has been proposed in the literature. In Chapter 2, we already presented some of these approaches. In [65] for example, the authors propose to discard the coefficients with respect to a probabilistic thresholding scheme. Generally, these techniques can also be used with compressed-cumulative WDEs. They simply have to be implemented within the compression step. An invariant in this context is that they ensure that after the compression, the remaining empirical coefficients fit into a preset amount of memory.

5.2.8.2 Compressed-cumulative WDEs over multivariate Data Streams

As discussed in the previous section, the generality of our approach allows us to plug-in other WDE types. This particularly refers to multivariate WDEs, i.e., we can exploit Algorithm 1 to compute WDEs over multivariate streams essentially in the same manner as over univariate streams. As already sketched in 3.5.2.3, WDEs over multivariate data are essentially generalizations of WDEs over univariate data. Their construction also relies on estimating the scaling and wavelet coefficients of the according wavelet series expansion. In order to compute them in an online fashion, we have to provide the following components to be compliant with the framework: computation of a block WDE, merge step, and compression step. We assume that a weighting strategy is already available.

The computation of a block WDE is the most difficult part since it requires the implementation of the multi-dimensional DWT, multi-dimensional wavelets etc. Provided that a

block WDE is computed, we can merge it with the current compressed-cumulative WDE as described in Section 5.2.2; we simply merge the corresponding empirical scaling and wavelet coefficients. To compress the resulting estimator, we can apply the same strategy as in the univariate case; we drop the empirical wavelet coefficients with smallest absolute value.

Compressed-cumulative WDEs over multivariate data streams can be implemented similar to those over univariate streams. Again, the crucial step is to organize the empirical coefficients in a suitable data structure which provides easy access and modification facilities.

5.2.8.3 Deletions in the Stream

Up to now, we assumed that once an element has been processed, it cannot be deleted subsequently from the stream.[12] Since deletions in the stream may also be relevant in real-world scenarios, let us conclude this chapter with a brief discussion of this topic.

Let $X_1, ..., X_k$ be the already processed stream elements and $X_l, l \in \{1, ..., k\}$ the element to be deleted. Formally, the deletion of X_l from the current compressed-cumulative WDE, which is based on $X_1, ..., X_k$, shall deliver the compressed-cumulative WDE based on $X_1, ..., X_{l-1}, X_{l+1}, ..., X_k$.

However, as a closer look at Algorithm 1 reveals, the exact deletion of X_l from the current compressed-cumulative WDE is not possible. The main problem is that we do not know in which of the processed data blocks the element to be deleted did occur. Even if we would know the corresponding block, we would need to know all the subsequent data blocks as the deleted element induces a shift of the elements belonging to each block. As a consequence, the corresponding block estimators also change and would have to be recomputed. But this proceeding violates the processing requirements for streams presented in 3.1.2.

A simple approach allows us to model deletions in the stream at least approximately. We maintain a secondary buffer that stores all elements that are to be deleted. If this buffer contains b elements, we build a block estimator for them, which must be suitably weighted. Then we subtract this block estimator from the current compressed-cumulative WDE, which must also be suitably weighted before. To perform the subtraction, we merge their empirical coefficients as presented in 5.2.2, except that we invert the sign of each empirical coefficient of the "deletion estimator" before. It is important to note that this method comes along with several problems. For example, the weights of the block estimator and the compressed-cumulative WDE must be suitably adjusted. Additionally, the compressed-cumulative WDE may take on negative values due to the subtraction. For those reasons, a deeper investigation of this subject is necessary.

Overall, these extensions complete our discussion of compressed-cumulative WDEs. In the next chapter, we present Cluster Kernels, our approach to kernel density estimation over data streams.

[12] See also Section 3.1.1.

Chapter 6

Cluster Kernels: Kernel Density Estimators over Data Streams

In this chapter we present our Cluster Kernel approach, a sophisticated solution for kernel density estimation over data streams. We presented the basic idea of Cluster Kernels in [85] and more detailed in [88]. On the one hand, we designed Cluster Kernels very general and parameterized them by exchangeable strategies. On the other hand, we developed concrete strategies and implementations for them, resulting in ready-to-use estimators. We start in Section 6.1 with an overview of Cluster Kernels. As Section 6.2 will clarify, a previously proposed approach termed M-Kernels can be described as one specialization of Cluster Kernels. In Section 6.3, we thoroughly discuss concrete Cluster Kernels for the case of univariate data streams. Section 6.4 concludes this chapter with reasonable extensions of Cluster Kernels.

6.1 Overview of Cluster Kernels

Our goal is to develop a suitable technique for kernel density estimation over data streams, which takes the rigid processing requirements presented in Section 3.1.2 into account. Before we go into the details of our solution, we first provide a grasp of our basic idea. For the sake of simplicity, we concentrate on Cluster Kernels over univariate data streams. Section 6.4.2 separately discusses Cluster Kernels over multivariate data streams. Recall the general form of a KDE as described in equation (3.4). We establish specific functions $CK_i^{(n)}$, termed *Cluster Kernels*, so that

$$\hat{f}^{(n)}(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h^{(n)}}K\left(\frac{x-X_i}{h^{(n)}}\right) \approx \frac{1}{n}\sum_{i=1}^{m} CK_i^{(n)}(x), x \in \mathbb{R} \tag{6.1}$$

holds. In this equation, $\hat{f}^{(n)}$ refers to the offline KDE with bandwidth $h^{(n)}$ based on $X_1, ..., X_n$, the first n elements of the data stream. $\hat{f}^{(n)}$ is quasi the current optimal KDE computed with unlimited resources. While the number n of processed elements of the data stream continuously increases, we keep the total number m of Cluster Kernels constant. But the Cluster Kernels themselves change during runtime as we will see later. For that reason, we index them with (n).

Parameter	*Description*
n	# processed stream elements
X_i	i-th element of the stream
K	kernel function
$h^{(n)}$	optimal bandwidth for the first n elements
$\hat{f}^{(n)}$	optimal KDE for the first n elements
m	maximum number of Cluster Kernels
$CK_i^{(n)}$	i-th Cluster Kernel after n processed elements
$\bar{X}_i^{(n)}$	mean of $CK_i^{(n)}$
$\hat{h}_i^{(n)}$	bandwidth of $CK_i^{(n)}$
$mergecosts_{i,j}^{(n)}$	merge costs between $CK_i^{(n)}$ and $CK_j^{(n)}$
$stat_i^{(n)}$	local statistics of $CK_i^{(n)}$
$S_i^{(n)}$	partition associated with $CK_i^{(n)}$
$c_i^{(n)}$	# elements in $S_i^{(n)}$
$\hat{X}_{i,j}^{(n)}$	j-th resampled element of $S_i^{(n)}$

Table 6.1: Description of parameters

Each Cluster Kernel $CK_i^{(n)}$ represents a local group of elements of the stream, called *partition*. To be more specific, each Cluster Kernel represents the kernels over the elements of the partition. We consider m partitions $S_i^{(n)}, i = 1, ..., m$ of the stream; each one is associated with a separate Cluster Kernel $CK_i^{(n)}$. The memory requirements render the storage of all elements of a partition impossible. On account of this restriction, we equip each Cluster Kernel with simple local statistics $stat_i^{(n)}$ that continuously summarize the partition elements. This includes among others $c_i^{(n)}$, the current number of elements in $S_i^{(n)}$.

Generally, a Cluster Kernel is characterized by a well-defined partition representative $\bar{X}_i^{(n)}$, which we call *mean*, as well as bandwidth, local statistics, and merge costs:

$$CK_i^{(n)} =< \bar{X}_i^{(n)}, \hat{h}_i^{(n)}, stat_i^{(n)}, (mergecosts_{i,j}^{(n)})_{j=1,...,i-1,i+1,...,m} > . \tag{6.2}$$

Section 6.1.2 gives an explanation of $\bar{X}_i^{(n)}$ and $mergecosts_{i,j}^{(n)}$. For the sake of generality, each Cluster Kernel has in the general approach its own bandwidth $\hat{h}_i^{(n)}$ in order to cope with arbitrary bandwidth strategies. We later present a suitable technique for an approximate online computation of the normal scale rule, which provides a global bandwidth. Additionally, we will show how to gain advantage from the Epanechnikov kernel; this kernel simplifies the subsequently explained merge of Cluster Kernels as well as their evaluation.

As each Cluster Kernel represents the kernels over a local "cluster" of elements, i.e. the partition elements, we decided for the name Cluster Kernel. For the sake of clarity, Table 6.1 summarizes the essential parameters used throughout this chapter.

6.1.1 Evaluation of Cluster Kernels

Let us consider the evaluation of a Cluster Kernel $CK_i^{(n)}$. With respect to the partitions $S_1^{(n)}, ..., S_m^{(n)}$, Cluster Kernels are designed to meet the following requirement

$$CK_i^{(n)}(x) \approx \sum_{X_j \in S_i^{(n)}} \frac{1}{h^{(n)}} K\left(\frac{x - X_j}{h^{(n)}}\right), i = 1, ..., m. \tag{6.3}$$

Therefore, Cluster Kernels have to deliver results that are equivalent to those obtained by evaluating the kernels over all elements of a partition. Note that this corresponds to processing requirement 6 of Section 3.1.2.

To cope with the memory requirements, we store only local statistics $stat_i^{(n)}$ that summarize the partition elements. With the help of those statistics, we *resample* elements $\hat{X}_{i,j}^{(n)}, j = 1, ..., c_i^{(n)}$ of the partition and utilize them for the evaluation of the corresponding Cluster Kernel:

$$CK_i^{(n)}(x) = \sum_{j=1}^{c_i^{(n)}} \frac{1}{\hat{h}_i^{(n)}} K\left(\frac{x - \hat{X}_{i,j}^{(n)}}{\hat{h}_i^{(n)}}\right), i = 1, ..., m. \tag{6.4}$$

A crucial prerequisite for the evaluation of Cluster Kernels is that the sum in (6.4) can be converted into a closed formula. Otherwise, the evaluation cost for all Cluster Kernels would be $O(n)$ due to the fact that $\sum_{i=1}^{m} c_i^{(n)} = n$ always holds. The kernel function and the resampling strategies we present in Section 6.3 guarantee that this requirement is met.

Combining (6.4) with (6.1) delivers:

$$\hat{f}^{(n)}(x) \approx \frac{1}{n} \sum_{i=1}^{m} \sum_{j=1}^{c_i^{(n)}} \frac{1}{\hat{h}_i^{(n)}} K\left(\frac{x - \hat{X}_{i,j}^{(n)}}{\hat{h}_i^{(n)}}\right). \tag{6.5}$$

This equation reflects the main objectives of Cluster Kernels: they continuously keep summaries of partitions of the stream and use them to resample the partition elements; the evaluation of all resampled elements of all partitions in turn approximates the optimal KDE over all processed elements. Figure 6.1 illustrates the general procedure. In Algorithm 2, we summarize the essential steps for evaluating Cluster Kernels. For the sake of simplicity, we denote in the following the estimator based on Cluster Kernels also as $\hat{f}^{(n)}$. Whether $\hat{f}^{(n)}$ refers to the KDE based on Cluster Kernels or to the optimal KDE for the first n elements will be apparent from the context.

It is worth mentioning that we can exploit Cluster Kernels in a similar manner to approximate the integral over a KDE or to approximate summary measures like mean or variance. We address these issues later in this chapter.

6.1.2 Merge of Cluster Kernels

After we have discussed the components of Cluster Kernels as well as their evaluation after n processed elements, let us examine how to process an incoming new element X_{n+1}. First, we

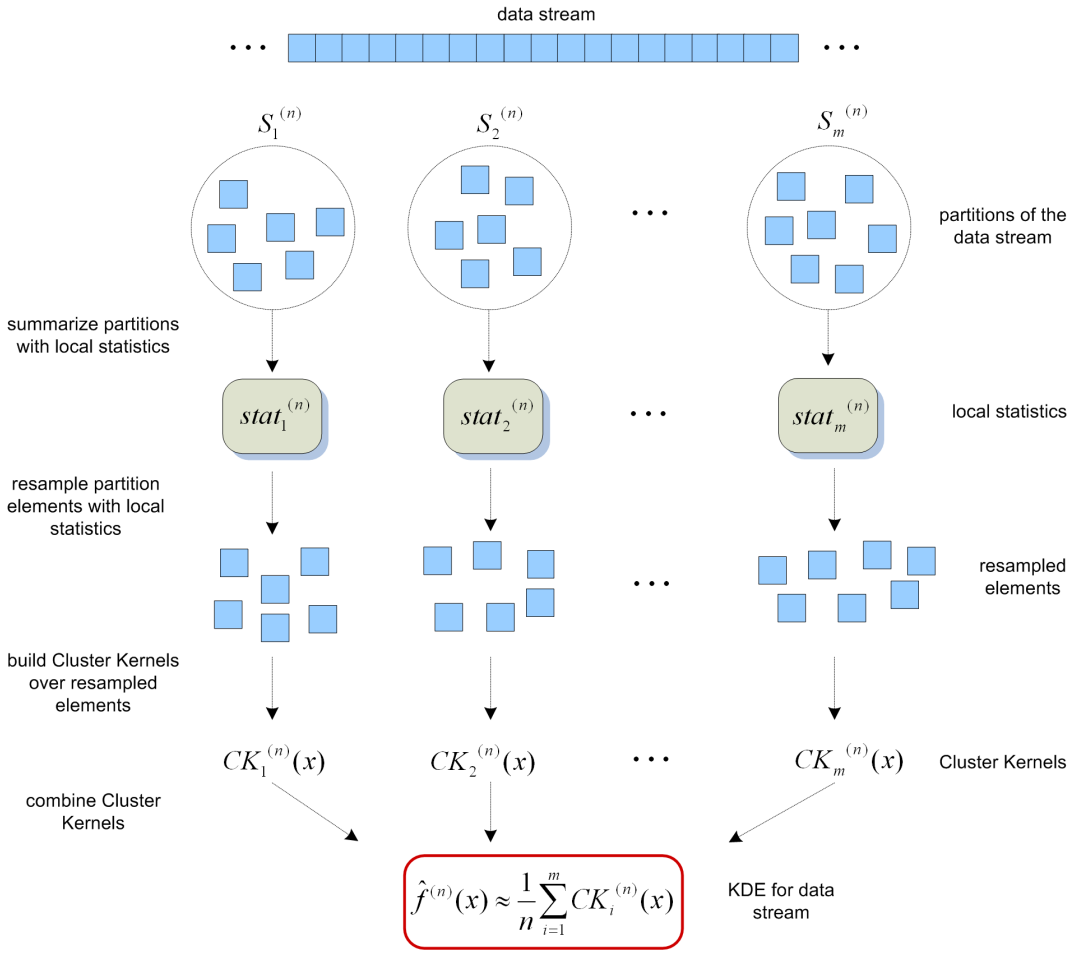


Figure 6.1: Resampling of partition elements

determine whether X_{n+1} equals the mean of an existing Cluster Kernel, i.e. $\exists i \in \{1, ..., m\} : X_{n+1} = \bar{X}_i^{(n)}$? If so, we update the local statistics $stat_i^{(n)}$ of this Cluster Kernel, which includes incrementing $c_i^{(n)}$, the element counter for the partition. If not, we build a new Cluster Kernel with mean X_{n+1} and properly initialized local statistics.

If the current number of Cluster Kernels plus the new one exceeds the maximum number m, we merge the two Cluster Kernels closest to each other into one Cluster Kernel. As this merge will be typically lossy, we introduce a *loss function* $loss(CK_i^{(n)}, CK_j^{(n)}, CK)$. This function describes the loss in accuracy that results from substituting $CK_i^{(n)}$ and $CK_j^{(n)}$ by the Cluster Kernel CK. By means of the loss function, we determine the *merge costs* of a Cluster Kernel pair as follows:

$$mergecosts_{i,j}^{(n)} = min\{loss(CK_i^{(n)}, CK_j^{(n)}, CK) : CK\ ClusterKernel\}. \tag{6.6}$$

The *merge kernel* of $CK_i^{(n)}$ and $CK_j^{(n)}$ is the Cluster Kernel that minimizes the loss function. The merge kernel will be located between $CK_i^{(n)}$ and $CK_j^{(n)}$, i.e., its mean will be between

Algorithm 2 Evaluation of Cluster Kernels

INPUT:	$CK_i^{(n)}$	set of m Cluster Kernels
	x	point to evaluate
OUTPUT:	$\hat{f}(x)$	approximated function value of the unknown density at point x

1: set the temporary variable *sum* to zero;
2: **for** $i = 1, ..., m$ **do**
3: resample $c_i^{(n)}$ elements of $S_i^{(n)}$ with $stat_i^{(n)}$;
4: evaluate $CK_i^{(n)}(x)$ for the resampled elements $\hat{X}_{i,j}^{(n)}$;
5: add the result to *sum*;
6: **end for**
7: return *sum* / n;

$\bar{X}_i^{(n)}$ and $\bar{X}_j^{(n)}$; its concrete location depends on the loss function. Besides the mean of the merge kernel, we also must set its bandwidth and its local statistics. To determine its local statistics, the local statistics $stat_i^{(n)}$ and $stat_j^{(n)}$ of the involved Cluster Kernels must be appropriately merged. The setting of the bandwidth depends on the underlying bandwidth strategy.

If the number of Cluster Kernels is $m + 1$ and has to be reduced, we choose among all pairs of Cluster Kernels the one with overall minimum merge costs and substitute them by their merge kernel. The number of potential merge pairs to consider is $(m + 1)m$. In case the Cluster Kernels can be totally ordered by their means $\bar{X}_i^{(n)}$, we can reduce the number of pairs to consider in order to reduce the computational effort. To be more concrete, the loss function has to ensure that the merge costs are monotonous with respect to the means of the corresponding Cluster Kernels, i.e. $mergecosts_{i,j}^{(n)} \leq mergecosts_{i,k}^{(n)}$ for $\bar{X}_i^{(n)} \leq \bar{X}_j^{(n)} \leq \bar{X}_k^{(n)}$. Hence, the merge of Cluster Kernels with closer means must cause lower merge costs. In case the Cluster Kernels are totally ordered by mean, it then suffices to compute the merge costs only between adjacent Cluster Kernels, which reduces the number of merge pairs to consider to m. Another advantage is that each Cluster Kernel only has to store the merge costs with its right neighbor, i.e. $CK_i^{(n)} =< \bar{X}_i^{(n)}, \hat{h}_i^{(n)}, stat_i^{(n)}, mergecosts_{i,i+1}^{(n)} >$. Note that the total ordering by mean is crucial for this pruning of potential merge pairs. Univariate data streams for instance have a total ordering, but multivariate ones with dimension $d > 1$ only have a partial ordering. The latter case will be addressed in 6.4.2.

The merge of Cluster Kernels renders their seamless integration into complex applications possible as it offers anytime-adaptation to changing system resources. In case of an increased maximum number m, we build separate Cluster Kernels for each new element, except duplicates, until the total number of Cluster Kernels equals m. In case of a decreased m, we perform the merge step sufficiently often. With the loss function, we have a reasonable tool to quantify the loss in accuracy induced by these merges.

In Algorithm 3, we summarize the essential steps that are required to build Cluster Kernels over a data stream.

Algorithm 3 Cluster Kernels over Data Stream

```
1: for each incoming element X do
2:   if ∃i ∈ {1,...,m} : X = X̄_i^(n) then
3:     update local statistics stat_i^(n);
4:   else
5:     insert new Cluster Kernel for X;
6:   end if
7:   while # Cluster Kernels > m do
8:     choose i, j with mergecosts_{i,j}^(n) = min;
9:     substitute CK_i^(n), CK_j^(n) by their merge kernel;
10:  end while
11:  return current set of Cluster Kernels;
12: end for
```

6.1.3 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. As a consequence, an analysis technique also has to take changes of the stream properly into account. To fulfil this need, we show an extension of Cluster Kernels that allows us to focus on recent trends of an evolving stream.

Let us examine an evolving stream from a formal point of view. According to Section 3.1.1, we consider a data stream as sample of an unknown random variable. Up to this point, we have assumed the stream to be stable, i.e., all stream elements follow the same distribution. Contrary to, the distribution underlying an evolving stream will change over time and, as a consequence, also the underlying pdf we want to estimate. In order to concentrate on the latest trends, we couple Cluster Kernels with exponential smoothing. Recall in this context the exponential weighting strategy presented in Section 5.1.2.

The basic idea of the coupling is to give older data less weight in the evaluation. Let us consider the current estimator after the insertion of a new element X_n and before the merge step is performed (the element shall not be a duplicate):

$$\hat{f}^{(n)}(x) \quad = \quad \frac{1}{n}\sum_{i=1}^{m} CK_i^{(n-1)}(x) + \frac{1}{n} CK^{(n)}(x)$$

for $n \geq 2$ and $\hat{f}^{(1)}(x) = CK_1^{(1)}(x)$. Each Cluster Kernel is equally weighted with $1/n$. With exponential smoothing, these equal weights are substituted by exponentially decreasing ones. More precisely, given $\alpha \in (0,1)$, the new Cluster Kernel receives weight α and triggers a re-scaling of older weights by a factor $(1-\alpha)$:

$$\hat{f}_\alpha^{(n)}(x) \quad = \quad (1-\alpha)\hat{f}_\alpha^{(n-1)}(x) + \alpha CK^{(n)}(x) \tag{6.7}$$

for $n \geq 2$ and $\hat{f}_\alpha^{(1)}(x) = CK_1^{(1)}(x)$. With the smoothing parameter α, we adjust the impact of old and new data respectively.

For illustration purposes, we assume the maximum number m of Cluster Kernels to be unbounded and determine the resulting weighting sequence. For $n \geq 2$, it follows

$$\hat{f}_{\alpha}^{(n)}(x) = (1-\alpha)^{n-1} CK_1^{(n)} + \sum_{i=2}^{n-1} (1-\alpha)^{n-i} \alpha CK_i^{(n)} + \alpha CK_n^{(n)}(x).$$

Instead of equal weights $1/n$ for each Cluster Kernel, we now have "discounted" weights: $(1-\alpha)^{n-i}\alpha$ and $(1-\alpha)^{n-1}$ for older Cluster Kernels and α for the new Cluster Kernel. The weighting sequence sums to 1 for each $n \in \mathbb{N}$. This is a necessary prerequisite for KDEs as otherwise the integration to 1, a fundamental property of each pdf, is violated. M-Kernels, a competitive technique, also support a weighting by means of a fadeout function [26], but the resulting weighting scheme violates this prerequisite. After examining the viability of Cluster Kernels, we take a closer look at M-Kernels.

6.1.4 Viability of Cluster Kernels

The presented Cluster Kernel approach meets the processing requirements for data streams postulated in Section 3.1.2. Each Cluster Kernel stores local statistics that summarize the elements of its partition. For that reason, each element is processed only once, i.e. requirement 1 is satisfied. Due to the merge procedure, we can keep the overall number of Cluster Kernels constant, which corresponds to requirement 3. The constant number also ensures a constant processing time per element in compliance with requirement 2. Since we can adjust the number of Cluster Kernels dynamically, we meet requirement 7. At each point in time, we have a valid estimator for the already processed stream at our disposal; requirement 4 is satisfied. With the resampling of partition elements, we aim to be close to the best offline KDE; requirement 6 is met. We tackle requirement 5, which addresses changes in the stream, with the weighing of recent data by means of exponential smoothing.

A major advantage of the Cluster Kernel approach is its modular design. Due to its exchangeable modules for the essential algorithm components, it constitutes a framework for the computation of KDEs over streaming data. In order to build Cluster Kernels, the following components must be defined: settings for kernel function and bandwidth, definition of local statistics, definition of a loss function, and development of resampling strategies. In the following sections, we elaborate concrete strategies and settings for those components for univariate as well as for multivariate streams. For the sake of applicability, we also attend to the implementation of Cluster Kernels and propose suitable solutions. Prior to these issues, let us first discuss M-Kernels.

6.2 M-Kernels over univariate Data Streams

The M-Kernel approach [26] initially inspired us to develop Cluster Kernels. We will briefly describe the essence of M-Kernels as, on the hand, we use them as competitive technique in our experiments and, on the other hand, they nicely fit into the general Cluster Kernel approach and can be described as one possible specialization for univariate data. Therefore, we discuss M-Kernels with respect to our approach, i.e., we discuss how they define the essential components required for Algorithm 2 and 3. An M-Kernel corresponds in our terminology to a Cluster Kernel.

6.2.1 Kernel Function and Bandwidth Settings

In [26], the authors propose to use the Gaussian kernel as underlying kernel function. Concerning the bandwidth, they equip each M-Kernel with a separate one. The bandwidth of a new M-Kernel is initially set to 1. In case two M-Kernels are merged, their merge kernel receives a new bandwidth, which, as we will see, depends on the loss function.

6.2.2 Local Statistics of M-Kernels

Each M-Kernel stores the number of elements it represents. With respect to our approach, this number constitutes the local statistics of an M-Kernel, i.e. $stat_i^{(n)} = \{c_i^{(n)}\}$. For a new M-Kernel, $c_i^{(n+1)}$ is set to 1. The counter is incremented if a new element X_{n+1} equals $\bar{X}_i^{(n)}$. In case of a merge between two M-Kernels, their merge kernel receives $c_i^{(n)} + c_j^{(n)}$ as its local statistics.

6.2.3 Loss Function for M-Kernels

The merge of two M-Kernels with means $\bar{X}_i^{(n)}$, $\bar{X}_j^{(n)}$ relies on detecting a merge kernel with mean X and bandwidth h so that

$$\frac{c_i^{(n)}}{\hat{h}_i^{(n)}} K\left(\frac{x - \bar{X}_i^{(n)}}{\hat{h}_i^{(n)}}\right) + \frac{c_j^{(n)}}{\hat{h}_j^{(n)}} K\left(\frac{x - \bar{X}_j^{(n)}}{\hat{h}_j^{(n)}}\right) \approx \frac{c_i^{(n)} + c_j^{(n)}}{h} K\left(\frac{x - X}{h}\right). \tag{6.8}$$

In order to measure the accuracy loss of a merge, the authors propose in [26] to minimize the L_1 distance, i.e. the mean absolute deviation, between the two M-Kernels and their merge kernel:

$$\int_{-\infty}^{+\infty} \left| \frac{c_i^{(n)}}{\hat{h}_i^{(n)}} K\left(\frac{x - \bar{X}_i^{(n)}}{\hat{h}_i^{(n)}}\right) + \frac{c_j^{(n)}}{\hat{h}_j^{(n)}} K\left(\frac{x - \bar{X}_j^{(n)}}{\hat{h}_j^{(n)}}\right) - \frac{c_i^{(n)} + c_j^{(n)}}{h} K\left(\frac{x - X}{h}\right) \right| dx. \tag{6.9}$$

In our terminology, (6.9) defines a loss function. This loss function produces monotonous merge costs. Thus, to determine the M-Kernel pair with minimum merge costs, it suffices to consider adjacent M-Kernels for a merge. It is important to note that this loss function has mean X and bandwidth h of the merge kernel as parameters, i.e., to determine the merge kernel, a two-dimensional function must be minimized. Moreover, transforming the integral in (6.9) into a closed formula, which would facilitate determining the minimum, is a difficult task. To overcome this problem, the authors propose to use numerical approximations of the minimum. More precisely, they exploit the downhill simplex algorithm [119], which can approximate the minimum of a multi-dimensional function.

6.2.4 Evaluation of M-Kernels

The entirety of M-Kernels constitutes an estimator $\hat{f}^{(n)}$ that is evaluated as follows:

$$\hat{f}^{(n)}(x) = \frac{1}{n} \sum_{i=1}^{m} \frac{c_i^{(n)}}{\hat{h}_i^{(n)}} K\left(\frac{x - \bar{X}_i^{(n)}}{\hat{h}_i^{(n)}}\right), x \in \mathbb{R}. \tag{6.10}$$

In terms of our approach, M-Kernels resample partition elements by generating $c_i^{(n)}$ identical instances of the corresponding mean, i.e. $\hat{X}_{i,j}^{(n)} = \bar{X}_i^{(n)}, j = 1, ..., c_i^{(n)}$. The use of those resampled elements in (6.4) in combination with (6.5) results in (6.10).

6.2.5 M-Kernels and evolving Streams

To emphasize recent data, M-Kernels also provide a weighting scheme. They use a so-called fade-out function to emphasize recent data. In accordance with [26], the resulting estimator for a new element X_n is defined as

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

with a parameter $\alpha \in (0, 1)$ to adjust the impact of recent data.

However, this weighting scheme causes serious problems as the resulting weights for the kernels do not sum to 1. Consider for example the estimator for the first two elements

$$\hat{f}^{(2)}(x) = (1-\alpha)\frac{1}{\hat{h}_1^{(2)}} K\left(\frac{x - X_1}{\hat{h}_1^{(2)}}\right) + (1+\alpha)\frac{1}{\hat{h}_2^{(2)}} K\left(\frac{x - X_2}{\hat{h}_2^{(2)}}\right).$$

For each α, the weights sum to 2. Keeping in mind that $K((x - X_i)/\hat{h}_i^{(n)})/\hat{h}_i^{(n)}$ integrates to 1, we see that the integral of $\hat{f}^{(2)}(x)$ will be 2. Consequently, the estimator violates an essential property of densities, namely the integration to 1.

6.2.6 Implementation of M-Kernels

With respect to (6.2), an M-Kernel is characterized by $< \bar{X}_i^{(n)}, \hat{h}_i^{(n)}, c_i^{(n)}, mergecosts_{i,i+1}^{(n)} >$. All M-Kernels are stored in a list sorted by mean $\bar{X}_i^{(n)}$. In case a new element arrives, either a new M-Kernel is inserted or an existing one is updated. The M-Kernels neighbored to the corresponding M-Kernel must update their merge costs as the merge costs depend on this M-Kernel's bandwidth, mean, and weight. In case of a subsequent merge, the adjacent M-Kernels with minimum merge costs are determined and merged. Again, the merge costs of neighbored M-Kernels must be updated.

6.2.7 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, renders an efficient evaluation as discussed in Section 3.4.6 impossible for M-Kernels.
- The proposed bandwidth setting has no theoretical foundation. It is not ensured that the bandwidth decreases with the sample size. But this is a fundamental premise for the consistency of KDEs according to their asymptotic properties described in Section 3.4.4.

- The numerical approximation of the minima in (6.9) causes additional computational effort and leads to less accurate values.
- M-Kernels capture evolving data streams by means of a fade-out function. But the resulting estimator violates the prerequisite of an integration to 1.

6.3 Cluster Kernels over univariate Data Streams

In this section, we thoroughly discuss our specialization of Cluster Kernels for univariate data streams. Not only do we examine elaborate settings and strategies for the essential components of Algorithm 3, we also present efficient solutions for their implementation. As our considerations rely on Algorithm 3, the processing requirements for data streams are inherently met.

6.3.1 Kernel Function and Bandwidth Settings

Contrary to the M-Kernel approach, we utilize a kernel function with bounded support because it facilitates the evaluation of a KDE as described in Section 3.4.6. Specifically, we decided to use the *Epanechnikov kernel*, which was already presented in Section 3.4.3, as underlying kernel function. Not only has this kernel function a simple form, its minimum roughness also improves the AMISE. We will see that the computation of the merge costs, the efficient evaluation of Cluster Kernels as well as their integration strongly rely on the simple form of this kernel function.

As mentioned in Section 3.4.2, the bandwidth as second parameter of a KDE is vital to the estimation quality. Theoretically founded and practically approved bandwidth strategies assign a global bandwidth to all kernels. As complex bandwidth strategies typically come along with a heavy computational burden, we concentrate on the normal scale rule, a simple yet convenient bandwidth strategy which is defined in (3.14). For the sake of an online application of this rule, we have to estimate the standard deviation 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 [35]. Given this estimate $\hat{\sigma}^{(n)}$, we can continuously compute a global bandwidth $\hat{h}^{(n)}$ while processing the data stream:

$$h^{(n)} = 1.06 \cdot \sigma^{(n)} \cdot n^{-\frac{1}{5}} \approx \hat{h}^{(n)} = 1.06 \cdot \hat{\sigma}^{(n)} \cdot n^{-\frac{1}{5}}. \tag{6.11}$$

The first bandwidth $\hat{h}^{(1)}$ is set to 1. Due to the global bandwidth, it follows $\hat{h}_i^{(n)} = \hat{h}^{(n)}$ for $i = 1, ..., m$. Therefore, we skip the index i of the bandwidth in the following.

6.3.2 Local Statistics of Cluster Kernels

To cope with the limited amount of available memory, we utilize local statistics to summarize the elements of a partition $S_i^{(n)}$. In this work, we concretely examine three strategies for computing the local statistics $stat_i^{(n)}$ of a univariate Cluster Kernel $CK_i^{(n)}$: one maintains a counter for the number of elements in the associated partition $S_i^{(n)}$, one additionally maintains minimum and maximum of the partition and one additionally maintains mean and variance.

For all strategies, we examine the setting of the according local statistics with respect to the following questions: How do we initialize the local statistics of a new Cluster Kernel $CK_i^{(n+1)}$ for a new element X_{n+1}? In case the new element X_{n+1} is equal to the mean of an existing Cluster Kernel, how do we update the local statistics of this Cluster Kernel? If we merge two Cluster Kernels, how do we set the local statistics of the resulting merge kernel?

The local statistics solely maintaining the element counter are defined as $stat_i^{(n)} = \{c_i^{(n)}\}$. The local statistics of a new Cluster Kernel are $stat_i^{(n+1)} = \{1\}$. In case of an update, they are set to $stat_i^{(n+1)} = \{c_i^{(n)} + 1\}$. In case of a merge of two Cluster Kernels $CK_i^{(n)}$ and $CK_j^{(n)}$, their merge kernel receives local statistics $stat^{(n)} = \{c_i^{(n)} + c_j^{(n)}\}$.

The local statistics based on element counter, minimum, and maximum are defined as $stat_i^{(n)} = \{c_i^{(n)}, min_i^{(n)}, max_i^{(n)}\}$. A new Cluster Kernel $CK_i^{(n+1)}$ has initial local statistics $stat_i^{(n+1)} = \{1, X_{n+1}, X_{n+1}\}$. In case of an update of $CK_i^{(n)}$, we only increment $c_i^{(n)}$; minimum and maximum are not affected. The local statistics are $stat_i^{(n+1)} = \{c_i^{(n)} + 1, min_i^{(n)}, max_i^{(n)}\}$. In case of a merge of $CK_i^{(n)}$ and $CK_j^{(n)}$, the local statistics of their merge kernel are set to $stat^{(n)} = \{c_i^{(n)} + c_j^{(n)}, min\{min_i^{(n)}, min_j^{(n)}\}, max\{max_i^{(n)}, max_j^{(n)}\}\}$.

The variance-based local statistics are similar to the microclusters used in [9]. We store sum $sum_i^{(n)}$ and squared sum $sqrdSum_i^{(n)}$ of the partition elements and utilize them to compute the sample standard deviation $\hat{\sigma}_i^{(n)}$ of $S_i^{(n)}$. The chief advantage of using sum and squared sum is that we can compute and merge them in an online manner. The local statistics of a new Cluster Kernel are initially $stat_i^{(n+1)} = \{1, X_{n+1}, X_{n+1} \cdot X_{n+1}\}$. In case of an update, the local statistics of the according Cluster Kernel $CK_i^{(n+1)}$ are $\{c_i^{(n)} + 1, sum_i^{(n)} + \bar{X}_i^{(n)}, sqrdSum_i^{(n)} + \bar{X}_i^{(n)} \cdot \bar{X}_i^{(n)}\}$. In case of a merge, the local statistics of the merge kernel are $stat^{(n)} = \{c_i^{(n)} + c_j^{(n)}, sum_i^{(n)} + sum_j^{(n)}, sqrdSum_i^{(n)} + sqrdSum_j^{(n)}\}$. From an implementation point of view, a drawback of this strategy is that the sums may become very large and difficult to represent.

For the sake of completion, we briefly describe how the sample variance and thus the sample standard deviation can be computed with sum and squared sum. With $\bar{X}$ as the average of the sample $X_1, ..., X_n$, the sample variance can be expressed as

$$\begin{aligned} \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2 &= \frac{1}{n-1}\left(\sum_{i=1}^{n} X_i^2 - 2n\bar{X}^2 + n\bar{X}^2\right) \\ &= \frac{1}{n-1}\left(\sum_{i=1}^{n} X_i^2 - n\bar{X}^2\right) \\ &= \frac{1}{n-1}\sum_{i=1}^{n} X_i^2 - \frac{1}{n(n-1)}\left(\sum_{i=1}^{n} X_i\right)^2. \end{aligned} \tag{6.12}$$

6.3.3 Resampling Strategies for Cluster Kernels

The basic idea of Cluster Kernels - see Section 6.1 - is to exploit their local statistics for resampling the elements of their partition. Instead of the original elements, we use the resampled elements for the essential applications of Cluster Kernels like evaluation, integration, and computation of summary measures. However, a crucial prerequisite in this context is

that a closed formula exists for each of these applications. Otherwise, the complexity would be linear in the stream size, which would violate the processing requirements for streams. In Section 6.1, we illustrated this necessity for the evaluation of Cluster Kernels.

With respect to the local statistics introduced in the previous section, let us present a set of convenient resampling strategies. As the later sections will clarify, we can determine for all of them closed formulas for the essential applications of Cluster Kernels.

One-value-resampling We start with the most simple strategy, which relies on the counter-based local statistics and the mean $\bar{X}_i^{(n)}$ of a Cluster Kernel's partition. Recall that the mean of a Cluster Kernel serves as representative for its partition $S_i^{(n)}$. By utilizing $c_i^{(n)}$, the number of elements in $S_i^{(n)}$, we generate $c_i^{(n)}$ identical instances of the mean

$$\hat{X}_{i,j}^{(n)} = \bar{X}_i^{(n)}, j = 1, ..., c_i^{(n)}. \tag{6.13}$$

We term this strategy *one-value-resampling*. Note that M-Kernels use this strategy according to Section 6.2.4. Even though this strategy is simple and has minimum storage requirements for its local statistics, it also has its drawbacks. Only one value represents the complete partition of a Cluster Kernel. If the partition has a wide range, we run the risk of not adequately representing it. Moreover, as Section 6.3.5 will reveal for the evaluation of Cluster Kernels with this strategy, large streams may induce an estimator that essentially consists of spikes at the means.

Min-max-resampling with equal Distances To circumvent the problem of spikes as possible with one-value-resampling, we aim at covering the complete partition with the resampled elements. We achieve this objective by means of the local statistics that store minimum and maximum of the partition besides the element counter. As minimum and maximum give us the value range of the partition elements, a natural approach is to distribute the resampled elements equally within this range. More precisely, min-max-resampling with equal distances distributes $c_i^{(n)}$ elements equidistantly over $[min_i^{(n)}, max_i^{(n)}]$. Formally, the elements of a Cluster Kernel $CK_i^{(n)}$ resampled with respect to this strategy are defined as

$$\hat{X}_{i,j}^{(n)} = min_i^{(n)} + (j-1) \cdot \frac{max_i^{(n)} - min_i^{(n)}}{c_i^{(n)} - 1}, j = 1, ..., c_i^{(n)}. \tag{6.14}$$

Figure 6.2 displays the general resampling procedure of this strategy, which we term *min-max-resampling with equal distances*. In comparison to one-value-resampling, this strategy does not run the risk of producing spikes. Instead of, the complete range is covered with elements. Intuitively, we expect this strategy to be particularly suitable for locally smooth distributions. However, this strategy may have problems with distributions exhibiting many outliers. An outlier will "expand" the range of the resampled elements even though the outlier itself is per se locally isolated or in a sparse region.

Min-max-resampling with exponential Distances An alternative resampling strategy also relies on distributing $c_i^{(n)}$ elements over $[min_i^{(n)}, max_i^{(n)}]$. While the previous strategy distributes the elements uniformly in the partition range, *min-max-resampling with exponential distances* also intends to cover the complete partition, but with a focus on distributing

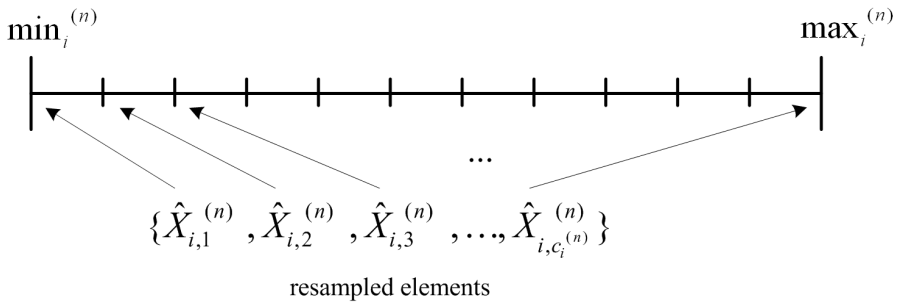


Figure 6.2: $c_i^{(n)}$ resampled elements equally distributed over $[min_i^{(n)}, max_i^{(n)}]$

most elements in the vicinity of the mean $\bar{X}_i^{(n)}$. Recall that the mean $\bar{X}_i^{(n)}$ serves as representative of the partition. We concretely place the resampled elements to the left and the right of the mean with exponentially increasing distances. Figure 6.3 provides a grasp of the resulting distribution of the resampled elements.

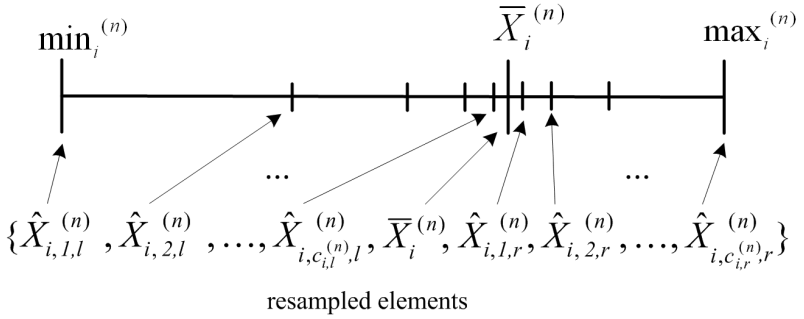


Figure 6.3: $c_i^{(n)}$ resampled elements distributed over $[min_i^{(n)}, max_i^{(n)}]$ with exponentially increasing distances from $\bar{X}_i^{(n)}$

We define the resampled elements for a given Cluster Kernel $CK_i^{(n)}$ as follows. We split $[min_i^{(n)}, max_i^{(n)}]$ in two parts located to the left and the right of the mean: $[min_i^{(n)}, \bar{X}_i^{(n)})$ and $(\bar{X}_i^{(n)}, max_i^{(n)}]$. While the mean itself constitutes one resampled element, we distribute the remaining $c_i^{(n)} - 1$ ones among the left and the right interval, proportional to the interval lengths. Let $c_{i,l}^{(n)}$ be the number of elements for $[min_i^{(n)}, \bar{X}_i^{(n)})$ and $c_{i,r}^{(n)}$ the one for $(\bar{X}_i^{(n)}, max_i^{(n)}]$:

$$c_{i,l}^{(n)} = \left\lceil \frac{(c_i^{(n)} - 1)(\bar{X}_i^{(n)} - min_i^{(n)})}{max_i^{(n)} - min_i^{(n)}} \right\rceil \text{ and } c_{i,r}^{(n)} = \left\lfloor \frac{(c_i^{(n)} - 1)(max_i^{(n)} - \bar{X}_i^{(n)})}{max_i^{(n)} - min_i^{(n)}} \right\rfloor. \tag{6.15}$$

The resampled elements $\hat{X}_{i,j,l}^{(n)}$ for $[min_i^{(n)}, \bar{X}_i^{(n)})$ are defined as

$$\hat{X}_{i,j,l}^{(n)} = \bar{X}_i^{(n)} - 2^{1-j}(\bar{X}_i^{(n)} - min_i^{(n)}),\ j = 1, ..., c_{i,l}^{(n)} \tag{6.16}$$

and analogously those for $(\bar{X}_i^{(n)}, max_i^{(n)}]$ as

$$\hat{X}_{i,j,r}^{(n)} = \bar{X}_i^{(n)} + 2^{j-c_{i,r}^{(n)}}(max_i^{(n)} - \bar{X}_i^{(n)}),\ j = 1, ..., c_{i,r}^{(n)}. \tag{6.17}$$

This strategy is less sensitive to outliers compared to the one based on equal distances. It is particularly designed for non-smooth distributions that exhibit local irregularities.

Mean-var-resampling with equal or exponential Distances We provide two other strategies closely related to the previous two strategies, except that they rely on the variance-based local statistics. To determine the range of the partition values, i.e. the resampling interval, we use the partition mean $\bar{X}_i^{(n)}$ and the estimated standard deviation $\hat{\sigma}_i^{(n)}$ to build $[\bar{X}_i^{(n)} - 2\hat{\sigma}_i^{(n)}, \bar{X}_i^{(n)} + 2\hat{\sigma}_i^{(n)}]$. The setting of the width of the resampling interval relies on the fact that the probability for a deviation from the mean by more than two standard deviations is less than 25% according to Chebyshev's inequality. In case of a normal distribution, this probability is only 5%.

Given the interval $[\bar{X}_i^{(n)} - 2\hat{\sigma}_i^{(n)}, \bar{X}_i^{(n)} + 2\hat{\sigma}_i^{(n)}]$ of a Cluster Kernel $CK_i^{(n)}$, we resample the elements in the same manner as for min-max-resampling by using equal or exponential distances. The formulas for the elements resampled with these *mean-var-resampling* strategies are identical to those for min-max-resampling, except that $min_i^{(n)}$ and $max_i^{(n)}$ are substituted by $\bar{X}_i^{(n)} - 2\hat{\sigma}_i^{(n)}$ and $\bar{X}_i^{(n)} + 2\hat{\sigma}_i^{(n)}$ respectively.

Generally, the setting of the resampling interval ensures that the mean-var-resampling strategies are more robust in the presence of outliers. A single outlier will not have as much impact on the resampling interval as it has with min-max-resampling.

Before we examine evaluation and integration of Cluster Kernels with respect to these resampling strategies, we introduce our loss function.

6.3.4 Loss Function for Cluster Kernels

Our definition of a loss function for two Cluster Kernels $CK_i^{(n)}$, $CK_j^{(n)}$ also bases on the objective implicitly described in equation (6.8). Consider the sum of kernels weighted with $c_i^{(n)}$ and $c_j^{(n)}$ over means $\bar{X}_i^{(n)}$ and $\bar{X}_j^{(n)}$ respectively. To merge these two kernels, we seek for the kernel that optimally approximates their sum. The crucial factors for the computation of the merge kernel are the means and the weights.

We concentrate on 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 Cluster

Kernels $CK_i^{(n)}, CK_j^{(n)}$:

$$
\begin{aligned}
loss(X) &= loss(CK_i^{(n)}, CK_j^{(n)}, CK) = \\
&\int_{-\infty}^{+\infty} \Bigg(\frac{c_i^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x-\bar{X}_i^{(n)}}{\hat{h}^{(n)}}\right) + \frac{c_j^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x-\bar{X}_j^{(n)}}{\hat{h}^{(n)}}\right) \\
&\quad - \frac{c_i^{(n)} + c_j^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x-X}{\hat{h}^{(n)}}\right) \Bigg)^2 dx \\
&= \int_{-\infty}^{+\infty} \Bigg(\frac{0.75 c_i^{(n)}}{\hat{h}^{(n)}} \Big(1 - \Big(\frac{x-\bar{X}_i^{(n)}}{\hat{h}^{(n)}}\Big)^2\Big) 1_{[\bar{X}_i^{(n)} - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]}(x) \\
&\quad + \frac{0.75 c_j^{(n)}}{\hat{h}^{(n)}} \Big(1 - \Big(\frac{x-\bar{X}_j^{(n)}}{\hat{h}^{(n)}}\Big)^2\Big) 1_{[\bar{X}_j^{(n)} - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]}(x) \\
&\quad - \frac{0.75(c_i^{(n)} + c_j^{(n)})}{\hat{h}^{(n)}} \Big(1 - \Big(\frac{x-X}{\hat{h}^{(n)}}\Big)^2\Big) 1_{[X - \hat{h}^{(n)}, X + \hat{h}^{(n)}]}(x) \Bigg)^2 dx \qquad (6.18)
\end{aligned}
$$

with X the mean of CK. We abbreviate $loss(CK_i^{(n)}, CK_j^{(n)}, CK)$ to $loss(X)$ to emphasize the fact that the mean X of CK is the only variable in (6.18). In contrast to this loss function, the M-Kernel approach 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 of $CK_i^{(n)}$ and $CK_j^{(n)}$, we set the value X^* that minimizes $loss(X)$, i.e. $loss(X^*) \leq loss(X)$ for all X. As we minimize the accuracy loss of the merge, the merge is optimal 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, the loss function is monotonous and it suffices to consider only the merge of adjacent Cluster Kernels. For that reason, we define $mergecosts_{i,i+1}^{(n)} = loss(X^*)$ as cost of merging the i-th and the $(i+1)$-th Cluster Kernel, provided that the Cluster Kernels are ordered by their means.

For illustrative purposes, we present in Figure 6.4 the shape of $loss(X)$ for two Cluster Kernels. The left y-axis describes the weighted kernels and the right one $loss(X)$, while the x-axis describes the support of the kernel functions as well as the possible means of the merge kernel. For reasons of simplicity, the bandwidth of both kernels is set to 1 in this example.

For the existence and the computation of the minimum of $loss(X)$, the following theorem is of utmost importance.

Theorem 6.1. *For arbitrary Cluster Kernels $CK_i^{(n)}, CK_j^{(n)}$, the minimum of $loss(X)$ exists and can be computed in constant time.*

In the proof of the theorem, we make us of the following lemma.

Lemma 6.2. *Let $Y_1, Y_2 \in \mathbb{R}$ be given with $Y_1 \geq Y_2$ and $|Y_1 - Y_2| < 2h$. Let $K(x)$ be the Epanechnikov kernel defined in Table 3.1. Under these assumptions, the following statement*

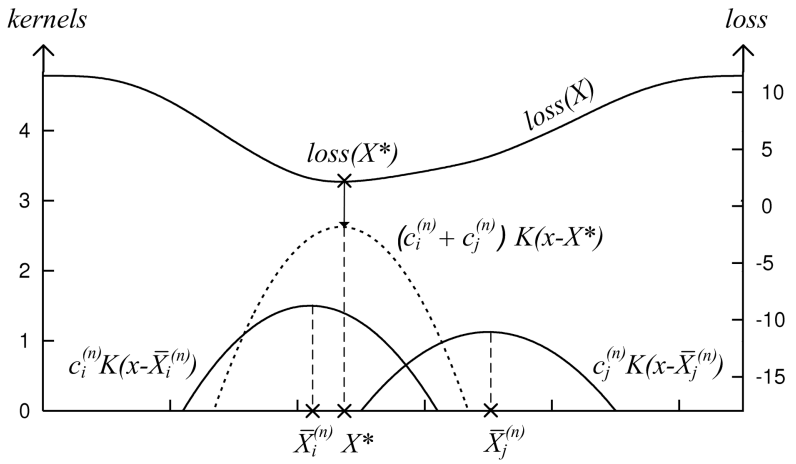


Figure 6.4: Merge of two Cluster Kernels with means $\bar{X}_i^{(n)}$, $\bar{X}_j^{(n)}$ and bandwidth $\hat{h}^{(n)} = 1$

holds with $h > 0$

$$\begin{aligned}\int_{-\infty}^{+\infty} \frac{1}{h^2} K\left(\frac{x-Y_1}{h}\right) K\left(\frac{x-Y_2}{h}\right) dx &= \frac{3}{160h^6}(Y_2 - Y_1 + h)^5 - \frac{3}{32h^5}(Y_2 - Y_1 + h)^4 \\ &\quad - \frac{3}{16h^4}(Y_2 - Y_1 + h)^3 + \frac{3}{16h^3}(Y_2 - Y_1 + h)^2 \\ &\quad + \frac{15}{32h^2}(Y_2 - Y_1 + h) + \frac{33}{160h}. \end{aligned} \tag{6.19}$$

Proof. With the definition of the Epanechnikov kernel, it follows

$$\begin{aligned}\int_{-\infty}^{+\infty} \frac{1}{h^2} K\left(\frac{x-Y_1}{h}\right) K\left(\frac{x-Y_2}{h}\right) dx &= \int_{-\infty}^{+\infty} \frac{3}{4h}\left(1 - \left(\frac{x-Y_1}{h}\right)^2\right) \cdot 1_{[Y_1-h, Y_1+h]}(x) \\ &\quad \cdot \frac{3}{4h}\left(1 - \left(\frac{x-Y_2}{h}\right)^2\right) \cdot 1_{[Y_2-h, Y_2+h]}(x) dx \\ &= \int_{Y_1-h}^{Y_2+h} \frac{9}{16h^2}\left(1 - \left(\frac{x-Y_1}{h}\right)^2\right)\left(1 - \left(\frac{x-Y_2}{h}\right)^2\right) dx. \end{aligned}$$

The integration borders in the second equation follow from computing the intersection between $[Y_1 - h, Y_1 + h]$ and $[Y_2 - h, Y_2 + h]$ and using $Y_1 \geq Y_2$ and $|Y_1 - Y_2| < 2h$. Equation (6.19) can be derived from this equation by means of algebraic conversions. □

Given this lemma, let us now prove Theorem 6.1.

Proof. In order to prove this theorem, we have to compute the minimum of $loss(X)$ as defined in equation (6.18). Let $CK_i^{(n)}, CK_j^{(n)}$ be two arbitrary Cluster Kernels with means $\bar{X}_i^{(n)}, \bar{X}_j^{(n)}$ and weights $c_i^{(n)}, c_j^{(n)}$ and let $\hat{h}^{(n)}$ be the current bandwidth. In case of $\bar{X}_i^{(n)} = \bar{X}_j^{(n)}$, the

minimum of $loss(X)$ equals $\bar{X}_i^{(n)}$. In the remainder of this proof, we assume $\bar{X}_i^{(n)} < \bar{X}_j^{(n)}$ without loss of generality.

The computation of the minimum of $loss(X)$ requires the following steps. First, we have to determine an equivalent representation of $loss(X)$ by solving the integral in (6.18). Second, we have to determine the first derivative of this representation of $loss(X)$. Third, we have to compute the roots of the derivative and choose among them the one that delivers the overall minimum of $loss(X)$.

In the first step, which is the most difficult one, we solve the integral in (6.18).

$$\begin{aligned}
loss(X) &= \int_{-\infty}^{+\infty} \left(\frac{c_i^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x-\bar{X}_i^{(n)}}{\hat{h}^{(n)}}\right) + \frac{c_j^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x-\bar{X}_j^{(n)}}{\hat{h}^{(n)}}\right) - \frac{c_i^{(n)}+c_j^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x-X}{\hat{h}^{(n)}}\right)\right)^2 dx \\
&= \int_{-\infty}^{+\infty} \left(\frac{c_i^{(n)}}{\hat{h}^{(n)}}\right)^2 K\left(\frac{x-\bar{X}_i^{(n)}}{\hat{h}^{(n)}}\right)^2 + \left(\frac{c_j^{(n)}}{\hat{h}^{(n)}}\right)^2 K\left(\frac{x-\bar{X}_j^{(n)}}{\hat{h}^{(n)}}\right)^2 \\
&\quad + \frac{2c_i^{(n)}c_j^{(n)}}{(\hat{h}^{(n)})^2} K\left(\frac{x-\bar{X}_i^{(n)}}{\hat{h}^{(n)}}\right) K\left(\frac{x-\bar{X}_j^{(n)}}{\hat{h}^{(n)}}\right) + \left(\frac{c_i^{(n)}+c_j^{(n)}}{\hat{h}^{(n)}}\right)^2 K\left(\frac{x-X}{\hat{h}^{(n)}}\right)^2 \\
&\quad - \frac{2c_i^{(n)}(c_i^{(n)}+c_j^{(n)})}{(\hat{h}^{(n)})^2} K\left(\frac{x-\bar{X}_i^{(n)}}{\hat{h}^{(n)}}\right) K\left(\frac{x-X}{\hat{h}^{(n)}}\right) \\
&\quad - \frac{2c_j^{(n)}(c_i^{(n)}+c_j^{(n)})}{(\hat{h}^{(n)})^2} K\left(\frac{x-\bar{X}_j^{(n)}}{\hat{h}^{(n)}}\right) K\left(\frac{x-X}{\hat{h}^{(n)}}\right) dx. \qquad (6.20)
\end{aligned}$$

With regard to the subsequent computation of the derivative of $loss(X)$, let us take a closer look at the summands that constitute the integrand. The first three summands do not depend on X. As a consequence, the value of their integral will be independent of X. Concerning the fourth summand, the use of Lemma 6.2 with $Y_1 = Y_2 = X$ shows that the value of the integral is also independent of X. Let *const* be the value of the integral over the first four summands. For the integral, it follows

$$\begin{aligned}
loss(X) &= const - \frac{2(c_i^{(n)}+c_j^{(n)})}{(\hat{h}^{(n)})^2} \int_{-\infty}^{+\infty} c_i^{(n)} K\left(\frac{x-\bar{X}_i^{(n)}}{\hat{h}^{(n)}}\right) K\left(\frac{x-X}{\hat{h}^{(n)}}\right) \\
&\quad + c_j^{(n)} K\left(\frac{x-\bar{X}_j^{(n)}}{\hat{h}^{(n)}}\right) K\left(\frac{x-X}{\hat{h}^{(n)}}\right) dx. \qquad (6.21)
\end{aligned}$$

As *const* is independent of X, it will vanish if we differentiate $loss(X)$ with respect to X. For that reason, the main question we have to tackle is how to compute the integral for the remaining two summands in equation (6.21). As we will see, both of them depend on X.

Each summand is the product of a kernel over $\bar{X}_i^{(n)}$ or $\bar{X}_j^{(n)}$ and a kernel over X. Since we use the Epanechnikov kernel, the supports of these three kernels are $[\bar{X}_i^{(n)} - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$, $[\bar{X}_j^{(n)} - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$, and $[X - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$. The intersection of the corresponding supports determines the support and thus the integration borders of a summand. Let I_i and

I_j be the supports of the first and the second summand respectively, i.e.

$$I_i = [\bar{X}_i^{(n)} - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}] \cap [X - \hat{h}^{(n)}, X + \hat{h}^{(n)}], \tag{6.22}$$

$$I_j = [\bar{X}_j^{(n)} - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}] \cap [X - \hat{h}^{(n)}, X + \hat{h}^{(n)}]. \tag{6.23}$$

While $\bar{X}_i^{(n)}$ and $\bar{X}_j^{(n)}$ are fix, the support of the kernel over X continuously varies in X. As a consequence, I_i and I_j also change with X. The evaluation of $loss(X)$ at an arbitrary $X \in \mathbb{R}$ requires to determine the resulting supports I_i and I_j.

We solve this task by introducing a suitable partitioning of the support of $loss(X)$, which uniquely defines the integration borders of both summands. As X determines the supports of both summands simultaneously, the first step is to examine the relative positions of $\bar{X}_i^{(n)}$ and $\bar{X}_j^{(n)}$ to each other. More precisely, we examine the relative positions of the supports $[\bar{X}_i^{(n)} - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$ and $[\bar{X}_j^{(n)} - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$ 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 $\bar{X}_j^{(n)} - \bar{X}_i^{(n)} \geq k\hat{h}^{(n)}$, i.e. $k = \lfloor (\bar{X}_j^{(n)} - \bar{X}_i^{(n)})/\hat{h}^{(n)} \rfloor$. If $k \in \{0, 1, 2, 3\}$, the supports intersect. Otherwise, they do not.

In case of $k \notin \{0, 1, 2, 3\}$, either I_i or I_j will be zero for a given $X \in \mathbb{R}$. A closer look at equation (6.21) reveals that the integral over the first summand becomes maximum for $X = \bar{X}_i^{(n)}$ since I_j gets maximum length in this case. The same is true for the second summand with $X = \bar{X}_j^{(n)}$. Due to the negative sign of the integral, it follows that $loss(X)$ has two minima: one at $\bar{X}_i^{(n)}$ and one at $\bar{X}_j^{(n)}$. Since $c_i^{(n)}$ and $c_j^{(n)}$ are a factor of the corresponding summand, the global minimum will be $\bar{X}_i^{(n)}$ if $c_i^{(n)} > c_j^{(n)}$ and $\bar{X}_j^{(n)}$ if $c_i^{(n)} < c_j^{(n)}$. If $c_i^{(n)} = c_j^{(n)}$, $\bar{X}_i^{(n)}$ and $\bar{X}_j^{(n)}$ both minimize (6.18), i.e., we can set the mean either to $\bar{X}_i^{(n)}$ or $\bar{X}_j^{(n)}$.

The case of $k \notin \{0, 1, 2, 3\}$ is more difficult as I_i and I_j may both be non-zero for a given $X \in \mathbb{R}$. In order to determine I_i and I_j, we slide a kernel with support $[X - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$ continuously varying in X over the x-axis and simultaneously evaluate the intersections of its support with $[\bar{X}_i^{(n)} - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$ and $[\bar{X}_j^{(n)} - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$. This delivers as result a partitioning of the support of $loss(X)$. For the case of $k = 0$, Figure 6.5 displays the different cases of intersections, the resulting partitioning, as well as the corresponding I_i and I_j within each partition. For $X \in [\bar{X}_j^{(n)} - 2\hat{h}^{(n)}, \bar{X}_i^{(n)}]$, the computation of I_i and I_j is illustrated as an example. The support partitionings for $k = 1, 2, 3$ can be determined analogously. In Appendix A, we summarize the partitionings and the corresponding I_i's and I_j's for $k = 0, 1, 2, 3$.

For each $k \in \{0, 1, 2, 3\}$, we know for each partition of the corresponding support partitioning the associated I_i and I_j. Given I_i and I_j, we can solve the integral in equation (6.21). In case I_i is empty, the integral of the first summand is zero. In case it is not empty, we conclude $|\bar{X}_i^{(n)} - X| < 2\hat{h}^{(n)}$ and we can also determine whether $\bar{X}_i^{(n)} \leq X$ or $\bar{X}_i^{(n)} \geq X$. This allows us to compute the integral of the first summand by means of Lemma 6.2. In the same manner, we can determine the value of the integral over the second summand. Overall, this gives us a piecewise definition of $loss(X)$ with respect to the support partitioning. For each partition, we determine the corresponding I_i and I_j and compute with their help the integral over the two summands in (6.21).

As each partition is associated with a combination of I_i and I_j, let us examine the different

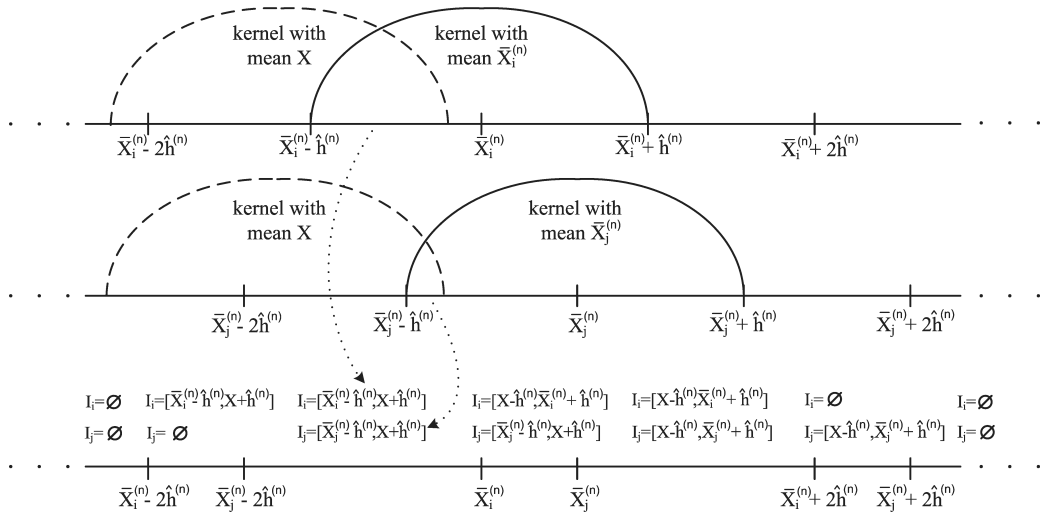


Figure 6.5: Integration intervals based on the support partitioning of $loss(X)$ for $k = 0$

combinations that can occur. I_i is either $[\bar{X}_i^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$, $[X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$, or $\varnothing$. Analogously, I_j is either $[\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$, $[X - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$, or $\varnothing$. Hence, we have 9 possible combinations for I_i and I_j. The combination $I_i = [X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$, $I_j = [X - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$ is not relevant as it implies $\bar{X}_i^{(n)} > \bar{X}_j^{(n)}$, a case we excluded before. For the remaining combinations, we utilize Lemma 6.2 to solve the corresponding integral over the summands. This gives us a closed formula for each combination of I_i and I_j and thus for each partition of the different support partitionings of $loss(X)$. Overall, we completed the first step of the proof, which was to solve the integral in (6.18).

With respect to the piece-wise definition of $loss(X)$ on the support partitions, we proceed with the second step, the computation of the derivative of $loss(X)$. As each partition is associated with one combination of I_i and I_j, Appendix A summarizes the resulting first derivatives for the different combinations of I_i and I_j. We see that each one is a polynomial of degree 4.

The third step is to determine the minima of $loss(X)$. For each partition, we compute the roots of the first derivative of $loss(X)$ in this partition. As $loss(X)$ is a polynomial of degree 4 in each partition, this corresponds to solving a quartic equation, a task we solved with Ferrari's formula. This gives us a set of local minima from the support partitions. By comparing them, we get the uniquely defined global minimum of $loss(X)$.

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

6.3.5 Evaluation of Cluster Kernels

In the previous sections, we thoroughly discussed the main building blocks of Cluster Kernels and their interplay. This particularly included the discussion of local statistics, accompanied by the introduction of resampling strategies that exploit these statistics. The objective of a resampling strategy is to provide resampled elements for each partition in order to approximate the KDE over all partition elements. A severe restriction in this context is that each

application of Cluster Kernels based on those resampled elements must have constant cost to meet our strict processing requirements. As a consequence, we must provide for each resampling strategy a closed formula for the corresponding application. Keeping this necessity in mind, we examine the core applications of Cluster Kernels, namely evaluation, integration, and computation of summary measures. The subsequent discussion of these applications will be of a more technical nature since our main goal is to derive closed formulas for each of the resampling strategies presented in Section 6.3.3. It is worth mentioning that other applications of Cluster Kernels and the corresponding closed formulas can be derived in a similar manner.

Let us start in this section with the evaluation of Cluster Kernels. According to equation (6.1), the sum of all Cluster Kernels constitutes the overall estimator. For that reason, it suffices to consider the evaluation of a single Cluster Kernel $CK_i^{(n)}$. We discuss the different resampling strategies with respect to equation (6.4).

One-value-resampling For the evaluation of a Cluster Kernel with one-value-resampling, we can derive the following theorem.

Theorem 6.3. *Let $CK_i^{(n)}$ be a Cluster Kernel with $stat_i^{(n)} = \{c_i^{(n)}\}$ and resampled elements $\hat{X}_{i,j}^{(n)}, j = 1, ..., c_i^{(n)}$ as defined in (6.13). For each $x \in \mathbb{R}$,*

$$CK_i^{(n)}(x) = \frac{c_i^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x - \bar{X}_i^{(n)}}{\hat{h}^{(n)}}\right). \tag{6.24}$$

Proof. We simply plug the resampled elements into (6.4). □

This strategy suffers from the aforementioned problem of spikes for extremely large data streams. Remember that the bandwidth tends to zero for an increasing sample size; so does our bandwidth setting in (6.11). Relating this fact to (6.5) and (6.24), we see that for very large n, a KDE based on one-value-resampling will become the sum of m peaks at the partition representatives $\bar{X}_i^{(n)}, i = 1, ..., m$. For the extreme case of an infinite stream, the KDE will be a kind of sum of m Dirac delta functions, i.e., it will consist of m singularities. Note that the resulting evaluation of a Cluster Kernel is identical to the evaluation of M-Kernels presented in Section 6.2.4. Consequently, M-Kernels suffer from the same problem.

Min-max-resampling with equal Distances For the evaluation of a Cluster Kernel based on min-max-resampling with equal distances, a closed formula exists according to the following theorem.

Theorem 6.4. *Let $CK_i^{(n)}$ be a Cluster Kernel with $stat_i^{(n)} = \{c_i^{(n)}, min_i^{(n)}, max_i^{(n)}\}$ and resampled elements $\hat{X}_{i,j}^{(n)}, j = 1, ..., c_i^{(n)}$ as defined in (6.14). For each $x \in \mathbb{R}$, indexes $k_1, k_2 \in$*

$\mathbb{N}$ *exist with* $k_1 \leq k_2$ *so that* $CK_i^{(n)}$ *is evaluated at* $x \in \mathbb{R}$ *as follows*

$$\begin{aligned} CK_i^{(n)}(x) &= \frac{0.75}{(\hat{h}^{(n)})^3}\Bigg((k_2-k_1+1)((\hat{h}^{(n)})^2-(x-min)^2) \\ &+\frac{(x-min_i^{(n)})(max_i^{(n)}-min_i^{(n)})}{c_i^{(n)}-1}\Big(-k_1^2+k_2^2+k_1+k_2\Big) \\ &-\frac{(max_i^{(n)}-min_i^{(n)})^2}{6(c_i^{(n)}-1)^2}\big(k_2(k_2+1)(2k_2+1) \\ &-(k_1-1)k_1(2k_1-1)\big)\Bigg). \end{aligned} \tag{6.25}$$

Proof. Let $x \in \mathbb{R}$ be an arbitrary point to be evaluated. We determine $k_1, k_2 \in \{1, ..., c_i^{(n)}\}$ so that $K((x - \hat{X}_{i,j}^{(n)})/\hat{h}^{(n)}) \neq 0$ only for $j \in \{k_1, ..., k_2\}$. With the Epanechnikov kernel as underlying kernel function, it follows

$$K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right) \neq 0 \Leftrightarrow 1_{[\hat{X}_{i,j}^{(n)}-\hat{h}^{(n)}, \hat{X}_{i,j}^{(n)}+\hat{h}^{(n)}]}(x) \neq 0.$$

Thus, k_1 is the smallest and k_2 the largest index from $\{1, ..., c_i^{(n)}\}$ so that $x \leq \hat{X}_{i,k_1}^{(n)} + \hat{h}^{(n)}$ and $x \geq \hat{X}_{i,k_2}^{(n)} - \hat{h}^{(n)}$ as well as $k_1 \leq k_2$:

$$k_1 = \left\lceil \frac{(c_i^{(n)}-1)(x-min_i^{(n)}-\hat{h}^{(n)})}{max_i^{(n)}-min_i^{(n)}} + 1 \right\rceil, k_2 = \left\lfloor \frac{(c_i^{(n)}-1)(x-min_i^{(n)}+\hat{h}^{(n)})}{max_i^{(n)}-min_i^{(n)}} + 1 \right\rfloor. \tag{6.26}$$

Given k_1 and k_2, the evaluation of $CK_i^{(n)}$ reduces to

$$CK_i^{(n)}(x) = \sum_{j=k_1}^{k_2} \frac{1}{\hat{h}^{(n)}} K\left(\frac{1}{\hat{h}^{(n)}}\left(x - min_i^{(n)} - (j-1)\cdot\frac{max_i^{(n)}-min_i^{(n)}}{c_i^{(n)}-1}\right)\right). \tag{6.27}$$

With the help of algebraic conversions, we can derive (6.25) from (6.27). □

Min-max-resampling with exponential Distances For the min-max-resampling strategy based on exponential distances, there also exists a closed formula.

Theorem 6.5. *Let* $CK_i^{(n)}$ *be a Cluster Kernel with* $stat_i^{(n)} = \{c_i^{(n)}, min_i^{(n)}, max_i^{(n)}\}$ *and resampled elements* $\hat{X}_{i,j,l}^{(n)}, j = 1, ..., c_{i,l}^{(n)}$ *and* $\hat{X}_{i,j,r}^{(n)}, j = 1, ..., c_{i,r}^{(n)}$ *as defined in* (6.16) *and* (6.17) *respectively. For each* $x \in \mathbb{R}$, *indexes* $k_{1,l}, k_{2,l}, k_{1,r}, k_{2,r} \in \mathbb{N}$ *exist so that* $CK_i^{(n)}$ *is*

evaluated at $x \in \mathbb{R}$ *as follows*

$$\begin{aligned} CK_i^{(n)}(x) &= \frac{0.75}{(\hat{h}^{(n)})^3}\Big(((\hat{h}^{(n)})^2 - (x-\bar{X}_i^{(n)})^2)1_{[\bar{X}_i^{(n)}-\hat{h}^{(n)},\bar{X}_i^{(n)}+\hat{h}^{(n)}]}(x) \\ &+(k_{2,l}-k_{1,l}+k_{2,r}-k_{1,r}+2)((\hat{h}^{(n)})^2-(x-\bar{X}_i^{(n)})^2) \\ &-8(\frac{1}{2^{k_{1,l}}}-\frac{1}{2^{k_{1,l}+1}})(x-\bar{X}_i^{(n)})(\bar{X}_i^{(n)}-min_i^{(n)}) \\ &-\frac{16}{3}(\frac{1}{4^{k_{2,l}}}-\frac{1}{4^{k_{2,l}+1}})(\bar{X}_i^{(n)}-min_i^{(n)})^2 \\ &+(2^{k_{2,r}-c_{i,r}^{(n)}+2}-2^{k_{1,r}-c_{i,r}^{(n)}+1})(max_i^{(n)}-\bar{X}_i^{(n)})(x-\bar{X}_i^{(n)}) \\ &-\frac{1}{3}(4^{k_{2,r}-c_{i,r}^{(n)}+1}-4^{k_{1,r}-c_{i,r}^{(n)}})(max-\bar{X}_i^{(n)})^2\Big). \end{aligned} \tag{6.28}$$

Proof. Analogous to the proof of Theorem 6.4, we determine maximum index sets $k_{1,l}, k_{2,l} \in \{1,...,c_{i,l}^{(n)}\}$ and $k_{1,r}, k_{2,r} \in \{1,...,c_{i,r}^{(n)}\}$ that guarantee $K((x-\hat{X}_{i,j,l}^{(n)})/\hat{h}^{(n)}) \neq 0$ and $K((x-\hat{X}_{i,j,r}^{(n)})/\hat{h}^{(n)}) \neq 0$ only for $j \in \{k_{1,l},...,k_{2,l}\}$ and $j \in \{k_{1,r},...,k_{2,r}\}$ respectively. $k_{1,l}$ is the smallest and $k_{2,l}$ the largest index from $\{1,...,c_{i,l}^{(n)}\}$ so that $x \leq \hat{X}_{i,k_{1,l},l}^{(n)} + \hat{h}^{(n)}$ and $x \geq \hat{X}_{i,k_{2,l},l}^{(n)} - \hat{h}^{(n)}$ as well as $k_{1,l} \leq k_{2,l}$:

$$k_{1,l} = \left\lceil \frac{1}{ln\,2} ln\left(\frac{2(\bar{X}_i^{(n)}-min_i^{(n)})}{\bar{X}_i^{(n)}+\hat{h}^{(n)}-x}\right)\right\rceil, \tag{6.29}$$

$$k_{2,l} = \left\lfloor \frac{1}{ln\,2} ln\left(\frac{2(\bar{X}_i^{(n)}-min_i^{(n)})}{\bar{X}_i^{(n)}-\hat{h}^{(n)}-x}\right)\right\rfloor. \tag{6.30}$$

For $k_{1,r}$ and $k_{2,r}$, it follows analogously:

$$k_{1,r} = \left\lceil c_{i,r}^{(n)} + \frac{1}{ln\,2} ln\left(\frac{x-\bar{X}_i^{(n)}-\hat{h}^{(n)}}{max_i^{(n)}-\bar{X}_i^{(n)}}\right)\right\rceil, \tag{6.31}$$

$$k_{2,r} = \left\lfloor c_{i,r}^{(n)} + \frac{1}{ln\,2} ln\left(\frac{x-\bar{X}_i^{(n)}+\hat{h}^{(n)}}{max_i^{(n)}-\bar{X}_i^{(n)}}\right)\right\rfloor. \tag{6.32}$$

Given these indexes and the resampled elements, the evaluation of $CK_i^{(n)}$ reduces to

$$\begin{aligned} CK_i^{(n)}(x) &= \frac{1}{\hat{h}^{(n)}}K\left(\frac{x-\bar{X}_i^{(n)}}{\hat{h}^{(n)}}\right) + \sum_{j=k_{1,l}}^{k_{2,l}} \frac{1}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j,l}^{(n)}}{\hat{h}^{(n)}}\right) \\ &+ \sum_{j=k_{1,r}}^{k_{2,r}} \frac{1}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j,r}^{(n)}}{\hat{h}^{(n)}}\right). \end{aligned} \tag{6.33}$$

We determine (6.28) from (6.33) with the help of algebraic conversions. □

Mean-var-resampling with equal or exponential Distances Besides the min-max-resampling strategies, we also examined mean-var-resampling strategies in Section 6.3.3. These two strategy types differ in the local statistics they use as well as in their setting of the resampling interval. We can derive the closed formulas for mean-var-resampling with equal or exponential distances by substituting in Theorem 6.4 and 6.5 $min_i^{(n)}$ by $\bar{X}_i^{(n)} - 2\hat{\sigma}_i^{(n)}$ and $max_i^{(n)}$ by $\bar{X}_i^{(n)} + 2\hat{\sigma}_i^{(n)}$.

Overall, we see that a closed formula for the evaluation of a Cluster Kernel exists for each resampling strategy.

6.3.6 Integration of Cluster Kernels

Subsequent to the evaluation of Cluster Kernels, we investigate their integration, which is of utmost importance for practical and theoretical purposes. Recall from Section 3.2 that probabilities of a continuous random variable X are computed via integrating its associated density. As we can express these integrals also in terms of the cumulative distribution function (cdf), the density's antiderivative, we examine how Cluster Kernels can be exploited to estimate the cdf.

The general method founds on the basic idea of Cluster Kernels presented in Section 6.1. To meet the processing requirements for streams, we maintain local statistics of partitions of the stream and use them to resample elements. While we evaluated the kernels over the resampled elements in the last section, we now integrate the kernels over them as follows. As we estimate the unknown density f with Cluster Kernels via

$$\begin{aligned} f(x) &\approx \frac{1}{n}\sum_{i=1}^{m} CK_i^{(n)}(x) \\ &= \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{c_i^{(n)}} \frac{1}{\hat{h}^{(n)}} K\left(\frac{x - \hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right), \end{aligned}$$

the natural approach is to estimate the cdf F via

$$\begin{aligned} F(t) &= \int_{-\infty}^{t} f(x)dx \\ &\approx \int_{-\infty}^{t} \frac{1}{n}\sum_{i=1}^{m} CK_i^{(n)}(x)dx \\ &= \frac{1}{n}\sum_{i=1}^{m} \int_{-\infty}^{t} CK_i^{(n)}(x)dx. \end{aligned} \tag{6.34}$$

Thereby, the main question is how to integrate a Cluster Kernel CK_i with respect to the resampled elements it relies on. Again, a crucial premise is the existence of a closed formula for the integration of a Cluster Kernel. The following considerations prepare the ground for the derivation of closed formulas for the presented resampling strategies.

We presuppose that the resampled elements $\hat{X}_{i,j}^{(n)}, j = 1, ..., c_i^{(n)}$ are in an increasing order, i.e. $\hat{X}_{i,j}^{(n)} \leq \hat{X}_{i,j+1}^{(n)}, j = 1, ..., c_i^{(n)} - 1$. Let us note that we use the Epanechnikov kernel as underlying kernel function.

$$\begin{aligned}
\int_{-\infty}^{t} CK_i^{(n)}(x)dx &= \int_{-\infty}^{t} \sum_{j=1}^{c_i^{(n)}} \frac{1}{\hat{h}^{(n)}} K\left(\frac{x - \hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right) dx \\
&= \sum_{j=1}^{c_i^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\hat{X}_{i,j}^{(n)})} K(x)dx \\
&= \sum_{j=1}^{c_i^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}\left(t-\hat{X}^{(n)}_{i,c_i^{(n)}-j+1}\right)} K(x)dx \\
&= \sum_{j=1}^{c_i^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}\left(t-\hat{X}^{(n)}_{i,c_i^{(n)}-j+1}\right)} 0.75 \cdot (1-x^2) \cdot 1_{[-1,1]}(x)dx \\
&= \sum_{j=1}^{c_i^{(n)}} \int_{\left(-\infty, \frac{1}{\hat{h}^{(n)}}\left(t-\hat{X}^{(n)}_{i,c_i^{(n)}-j+1}\right)\right] \cap [-1,1]} 0.75 \cdot (1-x^2)dx. \qquad (6.35)
\end{aligned}$$

To compute those integrals, the main step is to examine the different cases of intersections $(-\infty, \frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1})] \cap [-1, 1]$ for the resampled elements $\hat{X}^{(n)}_{i,c_i^{(n)}-j+1}, j = 1, ..., c_i^{(n)}$. Due to the ordering of the resampled elements, the change of the index from j to $c_i^{(n)} - j + 1$ in the upper equations implies $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) \leq \frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-(j+1)+1})$ for $j = 1, ..., c_i^{(n)} - 1$, i.e. an increasing ordering. This ordering is a prerequisite for determining the intersections. For illustration purposes, Figure 6.6 displays concrete intersections as well as the indexes k_1, k_2, which play an important role in the following theorem.

Theorem 6.6. *For an increasing sequence $\hat{X}_{i,j}^{(n)} \in \mathbb{R}, j = 1, ..., c_i^{(n)}$, indexes $k_1, k_2 \in \{1, ..., c_i^{(n)}\}$ exist such that*

$$\int_{-\infty}^{t} CK_i^{(n)}(x)dx = \begin{cases}
0, & \frac{1}{\hat{h}^{(n)}}(t - \hat{X}_{i,1}^{(n)}) \leq -1 & (6.36a) \\
c_i^{(n)}, & \frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}}) \geq 1 & (6.36b) \\
\sum_{j=k_1}^{k_2} \left(\frac{1}{4} \cdot \left(\frac{3}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) - \frac{1}{(\hat{h}^{(n)})^3}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1})^3\right)\right. & \\
\left. + c_i^{(n)} + \frac{1}{2}(1 - k_2 - k_1), \quad else. \right. & (6.36c)
\end{cases}$$

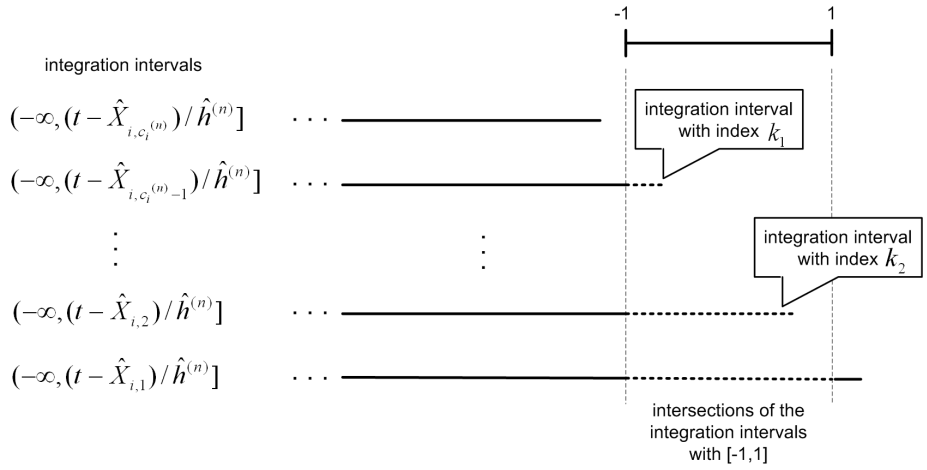


Figure 6.6: Intersections of integration intervals and $[-1,1]$ for a sequence of resampled elements

Proof. We examine the different cases of intersection between $(-\infty, \frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,c_i^{(n)}-j+1})]$ and $[-1,1]$ for $j=1,...,c_i^{(n)}$:

First case: $\frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,1}) \leq -1$

Due to the ordering, $\frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) \leq -1 \ \forall j \in \{1,...,c_i^{(n)}\}$ follows. Hence, all intersections are empty, i.e. $(-\infty, \frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,c_i^{(n)}-j+1})] \cap [-1,1] = \emptyset \ \forall j \in \{1,...,c_i^{(n)}\}$. For the integration, it follows

$$\int_{-\infty}^{t} CK_i^{(n)}(x)dx = \sum_{j=1}^{c_i^{(n)}} \int_{\emptyset} 0.75 \cdot (1-x^2)dx = 0.$$

Second case: $\frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,c_i^{(n)}}) \geq 1$

Due to the ordering, $\frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) \geq 1 \ \forall j \in \{1,...,c_i^{(n)}\}$ follows. All intersections are equal to $[-1,1]$, i.e. $(-\infty, \frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,c_i^{(n)}-j+1})] \cap [-1,1] = [-1,1] \ \forall j \in \{1,...,c_i^{(n)}\}$. For the integration, it follows

$$\int_{-\infty}^{t} CK_i^{(n)}(x)dx = \sum_{j=1}^{c_i^{(n)}} \int_{[-1,1]} 0.75 \cdot (1-x^2)dx = c_i^{(n)}.$$

Third case: $\frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,1}) > -1$ and $\frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,c_i^{(n)}}) < 1$

In this case, we determine indexes $k_1, k_2 \in \{1,...,c_i^{(n)}\}, k_1 \leq k_2$ which allow us to "split" the integration (see also Figure 6.6). The index k_1 refers to the smallest $j \in \{1,...,c_i^{(n)}\}$ with $\frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) \geq -1$. The index k_2 refers to the largest $j \in \{1,...,c_i^{(n)}\}$ with $\frac{1}{\hat{h}^{(n)}}(t-$

$\hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) \leq 1$. For all $j \in \{1, ..., k_1 - 1\}$, the intersection of $(-\infty, \frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1})]$ and $[-1, 1]$ is empty, while for all $j \in \{k_2 + 1, ..., c_i^{(n)}\}$ the intersection is $[-1, 1]$.

For the integration, it follows:

$$\begin{aligned}
\int_{-\infty}^{t} CK_i^{(n)}(x)dx &= \sum_{j=k_1}^{k_2} \int_{[-1, \frac{1}{\hat{h}^{(n)}}(t-\hat{X}^{(n)}_{i,c_i^{(n)}-j+1})]} 0.75 \cdot (1 - x^2)dx \\
&+ \sum_{j=k_2+1}^{c_i^{(n)}} \int_{[-1,1]} 0.75 \cdot (1 - x^2)dx \\
&= \sum_{j=k_1}^{k_2} \frac{1}{4} \cdot \Big(\frac{3}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) - \frac{1}{(\hat{h}^{(n)})^3}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1})^3 \Big) \\
&- \sum_{j=k_1}^{k_2} \frac{1}{4} \cdot (3(-1) - (-1)^3) + \sum_{j=k_2+1}^{c_i^{(n)}} 1 \\
&= \sum_{j=k_1}^{k_2} \frac{1}{4} \cdot \Big(\frac{3}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) - \frac{1}{(\hat{h}^{(n)})^3}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1})^3 \Big) \\
&+ \frac{1}{2}(k_2 - k_1 + 1) + c_i^{(n)} - k_2 \\
&= \sum_{j=k_1}^{k_2} \frac{1}{4} \cdot \Big(\frac{3}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) - \frac{1}{(\hat{h}^{(n)})^3}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1})^3 \Big) \\
&+ c_i^{(n)} + \frac{1}{2}(1 - k_2 - k_1).
\end{aligned}$$

□

While the first (6.36a) and the second case (6.36b) in Theorem 6.6 already provide a closed formula, the third case (6.36c) depends on the sum in the upper equation; the components of this sum in turn depend on the resampling strategy. With regard to the streaming context, each resampling strategy must provide an equivalent representation of the sum as a closed formula. The following lemma facilitates the derivation of the corresponding closed formulas.

Lemma 6.7. *Let $\alpha, \beta \in \mathbb{R}$ and let $h(j)$ be a function varying in $j \in \mathbb{N}$. Then*

$$\begin{aligned}
\sum_{j=k_1}^{k_2} \frac{1}{4}\big(3(\alpha + h(j)\beta) - (\alpha + h(j)\beta)^3\big) &= \frac{1}{4}\Big((k_2 - k_1 + 1)(3\alpha - \alpha^3) + 3(1 - \alpha^2)\beta \sum_{j=k_1}^{k_2} h(j) \\
&\quad -3\alpha\beta^2 \sum_{j=k_1}^{k_2} h(j)^2 - \beta^3 \sum_{j=k_1}^{k_2} h(j)^3 \Big). \qquad (6.37)
\end{aligned}$$

The proof of this lemma bases on simple algebraic conversions.

To apply this lemma for the evaluation of (6.36c), we interpret $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1})$ as an expression $\alpha + \beta h(j)$ with $\alpha, \beta \in \mathbb{R}$ and $h(j)$ a function varying in $j \in \mathbb{N}$.

With the help of Theorem 6.6 and Lemma 6.7, we develop the closed formulas for the previously discussed resampling strategies. We discuss them with respect to the case differentiations of Theorem 6.6.

One-value-resampling The integration of a Cluster Kernel with respect to one-value-resampling is given by the following theorem.

Theorem 6.8. *Let $CK_i^{(n)}$ be a Cluster Kernel with $stat_i^{(n)} = \{c_i^{(n)}\}$ and resampled elements $\hat{X}^{(n)}_{i,j}, j = 1, ..., c_i^{(n)}$ as defined in (6.13). For the integration of $CK_i^{(n)}$, the following statement holds*

$$\int_{-\infty}^{t} CK_i^{(n)}(x)dx = \begin{cases} 0, & \frac{1}{\hat{h}^{(n)}}(t - \bar{X}_i^{(n)}) \leq -1 \quad (6.38a) \\ c_i^{(n)}, & \frac{1}{\hat{h}^{(n)}}(t - \bar{X}_i^{(n)}) \geq 1 \quad (6.38b) \\ c_i^{(n)}\big(\frac{1}{4} \cdot (\frac{3}{\hat{h}^{(n)}}(t - \bar{X}_i^{(n)}) - \frac{1}{(\hat{h}^{(n)})^3}(t - \bar{X}_i^{(n)})^3) + \frac{1}{2}\big), & else. \quad (6.38c) \end{cases}$$

Proof. The formula directly results from Theorem 6.6. The indexes required for the third case are given by $k_1 = 1$ and $k_2 = c_i^{(n)}$. □

Min-max-resampling with equal Distances The following theorem provides a closed formula for the integration of Cluster Kernels using min-max-resampling with equal distances.

Theorem 6.9. *Let $CK_i^{(n)}$ be a Cluster Kernel with $stat_i^{(n)} = \{c_i^{(n)}, min_i^{(n)}, max_i^{(n)}\}$ and resampled elements $\hat{X}^{(n)}_{i,j}, j = 1, ..., c_i^{(n)}$ as defined in (6.14). For the integration of $CK_i^{(n)}$, the following case distinctions lead to*

First case: $\frac{1}{\hat{h}^{(n)}}(t - min_i^{(n)}) \leq -1$

$$\int_{-\infty}^{t} CK_i^{(n)}(x)dx = 0. \quad (6.39)$$

Second case: $\frac{1}{\hat{h}^{(n)}}(t - max_i^{(n)}) \geq 1$

$$\int_{-\infty}^{t} CK_i^{(n)}(x)dx = c_i^{(n)}. \quad (6.40)$$

Third case: $\frac{1}{\hat{h}^{(n)}}(t - min_i^{(n)}) > -1$ and $\frac{1}{\hat{h}^{(n)}}(t - max_i^{(n)}) < 1$

With suitably chosen constants $\alpha, \beta \in \mathbb{R}$, the following statement holds

$$\begin{aligned}\int_{-\infty}^{t} CK_i^{(n)}(x)dx &= \frac{1}{4}\Big((k_2-k_1+1)(3\alpha-\alpha^3) \\ &+\frac{3}{2}(1-\alpha^2)\beta(k_2(k_2+1)-k_1(k_1+1)) \\ &-\frac{1}{2}\alpha\beta^2(k_2(k_2+1)(2k_2+1)-(k_1-1)k_1(2k_1-1)) \\ &-\frac{1}{4}\beta^3((k_2(k_2+1))^2-((k_1-1)k_1)^2)\Big). \end{aligned} \tag{6.41}$$

Proof. The first and the second case directly base on Theorem 6.6. Therefore, we concentrate on the third case.

With respect to Theorem 6.6, we determine $k_1, k_2 \in \{1, ..., c_i^{(n)}\}$. The index k_1 is the smallest $j \in \{1, ..., c_i^{(n)}\}$ with $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) \geq -1$. This corresponds to

$$k_1 = \left\lceil c_i^{(n)} - (c_i^{(n)}-1) \cdot \frac{t + \hat{h}^{(n)} - min_i^{(n)}}{max_i^{(n)} - min_i^{(n)}} \right\rceil. \tag{6.42}$$

Analogously, it follows for k_2 as largest $j \in \{1, ..., c_i^{(n)}\}$ with $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1}) \leq 1$

$$k_2 = \left\lfloor c_i^{(n)} - (c_i^{(n)}-1) \cdot \frac{t - \hat{h}^{(n)} - min_i^{(n)}}{max_i^{(n)} - min_i^{(n)}} \right\rfloor. \tag{6.43}$$

To determine the overall formula with Theorem 6.6, we use Lemma 6.7. For that reason, we convert $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1})$ as follows

$$\begin{aligned}\frac{1}{\hat{h}^{(n)}}\Big(t - min_i^{(n)} - (c_i^{(n)}-j)\frac{max_i^{(n)} - min_i^{(n)}}{c_i^{(n)}-1}\Big) &= \underbrace{\frac{1}{\hat{h}^{(n)}}\Big(t - min_i^{(n)} - c_i^{(n)} \cdot \frac{max_i^{(n)} - min_i^{(n)}}{c_i^{(n)}-1}\Big)}_{=:\alpha} \\ &+ j \cdot \underbrace{\frac{max_i^{(n)} - min_i^{(n)}}{\hat{h}^{(n)}(c_i^{(n)}-1)}}_{=:\beta}.\end{aligned}$$

With respect to the lemma, we set $h(j) = j$. The powers of $h(j)$ are given by

$$\begin{aligned}\sum_{j=k_1}^{k_2} j &= \frac{1}{2}\big(k_2(k_2+1)-k_1(k_1+1)\big) \\ \sum_{j=k_1}^{k_2} j^2 &= \frac{1}{6}\big(k_2(k_2+1)(2k_2+1)-(k_1-1)k_1(2k_1-1)\big) \\ \sum_{j=k_1}^{k_2} j^3 &= \frac{1}{4}\big((k_2(k_2+1))^2-((k_1-1)k_1)^2\big).\end{aligned}$$

Combining $\alpha, \beta, k_1, k_2, h(j)$ with Lemma 6.7 delivers (6.41). □

Min-max-resampling with exponential Distances The integration of a Cluster Kernel with respect to min-max-resampling with exponential distances is discussed in the following theorem.

Theorem 6.10. *Let $CK_i^{(n)}$ be a Cluster Kernel with $stat_i^{(n)} = \{c_i^{(n)}, min_i^{(n)}, max_i^{(n)}\}$ and resampled elements $\hat{X}_{i,j,l}^{(n)}, j = 1, ..., c_{i,l}^{(n)}$ and $\hat{X}_{i,j,r}^{(n)}, j = 1, ..., c_{i,r}^{(n)}$ as defined in (6.16) and (6.17) respectively. For the integration of $CK_i^{(n)}$, the following statement holds*

$$\begin{aligned}\int_{-\infty}^{t} CK_i^{(n)}(x)dx &= \sum_{j=1}^{c_i^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\hat{X}_{i,j}^{(n)})} K(x)dx \\ &= \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\bar{X}_i^{(n)})} K(x)dx + \sum_{j=1}^{c_{i,l}^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\hat{X}_{i,j,l}^{(n)})} K(x)dx \\ &\quad + \sum_{j=1}^{c_{i,r}^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\hat{X}_{i,j,r}^{(n)})} K(x)dx. \end{aligned} \tag{6.44}$$

For the first summand, it holds

$$\int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\bar{X}_i^{(n)})} K(x)dx = \begin{cases} 0, \ \frac{1}{\hat{h}^{(n)}}(t-\bar{X}_i^{(n)}) \leq -1 & \text{(6.45a)} \\ 1, \ \frac{1}{\hat{h}^{(n)}}(t-\bar{X}_i^{(n)}) \geq 1 & \text{(6.45b)} \\ \frac{1}{4} \cdot \left(\frac{3}{\hat{h}^{(n)}}(t-\bar{X}_i^{(n)}) - \frac{1}{(\hat{h}^{(n)})^3}(t-\bar{X}_i^{(n)})^3\right) + \frac{1}{2}, \quad else. & \text{(6.45c)} \end{cases}$$

For the second summand, it holds with suitable constants $\alpha, \beta \in \mathbb{R}$:
First case: $\frac{1}{\hat{h}^{(n)}}(t - min_i^{(n)}) \leq -1$

$$\sum_{j=1}^{c_{i,l}^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\hat{X}_{i,j,l}^{(n)})} K(x)dx = 0. \tag{6.46}$$

Second case: $\frac{1}{\hat{h}^{(n)}}(t - \bar{X}_i^{(n)}) \geq 1$

$$\sum_{j=1}^{c_{i,l}^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\hat{X}_{i,j,l}^{(n)})} K(x)dx = c_{i,l}^{(n)}. \tag{6.47}$$

Third case: $\frac{1}{\hat{h}^{(n)}}(t - min_i^{(n)}) > -1$ *and* $\frac{1}{\hat{h}^{(n)}}(t - \bar{X}_i^{(n)}) < 1$

$$\sum_{j=1}^{c_{i,l}^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\hat{X}_{i,j,l}^{(n)})} K(x)dx = \frac{1}{4}\Bigg((k_{2,l}-k_{1,l}+1)(3\alpha-\alpha^3) + 3(1-\alpha^2)\beta(2^{k_{2,l}+1-c_{i,l}^{(n)}} - 2^{k_{1,l}-c_{i,l}^{(n)}}) - \alpha\beta^2(4^{k_{2,l}+1-c_{i,l}^{(n)}} - 4^{k_{1,l}-c_{i,l}^{(n)}}) - \frac{1}{7}\beta^3(8^{k_{2,l}+1-c_{i,l}^{(n)}} - 8^{k_{1,l}-c_{i,l}^{(n)}})\Bigg). \quad (6.48)$$

For the third summand, it holds with suitable constants $\alpha, \beta \in \mathbb{R}$*:*
First case: $\frac{1}{\hat{h}^{(n)}}(t-\bar{X}_i^{(n)}) \leq -1$

$$\sum_{j=1}^{c_{i,r}^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\hat{X}_{i,j,r}^{(n)})} K(x)dx = 0. \quad (6.49)$$

Second case: $\frac{1}{\hat{h}^{(n)}}(t-max_i^{(n)}) \geq 1$

$$\sum_{j=1}^{c_{i,r}^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\hat{X}_{i,j,r}^{(n)})} K(x)dx = c_{i,r}^{(n)}. \quad (6.50)$$

Third case: $\frac{1}{\hat{h}^{(n)}}(t-\bar{X}_i^{(n)}) > -1$ *and* $\frac{1}{\hat{h}^{(n)}}(t-max_i^{(n)}) < 1$

$$\sum_{j=1}^{c_{i,r}^{(n)}} \int_{-\infty}^{\frac{1}{\hat{h}^{(n)}}(t-\hat{X}_{i,j,r}^{(n)})} K(x)dx = \frac{1}{4}\Bigg((k_{2,r}-k_{1,r}+1)(3\alpha-\alpha^3) + 3(1-\alpha^2)\beta\left(\left(\frac{1}{2}\right)^{k_{1,r}-1} - \left(\frac{1}{2}\right)^{k_{2,r}}\right) - \alpha\beta^2\left(\left(\frac{1}{4}\right)^{k_{1,r}-1} - \left(\frac{1}{4}\right)^{k_{2,r}}\right) - \frac{1}{7}\beta^3\left(\left(\frac{1}{8}\right)^{k_{1,r}-1} - \left(\frac{1}{8}\right)^{k_{2,r}}\right)\Bigg). \quad (6.51)$$

Proof. In order to use Theorem 6.6 for the derivation of a closed formula, we interpret each summand as a separate Cluster Kernel with min-max-resampling which is to be integrated. The first summand is a Cluster Kernel with local statistics $stat_i^{(n)} = \{1, \bar{X}_i^{(n)}, \bar{X}_i^{(n)}\}$. The second summand is a Cluster Kernel with $stat_i^{(n)} = \{c_{i,l}^{(n)}, min_i^{(n)}, \bar{X}_i^{(n)}\}$. The third summand is a Cluster Kernel with $stat_i^{(n)} = \{c_{i,r}^{(n)}, \bar{X}_i^{(n)}, max_i^{(n)}\}$.

First summand:
The closed formula results from Theorem 6.8 using $c_i^{(n)} = 1$ as weight.

Second summand:
The first and the second case can be directly derived from Theorem 6.6.

For the third case, we determine indexes $k_{1,l}, k_{2,l} \in \{1, ..., c_{i,l}^{(n)}\}$. The index $k_{1,l}$ is the smallest $j \in \{1, ..., c_{i,l}^{(n)}\}$ with $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_{i,l}^{(n)}-j+1,l}) \geq -1$. This corresponds to

$$k_{1,l} = \left\lceil \frac{1}{ln\,2} ln \frac{\bar{X}_i^{(n)} - t - \hat{h}^{(n)}}{\bar{X}_i^{(n)} - min_i^{(n)}} + c_{i,l}^{(n)} \right\rceil .$$

Analogously, it follows for $k_{2,l}$ as largest $j \in \{1, ..., c_{i,l}^{(n)}\}$ with $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_{i,l}^{(n)}-j+1,l}) \leq 1$

$$k_{2,l} = \left\lfloor \frac{1}{ln\,2} ln \frac{\bar{X}_i^{(n)} - t + \hat{h}^{(n)}}{\bar{X}_i^{(n)} - min_i^{(n)}} + c_{i,l}^{(n)} \right\rfloor .$$

For the conversion of $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_{i,l}^{(n)}-j+1,l})$, we find that

$$\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_{i,l}^{(n)}-j+1,l}) = \underbrace{\frac{1}{\hat{h}^{(n)}}(t - \hat{X}_i^{(n)})}_{=:\alpha} + 2^{j-c_{i,l}^{(n)}} \underbrace{\frac{1}{\hat{h}^{(n)}}(\hat{X}_i^{(n)} - min_i^{(n)})}_{=:\beta} .$$

With respect to Lemma 6.7, we set $h(j) = 2^{j-c_{i,l}^{(n)}}$. For the powers of $h(j)$, this leads to

$$\sum_{j=k_{1,l}}^{k_{2,l}} 2^{j-c_{i,l}^{(n)}} = 2^{k_{2,l}+1-c_{i,l}^{(n)}} - 2^{k_{1,l}-c_{i,l}^{(n)}},$$

$$\sum_{j=k_{1,l}}^{k_{2,l}} 2^{2(j-c_{i,l}^{(n)})} = \frac{1}{3}(4^{k_{2,l}+1-c_{i,l}^{(n)}} - 4^{k_{1,l}-c_{i,l}^{(n)}}),$$

$$\sum_{j=k_{1,l}}^{k_{2,l}} 2^{3(j-c_{i,l}^{(n)})} = \frac{1}{7}(8^{k_{2,l}+1-c_{i,l}^{(n)}} - 8^{k_{1,l}-c_{i,l}^{(n)}}).$$

With Lemma 6.7, Theorem 6.6, α, β, and $k_{1,l}, k_{2,l}$ as defined above, we can determine (6.48).

Third summand:
The first and the second case can again be directly derived from Theorem 6.6.

For the third case, we determine indexes $k_{1,r}, k_{2,r} \in \{1, ..., c_{i,r}^{(n)}\}$. The index $k_{1,r}$ is the smallest $j \in \{1, ..., c_{i,r}^{(n)}\}$ with $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_{i,r}^{(n)}-j+1,r}) \geq -1$. This corresponds to

$$k_{1,r} = \left\lceil 1 - \frac{1}{ln\,2} \cdot ln \frac{t + \hat{h}^{(n)} - \bar{X}_i^{(n)}}{max_i^{(n)} - \bar{X}_i^{(n)}} \right\rceil .$$

Analogously, it follows for $k_{2,r}$ as largest $j \in \{1, ..., c_{i,r}^{(n)}\}$ with $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_{i,r}^{(n)}-j+1,r}) \leq 1$

$$k_{2,r} = \left\lfloor 1 - \frac{1}{ln\,2} \cdot ln \frac{t - \hat{h}^{(n)} - \bar{X}_i^{(n)}}{max_i^{(n)} - \bar{X}_i^{(n)}} \right\rfloor .$$

For the conversion of $\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_i^{(n)}-j+1,r})$, we derive

$$\frac{1}{\hat{h}^{(n)}}(t - \hat{X}^{(n)}_{i,c_{i,r}^{(n)}-j+1,r}) = \underbrace{\frac{1}{\hat{h}^{(n)}}(t - \hat{X}_i^{(n)})}_{=:\alpha} + \Big(\frac{1}{2}\Big)^j \underbrace{\frac{2}{\hat{h}^{(n)}}(\hat{X}_i^{(n)} - max_i^{(n)})}_{=:\beta}.$$

With respect to Lemma 6.7, we set $h(j) = (\frac{1}{2})^j$. For the powers of $h(j)$, this leads to:

$$\sum_{j=k_{1,r}}^{k_{2,r}} \Big(\frac{1}{2}\Big)^j = \Big(\frac{1}{2}\Big)^{k_{1,r}-1} - \Big(\frac{1}{2}\Big)^{k_{2,r}},$$

$$\sum_{j=k_{1,r}}^{k_{2,r}} \Big(\frac{1}{2}\Big)^{2j} = \frac{1}{3}\Big(\Big(\frac{1}{4}\Big)^{k_{1,r}-1} - \Big(\frac{1}{4}\Big)^{k_{2,r}}\Big),$$

$$\sum_{j=k_{1,r}}^{k_{2,r}} \Big(\frac{1}{2}\Big)^{3j} = \frac{1}{7}\Big(\Big(\frac{1}{8}\Big)^{k_{1,r}-1} - \Big(\frac{1}{8}\Big)^{k_{2,r}}\Big).$$

With Lemma 6.7, Theorem 6.6, α, β, and $k_{1,r}, k_{2,r}$ as defined above, we can determine (6.51). □

Mean-var-resampling with equal or exponential Distances The integration of a Cluster Kernel with respect to mean-var-resampling can be easily derived from the previous results for the integration with min-max-resampling. We only have to substitute $min_i^{(n)}$ by $\bar{X}_i^{(n)} - 2\hat{\sigma}_i^{(n)}$ and $max_i^{(n)}$ by $\bar{X}_i^{(n)} + 2\hat{\sigma}_i^{(n)}$.

6.3.7 Summary Measures of Cluster Kernels

After the discussion of the evaluation and integration of Cluster Kernels, let us examine the computation of important summary measures by means of Cluster Kernels. We focus on mean and variance, but also discuss other summary measures of a random variable. As in the previous sections, we discuss the derivation of a closed formula for the corresponding summary measure for each resampling strategy separately.

6.3.7.1 Expectation Value of Cluster Kernels

The expectation value of a random variable is of utmost importance as it describes the average behavior. Because its exact computation relies on the density, which is unknown in our scenario, a natural approach is to plug our Cluster Kernels into the computation. This approach delivers

$$EX \approx \int_{-\infty}^{+\infty} x\hat{f}(x)dx = \frac{1}{n}\sum_{i=1}^{m}\int_{-\infty}^{+\infty} xCK_i^{(n)}(x)dx. \tag{6.52}$$

With respect to this equation, we first compute the expectation value of a single Cluster Kernel. For this computation, we utilize the first two of the following properties of the

Epanechnikov kernel

$$\int_{-\infty}^{+\infty} K(x)dx = 1, \int_{-\infty}^{+\infty} xK(x)dx = 0, \int_{-\infty}^{+\infty} x^2K(x)dx = \frac{1}{5}. \tag{6.53}$$

The expectation value of a single Cluster Kernel is given by

$$\begin{aligned}
\int_{-\infty}^{+\infty} xCK_i^{(n)}(x)dx &= \sum_{j=1}^{c_i^{(n)}} \int_{-\infty}^{+\infty} \frac{x}{\hat{h}^{(n)}} K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right) dx \\
&= \sum_{j=1}^{c_i^{(n)}} \int_{-\infty}^{+\infty} \frac{x}{\hat{h}^{(n)}} K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right) dx - \hat{X}_{i,j}^{(n)} + \hat{X}_{i,j}^{(n)} \\
&= \sum_{j=1}^{c_i^{(n)}} \int_{-\infty}^{+\infty} \frac{x}{\hat{h}^{(n)}} K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right) dx - \hat{X}_{i,j}^{(n)} \int_{-\infty}^{+\infty} \frac{1}{\hat{h}^{(n)}} K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right) dx + \hat{X}_{i,j}^{(n)} \\
&= \sum_{j=1}^{c_i^{(n)}} \int_{-\infty}^{+\infty} \frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right) dx + \hat{X}_{i,j}^{(n)} \\
&= \sum_{j=1}^{c_i^{(n)}} \hat{h}^{(n)} \int_{-\infty}^{+\infty} xK(x)dx + \hat{X}_{i,j}^{(n)} \\
&= \sum_{j=1}^{c_i^{(n)}} \hat{X}_{i,j}^{(n)}.
\end{aligned}$$

We can compute the expectation value via

$$EX \approx \frac{1}{n} \sum_{i=1}^{m} \sum_{j=1}^{c_i^{(n)}} \hat{X}_{i,j}^{(n)}. \tag{6.54}$$

This equation confirms the intuitive notion of the expectation value as average of the underlying values. It serves as starting point for the development of closed formulas for the different resampling strategies.

One-value-resampling Since one-value-resampling generates $c_i^{(n)}$ identical instances of the mean, the corresponding expectation value is simply

$$EX \approx \frac{1}{n} \sum_{i=1}^{m} \sum_{j=1}^{c_i^{(n)}} \hat{X}_{i,j}^{(n)} = \frac{1}{n} \sum_{i=1}^{m} c_i^{(n)} \bar{X}_i. \tag{6.55}$$

Min-max-resampling with equal Distances With the uniform distribution of the resampled elements in $[min_i^{(n)}, max_i^{(n)}]$, it follows for the inner sum in (6.54)

$$\begin{aligned}\sum_{j=1}^{c_i^{(n)}} \hat{X}_{i,j}^{(n)} &= \sum_{j=1}^{c_i^{(n)}} min_i^{(n)} + (j-1)\frac{max_i^{(n)} - min_i^{(n)}}{c_i^{(n)} - 1} \\ &= c_i^{(n)} min_i^{(n)} + \frac{c_i^{(n)}(c_i^{(n)}-1)}{2} \cdot \frac{max_i^{(n)} - min_i^{(n)}}{c_i^{(n)} - 1} \\ &= \frac{c_i^{(n)}}{2}(max_i^{(n)} + min_i^{(n)}).\end{aligned}$$

For the overall sum, it follows

$$EX \approx \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{c_i^{(n)}} \hat{X}_{i,j}^{(n)} = \frac{1}{n}\sum_{i=1}^{m} c_i^{(n)} \frac{max_i^{(n)} + min_i^{(n)}}{2}. \tag{6.56}$$

Min-max-resampling with exponential Distances For the inner sum of the elements resampled with respect to min-max-resampling with exponential distances, it turns out that

$$\begin{aligned}\sum_{j=1}^{c_i^{(n)}} \hat{X}_{i,j}^{(n)} &= \bar{X}_i^{(n)} + \sum_{j=1}^{c_{i,l}^{(n)}} \hat{X}_{i,j,l}^{(n)} + \sum_{j=1}^{c_{i,r}^{(n)}} \hat{X}_{i,j,r}^{(n)} \\ &= \bar{X}_i^{(n)} + \sum_{j=1}^{c_{i,l}^{(n)}} \big(\bar{X}_i^{(n)} + 2^{1-j}(min_i^{(n)} - \bar{X}_i^{(n)})\big) + \sum_{j=1}^{c_{i,r}^{(n)}} \big(\bar{X}_i^{(n)} + 2^{j-c_{i,r}^{(n)}}(max_i^{(n)} - \bar{X}_i^{(n)})\big) \\ &= c_i^{(n)}\bar{X}_i^{(n)} + (2 - 2^{1-c_{i,l}^{(n)}})(min_i^{(n)} - \bar{X}_i^{(n)}) + (2 - 2^{1-c_{i,r}^{(n)}})(max_i^{(n)} - \bar{X}_i^{(n)}) \\ &= (c_i^{(n)} - 4 + 2^{1-c_{i,l}^{(n)}} + 2^{1-c_{i,r}^{(n)}})\bar{X}_i^{(n)} + (2 - 2^{1-c_{i,l}^{(n)}})min_i^{(n)} + (2 - 2^{1-c_{i,r}^{(n)}})max_i^{(n)}.\end{aligned}$$

If we plug this result into (6.54), we receive

$$\begin{aligned}EX &\approx \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{c_i^{(n)}} \hat{X}_{i,j}^{(n)} \\ &= \frac{1}{n}\sum_{i=1}^{m}(c_i^{(n)} - 4 + 2^{1-c_{i,l}^{(n)}} + 2^{1-c_{i,r}^{(n)}})\bar{X}_i^{(n)} + (2 - 2^{1-c_{i,l}^{(n)}})min_i^{(n)} + (2 - 2^{1-c_{i,r}^{(n)}})max_i^{(n)}.\end{aligned}$$

Mean-var-resampling with equal or exponential Distances The formulas are the same as for min-max-resampling, except that $min_i^{(n)}$ is substituted by $\bar{X}_i^{(n)} - 2\hat{\sigma}_i^{(n)}$ and $max_i^{(n)}$ by $\bar{X}_i^{(n)} + 2\hat{\sigma}_i^{(n)}$.

6.3.7.2 Variance of Cluster Kernels

The variance is besides the mean probably the most important characteristic of a random variable as it describes the deviation of a random variable from its mean. However, its

computation also relies on the unknown density. To circumvent this drawback, we utilize Cluster Kernels to approximately compute the variance. Due to the identity $Var(X) = EX^2 - (EX)^2$, we focus on the computation of EX^2 since the previous section already discussed the approximation of EX with Cluster Kernels.

To approximate EX^2, we utilize Cluster Kernels as follows

$$\begin{aligned} EX^2 &\approx \frac{1}{n}\sum_{i=1}^{m}\int_{-\infty}^{+\infty} x^2 CK_i^{(n)}(x)dx \\ &= \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{c_i^{(n)}}\int_{-\infty}^{+\infty}\frac{x^2}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right). \end{aligned} \tag{6.57}$$

For the subsequent computation, we need some elementary identities based on (6.53):

$$\hat{X}_{i,j}^{(n)} = \hat{X}_{i,j}^{(n)}\int_{-\infty}^{+\infty}\frac{1}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx = \int_{-\infty}^{+\infty}\frac{\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx, \tag{6.58}$$

$$\begin{aligned} \int_{-\infty}^{+\infty}\frac{x}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx &= \int_{-\infty}^{+\infty}\frac{x}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx - \int_{-\infty}^{+\infty}\frac{\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx + \hat{X}_{i,j}^{(n)} \\ &= \int_{-\infty}^{+\infty}\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx + \hat{X}_{i,j}^{(n)} \\ &= \hat{X}_{i,j}^{(n)}. \end{aligned} \tag{6.59}$$

With the help of these identities and (6.53), it follows

$$\begin{aligned} \int_{-\infty}^{+\infty}\frac{x^2}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx &= \int_{-\infty}^{+\infty}\frac{x^2}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx + \int_{-\infty}^{+\infty}\frac{-2\hat{X}_{i,j}^{(n)}x}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx \\ &+ 2(\hat{X}_{i,j}^{(n)})^2 + \int_{-\infty}^{+\infty}\frac{(\hat{X}_{i,j}^{(n)})^2}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx - (\hat{X}_{i,j}^{(n)})^2 \\ &= \int_{-\infty}^{+\infty}\frac{(x-\hat{X}_{i,j}^{(n)})^2}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx + (\hat{X}_{i,j}^{(n)})^2 \\ &= (\hat{h}^{(n)})^2\int_{-\infty}^{+\infty}\frac{1}{\hat{h}^{(n)}}\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)^2 K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx + (\hat{X}_{i,j}^{(n)})^2 \\ &= (\hat{h}^{(n)})^2\int_{-\infty}^{+\infty}x^2K(x)dx + (\hat{X}_{i,j}^{(n)})^2 \\ &= (\hat{X}_{i,j}^{(n)})^2 + \frac{(\hat{h}^{(n)})^2}{5}. \end{aligned}$$

For EX^2, it follows with this statement

$$\begin{aligned} EX^2 &\approx \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{c_i^{(n)}}\int_{-\infty}^{+\infty}\frac{x^2}{\hat{h}^{(n)}}K\left(\frac{x-\hat{X}_{i,j}^{(n)}}{\hat{h}^{(n)}}\right)dx \\ &= \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{c_i^{(n)}}(\hat{X}_{i,j}^{(n)})^2+\frac{(\hat{h}^{(n)})^2}{5} \\ &= \frac{(\hat{h}^{(n)})^2}{5}+\frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{c_i^{(n)}}(\hat{X}_{i,j}^{(n)})^2. \end{aligned} \tag{6.60}$$

The combination of this identity with the estimate of $(EX)^2$ delivers an estimate of the variance. To compute the variance for a concrete resampling strategy, we have to determine a closed formula for $\sum_{j=1}^{c_i^{(n)}}(\hat{X}_{i,j}^{(n)})^2$. Let us examine how to derive this formula for our resampling strategies.

One-value-resampling As $\hat{X}_{i,j}^{(n)}=\bar{X}_i^{(n)}, j=1,...,c_i^{(n)}$ for one-value-resampling, it follows for the sum

$$\sum_{j=1}^{c_i^{(n)}}(\hat{X}_{i,j}^{(n)})^2 = c_i^{(n)}(\bar{X}_i^{(n)})^2. \tag{6.61}$$

Min-max-resampling with equal Distances With respect to the definition of the resampled elements in (6.14), the sum for min-max-resampling with equal distances is computed as

$$\begin{aligned} \sum_{j=1}^{c_i^{(n)}}(\hat{X}_{i,j}^{(n)})^2 &= \sum_{j=1}^{c_i^{(n)}}\left(min_i^{(n)}+(j-1)\frac{max_i^{(n)}-min_i^{(n)}}{c_i^{(n)}-1}\right)^2 \\ &= \sum_{j=0}^{c_i^{(n)}-1}\left(min_i^{(n)}+j\frac{max_i^{(n)}-min_i^{(n)}}{c_i^{(n)}-1}\right)^2 \\ &= \sum_{j=0}^{c_i^{(n)}-1}\left(\frac{max_i^{(n)}-min_i^{(n)}}{c_i^{(n)}-1}\right)^2 j^2+2min_i^{(n)}\cdot\frac{max_i^{(n)}-min_i^{(n)}}{c_i^{(n)}-1}j+(min_i^{(n)})^2 \\ &= c_i^{(n)}(min_i^{(n)})^2+\frac{1}{6}\cdot\frac{(max_i^{(n)}-min_i^{(n)})^2}{c_i^{(n)}-1}\cdot c_i^{(n)}\cdot(2c_i^{(n)}-1) \\ &\quad +(max_i^{(n)}-min_i^{(n)})min_i^{(n)}c_i^{(n)}. \end{aligned} \tag{6.62}$$

Min-max-resampling with exponential Distances Since the variant with exponential distances defines its resampled elements separately for the according intervals $[min_i^{(n)},\bar{X}_i^{(n)})$

and $(\bar{X}_i^{(n)}, max_i^{(n)}]$, we consider each sum separately. More precisely, we concentrate on $\sum_{j=1}^{c_{i,l}^{(n)}}(\hat{X}_{i,j,l}^{(n)})^2$ and $\sum_{j=1}^{c_{i,r}^{(n)}}(\hat{X}_{i,j,r}^{(n)})^2$ due to the following identity

$$\sum_{j=1}^{c_i^{(n)}}(\hat{X}_{i,j}^{(n)})^2 = (\bar{X}_i^{(n)})^2 + \sum_{j=1}^{c_{i,l}^{(n)}}(\hat{X}_{i,j,l}^{(n)})^2 + \sum_{j=1}^{c_{i,r}^{(n)}}(\hat{X}_{i,j,r}^{(n)})^2. \tag{6.63}$$

For the left interval, it follows with algebraic conversions

$$\begin{aligned}
\sum_{j=1}^{c_{i,l}^{(n)}}(\hat{X}_{i,j,l}^{(n)})^2 &= \sum_{j=1}^{c_{i,l}^{(n)}}\Big(\bar{X}_i^{(n)} + 2^{1-j}(min_i^{(n)} - \bar{X}_i^{(n)})\Big)^2 \\
&= \sum_{j=1}^{c_{i,l}^{(n)}}(\bar{X}_i^{(n)})^2 + 2^{2-j}\bar{X}_i^{(n)}(min_i^{(n)} - \bar{X}_i^{(n)}) + 4^{1-j}(min_i^{(n)} - \bar{X}_i^{(n)})^2 \\
&= c_{i,l}^{(n)}(\bar{X}_i^{(n)})^2 + \frac{1}{3}(4 - 4^{1-c_{i,l}})(min_i^{(n)} - \bar{X}_i^{(n)})^2 \\
&\quad +(4 - 2^{2-c_{i,l}})\bar{X}_i^{(n)}(min_i^{(n)} - \bar{X}_i^{(n)}).
\end{aligned} \tag{6.64}$$

Analogously, it follows for the right interval

$$\begin{aligned}
\sum_{j=1}^{c_{i,r}^{(n)}}(\hat{X}_{i,j,r}^{(n)})^2 &= \sum_{j=1}^{c_{i,r}^{(n)}}\Big(\bar{X}_i^{(n)} + 2^{j-c_{i,r}^{(n)}}(max_i^{(n)} - \bar{X}_i^{(n)})\Big)^2 \\
&= c_{i,r}^{(n)}(\bar{X}_i^{(n)})^2 + \frac{1}{3}(4 - 4^{1-c_{i,r}^{(n)}})(max_i^{(n)} - \bar{X}_i^{(n)})^2 \\
&\quad +(4 - 2^{2-c_{i,r}^{(n)}})\bar{X}_i^{(n)}(max_i^{(n)} - \bar{X}_i^{(n)}).
\end{aligned} \tag{6.65}$$

Mean-var-resampling with equal or exponential Distances The formulas are the same as for the min-max-resampling variants; we only have to substitute $min_i^{(n)}$ by $\bar{X}_i^{(n)} - 2\hat{\sigma}_i^{(n)}$ and $max_i^{(n)}$ by $\bar{X}_i^{(n)} + 2\hat{\sigma}_i^{(n)}$.

6.3.7.3 Other Summary Measures

Besides for the approximation of mean and variance, we can also utilize Cluster Kernels to determine other important characteristics of a data stream. Let us briefly describe some of them.

Standard Deviation of Cluster Kernels Given the estimated variance, we can easily compute the standard deviation as its square root.

Value Range of Cluster Kernels The range of a data stream refers to minimum and maximum of the already processed stream elements. In case of min-max-resampling, we can determine those values exactly. We examine the local statistics of each Cluster Kernel and determine the corresponding minimum and maximum. This delivers the overall minimum and maximum for all processed elements. For one-value-resampling and mean-var-resampling, we cannot determine minimum and maximum exactly. In case of one-value-resampling, we can use the mean of the left-most Cluster Kernel minus $\hat{h}^{(n)}$ as minimum and the mean of the right-most Cluster Kernel plus $\hat{h}^{(n)}$ as maximum. In case of mean-var-resampling, we iterate over all Cluster Kernels and determine $\bar{X}_i^{(n)} - 2\hat{\sigma}_i^{(n)} - \hat{h}^{(n)}$ and $\bar{X}_i^{(n)} + 2\hat{\sigma}_i^{(n)} + \hat{h}^{(n)}$ as approximate minimum and maximum of each Cluster Kernel. Then we determine their overall minimum and maximum.

Entropy of Cluster Kernels The entropy as an information measure can also be numerically approximated with the help of Cluster Kernels.

$$H(X) = \int_{-\infty}^{+\infty} -x \log f(x) dx \approx \int_{-\infty}^{+\infty} -x \log \frac{1}{n} \sum_{i=1}^{m} CK_i(x) dx. \tag{6.66}$$

By means of the compound Simpson's rule [49], we can approximate the above integral numerically.

Number of Elements in an Interval We can approximate for an arbitrary interval $[a, b]$, $a, b \in \mathbb{R}$ the number of processed stream elements that fell into this interval. We simply have to compute

$$|\{X_i, i = 1, ..., n : X_i \in [a, b]\}| \approx \int_a^b \sum_{i=1}^{m} CK_i(x) dx. \tag{6.67}$$

If we introduce an additional factor $1/n$, we can also exploit Cluster Kernels to determine the selectivity of range queries over data streams.

6.3.8 Implementation and Algorithm Analysis

For practical purposes, we discuss two suitable implementations of univariate Cluster Kernels. Algorithm 3 gives us the main steps for the computation of Cluster Kernels. The implementations we present are aimed at supporting the essential processing steps of this algorithm. While one implementation relies on a sorted list, the other one relies on trees. As we will see, they differ in their update policy for the merge costs. The list-based implementation updates all merge costs continuously and the tree-based one only locally.

6.3.8.1 List-based Implementation

Similar to the implementation of M-Kernels presented in Section 6.2.6, we organize the entirety of Cluster Kernels $CK_i^{(n)} =< \bar{X}_i^{(n)}, stat_i^{(n)}, mergecosts_{i,i+1}^{(n)} >, i = 1, ..., m$ in a sorted list. Note that, in comparison to (6.2), we do not store the bandwidth for each Cluster Kernel separately as we use a single one for all Cluster Kernels, which can be globally stored. The

mean $\bar{X}_i^{(n)}$ serves as underlying ordering criterion for the list. For an illustration of this data structure, see Figure 6.7. With respect to Algorithm 3, this list has to support the insertion of new Cluster Kernels as well as the merge of adjacent Cluster Kernels.

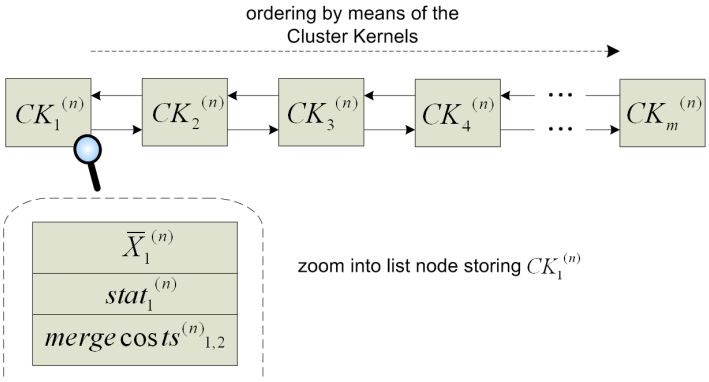


Figure 6.7: List-based implementation of Cluster Kernels

Insertion Step If a new element equals the mean of an existing Cluster Kernel, we update the local statistics as discussed in Section 6.3.2. If not, we insert a new Cluster Kernel in compliance with the ordering by mean.

Since the bandwidth $\hat{h}^{(n)}$ depends on the number of processed elements, a new element also triggers an update of the bandwidth. According to equation (6.18), the bandwidth is part of our loss function, i.e., we have to update the merge costs of all Cluster Kernels, except the last one, for each new element. While performing this update, we can determine the current Cluster Kernel pair with overall minimum merge costs to simplify the merge step.

Merge Step If the number of Cluster Kernels exceeds m, we substitute the adjacent Cluster Kernels with overall minimum merge costs by their merge kernel. As the merge kernel is located between these Cluster Kernels, this substitution does not violate the list ordering. Subsequent to this substitution, we must compute the merge costs between the merge kernel and its left as well as its right neighbor (if existent).

Algorithm Analysis The insertion step has complexity $O(m)$. Even though the ordering allows us to determine the location where to insert the according new Cluster Kernel in $O(\log m)$, the recomputation of all merge costs afterward leads to an overall complexity of $O(m)$. Provided we have already determined the Cluster Kernel pair with overall minimum merge costs, the subsequent merge as well as the update of the affected merge costs has complexity $O(1)$. Overall, the processing of a new element has complexity $O(m)$ in the list-based implementation.

6.3.8.2 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 Cluster Kernels after an update of the bandwidth. We only recompute the merge costs of those Cluster Kernels that are "locally" affected by an insertion or a merge.

Let us examine the requirements for processing a set of m Cluster Kernels. On the one hand, an ordering by mean is desirable as it facilitates the insertion and search of Cluster Kernels. On the other hand, an ordering by merge costs is desirable as it facilitates the detection of the Cluster Kernel pair with minimum merge costs. We unify these contrary requirements within a data structure consisting of two height-balanced binary search trees. We store the Cluster Kernels $CK_i^{(n)} =< \bar{X}_i^{(n)}, stat_i^{(n)}, mergecosts_{i,i+1}^{(n)} >, i = 1, ..., m$ in a binary search tree termed *mean tree* with the mean $\bar{X}_i^{(n)}$ as ordering criterion. Additionally, we maintain a second binary search tree termed *merge costs tree* with entries $\langle mergecosts_{i,i+1}^{(n)}, \bar{X}_i^{(n)} \rangle$ and the merge costs as ordering criterion. The basic idea of this implementation is illustrated in Figure 6.8.

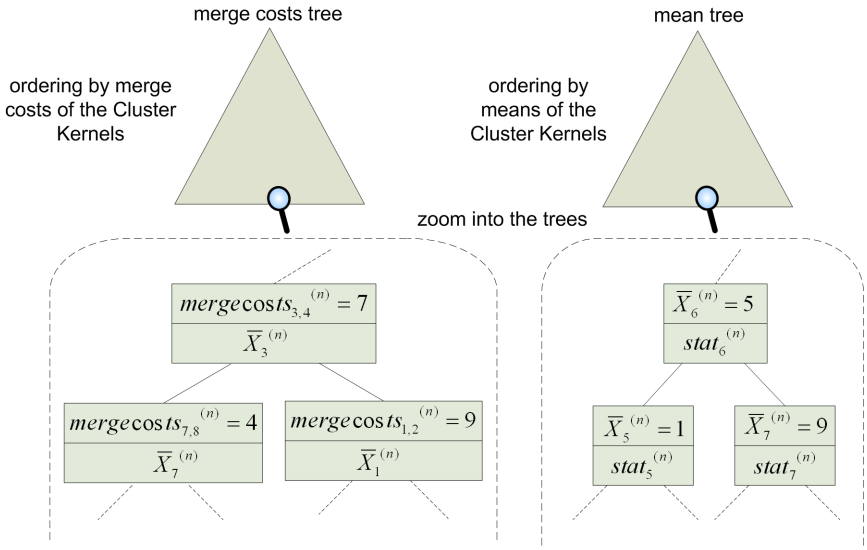


Figure 6.8: Tree-based implementation of Cluster Kernels

Insertion Step A new element implies that either the local statistics of an already inserted Cluster Kernel are updated or a new Cluster Kernel is inserted into the mean tree. In both cases, we recompute the merge costs between the symmetric predecessor (if existent) and the corresponding Cluster Kernel as well as between this Cluster Kernel and its symmetric 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 Cluster Kernels exceeds m after an insertion, we merge the adjacent Cluster Kernels with minimum merge costs. We remove the minimum merge costs from the merge costs tree and determine the associated Cluster Kernel in the mean tree. Recall that the merge costs of a Cluster Kernel refer to the merge costs with its right neighbor. In compliance with the mean tree ordering, we substitute the Cluster Kernel with minimum merge costs by its merge kernel and remove its symmetric successor. Then, we

update the merge costs between the merge kernel and its symmetric predecessor (if existent) as well as between the merge kernel and its new symmetric successor (if existent). While doing so, we keep the merge costs tree consistent by removing and inserting the associated old and new merge costs.

Algorithm Analysis The main advantage of the tree-based implementation is that the insertion step as well as the merge step have complexity $O(\log m)$. Hence, the tree-based implementation processes a new element in $O(\log m)$, compared to $O(m)$ in the case of the list-based implementation.

Comparison of list-based and tree-based Implementation Only kernels locally affected by an insertion or a merge receive an update of their merge costs with respect to the current bandwidth, i.e., the tree-based implementation is approximate. Contrary to, the list-based implementation recomputes all merge costs in case of an updated bandwidth at the expense of higher processing cost. 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, it 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 in Chapter 7 will show that the loss in accuracy by this approximation only has minor effects on the overall quality of the resulting estimators, whereas the processing time substantially improves compared to the list-based implementation.

6.3.8.3 Implementation of Exponential Smoothing

According to (6.7), the Cluster Kernel of a new element receives weight α, while the other Cluster Kernels are rescaled with $(1-\alpha)$. The rescaling requires accessing all Cluster Kernels. Therefore, the complexity of inserting a new element is $O(m)$ whatever implementation is used. In particular, this exacerbates the processing cost of tree-based Cluster Kernels from $O(\log m)$ to $O(m)$.

6.3.8.4 Evaluation and Integration of Cluster Kernels

Let us now examine how to implement the evaluation as well as the integration of Cluster Kernels.

Algorithm 2 describes the essential steps for evaluating a set of Cluster Kernels. For each Cluster Kernel, we resample the partition elements with respect to the current local statistics and a preset resampling strategy. For the presented resampling strategies, we provided in Section 6.3.5 closed formulas for the evaluation of a Cluster Kernel at a given point $x \in \mathbb{R}$. To implement the evaluation as described in Algorithm 2, we iterate over all Cluster Kernels and evaluate the corresponding formula for each Cluster Kernel. The computational complexity of the evaluation is $O(m)$.

For the case of one-value-resampling, it is not necessary to evaluate all Cluster Kernels in case the tree-based implementation is used. This resampling strategy resamples $c_i^{(n)}$ identical instances of the mean $\bar{X}_i^{(n)}$, which delivers (6.24) for the evaluation of a Cluster Kernel. Consequently, as each kernel over a resampled element has support $[\bar{X}_i^{(n)} - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$, the Cluster Kernel itself has the same support. To evaluate all Cluster Kernels at a given

point x, it suffices to consider those whose support covers x, i.e., $x \in [\bar{X}_i^{(n)} - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$. Since the means of all Cluster Kernels are stored in the mean tree, we can determine those Cluster kernels by posing a range query with range $[x - \hat{h}^{(n)}, x + \hat{h}^{(n)}]$.

In case of min-max-resampling and mean-var-resampling with equal or exponential distances, we cannot proceed analogously as they resample different $\hat{X}_{i,j}^{(n)}$ for a Cluster Kernel $CK_i^{(n)}$. The support of the Cluster Kernel is the union of $[\hat{X}_{i,j}^{(n)} - \hat{h}^{(n)}, \hat{X}_{i,j}^{(n)} + \hat{h}^{(n)}], j = 1, ..., c_i^{(n)}$. However, at the expense of more allocated memory, we can also reduce the number of Cluster Kernels to evaluate for the tree-based implementation. We introduce a third data structure which stores the supports of all Cluster Kernels. More precisely, we can use a dynamic priority search tree [126], which offers a dynamical management of a set of intervals. With respect to this data structure, we pose a stabbing query with x to determine all Cluster Kernels whose support covers x.

The integration of Cluster Kernels requires to iterate over all Cluster Kernels as a closer look in Section 6.3.6 reveals. We evaluate for each Cluster Kernel the corresponding formula in dependence on the resampling strategy. Overall, the computational complexity of the integration is $O(m)$.

6.3.9 Implementation in XXL

We incorporated ready-to-use implementations of Cluster Kernels into XXL. In compliance with XXL's development philosophy, we strived for a modular implementation to ensure flexibility, but we also provide concrete implementations of those modules. We primarily discuss the tree-based implementation of Cluster Kernels. The list-based implementation is similarly designed and implemented.

Implementation of Cluster Kernels The functionality of tree-based Cluster Kernels as described in the previous sections is unified in the class `ClusterKernelTreeEstimator`. From an abstract point of view, this class models a KDE based on a set of internally stored Cluster Kernels. To be applicable in an online environment, this class provides an `insert` method for new elements. For each element received, the internal Cluster Kernels are updated as described in the previous sections. If the element is a duplicate, we update the corresponding Cluster Kernel. If not, a new Cluster Kernel is built and inserted into the internal data structures. In case of an exceeded memory, the `merge` method is called, which performs a merge step. Specifically, this method implements the detection of the Cluster Kernel pair with minimum merge costs and their subsequent merge. The `insert` method and the `merge` method modify the two binary search trees each `ClusterKernelTreeEstimator` internally stores. One tree corresponds to the mean tree and the other one to the merge costs tree. Both trees are implemented as AVL trees.

To provide an easy and intuitive implementation of new variants of Cluster Kernels, we parameterized `ClusterKernelTreeEstimator` with a resampling strategy, a merge strategy, and a bandwidth strategy. The abstract class `ResamplingStrategy` models the resampling of partition elements as well as the computation of the associated local statistics. Its abstract methods additionally cover the resampling-dependent functionality of a Cluster Kernel, e.g., evaluation and integration of a Cluster Kernel, computation of mean and variance. The abstract class `MergeStrategy` has a single abstract method, which computes the mean of the

merge kernel and the corresponding merge costs for two Cluster Kernels. The bandwidth strategy is a `Function` that continuously computes the current bandwidth. To establish a new type of Cluster Kernels, one only has to provide implementations of these parameters; the processing logic and the interaction between the parameters is already preimplemented. For the sake of applicability, we provide concrete implementations of these parameters based on the previously discussed results for univariate Cluster Kernels.

The main functionality of `ClusterKernelTreeEstimator` results from implementing the interfaces `RealFunction`, `Integrable`, `SummaryMeasures`, and `MemoryManageable`. The according methods are implemented with respect to the considerations of the previous sections.

Implementation of Cluster Kernels over Data Streams With the already existing functionality of XXL, we can easily derive Cluster Kernels over data streams. Recall from Chapter 4 the following classes of XXL and PIPES: `BufferPipe`, `Aggregator`, `AggregationFunction`. While `BufferPipe` serves as buffer for a data stream and periodically transfers single elements, the `Aggregator` of PIPES uses an internal `AggregationFunction` to compute the Cluster Kernels in an online fashion.

To be more specific, we combine these classes as follows. We connect the `BufferPipe` with an `Aggregator`, which henceforth receives the elements from the buffer. The `Aggregator` internally stores the current estimator, i.e., an instance of `ClusterKernelTreeEstimator`, and updates this estimator for each new element. More precisely, we call with the new element and the current estimator the `invoke` method of the internal aggregation function. Within the `invoke` method, we update the estimator by calling its `insert` method with the new element. If necessary, the `merge` method is subsequently called. Eventually, the updated estimator is returned to the `Aggregator`, where it is stored and transferred afterward. Figure 6.9 illustrates the interplay between the different components.

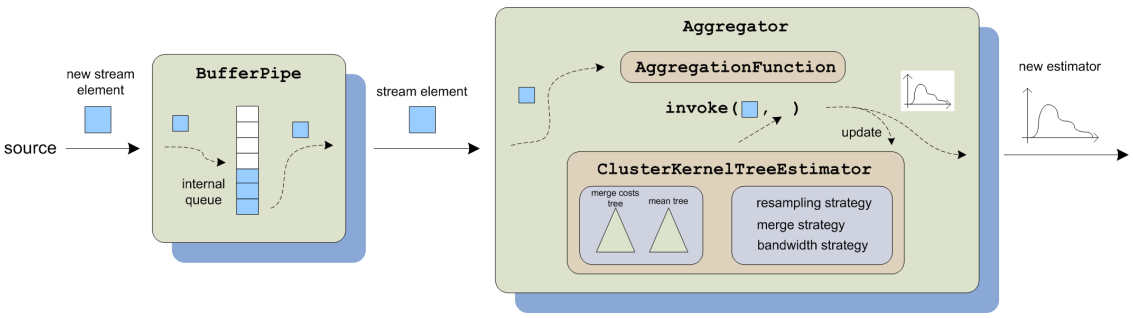


Figure 6.9: Main components of the Cluster Kernel implementation in XXL

Storage Requirements Concerning the storage requirements of this Cluster Kernel implementation during runtime, we focus on the `ClusterKernelTreeEstimator` internally stored in the `Aggregator`. Since the `BufferPipe` transfers single elements, we expect the number of elements in the buffer to be negligible except for situations of very high system load.

The storage requirements of a `ClusterKernelTreeEstimator` are determined by the following components: the bandwidth and the number of processed elements as static parameters and the Cluster Kernels stored in the trees as variable parameters. Important in this

context is that the resampling strategy also determines the storage requirements of a Cluster Kernel due to the local statistics it requires. While one-value-resampling solely stores $c_i^{(n)}$, min-max-resampling additionally stores minimum and maximum and mean-var-resampling mean and variance.

In comparison to the list-based implementation, the tree-based one allocates slightly more memory as it additionally maintains a tree for the merge costs.

With the implementation, we conclude the discussion of Cluster Kernels over univariate streams and turn our attention to extensions of Cluster Kernels.

6.4 Extensions

In this section, we briefly describe reasonable extensions of our Cluster Kernel approach. These extensions refer to other bandwidth strategies and so-called boundary kernels, multivariate Cluster Kernels as well as deletions in the stream. We only sketch these extensions; their elaboration is part of future work in this topic.

6.4.1 Other Bandwidth Strategies and Boundary Kernels

In the course of this work, we concentrated on the use of the normal scale rule for the computation of the bandwidth as it is simple and can be computed in an online fashion. However, as this rule may lead to an oversmoothing of the data, an investigation of other bandwidth strategies is reasonable. Besides the normal scale rule, a plethora of other bandwidth strategies exists; we presented a few of them in Section 3.4.2. However, their computational complexity makes their computation over data streams difficult. An interesting question to be addressed in future work is how these strategies can be suitably modified, typically at the expense of accuracy, so that they comply with the processing requirements for streams?

Another interesting aspect is the incorporation of boundary kernels in order to cope with discontinuities of the density. The KDE as defined in equation (3.4) has problems with discontinuities, indicated by a loss of MISE efficiency. As Section 6.2.3.5 in [128] clarifies, this problem can be circumvented in case the location of the discontinuity is known. More precisely, specific kernels called boundary kernels are used near the discontinuity. With regard to our Cluster Kernels, the question is how to integrate those boundary kernels and, if necessary, how to adapt the general algorithm for Cluster Kernels? In this context, let us emphasize that the use of boundary kernels presupposes the knowledge of the location of the discontinuity. On account of this prerequisite, an algorithm for detecting these discontinuities must be additionally integrated. This aspect has been addressed in [21].

6.4.2 Cluster Kernels over multivariate Data Streams

Up to this point, we have discussed the details of Cluster Kernels over univariate data. The strategies we presented for this case can be generalized, with some restrictions, to the case of multivariate data. Using Algorithm 3 as foundation, we will sketch the general procedure without going into details.

The starting point is the definition of a KDE over multivariate data in (3.23). For the

sake of simplicity, we limit the discussion to the case of one bandwidth h for all dimensions.[1] For the KDE, it follows

$$\hat{f}^{(n)}(x_1, ..., x_d) = \frac{1}{n} \sum_{i=1}^{n} \prod_{k=1}^{d} \frac{1}{h} K\left(\frac{x_k - X_{i,k}}{h}\right), x \in \mathbb{R}^d. \tag{6.68}$$

Analogous to the univariate case, we maintain local statistics for m partitions of the stream and exploit them to resample the partition elements. Then we evaluate the associated Cluster Kernels with respect to these elements. The entirety of Cluster Kernels constitutes the estimator for the already processed stream elements:

$$\begin{aligned} \hat{f}^{(n)}(x_1, ..., x_d) &\approx \frac{1}{n} \sum_{i=1}^{m} \underbrace{\sum_{j=1}^{c_i^{(n)}} \prod_{k=1}^{d} \frac{1}{\hat{h}^{(n)}} K\left(\frac{x_k - \hat{X}_{i,j,k}^{(n)}}{\hat{h}^{(n)}}\right)}_{=:CK_i^{(n)}(x_1,...,x_d)} \\ &= \frac{1}{n} \sum_{i=1}^{m} CK_i^{(n)}(x_1, ..., x_d). \end{aligned} \tag{6.69}$$

Kernel Function and Bandwidth Settings Due to the product form, we can plug in arbitrary kernel functions, in particular the Epanechnikov kernel. Analogous to the univariate case, the problem of the optimal bandwidth can only be overcome with an approximate solution. For practical purposes, we can use a multivariate analogue of the normal scale rule.

Local Statistics and Resampling Strategies The local statistics $stat_i^{(n)}$ of a multivariate Cluster Kernel $CK_i^{(n)}$ can be defined in the same manner as in the univariate case. A counter $c_i^{(n)}$ documents the number of elements in the associated partition. For the local statistics based on minimum and maximum, $min_i^{(n)}$ and $max_i^{(n)}$ are d-dimensional vectors that store minima and maxima for each dimension. The variance-based local statistics store sum and squared sum for each dimension. For both local statistics, updates and merges are defined componentwise.

The local statistics prepare the ground for the resampling of partition elements. The essential requirement for resampling strategies is that a closed formula for the evaluation of a Cluster Kernel over all resampled elements can be provided. The same requirement must be met for their integration and for the computation of summary measures. For univariate data, we developed in Section 6.3.5 different resampling strategies that meet this requirement. While one-value-resampling can be directly adapted to the multivariate case, the adaptation of min-max-resampling or mean-var-resampling is more difficult. For the evaluation of univariate Cluster Kernels, recall from the proofs of Theorem 6.4 and 6.5 that we determined indexes to confine the kernels to evaluate for a given point x. With respect to these indexes, we derived closed formulas for the evaluation of a Cluster Kernel. For the multivariate case, the computation of the corresponding indexes and in particular the derivation of the closed formula are complicated by the product form of a kernel.

[1] The case of a separate bandwidth for each dimension can be similarly examined.

Loss Function In the same way as in (6.18), we can utilize the mean squared deviation for the definition of a loss function for multivariate Cluster Kernels. Note that the corresponding integral is over $\mathbb{R}^d$ and that the variable X is d-dimensional. These facts render the computation of the minimum difficult. However, we can always compute an approximate solution with the help of numerical techniques, but at the expense of additional computational effort and perhaps less accurate values.

Another problem is the number of possible merge pairs for the merge step. In Section 6.1.2, we already discussed that the lack of a total ordering for a dimension $d > 1$ adversely affects the efficient detection of the Cluster Kernel pair with minimum merge costs. More precisely, we can not reduce the possible merge pairs to the adjacent ones as in the one-dimensional case. The interesting question in this context is whether we can exploit at least a partial ordering of the means of the Cluster Kernels to exclude irrelevant merge pairs.

Implementation of multivariate Cluster Kernels The implementations of univariate Cluster Kernels described in Section 6.3.8 can serve as templates for the implementation of multivariate Cluster Kernels. However, the lack of a total ordering by mean exacerbates their direct adaptation as they strongly rely on this ordering. However, we can adapt the general structures.

On the one hand, we can simply organize all Cluster Kernels in a list. On the other hand, we can use a combined data structure based on two trees. The merge costs are organized in a binary search tree and the Cluster Kernels are organized in a multi-dimensional tree. Generally, both implementations have to provide an efficient insertion, merge, and search of Cluster Kernels as well as the detection of the Cluster Kernel pair with minimum merge costs.

6.4.3 Deletions in the Stream

The last extension for Cluster Kernels we discuss refers to deletions in the stream. Formally, let $CK_1^{(n)}, ..., CK_m^{(n)}$ be the current set of Cluster Kernels, from which the estimator for the first n processed elements of the stream can be derived. For the sake of simplicity, we assume that the stream does not contain duplicates. Let $X_l, l \in \{1, ..., n\}$ be the element that shall be deleted. If $n < m$, Cluster Kernels have not been merged yet. To delete X_l in this case, we only have to remove the corresponding Cluster Kernel based on X_l. The case of $n > m$ is more difficult.

To investigate the deletion of X_l from the set of Cluster Kernels for $n > m$, recall Algorithm 3, which describes the online computation of Cluster Kernels. With respect to this algorithm, we see that an exact deletion of X_l is not possible due to dependencies induced by the processing of X_l. More precisely, we process X_l as follows. If it is equal to the mean of an existing Cluster Kernel, we increment the weight of the Cluster Kernel. If not, we build a new Cluster Kernel with mean X_l and weight 1. This Cluster Kernel joins the current set of Cluster Kernels. In case their overall number exceeds m, we merge the two Cluster Kernels with minimum merge costs. According to our definition of a loss function in (6.18), the merge costs depend on the means and the weights of the considered Cluster Kernels. Due to this dependence, X_l influences the processing of subsequent elements $X_k, k > l$. In case X_l would not have been processed, the computation of Cluster Kernels for subsequent elements and particularly their merge would have been different. For that reason, an exact deletion of X_l

is not possible; it would require to store all elements $X_k, k > l$ and to recompute the Cluster Kernels for them.

However, we can at least approximatively delete X_l from the current set of Cluster Kernels. Let us consider equation (6.5), which describes the main components of an online KDE derived from our Cluster Kernels: the scaling factor $1/n$, the bandwidth $\hat{h}^{(n)}$, and the resampled elements. The scaling factor is simple to adjust. For the bandwidth, we use the normal scale rule. With regard to the definition of this rule in equation (6.11), we must correct the factor $n^{-1/5}$ and the variance $\hat{\sigma}^{(n)}$. Our online computation of $\hat{\sigma}^{(n)}$ relies on the algorithm given in [35], which is similar to the online computation of the sample variance in equation (6.12). An important feature of this computation is that we can "subtract" the influence of a contributing element. Hence, we can delete X_l from $\hat{\sigma}^{(n)}$ and correct the bandwidth. With respect to equation (6.5), the remaining question is how to update the local statistics, which are utilized for the resampling of elements? Before we can address this question, we must clarify to which Cluster Kernel X_l should belong to.

In case X_l is equal to the mean of an existing Cluster Kernel, we choose this one. Otherwise, we build a new Cluster Kernel with mean X_l and weight 1. Then we determine its potential merge partner, i.e., the Cluster Kernel which is the closest in terms of minimum merge costs, and choose it as the one X_l should belong to.

In case we have determined the corresponding Cluster Kernel $CK_i^{(n)}$, we have to update its local statistics $stat_i^{(n)}$ by removing X_l properly. In Section 6.3.2, we presented three strategies for maintaining local statistics. One solely bases on the element counter $c_i^{(n)}$, one additionally on minimum $min_i^{(n)}$ and maximum $max_i^{(n)}$, and one additionally on sum $sum_i^{(n)}$ and squared sum $sqrdSum_i^{(n)}$. For all three strategies, we have to decrement their element counter $c_i^{(n)}$. For the local statistics based on minimum and maximum, we do not change $min_i^{(n)}$ and $max_i^{(n)}$. If $X_l \neq min_i^{(n)}$ and $X_l \neq max_i^{(n)}$, this "update" is exact. If $X_l = min_i^{(n)}$ or $X_l = max_i^{(n)}$, this update is inexact as we cannot reconstruct minimum and maximum of the partition elements without X_l. Concerning the local statistics based on sum and squared sum, we subtract X_l from $sum_i^{(n)}$ and $X_l \cdot X_l$ from $sqrdSum_i^{(n)}$, which delivers an exact update.

Overall, the topic of deletions in the stream completes our discussion of Cluster Kernels and their extensions. To convey an impression of their performance, the next chapter presents the results of an extensive experimental study.

Chapter 7

Experimental Evaluation

The previous chapters thoroughly discussed two techniques we developed for estimating the density of a data stream in a continuous fashion: compressed-cumulative WDEs and Cluster Kernels. We scrutinized both techniques as well as competitive ones in an extensive experimental study, whose results we discuss in this chapter. The experiments we carried out specifically address the following questions:

- *Given a small amount of available memory, how accurate is the estimation of the density of a data stream?* We addressed this question with two experiments: one examines the accuracy of compressed-cumulative WDEs (see Section 7.2) and the other one the accuracy of Cluster Kernels (see Section 7.3). In both experiments, we evaluated the accuracy of our density estimates by comparing them with the best possible density estimates.

- *How do compressed-cumulative WDEs and Cluster Kernels perform in comparison to other techniques?* We examined this question against the background of a common use case, namely selectivity estimation of range queries. We compared compressed-cumulative WDEs and Cluster Kernels with a set of competitive techniques that are capable of estimating pdf or cdf of a data stream. Section 7.4 presents the results of this comparison.

- *How time-consuming are construction and evaluation of our density estimators?* In the range query experiment, we additionally monitored the processing time a technique required to build and evaluate an estimator. We compare the measured processing times in Section 7.5.

- *How do compressed-cumulative WDEs and Cluster Kernels react to sudden changes of their available amount of memory?* To answer this question, we randomly varied the available amount of memory for each technique while estimating the density of a data stream. Section 7.6 examines the effects these changes had on the estimation quality.

Before we go into the details of the experiments, let us first discuss their general settings.

Name	*Size*	*Minimum*	*Maximum*	*Mean*	*Variance*
Burstin	50000	0.003	11.27	0.896	0.282
Claw	100000	-3.937	3.94	$6.98 \cdot 10^{-4}$	0.753
CP2	100000	$2.12 \cdot 10^{-6}$	1.0	0.508	0.06
Earthquake	4096	-0.523	0.516	$-5.09 \cdot 10^{-8}$	0.011
EEG Heart Rate	7200	10.76	44.78	28.097	15.712
Fluid Dynamics	10000	-1.0	1.0	-0.007	0.025
Network	18000	66.489	560.083	69.02	58.383
Physiological Data B1	17000	53.67	108.34	74.902	52.909
Physiological Data B2	17000	0.29	107.0	69.341	185.759
Power Data	35040	614.0	2152.0	1144.038	85688.389
Standard and Poor's 500	17610	4.4	456.33	82.959	9489.275
Tide	8746	-37.478	79.432	-0.051	172.816

Table 7.1: Descriptive statistics for the examined data streams

7.1 Experimental Settings

This section describes our settings concerning hardware and software equipment, examined data streams, competing techniques, and accuracy measures.

7.1.1 Hardware and Software Equipment

The experiments were performed on a PC equipped with an Intel Dual Core 3.0 GHz CPU and 3 GB RAM. As operating system, this PC runs Windows XP SP 2. The installed Java edition is 1.5.0. We executed all experiments within Eclipse 3.2 [2], a development platform and application framework particularly suited for Java.

7.1.2 Examined Data Streams

In order to assess the performance and robustness of an analysis technique for data streams, it is crucial to consider heterogeneous streams. For that reason, we conducted our experiments on a dozen different data streams, including synthetic as well as real-world streams. Generally, we concentrated on one-dimensional data streams. For the streams we concretely examined, some descriptive statistics are given in Table 7.1. To convey a visual impression of the streams and their features, Figure 7.1 additionally plots their densities. In case of real-world data streams, where the according densities are per se unknown, we plot the KDEs based on the complete streams instead.

7.1.2.1 Synthetic Data Streams

As synthetic data streams, we used samples drawn from the Claw and the CP2 distribution.

- *Claw:* The Claw density introduced in [113] is strongly multimodal. It is a Gaussian mixture density, i.e., a scaled sum of Gaussian densities with different means and variances.

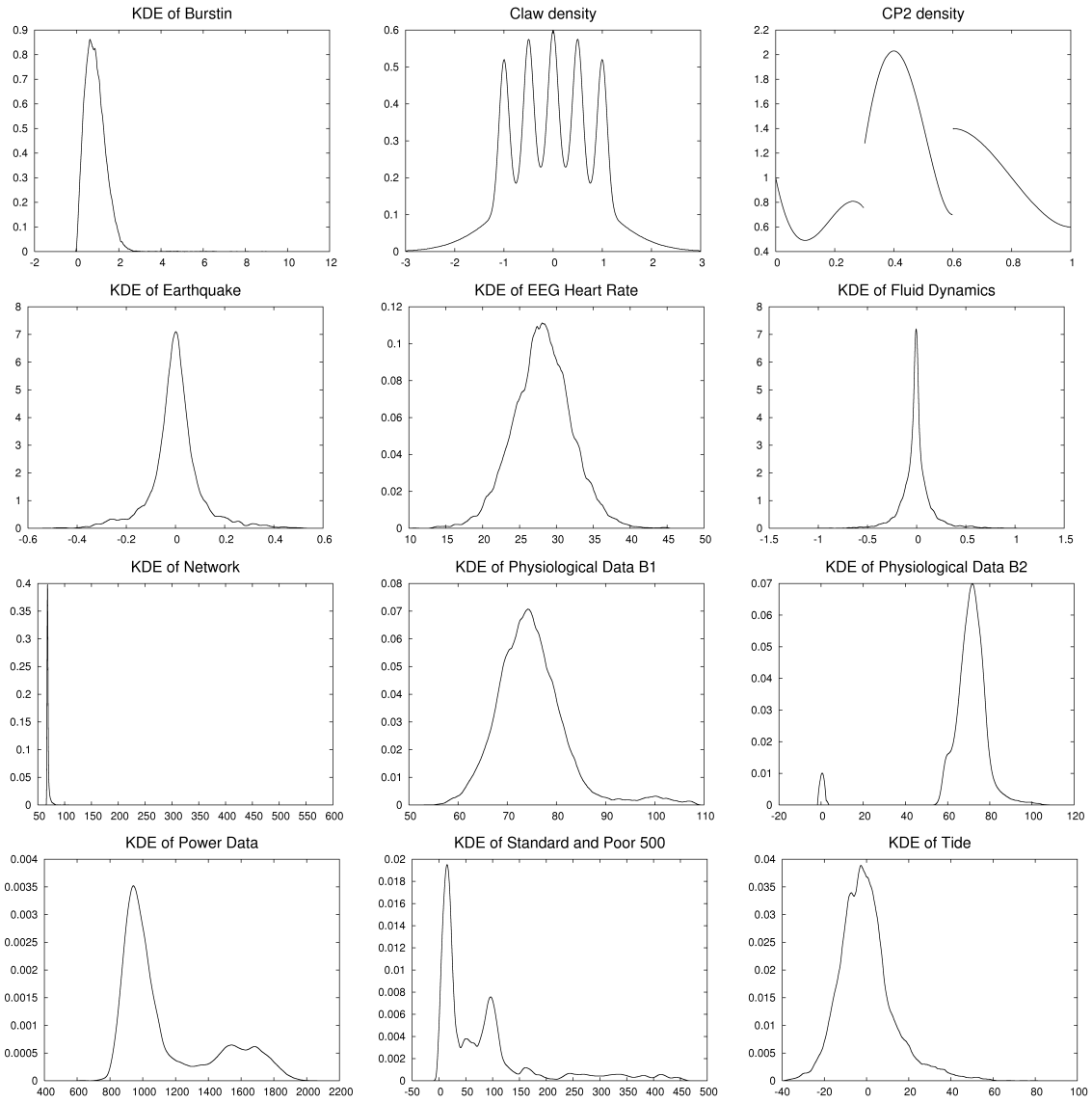


Figure 7.1: Plots of the original or estimated densities of the examined data streams

- *CP2:* The CP2 density introduced in [21] is piecewise smooth, but consists of two abrupt jumps.

For the analytic forms of these densities and a description of how we generated the synthetic streams, we refer to Appendix B.1.

7.1.2.2 Real Data Streams

The UCR Time Series Data Mining Archive [98] provides a collection of time series from diverse application domains like facility monitoring, networking, medicine. These time series exhibit different characteristics, e.g. noisy/smooth, stationary/non-stationary. For our ex-

periments, we chose a subset of those time series and used them as data streams. Since some of the time series we considered are multivariate, we selected one of their variables to obtain a one-dimensional data stream. The following brief description of the examined time series relies on [98] and the external sources quoted in the documentation of the archive.

- *Burstin:* This time series serves as test data set for an algorithm that tackles efficient elastic burst detection in data streams [146].
- *Earthquake:* This time series describes recordings collected from earthquakes and explosions.
- *EEG Hearte Rate:* This bivariate time series represents EEG recordings of a patient and the associated recording times. In our experiments, we examined the EEG recordings.
- *Fluid Dynamics:* This time series consists of measurements of a signal collected from boundary layer experiments in the fluid dynamics research domain.
- *Network:* This time series describes packet Round Trip Time (RTT) delays. While sending packets of size 400 bytes each 20 milliseconds from the University of California to the Carnegie Mellon University, the RTT delay of each packet was measured.
- *Physiological Data B1 and B2:* This is a multivariate time series with measurements of vital signs recorded from a patient in a sleep laboratory. The file has been split into two sequential parts, Physiological Data B1 and Physiological Data B2. The time series consists of three measured variables: heart rate, chest volume, and blood oxygen concentration. We examined the heart rate in our experiments.
- *Power Data:* This time series consists of the 15 minutes averaged values of power demand in a research center in the full year 1997.
- *Standard and Poor's 500:* This bivariate time series gives the dates and the closing values of the Standard and Poor's 500 Index from 1926 to 1993. We examined the closing values.
- *Tide:* This time series is a low-passed version of hourly water level observations recorded at the coast of California.

For these streams, we implicitly have an arrival order for the elements, which is given by the timestamp of their measurement, i.e., by the time they were recorded. On the contrary, the arrival order for the synthetic streams was random.

Some of these time series exhibit a strong temporal locality in the sense that some of their values first appear after a certain period of time. Hence, the first n elements of the stream do not necessarily constitute an *iid* sample of the complete stream. For example, the first time the Standard and Poor's 500 Index closed with a value higher than 200 was around 1986. Other time series showing high temporal locality are Physiological Data B1 and B2 as well as Power Data. An interesting question will be how our techniques cope with these temporal effects. The experimental study of the online density estimators presented in [124] also includes streams exhibiting temporal locality.

7.1.3 Competing Techniques

In order to assess compressed-cumulative WDEs and Cluster Kernels, we compared them with a set of other techniques that are capable of estimating pdf or cdf of a data stream: M-Kernels, sample-based KDEs, sample-based WDEs, sample-based ECDFs, compressed-cumulative KDEs, and approximate quantiles. Before we take a closer look at each technique, let us briefly describe the concepts underlying their implementation as well as the general memory settings.

7.1.3.1 Implementation

With respect to the considerations in Chapter 4, we implemented all techniques as operators in PIPES; this allowed us to design and execute the experiments as operator graphs. For the implementation as an operator, we had to distinguish between techniques that build separate estimators for data blocks and successively merge them and those that process each element separately and update one estimator.

For the techniques processing blocks of data, namely compressed-cumulative WDEs and compressed-cumulative KDEs, we exploited the `BlockBasedMergeAggregator` operator in combination with the `MergeAggregateFunction`; both were presented in Section 5.1.5. The `MergeAggregateFunction` encapsulates the construction of an estimator by specifying an appropriate aggregation function, a weighting function, and a merge and compress function. The aggregation function computes a new estimator for each data block it receives. The weighting function weights current and new estimator, while the merge and compress function merges them and compresses the result afterward.

For the estimators that process each element separately, we coupled an `Aggregator` with specific instances of `AggregationFunction`. The `Aggregator` receives a new element, updates the internally stored current estimator by means of its `AggregationFunction`, and transfers the updated estimator afterward. As discussed in Section 6.3.9, we used this concept to implement Cluster Kernels. For an example of an `AggregationFunction`, we refer to Section 4.5, which presents the corresponding instance for the computation of sample-based KDEs.

7.1.3.2 Memory Settings

To guarantee a fair comparison between the techniques, we set their parameters so that each technique occupied the same amount of memory. Except were otherwise stated, this amount was set to 5 kilobytes. A small amount of available memory allows us to examine whether a technique can still produce reasonable estimates under these circumstances. We also performed experiments with larger memory. Basically, we observed the same tendencies for the techniques, except that they reached higher degrees of accuracy.

To determine the memory requirements of a technique, we analyzed its operator implementation. Each technique internally keeps specific data structures which are continuously updated while processing the stream. By means of these data structures, the estimators are constructed. For the analysis of the memory allocation, we concentrated on the main data structures of a technique, i.e., we did not include temporary variables or structures. Generally, the data structures of each technique store a set of numerical values. For example, an empirical wavelet coefficient in a coefficient set comprises resolution, translation index, and the value itself. For the sake of simplicity, we assume all numerical values to be of type

`double`, a 8-byte floating point primitive in Java. The entirety of these values determines the current memory requirements of a technique.

7.1.3.3 Techniques

We briefly describe the examined techniques, their memory requirements, and the parameter settings that ensure an allocated memory amount of maximum 5 kilobytes.

Cluster Kernels For univariate data streams, we developed in Section 6.3 specific Cluster Kernels which use the Epanechnikov kernel as kernel function, the normal scale rule as bandwidth strategy, and the loss function defined in equation (6.18). We also developed different resampling strategies in Section 6.3.3. To implement Cluster Kernels, we can either use lists or trees according to Section 6.3.8. Overall, we can construct different variants of Cluster Kernels, which differ in the underlying implementation and in the setting of the resampling strategy. For the sake of clarity, we confine the discussion of list-based Cluster Kernels to the variant using one-value-resampling. For tree-based Cluster Kernels, we discuss all resampling strategies.

The storage requirements of Cluster Kernels were examined in Section 6.3.9. Let us list the resulting maximum number m of Cluster Kernels for the different variants. For list-based Cluster Kernels with one-value-resampling, m is 159. For tree-based Cluster Kernels with one-value-resampling, m is 106. For tree-based Cluster Kernels with one of the other resampling strategies, m is 79.

Compressed-cumulative WDEs Section 5.2 presented compressed-cumulative WDEs with respect to the general framework introduced in Section 5.1. Compressed-cumulative WDEs are built by successively merging weighted block WDEs. In our experiments, we built block WDEs based on data blocks of 500 elements and weighted them equally, i.e., we applied the arithmetic weighting strategy. We considered three types of WDEs: linear WDEs and WDEs with soft or hard thresholding. For the thresholded WDEs, we set $0.75 \max_{k \in \mathbb{Z}} |\hat{d}_{j,k}|$ as level-dependent threshold. For linear as well as thresholded WDEs, we used Daubechies5 wavelets as underlying wavelet family since they performed best.

To determine the memory requirements of compressed-cumulative WDEs during runtime, we have to consider the data buffer as well as the current overall WDE. We assume the system not to be saturated. Therefore, the data buffer will contain block size / 2, i.e. 250 elements, on average. This confines the available memory for storing the empirical coefficients of the overall WDE to 5 kilobytes minus 250*8 bytes. Thus, the number of empirical coefficients is limited to 130.

Compressed-cumulative KDEs Another application of the aforementioned framework is the construction of compressed-cumulative KDEs (see also Section 5.1.4). To build the underlying block KDEs, we set the Epanechnikov kernel as kernel function and the normal scale rule as bandwidth strategy throughout the experiments. Again, we used data blocks of 500 elements and weighted the corresponding block KDEs with the arithmetic weighting strategy. For the merge and compress step of compressed-cumulative KDEs, we used cubic Beziér splines in accordance with [21].

Analogous to compressed-cumulative WDEs, compressed-cumulative KDEs distribute their memory among the data buffer and the overall estimator. We assume the data buffer to contain 250 elements on average. As a consequence, the storage of the spline coefficients of the overall estimator must not exceed 5 kilobytes minus 250*8 bytes. Thus, the maximum number of spline coefficients is 390. For details of the spline compression and the corresponding memory requirements, we refer to [21].

Sample-based Estimators Section 3.7 presented a naive approach to compute complex estimators over streaming data. The basic idea is to continuously maintain an *iid* sample of the data stream, which serves as data base for the construction of an estimator. To maintain this sample, we exploited reservoir sampling [139]. In the experiments, we used this approach to build KDEs, WDEs, as well as ECDFs over a data stream.

Concerning the memory distribution, this approach can use the complete amount of available memory to store the sample elements. The resulting sample size is 640.

M-Kernels As described in Section 6.2, M-Kernels continuously compute KDEs over data streams; they fit into our general Cluster Kernel approach. Concerning their parameter settings, they use the Gaussian kernel as kernel function and a bandwidth strategy depending on the loss function defined in equation (6.9). Due to its complicated form, the minimum of the loss function must be numerically determined. The authors propose to use the downhill simplex algorithm [119] for an approximation of the minimum. In each step of this algorithm, the evaluation of the loss function requires the numerical evaluation of an integral. For that reason, we utilize the compound Simpson's Rule [49], which computes the Simpson's rule for the partitions of the partitioned support of the integrand. In the experiments, we set the number of iterations of the simplex algorithm to 30 and the number of partitions for the numerical integration to 50. We also performed experiments with higher numbers of iterations and partitions. This massively increased the processing time, but did not improve the estimation accuracy significantly.

For the implementation of M-Kernels as well as their memory requirements, we refer to Section 6.2. The maximum number m of M-Kernels is restricted to 128.

Approximate Quantiles In [110], the authors presented an algorithm for computing approximate quantiles of large data sets in a single pass. By means of *approximate quantiles*, we can estimate the selectivity of a range query. More specifically, we have to determine the quantiles that match the borders of the range query. Thus, we need the inverses of the quantiles; they can be determined with an interpolation search. We decided to include approximate quantiles in our experiments for the following reasons: First, they are computable in an online manner. Second, their memory footprint is small. Third, their computations during runtime are inexpensive.

Let us sketch the basic idea of approximate quantiles. The algorithm uses b buffers each of which can store k elements. While processing a data sequence in one pass, it probabilistically selects elements and fills them into the buffers. In case one ore more buffers are full, they are lossily compressed into one buffer. With respect to those buffers, the quantiles of the original data sequence can be computed approximately.

For the implementation, we exploited the COLT library [1], which comprises open source Java libraries for scientific and technical computing. A class named `QuantileBin1D` provides an implementation of the approximate quantiles algorithm, including the computation of the inverse of a quantile. By setting the parameters of this class appropriately, we were able to exactly adjust the amount of memory approximate quantiles allocate. The documentation [1] describes these parameters as follows:

- `epsilon`: the maximum allowed approximation error on quantiles
- `delta`: the probability allowed that the approximation error fails to be smaller than epsilon
- `quantiles`: the number of quantiles to be computed
- `knownN`: specifies whether the exact size of the dataset over which quantiles are to be computed is known

The following parameter settings ensure that approximate quantiles occupy exactly 5 kb: `epsilon`=0.06, `delta`=0.25, `quantiles`=`Integer.MAX_VALUE`, `knownN`=`false`.

7.1.4 Accuracy Measures

In the experiments, we evaluated how accurately our techniques estimate the density of a data stream. We also evaluated how accurately they estimate the selectivities of range queries over a data stream. To assess the accuracy, we used the following measures.

7.1.4.1 Density Estimation

For the case of density estimation, we examined the accuracy of compressed-cumulative WDEs and Cluster Kernels in two separate experiments. In both experiments, we had to distinguish between synthetic and real data streams. While we knew the densities of the synthetic streams, we did not know those of the real streams. For the case of a real stream, we computed the best offline density estimates. More precisely, we considered two offline density estimates: $\hat{f}^{(opt,n)}$, the best offline estimate for the first n elements of the data stream, and $\hat{f}^{(opt)}$, the best offline estimate for the complete data stream. For the synthetic streams, we also denote their known densities as $\hat{f}^{(opt)}$ for the sake of simplicity.

For the wavelet experiment, the best density estimate for a real stream refers to the best WDE for each WDE type, i.e., the best linear WDE as well as the best WDE based on soft or hard thresholding. For the kernel experiment, the best density estimate refers to the best KDE using the Epanechnikov kernel and the normal scale rule.

To assess the accuracy of an online density estimator, we compare it with the best offline estimate. Let $\hat{g}^{(n)}$ be an online density estimator for the first n processed elements. We compare $\hat{g}^{(n)}$ with $\hat{f}^{(opt,n)}$ (or $\hat{f}^{(opt)}$) by measuring their difference in terms of the mean squared error (MSE):

$$MSE(n) = \frac{1}{500} \sum_{i=1}^{500} \left(\hat{f}^{(opt,n)}(x_i) - \hat{g}^{(n)}(x_i) \right)^2 \tag{7.1}$$

with $x_1, ..., x_{500}$ an equidistant partition of the support of $\hat{f}^{(opt,n)}$. The MSE with respect to $\hat{f}^{(opt)}$ is defined analogously. Let us emphasize that the MSE measures how close an online computed density estimate is to its offline counterpart. Except for the synthetic streams, where $\hat{f}^{(opt)}$ is the original density, the MSE only indirectly measures how close the online estimate is to the real density of the considered stream.

By evaluating the MSE with $\hat{f}^{(opt)}$ or $\hat{f}^{(opt,n)}$, we can examine different aspects:

Computing the MSE with $\hat{f}^{(opt)}$ allows us to assess the "convergence" of an online density estimator. The online density estimator relies on the already processed elements and only has a limited amount of memory at its disposal. In contrast to the online estimator, $\hat{f}^{(opt)}$ relies on the complete stream and has an unlimited amount of memory at its disposal. By evaluating the MSE for an increasing number of processed elements, we can determine whether the online density estimator converges to $\hat{f}^{(opt)}$ and if so, how fast. However, if the first n elements of the stream do not constitute an *iid* sample of the complete stream, which is the case for some of our real streams, the MSE will per se be high in the beginning.

Computing the MSE with $\hat{f}^{(opt,n)}$ allows us to assess the impact of the limited amount of available memory on the accuracy. The online density estimator and $\hat{f}^{(opt,n)}$ both rely on the first n processed elements. They only differ in the amount of available memory, which is limited for the online estimator and unlimited for $\hat{f}^{(opt,n)}$. The more elements are processed, the more their memory allocations deviate. The question that arises is how this deviation affects the estimation quality in terms of the MSE.

7.1.4.2 Range Query Selectivity Estimation

In another experiment, we estimated the selectivity of range queries over data streams by means of different techniques. To assess the estimation accuracy of the techniques, we examined the relative error as follows. For a given set of k range queries, let $\{sel_{1,n}^{real}, ..., sel_{k,n}^{real}\}$ and $\{sel_{1,n}^{est}, ..., sel_{k,n}^{est}\}$ be the real and the estimated selectivities of the range queries after n elements have been processed. With respect to these selectivities, we define the mean relative error (MRE):

$$MRE(n) = \frac{1}{k}\sum_{i=1}^{k} err_{i,n} \text{ with } err_{i,n} = \begin{cases} |sel_{i,n}^{est}|, & sel_{i,n}^{real} = 0 \\ \frac{|sel_{i,n}^{real} - sel_{i,n}^{est}|}{sel_{i,n}^{real}}, & \text{else.} \end{cases} \tag{7.2}$$

We adopted the MRE as it is an objective measure and allows us to compare different techniques with each other. In the context of range query selectivity estimation, it is commonly used, e.g., [79], [100].

7.2 Density Estimation with compressed-cumulative WDEs

In the first set of experiments, we examined how compressed-cumulative WDEs capture the densities of the synthetic and the real streams, given only 5 kb of available memory. Besides compressed-cumulative WDEs, we also examined sample-based WDEs. For the sake of clarity, we only examined sample-based linear WDEs. Each technique is associated with a specific line type in the charts; Figure 7.2 shows the assignments.

Within the experiments, we used the MSE as quality measure. While processing a stream, we computed the MSE each 500 processed elements. As previously described, we can address

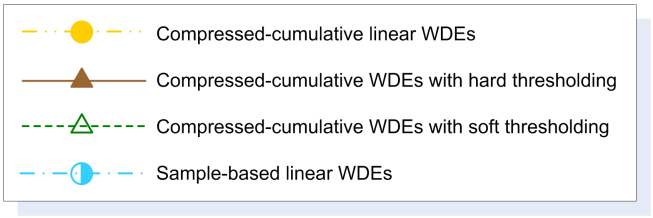


Figure 7.2: Line types of compressed-cumulative and sample-based WDEs

Technique	*Minimum*	*Maximum*	*Mean*	*Variance*
Compressed-cumulative WDEs hard thresholding	$1.7 \cdot 10^{-5}$	0.05	0.007	$2.1 \cdot 10^{-4}$
Compressed-cumulative WDEs soft thresholding	$1.1 \cdot 10^{-5}$	0.024	0.003	$4.9 \cdot 10^{-5}$
Compressed-cumulative linear WDEs	$2.8 \cdot 10^{-8}$	0.105	0.011	$9.1 \cdot 10^{-4}$
Sample-based linear WDEs	$6.9 \cdot 10^{-8}$	0.171	0.021	0.003

Table 7.2: Descriptive statistics for the average MSEs of compressed-cumulative WDEs w.r.t. $\hat{f}^{(opt,n)}$

different aspects by evaluating the MSE with respect to $\hat{f}^{(opt,n)}$ or $\hat{f}^{(opt)}$: the impact of the limited memory on the accuracy on the one hand and the convergence properties on the other hand.

To evaluate the performance of compressed-cumulative WDEs, we examined the MSE in two ways. One is the visual examination of the development of the MSE for an increasing number of elements. The second one is the examination of summary statistics of the MSE. These summary statistics comprise minimum, maximum, mean, and variance of the average MSEs. More precisely, we averaged the MSE values of each data stream. This delivered a set of average MSEs, one for each stream. Then, we computed for this set of average MSEs the corresponding minimum, maximum, mean, and variance. This gives us a global measure for all streams.

7.2.1 Impact of Limited Memory on Accuracy

We first examine which impact the limited memory had on the accuracy. Figure 7.3 depicts the MSEs compressed-cumulative WDEs and sample-based linear WDEs produced for the different streams. In the charts, the x-axis displays the number of processed elements and the y-axis, which is logarithmically scaled, the corresponding MSE. Table 7.2 provides the summary statistics of the average MSEs. An analysis of the charts and the summary statistics reveals the following trends:

Linear vs. thresholded compressed-cumulative WDEs The main parameter that distinguishes the different variants of compressed-cumulative WDEs from each other is the type of WDE they use to build the block estimates. The linear and the thresholded variants exhibit a relatively stable behavior without abrupt jumps or oscillations. For the majority of cases, compressed-cumulative linear WDEs were superior to their thresholded counterparts

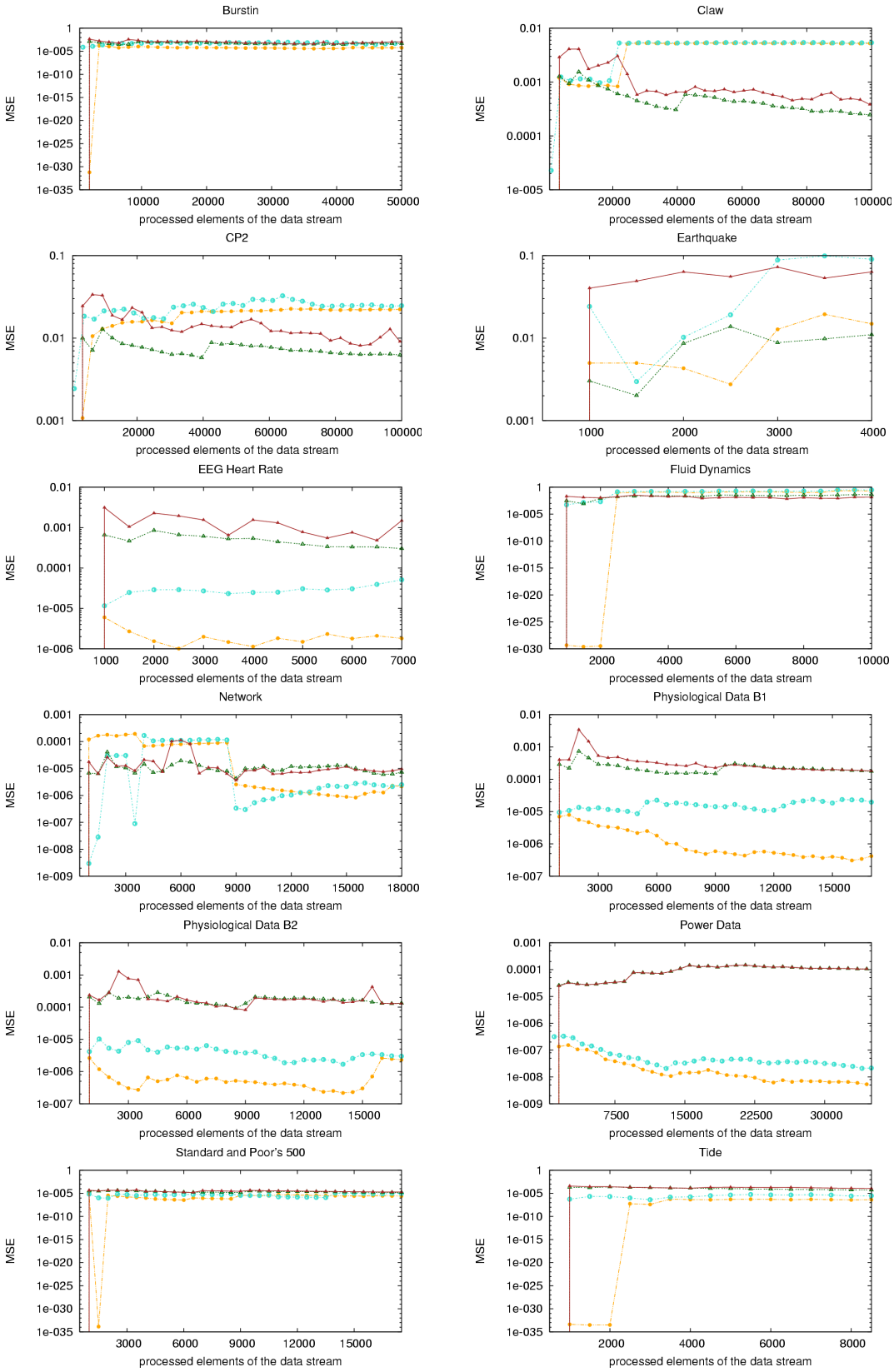


Figure 7.3: Results of compressed-cumulative WDEs with MSE based on $\hat{f}^{(opt,n)}$

in terms of a lower MSE. The reason for their superiority is that compressed-cumulative thresholded WDEs are more sensitive to the available memory than the linear ones. Recall from Section 3.5.2 that thresholded WDEs typically produce far more empirical coefficients than linear WDEs due to higher initial resolutions. Therefore, the limited memory has a higher impact on the estimation accuracy as a larger fraction of coefficients has to be discarded.

The two thresholding-based variants themselves performed similar for most streams. For Claw, CP2, and Earthquake, the soft thresholding variant was superior to the hard thresholding one.

Accuracy of sample-based linear WDEs For all but one data stream, namely Power Data, sample-based linear WDEs produced a slightly increasing MSE. We expected this behavior since the quality of a sample-based estimator depends on the ratio between sample size and size of the data set; the better this ratio, the better the estimation quality. In this experiment, the sample size remained constant while the size of the stream continuously increased. As a consequence, the estimation accuracy deteriorated the more elements were processed.

Generally, sample-based linear WDEs performed comparable to compressed-cumulative linear WDEs for the majority of streams, except that their MSE was typically higher.

Comparison of Accuracy Let us now compare the accuracies of the different techniques by evaluating the summary statistics in Table 7.2.

According to the summary statistics, compressed-cumulative WDEs with soft thresholding performed best, followed by compressed-cumulative WDEs with hard thresholding, compressed-cumulative linear WDEs, and sample-based linear WDEs. This result is surprising at first sight as it contradicts with the visual results given in Figure 7.3. According to this figure, compressed-cumulative thresholded WDEs were in the majority of cases inferior to compressed-cumulative linear WDEs. As reason for the inferiority of compressed-cumulative linear WDEs in terms of the average MSEs, we identified their poor performance for the Fluid Dynamics and the Earthquake stream. The average MSE they produced for these streams dominated the summary statistics. The same effect occurred for sample-based linear WDEs.

The summary statistics also show that the variance was very low for all compressed-cumulative WDE variants. This indicates that they are very robust and achieve a certain estimation accuracy relatively independent of the concrete stream.

Concerning the development of the MSE, we observe some general tendencies in Figure 7.3. In the beginning, all compressed-cumulative WDE variants produce a low MSE, in a few cases even extremely low. Then it starts increasing, but mostly approaches on average to a constant. If it continues increasing, then only slightly. In a few cases, compressed-cumulative linear WDEs even produce a decreasing MSE.

With respect to the question posed at the beginning, we can conclude that even a limited amount of memory suffices for compressed-cumulative WDEs to keep pace with the data stream.

Technique	*Minimum*	*Maximum*	*Mean*	*Variance*
Compressed-cumulative WDEs hard thresholding	$3.3 \cdot 10^{-5}$	0.226	0.023	0.004
Compressed-cumulative WDEs soft thresholding	$3.1 \cdot 10^{-5}$	0.137	0.017	0.002
Compressed-cumulative linear WDEs	$1.4 \cdot 10^{-7}$	0.251	0.034	0.006
Sample-based linear WDEs	$1.1 \cdot 10^{-7}$	0.309	0.045	0.01

Table 7.3: Descriptive statistics for the average MSEs of compressed-cumulative WDEs w.r.t. $\hat{f}^{(opt)}$

7.2.2 Convergence of compressed-cumulative WDEs

Let us now assess the convergence of compressed-cumulative WDEs and sample-based linear WDEs. For the different streams, we evaluated the MSE with respect to $\hat{f}^{(opt)}$. Figure 7.4 and Table 7.3 summarize the results of this experiment.

Linear vs. thresholded compressed-cumulative WDEs As in the previous experiment, compressed-cumulative linear and thresholded WDEs behave different. Typically, compressed-cumulative linear WDEs were superior in terms of a lower MSE. The two variants based on thresholding performed almost identical for most streams. An interesting fact is that compressed-cumulative thresholded WDEs performed best for both synthetic streams, i.e. Claw and CP2. For all variants, we observe a smooth development of the MSE.

Accuracy of sample-based linear WDEs For 6 out of 12 streams, sample-based linear WDEs performed similar to compressed-cumulative linear WDEs. More precisely, their MSE developed in a similar manner, but its absolute values were typically much higher. For most streams, including the synthetic ones, the MSE of sample-based WDEs was more or less constant. An MSE being constant complies with our expectations. We compare the linear WDE over a constant size sample with the linear WDE over the complete stream. Provided that the current sample elements are representative for the complete stream, sample-based linear WDEs should achieve more or less the same quality.

However, the MSE also decreased clearly for some streams. As a decreasing MSE is surprising at first sight, let us take a closer look at this phenomenon. To compute a sample while processing the stream, we exploit reservoir sampling. Reservoir sampling guarantees to deliver an *iid* sample of the already processed elements. In case the first n stream elements are not an *iid* sample of the complete stream, the samples computed with reservoir sampling will also not be *iid*. We observe this effect for the streams exhibiting temporal localities: Physiological Data B1 and B2, Power Data, Standard and Poor's 500. Some of their features first appear after a certain time period. As the sample cannot contain these features before, the MSE will be high in the beginning since the comparison relies on the best estimate over the complete stream, which incorporates these features.

Comparison of Accuracy We compare the performance of the different techniques by examining the development of the MSE as well as the summary statistics of the average

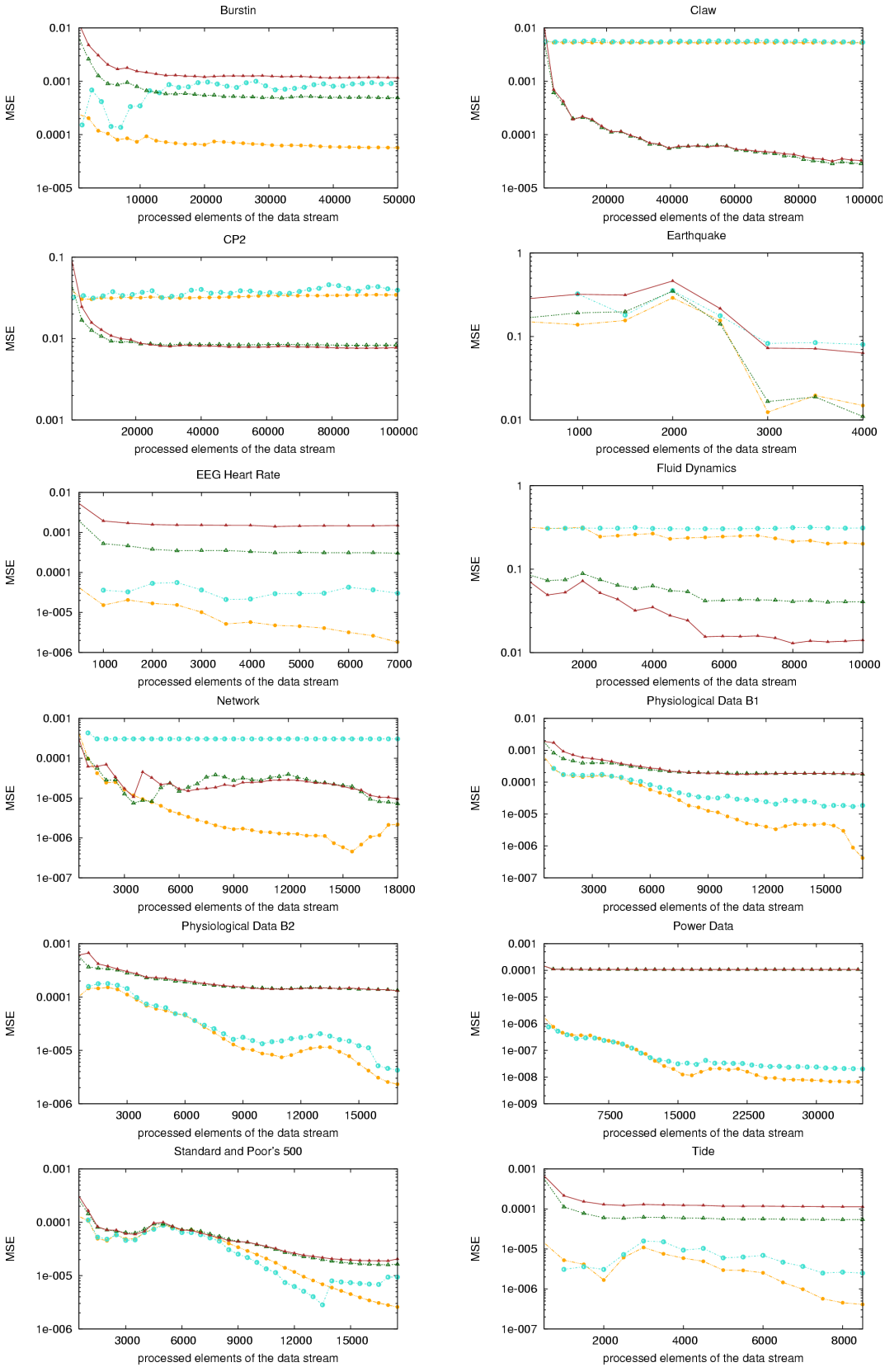


Figure 7.4: Results of compressed-cumulative WDEs with MSE based on $\hat{f}^{(opt)}$

MSEs.

According to Table 7.3, compressed-cumulative WDEs with soft thresholding were superior to the other techniques. Compressed-cumulative WDEs with hard thresholding performed second best, followed by compressed-cumulative linear WDEs, and sample-based linear WDEs. Again, these results do not reflect the results from visually comparing the MSE, which would mark compressed-cumulative linear WDEs as the best technique. The reason is as in the previous experiment the fact that compressed-cumulative linear WDEs produced a high average MSE for the Fluid Dynamics and the Earthquake stream.

As for sample-based WDEs, the temporal locality of some streams also affected the estimation accuracy of compressed-cumulative WDEs; their MSE was per se high in the beginning.

Concerning the general development of the MSE, we observe that all compressed-cumulative WDE variants exhibit a decreasing MSE for an increasing number of processed elements. In a few cases, however, the MSE decreased very slowly, e.g. compressed-cumulative linear WDEs for the Claw stream.

Overall, we can state that compressed-cumulative WDEs converged in terms of a decreasing MSE for most streams.

7.2.3 Approximation Properties of compressed-cumulative WDEs

In the latter two experiments, we evaluated the accuracy of compressed-cumulative WDEs in terms of the MSE. The MSE averages the squared differences between current compressed-cumulative WDE and best offline WDE at equidistant grid points. In order to gain deeper insights into the behavior of the MSE, we also took a closer look at its underlying differences. This gave us valuable information about the problematic parts of the offline WDE, i.e., those that were difficult to estimate. Additionally, we visually examined the resulting WDEs to get an idea of their general shape and their steadiness. Based on these informations, we extracted some general tendencies concerning the approximation properties of compressed-cumulative WDEs.

Compressed-cumulative linear WDEs Compressed-cumulative linear WDEs estimated smooth and steady regions at a very high level of accuracy. However, they had problems with estimating unsteady and spiky regions. For the examined streams, those regions typically occurred at the tails or in the vicinity of a global peak. Compressed-cumulative linear WDEs mostly oversmoothed these regions, i.e., they did not resolve all local details.

The missing ability to resolve local details may have two reasons. First, the empirical wavelet coefficients that describe the details were computed, but dropped due to the limited amount of memory. We took a closer look at the coefficient numbers during runtime and observed that for all but two streams, namely Network and Standard and Poor's 500, the overall number of empirical coefficients was significantly lower than 130, the maximum allowed number; it typically was between 40 and 60. For that reason, the memory restrictions can not be primarily responsible. Second, the initial resolutions of the block WDEs were too low to reach the required level of detail. More specifically, the size of the underlying data blocks was too low. We verified this statement experimentally by keeping the other parameters fixed and increasing the block size. The resulting estimates showed a better resolution of local details. Therefore, the size of the underlying blocks is crucial for the accuracy of compressed-cumulative linear WDEs.

Compressed-cumulative thresholded WDEs For compressed-cumulative thresholded WDEs, we observed a higher degree of spatial adaptivity. Generally, thresholded WDEs are superior in representing local details and peaks in comparison to compressed-cumulative linear WDEs. However, for many streams, the resulting compressed-cumulative WDEs were extremely wiggly.

To get deeper insights, we additionally examined the thresholded WDEs over the complete streams, i.e. $\hat{f}^{(opt)}$, and observed the same effects. We investigated the cause of this behavior and extracted an interesting general fact for thresholded WDEs. By examining the best thresholded WDEs for all streams, we detected that wiggly estimates occurred for all streams with a large value range. For those streams, the scaling functions at j_0^{thr} were too "local", i.e., their support was too small, which implied spiky local features. As described in Section 3.5.2.2, thresholded WDEs essentially depend on the resolutions j_0^{thr}, j_1^{thr} and the level-dependent threshold λ_j. According to equation (3.50), the resolutions solely depend on the sample size. Hence, the setting of the resolution should also incorporate a measure for the spread of the stream like the variance or the value range. The higher the spread, the lower the resolution j_0^{thr} should be set. To exclude the sample size as another reason for these effects, we performed additional experiments with the synthetic streams, which both have a small value range. We generated samples with 1 million elements and examined the resulting thresholded WDEs, which were not wiggly. We conclude that the development of appropriate resolution settings which incorporate the spread of the stream is an important topic for future work.

We also examined the number of empirical coefficients of compressed-cumulative thresholded WDEs. Except for the Earthquake stream, compressed-cumulative thresholded WDEs used the maximum number of 130 coefficients for all streams. For test purposes, we set the maximum number to infinity while keeping the block size constant. We observed that far more empirical coefficients were computed, e.g. 3386 for Tide, accompanied by a significant increase of the estimation quality. In contrast to this effect, the effect of larger block sizes on the estimation quality was minor.

Generally, the impact of the block size and the maximum coefficient number on the estimation quality of compressed-cumulative linear and thresholded WDEs is in accordance with the theoretical expectations. As already discussed in Section 3.5, the spatial adaptivity of linear WDEs essentially depends on their initial resolution. Thresholded WDEs start at a high initial resolution, but drop some of the details at this resolution by means of the thresholding procedure performed in lower resolutions.

To convey a visual impression of the discussed effects, Figure 7.5 presents an illustrative example for the Earthquake stream. The left chart shows the best linear WDE over the complete stream and the compressed-cumulative linear WDE after processing the complete stream. The right chart and the one in the middle show the best thresholded WDE for the complete stream and the corresponding compressed-cumulative WDEs after processing the complete stream. Note that the estimates are negative at the tails. We observed this effect also for a few other streams.

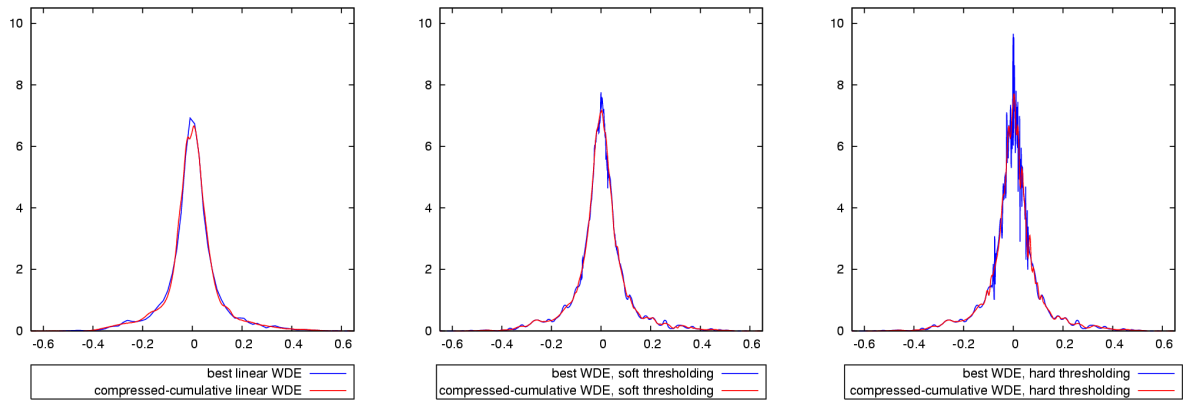


Figure 7.5: Best WDEs and compressed-cumulative WDEs for Earthquake stream

7.3 Density Estimation with Cluster Kernels

In another set of experiments, we examined how accurately Cluster Kernels estimate the density for the different data streams. As competitive techniques, we additionally examined sample-based KDEs and M-Kernels. Figure 7.6 displays the line types of the different techniques used in the charts.

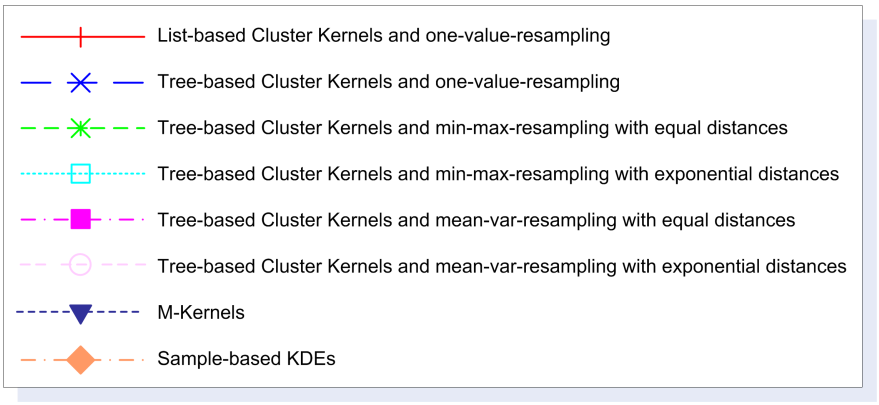


Figure 7.6: Line types of the Cluster Kernel variants, sample-based KDEs, and M-Kernels

With these techniques, we continuously computed density estimates and compared them with the best offline KDEs. More precisely, we compared them in terms of the MSE, which we computed each 500 processed elements. Analogous to the compressed-cumulative WDE experiment, we evaluated the MSE in one experiment with respect to $\hat{f}^{(opt,n)}$ and in a second one with respect to $\hat{f}^{(opt)}$. This allowed us to evaluate the impact of the limited memory on the accuracy of Cluster Kernels as well as their convergence.

We evaluated the performance of a technique by examining the development of the MSE on the one hand, and, on the other hand, by examining summary statistics of the MSE over all streams. Again, we averaged the MSE values for each stream. For the resulting average MSEs for all streams, we computed the extrema as well as mean and variance.

Technique	*Minimum*	*Maximum*	*Mean*	*Variance*
List-based Cluster Kernels one-value-resampling	$8.8 \cdot 10^{-12}$	$3.9 \cdot 10^{-5}$	$6.4 \cdot 10^{-6}$	$1.5 \cdot 10^{-10}$
Tree-based Cluster Kernels one-value-resampling	$3.4 \cdot 10^{-10}$	$2.2 \cdot 10^{-4}$	$3.5 \cdot 10^{-5}$	$4.3 \cdot 10^{-9}$
Tree-based Cluster Kernels mean-var-equal-dist-resampling	$4.2 \cdot 10^{-11}$	$3.9 \cdot 10^{-4}$	$5.3 \cdot 10^{-5}$	$1.4 \cdot 10^{-8}$
Tree-based Cluster Kernels mean-var-exp-dist-resampling	$2.6 \cdot 10^{-10}$	$6.5 \cdot 10^{-4}$	$9.8 \cdot 10^{-5}$	$3.97 \cdot 10^{-8}$
Tree-based Cluster Kernels min-max-equal-dist-resampling	$1.9 \cdot 10^{-9}$	$5.9 \cdot 10^{-4}$	$8.4 \cdot 10^{-5}$	$3.1 \cdot 10^{-8}$
Tree-based Cluster Kernels min-max-exp-dist-resampling	$2.6 \cdot 10^{-10}$	$3.7 \cdot 10^{-4}$	$7.3 \cdot 10^{-5}$	$1.7 \cdot 10^{-8}$
M-Kernels	$1.6 \cdot 10^{-6}$	2.693	0.362	0.652
Sample-based KDEs	$9.2 \cdot 10^{-9}$	0.021	0.0058	$6.5 \cdot 10^{-5}$

Table 7.4: Descriptive statistics for the average MSEs of Cluster Kernels w.r.t. $\hat{f}^{(opt,n)}$

7.3.1 Impact of Limited Memory on Accuracy

We first discuss the impact of the limited memory on the accuracy. Figure 7.7 depicts the results of this experiment. The x-axis displays the number of processed elements and the y-axis, which is logarithmically scaled, the MSE. Table 7.4 gives the summary statistics of the average MSEs.

Impact of Resampling Strategy on Accuracy One essential parameter of Cluster Kernels is the resampling strategy. For tree-based Cluster Kernels, we examined all resampling strategies presented in Section 6.3.3.

As the results in Figure 7.7 and Table 7.4 indicate, the corresponding Cluster Kernel variants mostly exhibited a comparable behavior. For the majority of streams, the MSE of the different variants developed in the same manner, reaching comparable levels of accuracy.

In particular, Cluster Kernels based on min-max-resampling with exponential distances and those based on mean-var-resampling with exponential distances performed almost identically for all streams, except for the Earthquake stream. Even in this case, the differences were only small.

On the contrary, the two variants based on the resampling strategies with equal distances exhibited a more different behavior. The development and shape of their MSEs were comparable, but the mean-var-resampling variant typically produced a lower MSE. For Physiological Data B2 and CP2, their MSEs were almost identical.

List-based vs. tree-based Cluster Kernels The second parameter of Cluster Kernels is their implementation, which can be either founded on lists or trees. The fundamental difference between list-based and tree-based Cluster Kernels is their update policy for the merge costs. List-based Cluster Kernels are always up-to-date with respect to the current bandwidth at the expense of higher processing costs. Tree-based Cluster Kernels update the

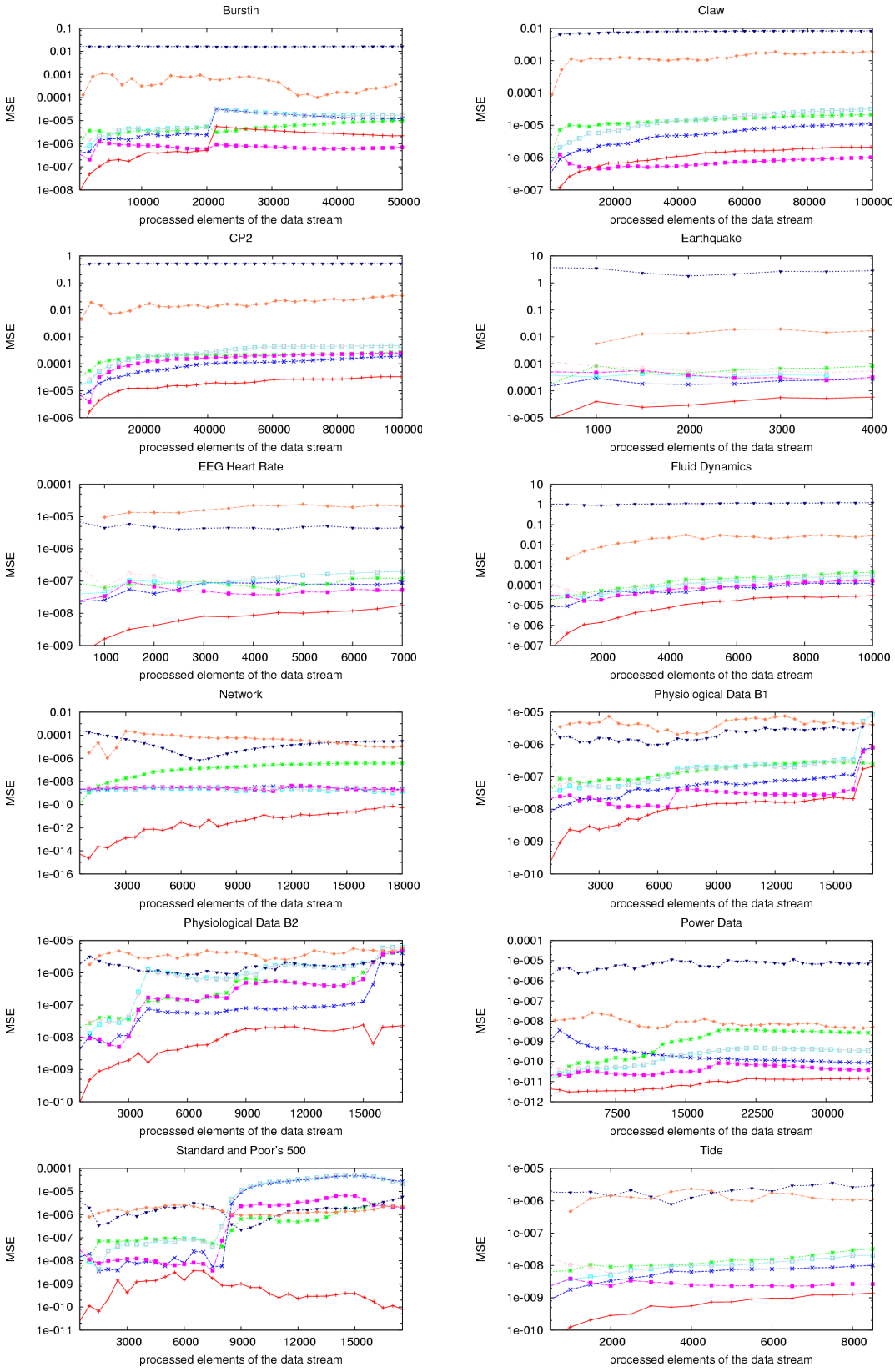


Figure 7.7: Results of Cluster Kernels with MSE based on $\hat{f}^{(opt,n)}$

merge costs only locally at the expense of merge costs that are not always up-to-date. To compare list- and tree-based Cluster Kernels, we examine the variants based on one-value-resampling.

As the results in Figure 7.7 suggest, the two variants performed similar, except that list-based Cluster Kernels were more accurate in terms of a lower MSE according to Table 7.4. Only for the Standard and Poor's 500 and the Network stream, we observed a significantly different development of the MSE.

Accuracy of Cluster Kernels Overall, the different Cluster Kernel variants produced estimates of a high accuracy. Even though the MSE is not suitable for comparing different techniques as it measures the absolute, not the relative differences, we see that the MSE was very low in all cases, ranging from 10^{-18} to 10^{-4}. This also reflects in the summary statistics of the average MSEs. The means over all streams are between $6.4 \cdot 10^{-6}$ and $9.8 \cdot 10^{-5}$. As the corresponding variances are also very low, we conclude that Cluster Kernels are very robust as they achieve similar accuracies for all streams.

Another interesting aspect is the steadiness in the development of the MSE. In the majority of cases, all Cluster Kernel variants produced a smooth MSE. Only for Physiological Data B1 and B2, Burstin, and Standard and Poor's 500, the MSE exhibited abrupt jumps and a more unsteady behavior.

Accuracy of sample-based KDEs For sample-based KDEs, we observed the same tendencies as for sample-based WDEs. In the majority of cases, the MSE increased slightly or remained constant. We expected this development for the same reason as for sample-based WDEs: the more elements are processed, the more worsens the ratio between sample size and size of the data set the sample relies on. As this ratio determines the estimation quality, the MSE will typically not decrease.

Accuracy of M-Kernels M-Kernels showed an identical behavior for all streams; they produced constant MSEs with high absolute values. Only for the Network stream, the MSE temporarily decreased. As reason for this poor performance, we identified an inappropriately chosen bandwidth. In most cases, the bandwidth was too high to capture local details. Consequently, M-Kernels produced oversmoothed estimates. Only for Physiological Data B2 and Standard and Poor's 500, the bandwidth was suitable, resulting in reasonable estimates.

The statistics of the average MSEs clearly show that M-Kernels fail to estimate the unknown density. The corresponding mean is 0.362, accompanied by a large variance of 0.652.

Comparison of Accuracy After discussing the different techniques separately, let us compare them.

As Figure 7.7 documents, Cluster Kernels outperformed sample-based KDEs and M-Kernels in most cases by orders of magnitude. Only for Physiological Data B1 and B2 as well as for Standard and Poor's 500, sample-based KDEs and M-Kernels achieved comparable levels of accuracy. In most cases, M-Kernels showed the worst performance. The average MSEs of the different techniques validate these results. While the mean of the average MSEs

Technique	*Minimum*	*Maximum*	*Mean*	*Variance*
List-based Cluster Kernels one-value-resampling	$1.4 \cdot 10^{-7}$	0.11	0.012	$9.97 \cdot 10^{-4}$
Tree-based Cluster Kernels one-value-resampling	$1.4 \cdot 10^{-7}$	0.11	0.012	$9.9 \cdot 10^{-4}$
Tree-based Cluster Kernels mean-var-equal-dist-resampling	$1.4 \cdot 10^{-7}$	0.108	0.012	$9.5 \cdot 10^{-4}$
Tree-based Cluster Kernels mean-var-exp-dist-resampling	$1.4 \cdot 10^{-7}$	0.108	0.012	$9.6 \cdot 10^{-4}$
Tree-based Cluster Kernels min-max-equal-dist-resampling	$1.4 \cdot 10^{-7}$	0.112	0.012	0.001
Tree-based Cluster Kernels min-max-exp-dist-resampling	$1.4 \cdot 10^{-7}$	0.108	0.012	$9.61 \cdot 10^{-4}$
M-Kernels	$5.4 \cdot 10^{-6}$	2.833	0.391	0.732
Sample-based KDEs	$1.1 \cdot 10^{-7}$	0.101	0.015	$9.7 \cdot 10^{-4}$

Table 7.5: Descriptive statistics for the average MSEs of Cluster Kernels w.r.t. $\hat{f}^{(opt)}$

of the best Cluster Kernel variant was $6.4 \cdot 10^{-6}$, the corresponding mean for sample-based KDEs was 0.0058 and 0.362 for M-Kernels.

Within the different Cluster Kernel variants, the list-based one using one-value-resampling performed best for all streams, except for Burstin and Claw, where tree-based Cluster Kernels using mean-var-resampling with equal distances performed better. If we rank the different Cluster Kernel variants, we receive the following results, starting with the best: list-based and one-value-resampling, tree-based and one-value-resampling, tree-based and mean-var-resampling with equal distances, tree-based and min-max-resampling with exponential distances, tree-based and min-max-resampling with equal distances, tree-based and mean-var-resampling with exponential distances.

Concerning the steadiness of the MSE, all techniques produced relatively smooth MSEs. Only for a few streams, some Cluster Kernel variants exhibited jumps and fluctuations.

Overall, we can state that Cluster Kernels kept very close to their offline counterparts while allocating only a small amount of memory. In comparison to Cluster Kernels, sample-based KDEs and M-Kernels mostly failed in doing so.

7.3.2 Convergence of Cluster Kernels

In order to examine the convergence of Cluster Kernels, we evaluated the MSE with respect to $\hat{f}^{(opt)}$. Figure 7.8 and Table 7.5 display the results of this experiment.

Impact of Resampling Strategy on Accuracy In contrast to the previous experiment, the Cluster Kernel variants based on the different resampling strategies mostly achieved the same accuracy. As Figure 7.8 clarifies, the according Cluster Kernels produced for all but the Burstin stream estimates of nearly the same quality in terms of MSE. For the Burstin stream, the MSE of some Cluster Kernel variants jumped after half the stream was processed. For the other streams, we only observed for the last processed elements of the stream larger

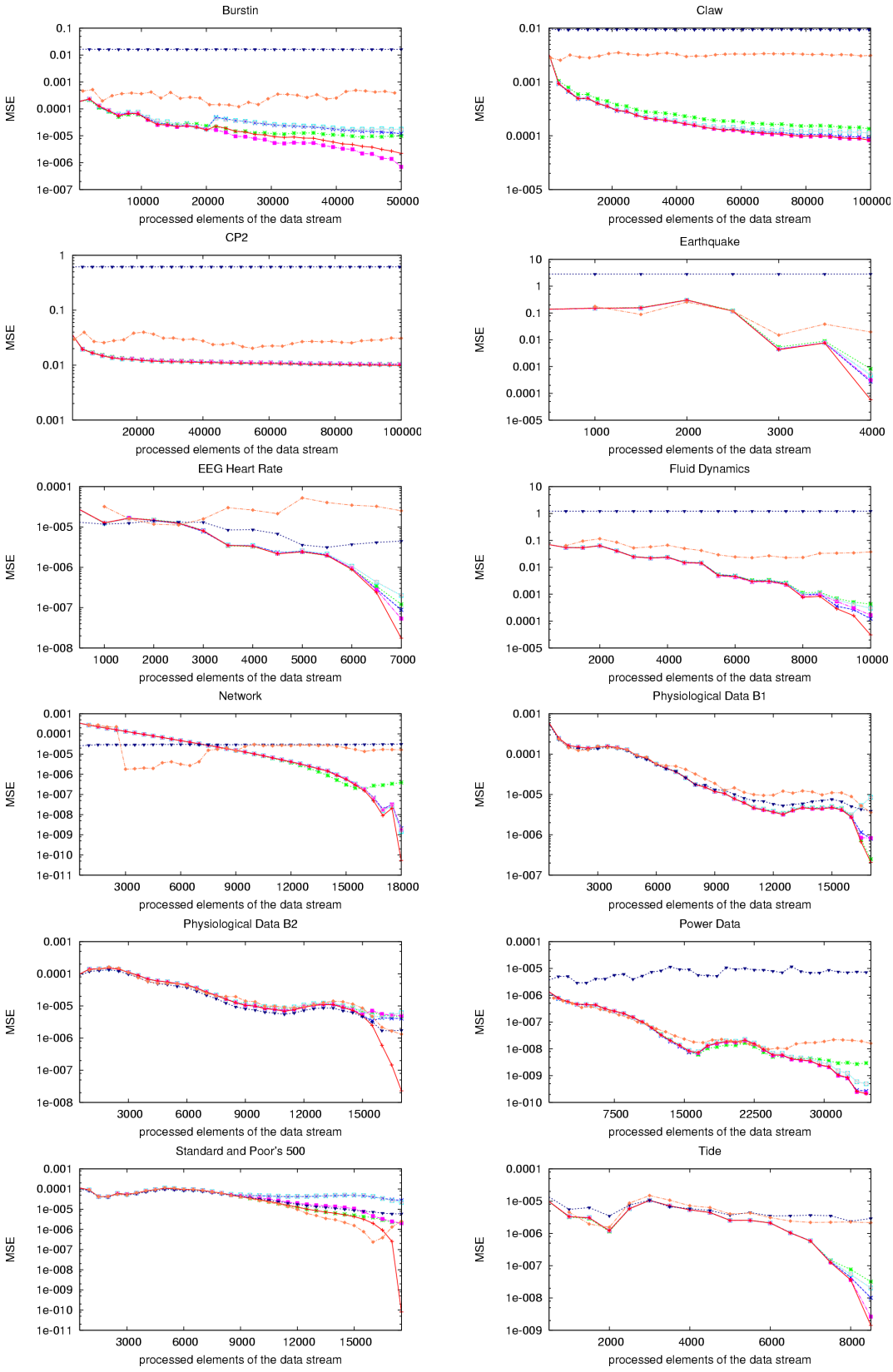


Figure 7.8: Results of Cluster Kernels with MSE based on $\hat{f}^{(opt)}$

differences between the MSEs; but this effect only occurred in a few cases. The similar performance of the Cluster Kernel variants also reflects in the average MSEs. The mean is 0.012 for all variants and their variance is nearly identical.

List-based vs. tree-based Cluster Kernels As the results suggest, the differences between list-based and tree-based Cluster Kernels with one-value-resampling were marginal. The list-based variant only became superior for the last processed elements of a stream. The fact that list-based and tree-based Cluster Kernels achieved nearly the same accuracy is very important. Recall that list-based Cluster Kernels update all merge costs, at the expense of higher processing cost. On the contrary, tree-based Cluster Kernels update the merge costs only locally, at the expense of merge costs not being up-to-date. However, if the loss in accuracy of tree-based Cluster Kernels is only minor, the processing cost becomes the relevant factor. As we will see in Section 7.5, tree-based Cluster Kernels are significantly faster than the list-based ones. Therefore, we can state that tree-based Cluster Kernels are better suited for practical purposes.

Accuracy of Cluster Kernels For the entirety of streams, all Cluster Kernel variants converged very fast to the corresponding densities in terms of a rapidly decreasing MSE. The final MSEs after processing the complete streams were very low, ranging from 10^{-2} to 10^{-10}. We also observed a very steady development of the MSE without abrupt jumps or fluctuations.

Generally, Cluster Kernels have proved to very robust in terms of mean and variance of the average MSEs. Their performance given a limited amount of memory is relatively independent of the concrete stream.

In comparison to the previous experiment, it is interesting to note that the differences between the Cluster Kernel variants were far smaller.

Accuracy of sample-based KDEs In Section 7.2.1 and 7.2.2, we examined the performance of sample-based WDEs, given the same experimental setting. Since sample-based KDEs exhibited the same effects as sample-based WDEs, let us briefly recapitulate the conclusions we drew for sample-based WDEs. The more or less constant development of the MSE can be traced back to the constant size of the sample. The reasoning for a decreasing MSE is the temporally varying nature of some streams. Features of these streams that first occur later in time can not be estimated before. Due to the comparison with the best offline estimate, which incorporates these features, the MSE will consequently be higher in the beginning. Following this argumentation, we can explain the constant as well as the decreasing MSE produced by sample-based KDEs.

Accuracy of M-Kernels The results in Figure 7.8 indicate that M-Kernels mostly failed to capture the unknown density. They produced a more or less constant MSE with high absolute values in most cases. The mean of the average MSEs was 0.391 and the variance 0.732. As in the previous experiment, this effect results from an inappropriate bandwidth strategy. Due to a bandwidth typically chosen too high, M-Kernels produced oversmoothed estimates that did not resolve local details. However, for Physiological Data B1 and B2 as

well as for Standard and Poor's 500, M-Kernels provided estimates that were competitive with Cluster Kernels.

Comparison of Accuracy The comparison of the different techniques delivered the following ranking: M-Kernels and sample-based KDEs performed worst. Except for a few cases, Cluster Kernels outperformed both techniques by orders of magnitude. Only for Physiological Data B2 and Standard and Poor's 500, M-Kernels and sample-based KDEs were slightly superior to some Cluster Kernel variants.

Among the Cluster Kernel variants, we did not observe significant differences; they mostly delivered estimates of nearly identical quality. Only for the last processed elements of a few streams, list-based Cluster Kernels were superior to the tree-based ones.

Concerning the steadiness, we see that all techniques produced a mostly stable MSE. Only for sample-based KDEs, we observed fluctuations in some cases.

Overall, Cluster Kernels succeeded in capturing an unknown density given only a limited amount of available memory. In comparison to sample-based KDEs and M-Kernels, they were typically better by orders of magnitude.

7.3.3 Approximation Properties of Cluster Kernels

The latter two experiments evaluated the accuracy of Cluster Kernels in terms of the MSE. Analogous to compressed-cumulative WDEs, we also examined the squared differences the MSE is based on; this allowed us to gain insight into the general approximation properties of Cluster Kernels. We observed the following trends for the different Cluster Kernel variants.

One-value-resampling According to Section 6.3.5, Cluster Kernels based on one-value-resampling resample the elements of a partition by generating $c_i^{(n)}$ identical instances of the partition mean $\bar{X}_i^{(n)}$, i.e., one element represents the complete partition. This may become problematic if the partition elements span a wide range of values.

In the experiments, this "underrepresentation" became apparent in the shape of the produced density estimates. More precisely, the density estimates exhibited small local bumps, which were not present in the best offline density estimate. These spurious bumps correspond to the Cluster Kernels. We typically observed them in the tails or in the vicinity of a local maximum.

Resampling strategies with equal Distances To circumvent the underrepresentation of a partition, we developed more sophisticated resampling strategies. One approach is to resample elements by distributing $c_i^{(n)}$ elements equidistantly over the value range covered by the partition elements. To determine this range, we can either use minimum and maximum or mean and variance. The equidistant distribution ensures that the complete range is covered with elements. However, if the partition elements are non-uniformly distributed, this resampling may lead to an oversmoothing.

In fact, we observed slight oversmoothing effects in the experiments. Both resampling strategies based on equal distances corrected the problem of spurious bumps, but oversmoothed near peaks or in the tails.

Resampling strategies with exponential Distances Two of our resampling strategies use exponential distances instead of equal ones. These strategies also ensure that the complete range of a partition is covered with elements, but they distribute most elements around the mean of the partition.

As the experiments revealed, these strategies avoided oversmoothing, but still produced spurious artifacts in the form of small bumps. Again, they typically occurred in the tails or near peaks.

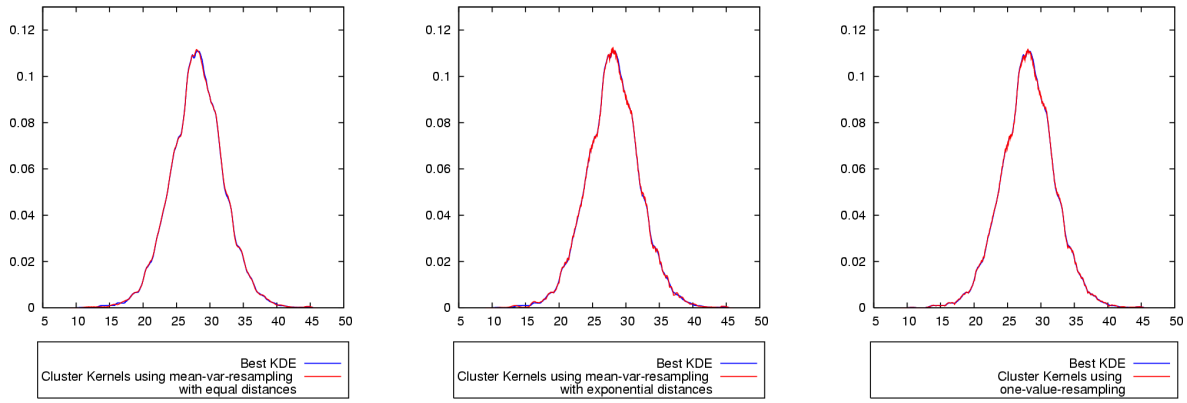


Figure 7.9: Best KDE and Cluster Kernels for EEG Heart Rate stream

To visualize these effects for the different resampling strategies, Figure 7.9 provides an example for the EEG Heart Rate stream. The figure displays Cluster Kernel variants after processing the complete stream as well as the best offline KDE for this stream. The left chart and the one in the middle display Cluster Kernels using mean-var-resampling with equal distances and exponential distances respectively. The chart to the right displays Cluster Kernels using one-value-resampling.

Detection of locally isolated Features For densities with small, locally isolated features, we made an interesting observation. Cluster Kernels had problems to detect these local features if they occurred later in time. For example, the Physiological Data B2 density exhibits an isolated local maximum at the left side, which first appears at the very end of the data stream.

To explain this behavior, recall the merge procedure for Cluster Kernels and the proof of Theorem 6.1. Generally, we build a new Cluster Kernel with weight 1 for each new element. In case we must merge the new Cluster Kernel with its "distant" neighbor, $k \notin \{0, 1, 2, 3\}$ to speak in terms of the proof, we set the mean of the merge kernel as the mean of the Cluster Kernel having the higher weight. Thus, the new Cluster Kernel and the corresponding local detail disappear after the merge.

Impact of larger Memory on Accuracy We discussed the experiments based on a setting of 5 kb of available memory. We also performed additional experiments using larger amounts of available memory. Generally, the more memory is available, the less Cluster Kernels must be merged and the less is the overall loss in accuracy caused by the merges.

Our additional experiments substantiated that a larger amount of available memory induces a higher estimation accuracy.

7.4 Selectivity Estimation of Range Queries over Data Streams

The previous two experiments evaluated the performance of compressed-cumulative WDEs and Cluster Kernels against the background of density estimation over data streams. Besides those two techniques, we evaluated in these experiments a set of competitive techniques that are capable of online density estimation. We measured their accuracy by comparing the estimates they delivered with the best offline KDEs and WDEs respectively. However, this does not allow us to compare the kernel-based and the wavelet-based techniques with each other. For that reason, we conducted an experiment specifically designed to compare the behavior of the different techniques. This experiment addresses range selectivity queries over data streams.

7.4.1 Experimental Setup

To evaluate range selectivity queries over data streams, a technique has to deliver estimates of the pdf or the cdf of a stream or of its quantiles. The following techniques besides Cluster Kernels and compressed-cumulative WDEs meet this requirement: compressed-cumulative KDEs, M-Kernels, sample-based ECDFs, and approximate quantiles. Even though sample-based WDEs and KDEs also meet this requirement, we do not consider them for the sake of clarity. For the same reason, we do not consider all variants of Cluster Kernels and compressed-cumulative WDEs. Instead we concentrated on the following variants, which have proved their suitability in the latter two experiments: compressed-cumulative linear WDEs, compressed-cumulative WDEs with hard thresholding, list-based Cluster Kernels with one-value resampling, and tree-based Cluster Kernels with mean-var-resampling based on equal distances. Generally, we refer to Section 7.1.3 for a description of these techniques, their parameter settings, and their implementation. Figure 7.10 summarizes the line types the techniques are associated with in the subsequent charts.

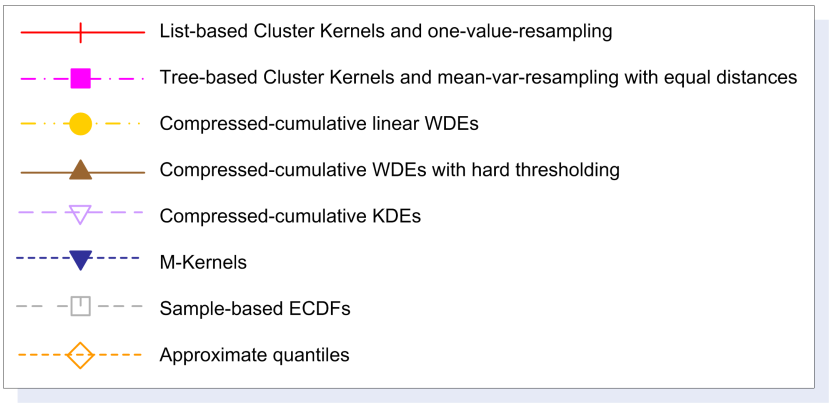


Figure 7.10: Line types of the techniques for range query selectivity estimation

By means of those techniques, we estimated the selectivities of a set of range queries over

the real streams presented in Section 7.1.2.2. More precisely, we estimated each 500 processed elements the selectivities of 1000 range queries. It is important to note that each query is processed with respect to the data seen so far, not with respect to the complete stream. Given the estimated selectivities, we measured their deviations from the true selectivities in terms of the MRE (see Section 7.1.4.2). Since the MRE is an objective measure, it is particularly suited for comparing the different techniques with each other.

Concerning the range queries, we generated four different query workloads for each stream. The selectivity of their range queries distinguishes the workloads from each other. While the first and second workload have small selectivities of approximately 1% and 2% respectively, the third and fourth workload have relatively high selectivities of approximately 5% and 10% respectively.

Let us briefly sketch how we determined for a given data stream 1000 range queries, all having approximately the same selectivity. For the sake of simplicity, let this selectivity be 1% in the following. First, we randomly selected 1000 range queries, whose selectivity with respect to the complete stream is exactly 1%. Second, we computed their selectivities with respect to the different subsets of the stream and obtained values of approximately 1%. The different subsets refer in this context to the first 500, 1000, 1500,... elements of the stream, i.e., to the points where we evaluate the MRE. With regard to the additionally monitored processing time, it is worth mentioning that we computed and stored the true selectivities for the subsets in a preprocessing phase, not while processing the stream.

Basically, the techniques showed the same tendencies for all workloads throughout our experiments; the results for the workloads mostly differed only in the absolute values of the MRE. For that reason, we concentrate on the results for workload 4. The range queries of this workload have a selectivity of approximately 10%. In Section 7.4.4, we discuss the impact of different selectivities on the estimation accuracy. For the record, we provide in Appendix B.2 the experimental results for all workloads and, in particular, for all Cluster Kernel and compressed-cumulative WDE variants.

7.4.2 Comparison of Accuracy

The results of the experiment are displayed in Figure 7.11. The charts within this figure use the x-axis for the number of processed elements and the logarithmically scaled y-axis for the corresponding MRE. Besides plotting the MRE for each technique and each data stream, we also display summary statistics of the average MREs in Table 7.6. As in the previous experiments, the average MRE of a stream refers to the average of its MRE values. To obtain a measure for the entirety of streams, we computed minimum, maximum, mean, and variance of the average MREs for each technique.

Before we compare the different techniques with each other, let us first examine them separately.

Accuracy of compressed-cumulative WDEs The comparison of compressed-cumulative WDEs revealed the following trends. Except for Physiological Data B1 and Physiological Data B2, where they achieved a similar accuracy, compressed-cumulative linear WDEs were clearly inferior to compressed-cumulative WDEs with hard thresholding. As mentioned in Section 3.5.2, WDEs using a thresholding procedure can achieve higher rates of convergence for non-smooth densities due to their localization properties, while linear WDEs are preferable

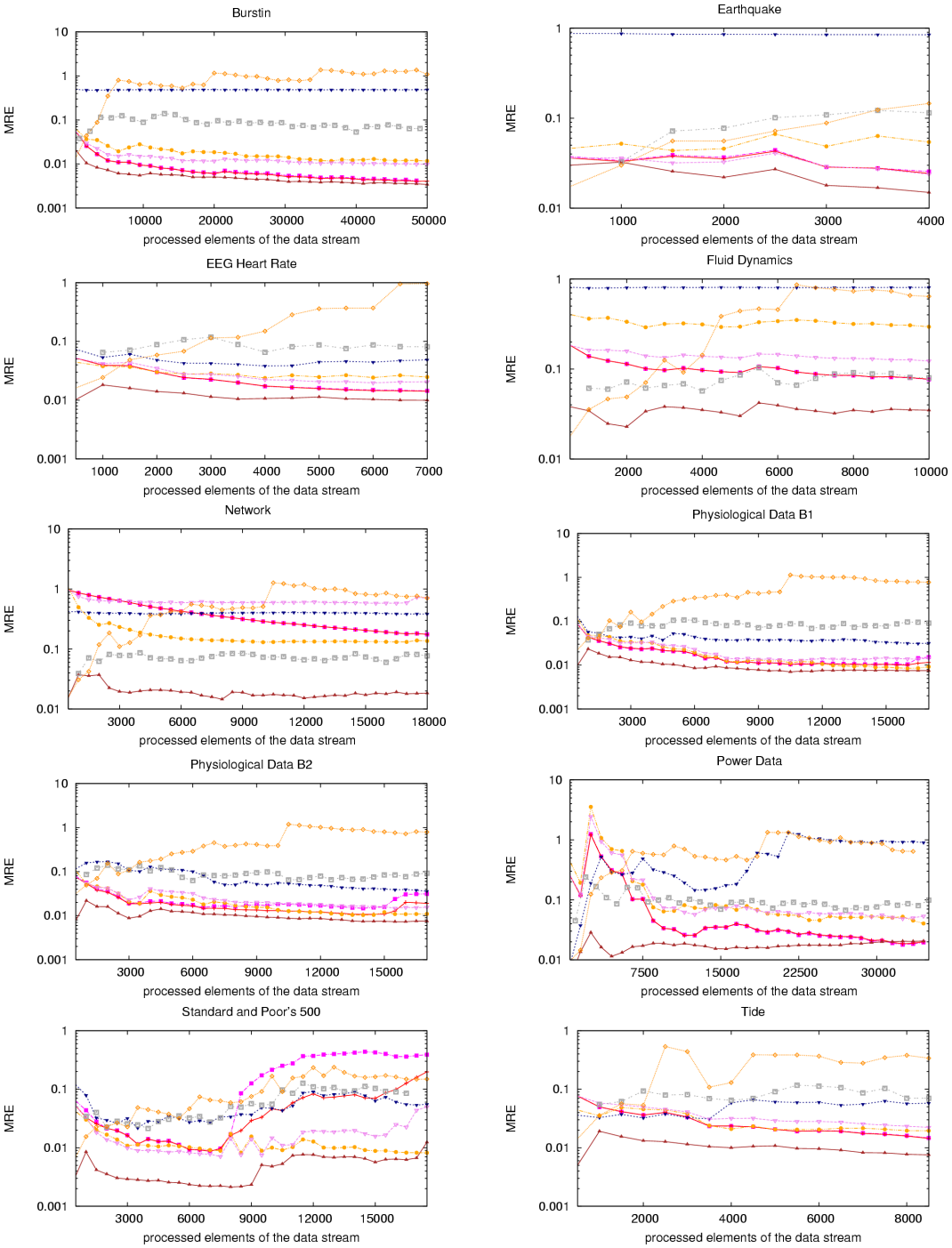


Figure 7.11: Results of all techniques for range selectivity queries

Technique	*Minimum*	*Maximum*	*Mean*	*Variance*
Compressed-cumulative WDEs hard thresholding	0.005	0.034	0.015	$8.3 \cdot 10^{-5}$
Compressed-cumulative linear WDEs	0.012	0.327	0.1	0.015
List-based Cluster Kernels one-value-resampling	0.008	0.387	0.079	0.013
Tree-based Cluster Kernels mean-var-equal-dist-resampling	0.008	0.386	0.093	0.014
Compressed-cumulative KDEs	0.014	0.622	0.117	0.036
Sample-based ECDFs	0.062	0.093	0.082	$8.9 \cdot 10^{-5}$
M-Kernels	0.042	0.856	0.338	0.108
Approximate Quantiles	0.074	0.862	0.434	0.0651

Table 7.6: Descriptive statistics for the average MREs of all techniques

for smooth densities. For the sake of completion, let us mention that compressed-cumulative WDEs with soft thresholding, though not displayed in Figure 7.11, achieved accuracies similar to those of the variant based on hard thresholding.

Generally, compressed-cumulative WDEs achieved a very high degree of accuracy in terms of the average MREs. The thresholding-based variant achieved 1.5% as mean of the average MREs. A noticeable fact is the low variance of the average MREs, which was only $8.3 \cdot 10^{-5}$ over all streams. This result underlines the robustness of compressed-cumulative WDEs.

Another interesting aspect is the development of the MRE. Except for a few local peaks, the MRE generally exhibited a smooth development with the tendency to decrease. Therefore, the accuracy of compressed-cumulative WDEs improves with the number of processed elements. This constant improvement can be traced back to the convergence properties of compressed-cumulative WDEs (see also the experimental results in Section 7.2). The more elements they process, the better is the estimate of the unknown pdf and, as a consequence, the estimate of the associated cdf. As a consequence, the accuracy of the selectivity estimates for the range queries will also improve with the number of processed elements.

Accuracy of Cluster Kernels The two Cluster Kernel variants we examined in this experiment showed an almost identical performance for all streams, except for Physiological Data B2 and Standard and Poor's 500. While the differences for Physiological Data B2 were very small, those for Standard and Poor's 500 were higher due to an abruptly increasing MRE of Cluster Kernels using mean-var-resampling with equal distances.

The fact that the two Cluster Kernel variants achieved the same accuracy in most cases is important with regard to their practical applicability. Recall that one variant relies on the list implementation and the other one on the tree implementation. As we noted, the gain in accuracy induced by the list implementation is mostly negligible, but the processing costs are significantly higher as we will see in Section 7.5. Therefore, we come to the same conclusion as in Section 7.3. Tree-based Cluster Kernels are preferable for practical purposes.

The similar performance of the two Cluster Kernel variants also reflects in their average MREs. According to Table 7.6, both had average MREs between 0.8% and 39%. The high average MRE of 39%, which occurred for the Network stream, is an exception as the mean

and the variance document. While the tree-based variant had a slightly higher mean of 9% compared to 8%, the variance of both variants was almost equal at around 0.014. In combination with the almost identical development of their MREs, we can conclude that Cluster Kernels mostly, not always, achieved nearly the same accuracy.

For the majority of streams, the MRE developed smoothly for both techniques. Only for Standard and Poor's 500, it developed irregularly; it decreased in the beginning, but increased after half of the stream has been processed. For the first processed elements of the Power Data stream, the MRE also increased, but decreased henceforth. In all other cases, the MRE decreased relatively constant. Hence, Cluster Kernels also improve the more elements they process. Analogous to compressed-cumulative WDEs, this behavior can be explained by their convergence properties.

Accuracy of M-Kernels M-Kernels achieved different levels of accuracy while processing the streams. The average MREs ranged from 4% up to 86%, accompanied by a mean of 34% and a variance of 0.108. To understand this unsteady behavior, let us recall the experiment examining the density estimation capabilities of M-Kernels (see Section 7.3.1 and 7.3.2). If we take a closer look at the results of this experiment displayed in Figure 7.7 and 7.8 and combine them with those in Figure 7.11, the following fact reveals. In case M-Kernels failed to estimate the density of a stream, they also failed to estimate the selectivity of range queries posed with respect to this stream. More specifically, we observed a high MSE for the following streams: Burstin, Earthquake, Fluid Dynamics, Power Data. For these streams, we also observed a high MRE. The reason for M-Kernels failing to estimate the density was basically their inappropriate bandwidth strategy, which mostly induced oversmoothed estimates hiding local features.

These facts also reflect in the development of the MRE for the different streams. In most cases, M-Kernels produced a constantly high MRE. Only for Physiological Data B1 and B2, the MRE slightly decreased. On the contrary, it increased for the case of Power Data and Standard and Poor's 500.

Overall, M-Kernels are not suitable for range selectivity queries over data streams. Neither did they improve with the number of processed elements, nor did they exhibit a stable performance for all streams.

Accuracy of compressed-cumulative KDEs For compressed-cumulative KDEs, we observed a partially unstable behavior. Due to high MREs for the Network stream, the average MREs ranged from 1% to 60% with a mean of 12%.

Except for Network and Standard and Poor's 500, compressed-cumulative KDEs improved the more elements they processed, indicated by a smoothly decreasing MRE. Again, we can explain the improvement with the convergence properties of compressed-cumulative KDEs. Even though we did not evaluate their density estimation capabilities in our experiments, let us mention that they behaved similar to compressed-cumulative WDEs in the sense that their density estimation quality improves the more blocks of data they process. For a more detailed discussion, we refer to the experiments in [22] and [21]. In the latter work, the author conducted an extensive experimental study on compressed-cumulative KDEs.

An examination of the development of compressed-cumulative KDEs compared to other techniques revealed an interesting aspect. In a few cases, compressed-cumulative KDEs

performed similar to compressed-cumulative linear WDEs. Consider for example the results for the Physiological Data B1 stream or the Burstin stream.

Accuracy of sample-based ECDFs Sample-based ECDFs produced a very similar MRE for all streams. In fact, the minimum of the average MREs was 6%, while the maximum was 9%. The variance of the average MREs was around $9 \cdot 10^{-5}$. This is noteworthy since it indicates that the accuracy sample-based ECDFs can achieve is more or less independent of the examined stream. A reason for the good performance of sample-based ECDFs is that their low storage requirements allow them to store large fractions of the stream with 5 kb of available memory.

Generally, sample-based ECDFs produced a relatively constant MRE for all streams. Only for Earthquake and Standard and Poor's 500, we observed an increasing MRE. The MRE rarely exhibited peaks or fluctuations; it was typically steady. The reason for occasional local fluctuations is the sample quality which can vary and therefore induce better or worse estimates.

Taking the constant nature of the MRE into account, we conclude that sample-based ECDFs are only partially suitable for the estimation of range selectivity queries over streams, even though they produced a relatively low MRE.

Accuracy of Approximate Quantiles Approximate quantiles produced without exception high MREs for each stream. In fact, we observed average MREs from 7% up to 86% with a mean of 43%.

Moreover, the MRE significantly increased with the number of processed elements for each stream. This contradicts massively with our initial aim to learn from the stream and to improve the estimation accuracy during runtime. To understand the unsatisfying performance of approximate quantiles, let us recapitulate their basic idea. For details, we refer to Section 7.1.3 and particularly to [110]. To approximate the quantiles of a data set, elements are filled into buffers. If the buffers are full, they are compressed, which introduces a compression error. The more elements are processed, the more often the compression step is executed. Consequently, the quality of the approximated quantiles deteriorates with the number of processed elements. Overall, we can state that approximate quantiles fail to estimate the selectivity of range queries over data streams.

Accuracy Comparison of all Techniques After we have discussed the different techniques separately, let us now compare their accuracy. We particularly refer to Table 7.6 in the subsequent discussion as it gives a comprehensive overview of the performance of each technique. Additionally, we consider the development of the MRE as depicted in Figure 7.11.

The best performance among all techniques was achieved by compressed-cumulative WDEs with hard thresholding. They performed best for all streams. Not only was the mean of their average MREs with 1.5% very low, the corresponding variance was also very low with only $8.3 \cdot 10^{-5}$. The low variance substantiates the robustness of compressed-cumulative WDEs with hard thresholding.

For the majority of streams, the two Cluster Kernel variants performed second best. In a few cases, e.g. Network and Standard and Poor's 500, they were inferior to other techniques. Overall, the mean of their average MREs was 7.9% and 9.3% respectively.

In comparison to Cluster Kernels, compressed-cumulative linear WDEs and compressed-cumulative KDEs were mostly inferior with 10% and 11.7% as means of their average MREs. However, they typically performed better than M-Kernels, sample-based ECDFs, or approximate quantiles.

Among the latter three techniques, sample-based ECDFs have proved to be the best one. They performed second best for Fluid Dynamics and Network and also achieved good results for Power Data and Standard and Poor's 500. With a mean of 8.2%, they were even superior to tree-based Cluster Kernels. This results from the fact that they performed significantly better for the Network stream, where tree-based Cluster Kernels produced a high MRE. If we exclude the Network stream from the computation of the mean, tree-based Cluster Kernels are superior. Sample-based ECDFs were superior to compressed-cumulative KDEs and compressed-cumulative linear WDEs.

M-Kernels and approximate quantiles showed the worst performance with means of 33.8% and 43.4% respectively. Moreover, approximate quantiles deteriorated the more elements they processed. They were only competitive to our techniques for the first processed elements. Then their MRE massively increased in contrast to our techniques, which produced a decreasing MRE. Even though the MRE of M-Kernels did not increase, they nevertheless were not competitive to our techniques due to high absolute values of the MRE.

In order to recapitulate the results of this experiment, we can state that Cluster Kernels and compressed-cumulative WDEs both succeeded in estimating the selectivity of range queries over data streams. They produced robust estimators and typically improved in quality the more elements they processed. They achieved high levels of accuracy for each stream. Compressed-cumulative KDEs and sample-based ECDFs also achieved a comparably good accuracy. M-Kernels and approximate quantiles, however, were clearly inferior.

7.4.3 Comparison with best offline Estimators

Let us now evaluate how valuable Cluster Kernels and compressed-cumulative WDEs are by comparing them with their best offline counterparts. For each of the streams, we computed the best linear WDE, the best WDE with hard thresholding, and the best KDE. Then, we estimated the range query selectivities with respect to the best estimators and computed the associated MRE. We used query workload 4, whose range queries have a selectivity of 10%. For each stream, we compared the MRE of the best estimators with the last MRE produced by Cluster Kernels and compressed-cumulative WDEs. The last MRE refers to the final MRE computed after the last stream element was processed. We considered list-based Cluster Kernels with one-value-resampling, tree-based Cluster Kernels with mean-var-resampling and equal distances, compressed-cumulative linear WDEs, and compressed-cumulative WDEs with hard thresholding. Figure 7.12 shows the results of this comparison, separated for each technique. The x-axis denotes the examined streams and the y-axis the MRE. In order to compare KDEs and WDEs with each other, we scaled each y-axis equally.

The MREs of compressed-cumulative thresholded WDEs were close to the MREs of their best offline counterparts. Compressed-cumulative thresholded WDEs were even better in case of Earthquake and Tide. For the other streams, offline WDEs were slightly superior in terms of lower MREs. We observed similar results for compressed-cumulative linear WDEs. They were superior to their offline counterparts for Standard and Poor's 500 and inferior for Fluid Dynamics.

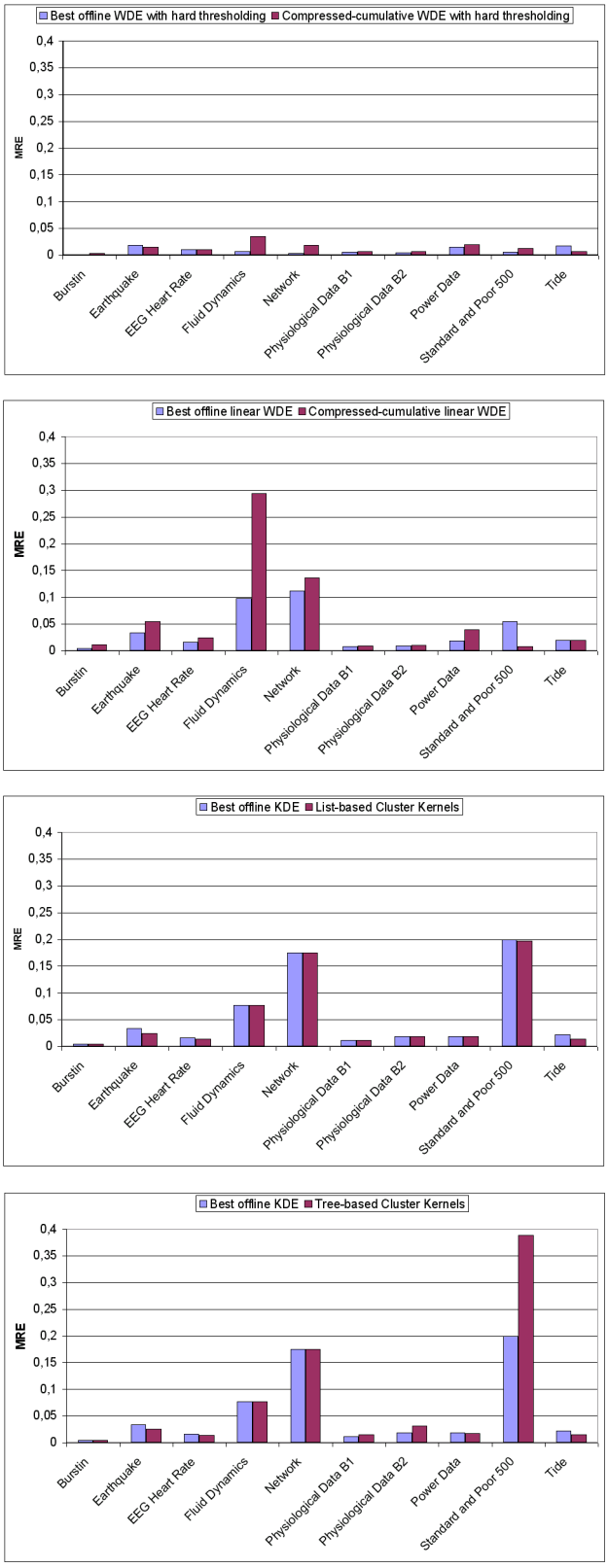


Figure 7.12: Comparison between offline and online estimators

In comparison to compressed-cumulative WDEs, Cluster Kernels performed far more close to the best offline KDEs. Only for the case of tree-based Cluster Kernels with mean-var-resampling and equal distances over the Standard and Poor's 500 stream, we observe a significant deviation. For all other streams, Cluster Kernels and the best offline KDEs produced MREs that were very close to each other.

Generally, this comparison reveals three important facts. First, neither offline KDEs nor offline WDEs estimate the selectivities of range queries exactly, even though they were based on the complete stream and had an unlimited amount of memory at their disposal. Second, WDEs achieved in comparison to KDEs a higher accuracy in terms of the absolute values. Third, Cluster Kernels were very close to their best offline counterpart, even though they were assigned only a small amount of memory.

7.4.4 Impact of Selectivity on Estimation Accuracy

Let us assess the impact of the selectivity on the estimation accuracy. Up to this point, we have discussed the results for workload 4, whose range queries have a selectivity of approximately 10%. We confined our discussion on this workload since we observed the same tendencies for the other workloads. More precisely, we observed the following tendencies while comparing the results for the different workloads.

In the majority of cases, the MREs of the different techniques developed in a very similar manner, except that they were higher the lower the selectivity of the corresponding workload was. For a given stream, the ranking of the techniques with respect to their estimation quality typically did not change for the different workloads.

One reason for a higher MRE in case of lower selectivities is the definition of the MRE as relative error. As described in [79], even a small absolute difference between estimated and true selectivity may induce a large relative error if the true selectivity is very small.

To get an impression of the impact the selectivity had on the estimation accuracy, we display in Figure 7.13 the results for the Physiological Data B2 stream and all workloads. The charts clearly show the described tendencies: while the MREs develop very similar, their absolute values are higher for lower selectivities.

7.5 Construction and Evaluation Complexity

A crucial aspect for the practical applicability of an online technique is its processing time. Therefore, we compare the processing times of the different techniques.

In the previous experiment, we additionally monitored the processing time. We performed two runs of the experiment with the parameters set as before. In the first run, we only computed the current estimator each 500 processed elements. In the second run, we additionally used the current estimator to evaluate the range queries of workload 4. This allows us to form an opinion of a technique's construction time on the one hand and, on the other hand, of its evaluation time. Figure 7.14 depicts the results of the two experiments. The charts in this figure display on the x-axis the number of processed elements and on the y-axis the logarithmically scaled processing time in seconds. We draw the following conclusions from these charts.

Approximate quantiles have proved to be the "fastest" technique in terms of the lowest

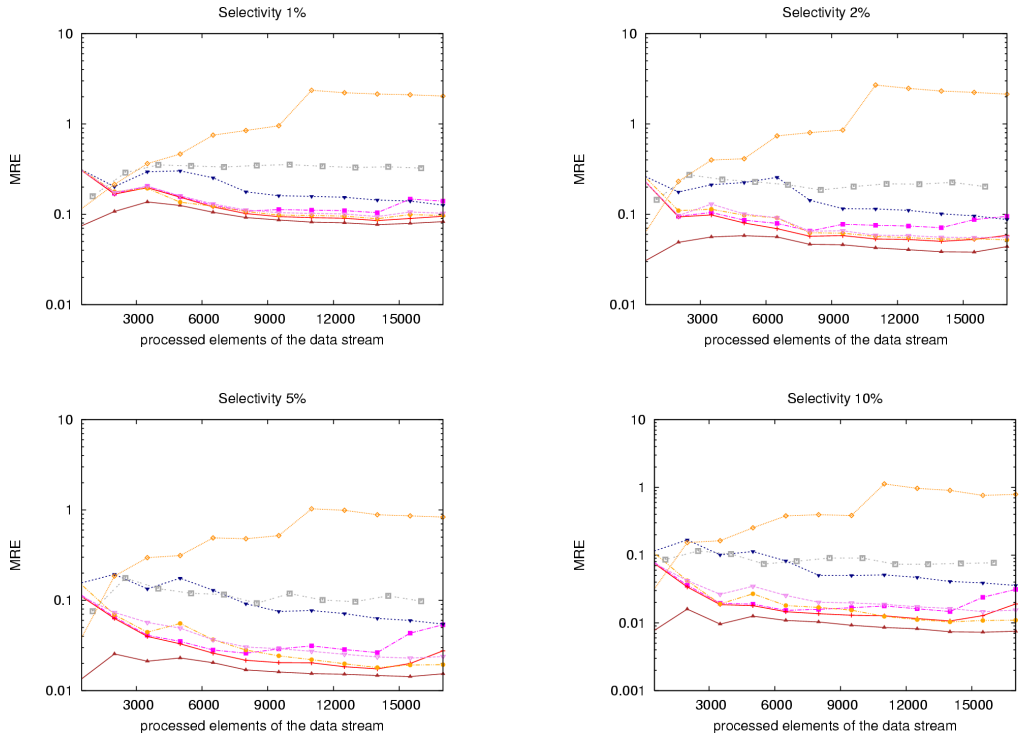


Figure 7.13: Results for Physiological Data B2 stream and all range query workloads

processing time with evaluation of the estimator. But their processing time with an additional evaluation was higher than the one for compressed-cumulative WDEs.

The techniques based on processing blocks of data also constructed and evaluated its estimators in a small period of time. Among them, compressed-cumulative KDEs were superior to the compressed-cumulative WDE variants. Thus, building block KDEs and merging them is computationally less expensive than building and merging block WDEs. Between the two compressed-cumulative WDE variants, the thresholding-based one had higher processing cost.

Sample-based ECDFs placed themselves in terms of processing time between compressed-cumulative KDEs and compressed-cumulative WDEs. Even though they process each element separately, their construction of the current estimator is computationally inexpensive.

Concerning Cluster Kernels, the tree-based variant was, as expected, clearly superior to the list-based variant. Both variants were inferior to the block-based techniques. In comparison to M-Kernels, tree-based Cluster Kernels were significantly faster while list-based Cluster Kernels performed similar to M-Kernels.

Concerning the differences between the processing times with and without evaluating the estimators, we see that the evaluation of approximate quantiles and the block-based techniques only caused little additional effort whereas the evaluation of the other techniques required more time.

Overall, we observe significant differences between the "fastest" and the "slowest" technique. While approximate quantiles elapsed only 0.64 seconds for processing the complete stream with an evaluation of the quantiles, M-Kernels required 3369 seconds. Generally, the

processing time of each technique was linear in the stream size.

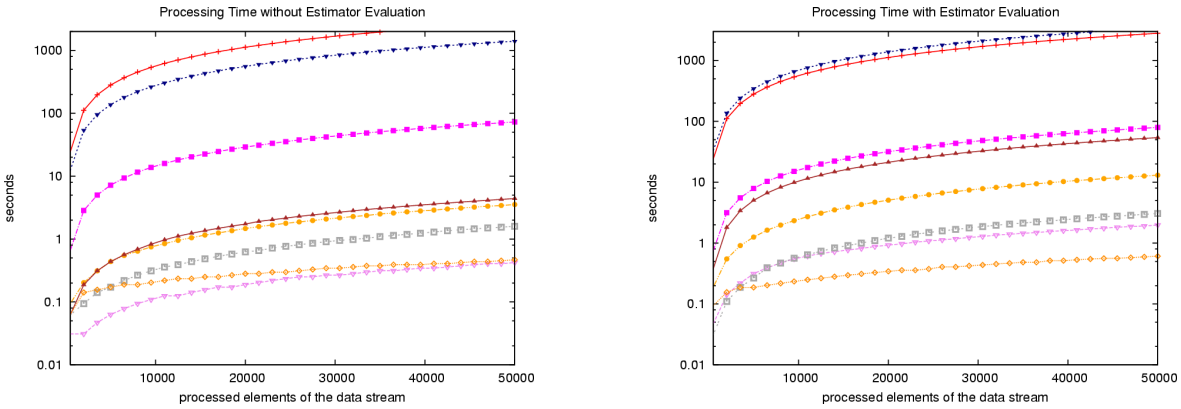


Figure 7.14: Processing time without (left) and with (right) evaluation of the estimator

7.6 Resource-Awareness of Cluster Kernels and compressed-cumulative WDEs

We emphasized in this work the necessity of resource-awareness as it is a fundamental prerequisite for using an online analysis technique in a complex application. For that reason, we examined how Cluster Kernels and compressed-cumulative WDEs react to sudden changes of their available amount of memory.

The experiment we conducted had the following setup: We computed over a stream of CP2 data compressed-cumulative linear WDEs as well as Cluster Kernels with one-value-resampling. While processing the stream, we randomly varied the number of Cluster Kernels and the number of empirical coefficients of compressed-cumulative WDEs from minimum 10 to maximum 100 always after 5000 elements had been processed. To measure the effects of these memory modifications on the estimation accuracy, we computed the MSE between the current estimator and the CP2 density. Figure 7.15 summarizes the results of this experiment for both techniques. In the charts, the x-axis displays the number of processed elements and the left y-axis the current MSE. The right y-axis displays the current number of Cluster Kernels and empirical coefficients.

For both techniques, we observed very similar reactions to the memory modifications. Both reacted very flexibly to the changes. After significant reductions of the available memory, their accuracy deteriorated, indicated by a suddenly increasing MSE. This effect is apparent after 20.000 processed elements, where the maximum number of Cluster Kernels and empirical coefficients dropped to circa 10. However, they "recovered" again in terms of a henceforth decreasing MSE. An interesting fact is that Cluster Kernels and compressed-cumulative WDEs did react very similar to changes of their available memory in terms of the heights of the jumps of the MSE. Only at 70.000 processed elements, we observe a significant difference as the MSE of compressed-cumulative WDEs jumped far higher than the one of Cluster Kernels.

With respect to the results of this experiment, we can state that Cluster Kernels as well

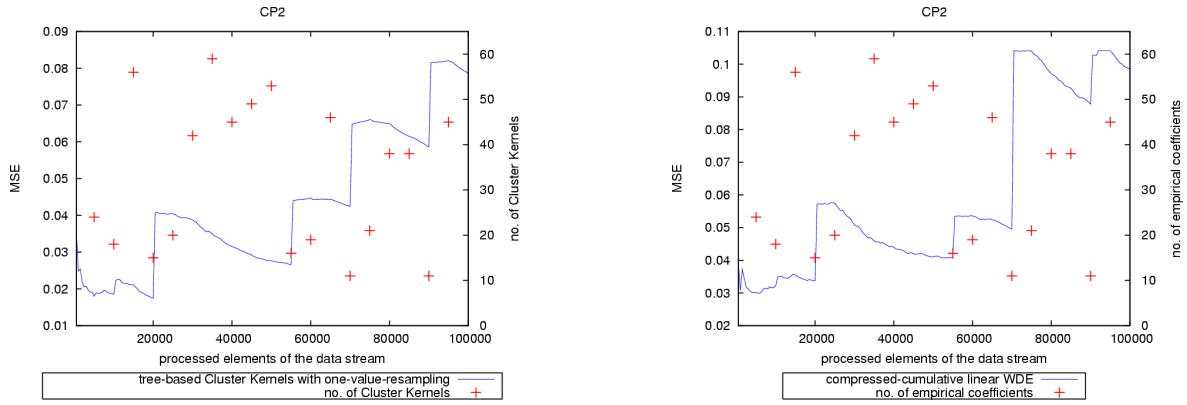


Figure 7.15: Development of the MSE for varying amounts of available memory

as compressed-cumulative WDEs are both resource-aware; they quickly adapt to changing amounts of available memory and succeed in providing suitable estimators meanwhile.

Overall, with the discussion of resource-awareness, we have completed the experimental evaluation of our techniques. In the next chapter, we conclude this work with a summary of its salient results and an outlook on future work.

Chapter 8

Conclusions and Future Work

This chapter concludes this thesis. In Section 8.1, we briefly recapitulate our main results for density estimation over data streams. In Section 8.2, we give an outlook on future research directions, which could utilize our density estimators as a functional starting point for further development.

8.1 Conclusions

The increasing number of real-world applications based on transient data streams necessitates the development of specific stream mining and analysis techniques as the rigid processing requirements for streams render the use of common techniques impossible. In this thesis, we particularly concentrated on density estimation, which serves as building block in many mining and analysis approaches due to its ability to capture the essence of a data distribution in a convenient and comprehensive manner. The importance of density estimation has led to the development of many estimation techniques, among which the nonparametric ones are the most promising. In this work, we investigated kernel and wavelet density estimators, two well-established nonparametric density estimation methods with a solid theoretical foundation. However, neither of them can be directly computed over data streams due to their computational burden. For this reason, we addressed in this work the question of how to adapt kernel and wavelet density estimators to real-valued data streams in compliance with the processing requirements for streams.

We introduced compressed-cumulative WDEs, a novel approach for wavelet density estimators over data streams. Compressed-cumulative WDEs are derived from our generic framework for maintaining nonparametric estimators over data streams. Estimators computed with this framework rely on processing the stream in blocks of data, where each block is associated with a separate estimator. The entirety of these block estimators is convex-linear combined, resulting in an overall estimator for the already processed stream elements. To meet the requirement of online computability, the convex linear combination of all block estimators can be iteratively computed by successively merging the current overall estimator with the new block estimator. As the overall estimator is only allowed to allocate a preset amount of memory, it is further compressed. For the case of compressed-cumulative WDEs, the essential processing steps of the framework can be suitably defined with the help of the wavelet series expansion of a block estimator. This representation in terms of the empirical

scaling and wavelet coefficients allows us to confine the convex merge step to the merge of the corresponding empirical coefficients. The compression step also reaps the benefits of the wavelet series expansion since we can reduce the storage space of a compressed-cumulative WDE by discarding a sufficient number of empirical coefficients. A convenient selection scheme for the coefficients to drop is to select the wavelet coefficients with lowest absolute value; this strategy ensures a minimum error. To determine the resulting loss in accuracy, we introduced the relative compression error as loss measure.

Cluster Kernels are our approach to kernel density estimation over data streams. Cluster Kernels are local clusters of the data stream at different data localities called partitions. A Cluster Kernel maintains local statistics which continuously summarize the elements of its associated partition. By means of these local statistics, the elements of each partition can be approximately resampled with respect to an appropriate resampling strategy. The entirety of Cluster Kernels and the corresponding elements resampled from their partitions can be utilized to build an estimator for the already processed stream elements. With regard to the limited resources, we introduced an intelligent merge scheme for Cluster Kernels, which allows us to reduce their number in a flexible manner, with a minimum loss in accuracy. If necessary, the two Cluster Kernels being closest to each other among all possible Cluster Kernel pairs are substituted by their merge kernel. The notion of closeness between two Cluster Kernels relies on the definition of a loss function, which measures the deviation of the merge kernel from the two Cluster Kernels. Overall, the generic and modular design of Cluster Kernels provides an intuitive way to construct online computable kernel density estimators by specifying exchangeable strategies for kernel function and bandwidth setting, computation of local statistics, and defining a loss function. We thoroughly discussed the specification of Cluster Kernels for the case of univariate data and extended the presented strategies to the multivariate case. Besides an adaptation of theoretically well-founded settings for kernel function and bandwidth, we defined a suitable loss function and investigated resampling strategies based on simple local statistics.

We complemented the presentation of Cluster Kernels and compressed-cumulative WDEs with a discussion of their implementation concepts in XXL and PIPES in order to provide an easy access to their use. We implemented both techniques as complex operators in PIPES, which allows us to combine several analysis tasks over multiple data streams in a single operator graph. This implementation concept particularly makes the integration of new analysis functionality on top of our online density estimators a much easier task.

In order to scrutinize compressed-cumulative WDEs and Cluster Kernels, we conducted an extensive experimental study. We carried out a large set of experiments for real-world data streams originating from different application scenarios as well as for synthetic streams. The experiments were designed specifically to evaluate the efficacy and efficiency our density estimators achieve with respect to estimation quality, processing time, and memory allocation. The results of the experiments revealed that our estimators delivered estimates of high quality with very small deviations from the best possible density estimates. Even though our estimators were assigned only a small amount of memory, they achieved high convergence rates for all streams, a fact which substantiates their robustness. In comparison to a set of competitive techniques including M-Kernels, approximate quantiles, as well as sample-based kernel and wavelet density estimators, our estimators were clearly superior in terms of estimation accuracy and, for the case of M-Kernels, also in terms of processing time. Concerning

the resource-awareness of our estimators, we observed that they adapted very flexibly to sudden changes of their available memory. Even in the presence of massive memory reductions, they still produced reasonable estimates.

Overall, we conclude that compressed-cumulative WDEs and Cluster Kernels both succeed in estimating the unknown density of a data stream while having only limited resources at their disposal. Both techniques can be exploited as building blocks in further analysis tasks over data streams.

8.2 Future Work

For compressed-cumulative WDEs and Cluster Kernels, we already examined extensions to be addressed in future work on data stream analysis. The extensions for compressed-cumulative WDEs presented in Section 5.2.8 comprise for example the integration of other WDE types, deletions in the stream, and multivariate WDEs. The extensions for Cluster Kernels presented in Section 6.4 refer, amongst other things, to the integration of other bandwidth strategies, multivariate Cluster Kernels, and deletions in the stream.

In this section, we sketch preliminary approaches to augment stream mining and analysis by using our online density estimators as starting point. Our intention is not to present complete solutions, but rather to convey a feeling for the wide scope of applications based on data streams that can reap the benefits of our techniques. A more detailed elaboration of the presented approaches remains to be done in future work.

Coupling with Temporal Information We examined compressed-cumulative WDEs and Cluster Kernels against the background of maintaining a density estimator over the complete data stream, i.e., over the entirety of already processed elements. However, it may also be interesting to consider density estimators over different time horizons, e.g., to examine temporal areas of interest or to track evolving data streams. To achieve this goal, it is necessary to couple our estimators with temporal information. Note that most streams attribute their elements with explicit temporal information in the form of timestamps.

For the case of Cluster Kernels, temporal information can be summarized with simple statistics similar to the local statistics we used for summarizing the stream elements of each partition. For this purpose, the pyramidal time frame presented in [9] can be utilized to develop a hierarchy of temporal and value-based summary statistics for different time horizons. The finer the granularity of the hierarchy is, the more recent the elements and the corresponding statistics are. To derive a kernel density estimator for a given time horizon, the corresponding summary statistics can be used to resample the according elements and to establish a kernel density estimator over them.

For the case of compressed-cumulative WDEs, one can examine whether the approach for mining streams under block evolution presented in [63] could be adapted to suit the maintenance of a set of block estimators. These block estimators can be combined flexibly to build an overall estimator for a specific time horizon.

Detection of Change Points As data streams often occur in unstable environments, their characteristics may change during runtime. A crucial question in this context is how to identify these change points. From a formal point of view, a change point indicates that

the underlying distribution has altered. The detection of these change points is a vital field of study in mathematical statistics. To detect them online while processing a data stream is a difficult task as each element can be accessed only once. In [18], the authors developed a statistical test for detecting change in data streams.

Our online density estimators can efficiently support the detection of change points. Following the idea in [18], pairs of density estimators can be maintained continuously. While one estimator of each pair is computed with respect to a sliding window, the other one is computed over the complete stream with the help of our techniques. The latter one is reset when a change is detected. These pairs of density estimators can be compared continuously. In case of large deviations, a change in the underlying distribution is likely to occur. For the comparison of two estimators and their differences, an appropriate statistical test has to be developed. For example, the Kolmogorov-Smirnov [129] test could be modified for this purpose. By using a set of estimator pairs, long term and short term changes in the stream behavior can be detected by means of sliding windows with different lengths.

Detection of Outliers Another important task in stream analysis is to detect outliers in a timely fashion as they are often critical values in the underlying application. As we already sketched in [86], our density estimators can be efficiently utilized to detect outliers in an intuitive manner. In the context of sensor networks, [131] also tackles outlier detection for data streams with kernel density estimators, which are computed over sliding windows.

Intuitively, an outlier is unusual or unexpected in comparison to the other elements; its occurrence is "improbable". In terms of the underlying probability density function, an outlier is expected to lie in a region with low density. With this notion of outliers, their detection while processing a data stream can be implemented with density estimators. Given a continuously computed estimator, we examine for each new element whether the probability of its local neighborhood is below a preset threshold. If that is the case, it is labelled as an outlier. The crucial question in this context is how to define the width of the neighborhood and the threshold. Instead of strictly labelling an element as an outlier or not, an alternative is to equip each element with the probability of being an outlier, which corresponds to the probability of its neighborhood. A comparable idea was introduced in [24] with the concept of local outlier factors, which describe the degree of outlier-ness of an element. In order to detect outliers with respect to recent tendencies of the stream, the variants of our density estimators using exponential smoothing to emphasize recent data can be utilized.

Biased Sampling In [99], an approach to determine biased samples of a data set with kernel density estimators is presented. A biased sample refers to a random sample of the data set, where elements from regions of interest have a higher probability to enter the sample. These probabilities are determined with respect to a kernel density estimator computed for the data set.

Biased samples over data streams can be computed in an analogous manner. A continuously computed density estimator for the data stream can be used to determine the probability for each element to enter the sample, which depends on the local density of the element. By emphasizing in the selection process either low or high density regions of the estimator, the sample can be biased toward sparse or dense regions of the data stream. As the density estimator can keep pace with recent trends in the stream, the current elements

of the sample also reflect these trends.

Probabilistic Queries in Sensor Networks Many applications based on data streams are confronted with interfering effects such as noise in the data, transmission errors, limited precision. As a consequence, those streams exhibit an inherent uncertainty, a fact that adversely affects the processing of queries over them; the query results may be unreliable. Data streams originating in sensor networks are particularly prone to these deficiencies. An adequate data processing in sensor networks is also exacerbated by the limited resources of the participating sensors. To reduce the energy consumption in sensor networks, different methods have been recently proposed, e.g., exploiting data redundancies to efficiently process spatio-temporal region queries [41].

A promising way to deal with the limited resources in network environments and with the uncertainty of the associated data streams is to abandon the notion of exact query answers in favor of probabilistic answers. More precisely, probabilistic queries can be developed, which annotate their results with a probabilistic confidence of the accuracy. These queries can be based on statistical models of the sensor streams, which can capture relationships and correlations between sensors as well as their attributes. The statistical models can also reduce the energy consumption of the sensors. Only if the model changes significantly does the sensor have to transmit an update. As a statistical model of a sensor stream, continuously computed density estimators can be utilized. Recently, several approaches picked up on the idea of modeling sensor streams with density estimators and exploiting them for probabilistic query answering [40, 50, 51, 111]. However, none of these approaches make use of nonparametric density estimation techniques for streams. For this reason, an interesting starting point for further investigations is to develop probabilistic queries with respect to our online density estimators. These queries could in particular incorporate the aforementioned analysis tasks of detecting outliers or changes as a new class of queries. The development of these probabilistic queries should be accompanied by the discussion of how to define them in an adequate SQL or CQL syntax in order to provide the user an intuitive access to the new functionality.

Support of Decisions in a DSMS As our online density estimators provide a comprehensive summary of a data stream, a kind of statistical synopsis, their analysis facilities can be exploited to augment decision making in a DSMS. Due to the brittle nature of data streams, a DSMS has to adapt its system decisions very quickly to changing conditions. On account of this necessity, suitable statistical models of the streams, ones which keep pace with the development of the streams, can play an important role in the process of decision making. A density estimator can serve as a kind of complex metadata, which provides far more complex analysis facilities than the simple descriptive statistics which are typically collected in a DSMS. For example, besides using transformation rules, the optimizer can also consult the statistical model of each stream before it generates or modifies its query plans.

Overall, the brief discussion of these preliminary approaches illustrates that our density estimators can prepare the ground for a plethora of further stream mining and analysis tasks; they can be utilized as major building blocks. Apart from the approaches mentioned here, we are sure that many other stream related topics can reap the benefits of sophisticated density estimators over data streams, as they capture the essential characteristics of a stream in a convenient manner.

Appendix A

Appendix to the Proof of Theorem 6.1

Theorem 6.1 states that the minimum of the loss function $loss(X)$ exists and can be computed in constant time. In the proof of this theorem, the support partitioning of $loss(X)$ plays an important role. Also an important plays the first derivative of $loss(X)$, which we need to determine the minimum of $loss(X)$. For that reason, we give in this appendix the different support partitionings as well as the closed formulas for the first derivative of $loss(X)$.

A.1 Support Partitioning for Loss Function

The evaluation of $loss(X)$ at an arbitrary $X \in \mathbb{R}$ requires to determine the supports I_i and I_j of the summands given in equation (6.21). To achieve this goal, we introduced a partitioning of the support of $loss(X)$, which uniquely determines the supports I_i and I_j for each $X \in \mathbb{R}$. The concrete support partitioning depends on the relative positions of I_i and I_j to each other, which we described by the parameter k. The following tables give the support partitionings for the different cases of k with $k \in \{0, 1, 2, 3\}$.

support partition	I_i	I_j
$(-\infty, \bar{X}_i^{(n)} - 2\hat{h}^{(n)}]$	$\varnothing$	$\varnothing$
$(\bar{X}_i^{(n)} - 2\hat{h}^{(n)}, \bar{X}_j^{(n)} - 2\hat{h}^{(n)}]$	$[\bar{X}_i^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$	$\varnothing$
$(\bar{X}_j^{(n)} - 2\hat{h}^{(n)}, \bar{X}_i^{(n)}]$	$[\bar{X}_i^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$	$[\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$
$(\bar{X}_i^{(n)}, \bar{X}_j^{(n)}]$	$[X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$	$[\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$
$(\bar{X}_j^{(n)}, \bar{X}_i^{(n)} + 2\hat{h}^{(n)}]$	$[X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$	$[X - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$
$(\bar{X}_i^{(n)} + 2\hat{h}^{(n)}, \bar{X}_j^{(n)} + 2\hat{h}^{(n)}]$	$\varnothing$	$[X - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$
$(\bar{X}_j^{(n)} + 2\hat{h}^{(n)}, \infty)$	$\varnothing$	$\varnothing$

Table A.1: Support partitioning for $k = 0$

support partition	I_i	I_j
$(-\infty, \bar{X}_i^{(n)} - 2\hat{h}^{(n)}]$	$\varnothing$	$\varnothing$
$(\bar{X}_i^{(n)} - 2\hat{h}^{(n)}, \bar{X}_j^{(n)} - 2\hat{h}^{(n)}]$	$[\bar{X}_i^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$	$\varnothing$
$(\bar{X}_j^{(n)} - 2\hat{h}^{(n)}, \bar{X}_i^{(n)}]$	$[\bar{X}_i^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$	$[\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$
$(\bar{X}_i^{(n)}, \bar{X}_j^{(n)}]$	$[X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$	$[\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$
$(\bar{X}_j^{(n)}, \bar{X}_i^{(n)} + 2\hat{h}^{(n)}]$	$[X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$	$[X - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$
$(\bar{X}_i^{(n)} + 2\hat{h}^{(n)}, \bar{X}_j^{(n)} + 2\hat{h}^{(n)}]$	$\varnothing$	$[X - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$
$(\bar{X}_j^{(n)} + 2\hat{h}^{(n)}, \infty)$	$\varnothing$	$\varnothing$

Table A.2: Support partitioning for $k = 1$

support partition	I_i	I_j
$(-\infty, \bar{X}_i^{(n)} - 2\hat{h}^{(n)}]$	$\varnothing$	$\varnothing$
$(\bar{X}_i^{(n)} - 2\hat{h}^{(n)}, \bar{X}_i^{(n)}]$	$[\bar{X}_i^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$	$\varnothing$
$(\bar{X}_i^{(n)}, \bar{X}_j^{(n)} - 2\hat{h}^{(n)}]$	$[X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$	$\varnothing$
$(\bar{X}_j^{(n)} - 2\hat{h}^{(n)}, \bar{X}_i^{(n)} + 2\hat{h}^{(n)}]$	$[X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$	$[\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$
$(\bar{X}_i^{(n)} + 2\hat{h}^{(n)}, \bar{X}_j^{(n)}]$	$\varnothing$	$[\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$
$(\bar{X}_j^{(n)}, \bar{X}_j^{(n)} + 2\hat{h}^{(n)}]$	$\varnothing$	$[X - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$
$(\bar{X}_j^{(n)} + 2\hat{h}^{(n)}, \infty)$	$\varnothing$	$\varnothing$

Table A.3: Support partitioning for $k = 2$

support partition	I_i	I_j
$(-\infty, \bar{X}_i^{(n)} - 2\hat{h}^{(n)}]$	$\varnothing$	$\varnothing$
$(\bar{X}_i^{(n)} - 2\hat{h}^{(n)}, \bar{X}_i^{(n)}]$	$[\bar{X}_i^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$	$\varnothing$
$(\bar{X}_i^{(n)}, \bar{X}_j^{(n)} - 2\hat{h}^{(n)}]$	$[X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$	$\varnothing$
$(\bar{X}_j^{(n)} - 2\hat{h}^{(n)}, \bar{X}_i^{(n)} + 2\hat{h}^{(n)}]$	$[X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}]$	$[\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$
$(\bar{X}_i^{(n)} + 2\hat{h}^{(n)}, \bar{X}_j^{(n)}]$	$\varnothing$	$[\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$
$(\bar{X}_j^{(n)}, \bar{X}_j^{(n)} + 2\hat{h}^{(n)}]$	$\varnothing$	$[X - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$
$(\bar{X}_j^{(n)} + 2\hat{h}^{(n)}, \infty)$	$\varnothing$	$\varnothing$

Table A.4: Support partitioning for $k = 3$

A.2 Derivative of Loss Function

An essential part of the proof of Theorem 6.1 is the computation of a partitioning of the support of $loss(X)$. The resulting partitionings for the different cases of $k \in \{0,1,2,3\}$ are given in Section A.1. By means of the support partitioning, we can determine I_i and I_j, the integration borders of the summands in equation (6.21). More precisely, we can determine I_i and I_j for each partition of the corresponding support partitioning. This allows us to solve the integral in (6.21) with respect to I_i and I_j.

Overall, this delivers a piecewise definition of $loss(X)$ for each partition. Since each partition is associated with one combination of I_i and I_j, we present the first derivative of $loss(X)$ for the different combinations of I_i and I_j. In combination with the support partitionings given in Appendix A.1, we can derive a piecewise definition of the first derivative of $loss(X)$ for all support partitionings.

$$
\begin{aligned}
I_i = [\bar{X}_i^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}], I_j = [\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]
\end{aligned}
$$

$$
\begin{aligned}
loss'(x) \;=\; & -\frac{8}{9(c_i^{(n)} + c_j^{(n)})}\Bigg(X^4 \frac{c_i^{(n)} + c_j^{(n)}}{6(\hat{h}^{(n)})^6} \\
& + X^3\Big(\frac{2}{3(\hat{h}^{(n)})^6}\big(c_i^{(n)}(\hat{h}^{(n)} - \bar{X}_i^{(n)}) + c_j^{(n)}(\hat{h}^{(n)} - \bar{X}_j^{(n)})\big) - \frac{2}{3(\hat{h}^{(n)})^5}(c_i^{(n)} + c_j^{(n)})\Big) \\
& + X^2\Big(\frac{1}{(\hat{h}^{(n)})^6}\big(c_i^{(n)}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^2 + c_j^{(n)}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^2\big) \\
& \quad - \frac{2}{(\hat{h}^{(n)})^5}\big(c_i^{(n)}(\hat{h}^{(n)} - \bar{X}_i^{(n)}) + c_j^{(n)}(\hat{h}^{(n)} - \bar{X}_j^{(n)})\big) + \frac{1}{(\hat{h}^{(n)})^4}(c_i^{(n)} + c_j^{(n)})\Big) \\
& + X^1\Big(\frac{2}{3(\hat{h}^{(n)})^6}\big(c_i^{(n)}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^3 + c_j^{(n)}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^3\big) \\
& \quad - \frac{2}{(\hat{h}^{(n)})^5}\big(c_i^{(n)}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^2 + c_j^{(n)}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^2\big) \\
& \quad - \frac{2}{(\hat{h}^{(n)})^4}\big(c_i^{(n)}(\hat{h}^{(n)} - \bar{X}_i^{(n)}) + c_j^{(n)}(\hat{h}^{(n)} - \bar{X}_j^{(n)})\big) + \frac{2}{3(\hat{h}^{(n)})^3}(c_i^{(n)} + c_j^{(n)})\Big) \\
& + X^0\Big(\frac{1}{6(\hat{h}^{(n)})^6}\big(c_i^{(n)}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^4 + c_j^{(n)}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^4\big) \\
& \quad - \frac{2}{3(\hat{h}^{(n)})^5}\big(c_i^{(n)}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^3 + c_j^{(n)}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^3\big) \\
& \quad - \frac{1}{(\hat{h}^{(n)})^4}\big(c_i^{(n)}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^2 + c_j^{(n)}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^2\big) \\
& \quad + \frac{2}{3(\hat{h}^{(n)})^3}\big(c_i^{(n)}(\hat{h}^{(n)} - \bar{X}_i^{(n)}) + c_j^{(n)}(\hat{h}^{(n)} - \bar{X}_j^{(n)})\big) + \frac{5}{6(\hat{h}^{(n)})^2}(c_i^{(n)} + c_j^{(n)})\Big)\Bigg).
\end{aligned}
$$

$$I_i = [X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}], I_j = [\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$$

$$\begin{aligned}
loss'(x) \;=\; & -\frac{8}{9(c_i^{(n)} + c_j^{(n)})}\Big(X^4 \frac{-c_i^{(n)} + c_j^{(n)}}{6(\hat{h}^{(n)})^6} \\
& + X^3\Big(\frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^6}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) - \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^5} + \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_j^{(n)}) - \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^5}\Big) \\
& + X^2\Big(-\frac{c_i^{(n)}}{(\hat{h}^{(n)})^6}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^2 + \frac{2c_i^{(n)}}{(\hat{h}^{(n)})^5}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) + \frac{c_i^{(n)}}{(\hat{h}^{(n)})^4} \\
& \quad + \frac{c_j^{(n)}}{(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^2 - \frac{2c_j^{(n)}}{(\hat{h}^{(n)})^5}(\hat{h}^{(n)} - \bar{X}_j^{(n)}) - \frac{c_j^{(n)}}{(\hat{h}^{(n)})^4}\Big) \\
& + X^1\Big(\frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^6}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^3 - \frac{2c_i^{(n)}}{(\hat{h}^{(n)})^5}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^2 - \frac{2c_i^{(n)}}{(\hat{h}^{(n)})^4}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) \\
& \quad + \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^3} + \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^3 - \frac{2c_j^{(n)}}{(\hat{h}^{(n)})^5}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^2 \\
& \quad - \frac{2c_j^{(n)}}{(\hat{h}^{(n)})^4}(\hat{h}^{(n)} - \bar{X}_j^{(n)}) + \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^3}\Big) \\
& + X^0\Big(-\frac{c_i^{(n)}}{6(\hat{h}^{(n)})^6}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^4 + \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^5}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^3 + \frac{c_i^{(n)}}{(\hat{h}^{(n)})^4}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^2 \\
& \quad - \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^3}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) - \frac{5c_i^{(n)}}{6(\hat{h}^{(n)})^2} + \frac{c_j^{(n)}}{6(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^4 \\
& \quad - \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^5}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^3 - \frac{c_j^{(n)}}{(\hat{h}^{(n)})^4}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^2 \\
& \quad + \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^3}(\hat{h}^{(n)} - \bar{X}_j^{(n)}) + \frac{5c_j^{(n)}}{6(\hat{h}^{(n)})^2}\Big)\Big).
\end{aligned}$$

$$I_i = [X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}], I_j = [X - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$$

$$\begin{aligned}
loss'(x) \;=\; & -\frac{8}{9(c_i^{(n)} + c_j^{(n)})}\Big(-X^4\frac{c_i^{(n)} + c_j^{(n)}}{6(\hat{h}^{(n)})^6} \\
& +X^3\Big(\frac{2}{3(\hat{h}^{(n)})^6}(c_i^{(n)}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) + c_j^{(n)}(\bar{X}_j^{(n)} + \hat{h}^{(n)})) - \frac{2}{3(\hat{h}^{(n)})^5}(c_i^{(n)} + c_j^{(n)})\Big) \\
& +X^2\Big(-\frac{1}{(\hat{h}^{(n)})^6}(c_i^{(n)}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^2 + c_j^{(n)}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^2) \\
& \quad +\frac{2}{(\hat{h}^{(n)})^5}(c_i^{(n)}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) + c_j^{(n)}(\bar{X}_j^{(n)} + \hat{h}^{(n)})) + \frac{1}{(\hat{h}^{(n)})^4}(c_i^{(n)} + c_j^{(n)})\Big) \\
& +X^1\Big(\frac{2}{3(\hat{h}^{(n)})^6}(c_i^{(n)}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^3 + c_j^{(n)}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^3) \\
& \quad -\frac{2}{(\hat{h}^{(n)})^5}(c_i^{(n)}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^2 + c_j^{(n)}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^2) \\
& \quad -\frac{2}{(\hat{h}^{(n)})^4}(c_i^{(n)}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) + c_j^{(n)}(\bar{X}_j^{(n)} + \hat{h}^{(n)})) + \frac{2}{3(\hat{h}^{(n)})^3}(c_i^{(n)} + c_j^{(n)})\Big) \\
& +X^0\Big(-\frac{1}{6(\hat{h}^{(n)})^6}(c_i^{(n)}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^4 + c_j^{(n)}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^4) \\
& \quad +\frac{2}{3(\hat{h}^{(n)})^5}(c_i^{(n)}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^3 + c_j^{(n)}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^3) \\
& \quad +\frac{1}{(\hat{h}^{(n)})^4}(c_i^{(n)}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^2 + c_j^{(n)}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^2) \\
& \quad -\frac{2}{3(\hat{h}^{(n)})^3}(c_i^{(n)}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) + c_j^{(n)}(\bar{X}_j^{(n)} + \hat{h}^{(n)})) - \frac{5}{6(\hat{h}^{(n)})^2}(c_i^{(n)} + c_j^{(n)})\Big)\Big).
\end{aligned}$$

$$I_i = [\bar{X}_i^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}], I_j = \varnothing$$

$$\begin{aligned}
loss'(x) \;=\; & -\frac{8}{9(c_i^{(n)} + c_j^{(n)})}\Big(X^4 \frac{c_i^{(n)}}{6(\hat{h}^{(n)})^2} \\
& + X^3\Big(\frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_i^{(n)}) - \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^5}\Big) \\
& + X^2\Big(\frac{c_i^{(n)}}{(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^2 - \frac{2c_i^{(n)}}{(\hat{h}^{(n)})^5}(\hat{h}^{(n)} - \bar{X}_i^{(n)}) - \frac{c_i^{(n)}}{(\hat{h}^{(n)})^4}\Big) \\
& + X^1\Big(\frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^3 - \frac{2c_i^{(n)}}{(\hat{h}^{(n)})^5}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^2 - \frac{2c_i^{(n)}}{(\hat{h}^{(n)})^4}(\hat{h}^{(n)} - \bar{X}_i^{(n)}) \\
& \quad + \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^3}\Big) \\
& + X^0\Big(\frac{c_i^{(n)}}{6(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^4 - \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^5}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^3 - \frac{c_i^{(n)}}{(\hat{h}^{(n)})^4}(\hat{h}^{(n)} - \bar{X}_i^{(n)})^2 \\
& \quad + \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^3}(\hat{h}^{(n)} - \bar{X}_i^{(n)}) + \frac{5c_i^{(n)}}{6(\hat{h}^{(n)})^2}\Big)\Big).
\end{aligned}$$

$$I_i = [X - \hat{h}^{(n)}, \bar{X}_i^{(n)} + \hat{h}^{(n)}], I_j = \varnothing$$

$$\begin{aligned}
loss'(x) \;=\; & -\frac{8}{9(c_i^{(n)} + c_j^{(n)})}\Big(X^4 \frac{-c_i^{(n)}}{6(\hat{h}^{(n)})^6} \\
& + X^3\Big(\frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^6}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) - \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^5}\Big) \\
& + X^2\Big(\frac{-c_i^{(n)}}{(\hat{h}^{(n)})^6}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^2 + \frac{2c_i^{(n)}}{(\hat{h}^{(n)})^5}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) + \frac{c_i^{(n)}}{(\hat{h}^{(n)})^4}\Big) \\
& + X^1\Big(\frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^6}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^3 - \frac{2c_i^{(n)}}{(\hat{h}^{(n)})^5}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^2 - \frac{2c_i^{(n)}}{(\hat{h}^{(n)})^4}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) \\
& \quad + \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^3}\Big) \\
& + X^0\Big(\frac{-c_i^{(n)}}{6(\hat{h}^{(n)})^6}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^4 + \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^5}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^3 + \frac{c_i^{(n)}}{(\hat{h}^{(n)})^4}(\bar{X}_i^{(n)} + \hat{h}^{(n)})^2 \\
& \quad - \frac{2c_i^{(n)}}{3(\hat{h}^{(n)})^3}(\bar{X}_i^{(n)} + \hat{h}^{(n)}) - \frac{5c_i^{(n)}}{6(\hat{h}^{(n)})^2}\Big)\Big).
\end{aligned}$$

$$I_i = \varnothing, I_j = [X - \hat{h}^{(n)}, \bar{X}_j^{(n)} + \hat{h}^{(n)}]$$

$$\begin{aligned}
loss'(x) \;=\; & -\frac{8}{9(c_i^{(n)} + c_j^{(n)})}\Big(X^4 \frac{-c_j^{(n)}}{6(\hat{h}^{(n)})^6} \\
& + X^3\Big(\frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^6}(\bar{X}_j^{(n)} + \hat{h}^{(n)}) - \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^5}\Big) \\
& + X^2\Big(\frac{-c_j^{(n)}}{(\hat{h}^{(n)})^6}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^2 + \frac{2c_j^{(n)}}{(\hat{h}^{(n)})^5}(\bar{X}_j^{(n)} + \hat{h}^{(n)}) + \frac{c_j^{(n)}}{(\hat{h}^{(n)})^4}\Big) \\
& + X^1\Big(\frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^6}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^3 - \frac{2c_j^{(n)}}{(\hat{h}^{(n)})^5}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^2 - \frac{2c_j^{(n)}}{(\hat{h}^{(n)})^4}(\bar{X}_j^{(n)} + \hat{h}^{(n)}) \\
& \quad + \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^3}\Big) \\
& + X^0\Big(\frac{-c_j^{(n)}}{6(\hat{h}^{(n)})^6}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^4 + \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^5}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^3 + \frac{c_j^{(n)}}{(\hat{h}^{(n)})^4}(\bar{X}_j^{(n)} + \hat{h}^{(n)})^2 \\
& \quad - \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^3}(\bar{X}_j^{(n)} + \hat{h}^{(n)}) - \frac{5c_j^{(n)}}{6(\hat{h}^{(n)})^2}\Big)\Big).
\end{aligned}$$

$$I_i = \varnothing, I_j = [\bar{X}_j^{(n)} - \hat{h}^{(n)}, X + \hat{h}^{(n)}]$$

$$\begin{aligned}
loss'(x) \;=\; & -\frac{8}{9(c_i^{(n)} + c_j^{(n)})}\Big(X^4 \frac{c_j^{(n)}}{6(\hat{h}^{(n)})^2} \\
& + X^3\Big(\frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_j^{(n)}) - \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^5}\Big) \\
& + X^2\Big(\frac{c_j^{(n)}}{(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^2 - \frac{2c_j^{(n)}}{(\hat{h}^{(n)})^5}(\hat{h}^{(n)} - \bar{X}_j^{(n)}) - \frac{c_j^{(n)}}{(\hat{h}^{(n)})^4}\Big) \\
& + X^1\Big(\frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^3 - \frac{2c_j^{(n)}}{(\hat{h}^{(n)})^5}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^2 - \frac{2c_j^{(n)}}{(\hat{h}^{(n)})^4}(\hat{h}^{(n)} - \bar{X}_j^{(n)}) \\
& \quad + \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^3}\Big) \\
& + X^0\Big(\frac{c_j^{(n)}}{6(\hat{h}^{(n)})^6}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^4 - \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^5}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^3 - \frac{c_j^{(n)}}{(\hat{h}^{(n)})^4}(\hat{h}^{(n)} - \bar{X}_j^{(n)})^2 \\
& \quad + \frac{2c_j^{(n)}}{3(\hat{h}^{(n)})^3}(\hat{h}^{(n)} - \bar{X}_j^{(n)}) + \frac{5c_j^{(n)}}{6(\hat{h}^{(n)})^2}\Big)\Big).
\end{aligned}$$

Appendix B

Appendix to the Experimental Evaluation

In this appendix, we present the densities of the synthetic streams we used in our experimental evaluation as well as the results of the range selectivity experiment for all techniques.

B.1 Densities of synthetic Streams

In our experiments, we used two synthetic streams, namely Claw and CP2. Their graphs are displayed in Figure 7.1. For both densities, a closed formula exists for their evaluation:
Claw:

$$\begin{aligned} f(x) &= \left(0.32 + 0.6x + 0.3e^{-100(x-0.3)^2}\right) I_{[0,0.8]}(x) \\ &\quad + \left(-0.28 + 0.6x + 0.3e^{-100(x-1.3)^2}\right) I_{(0.8,1]}(x) \end{aligned}$$

CP2:

$$f(x) = \begin{cases} 1 - \frac{35}{3}x + \frac{725}{9}x^2 - \frac{4000}{27}x^3, & 0 \le x \le \frac{3}{10} \\ -\frac{546979}{40399150} - \frac{653372272}{12119745}x + \frac{2949790460}{7271847}x^2 + \frac{6469685560}{7271847}x^3 + \frac{5500}{9}x^4, & \frac{3}{10} < x < \frac{6}{10} \\ -\frac{47}{5} + 45x - 60x^2 + 25x^3, & \frac{6}{10} \le x \le 1. \end{cases}$$

Given these densities, we utilized rejection sampling [140] to generate the corresponding synthetic streams. Basically, rejection sampling uses a given density f to generate an arbitrary number of elements which all follow the distribution of the random variable with density f. The basic idea of rejection sampling is to envelope f with a suitable multiple c of a different density g, i.e. $f(x) \le cg(x)$. A prerequisite is that we can easily draw samples from g. A value x drawn from g is accepted with probability $f(x)/cg(x)$. One can show that each accepted value follows the distribution of f.

B.2 Range Query Selectivity Estimation

In Section 7.4, we discussed the results of the range selectivity estimation experiment for a subset of the available techniques. For the sake of completion, the following charts display the results of the experiment for all techniques and all streams. Figure B.1 summarizes the line types of the techniques.

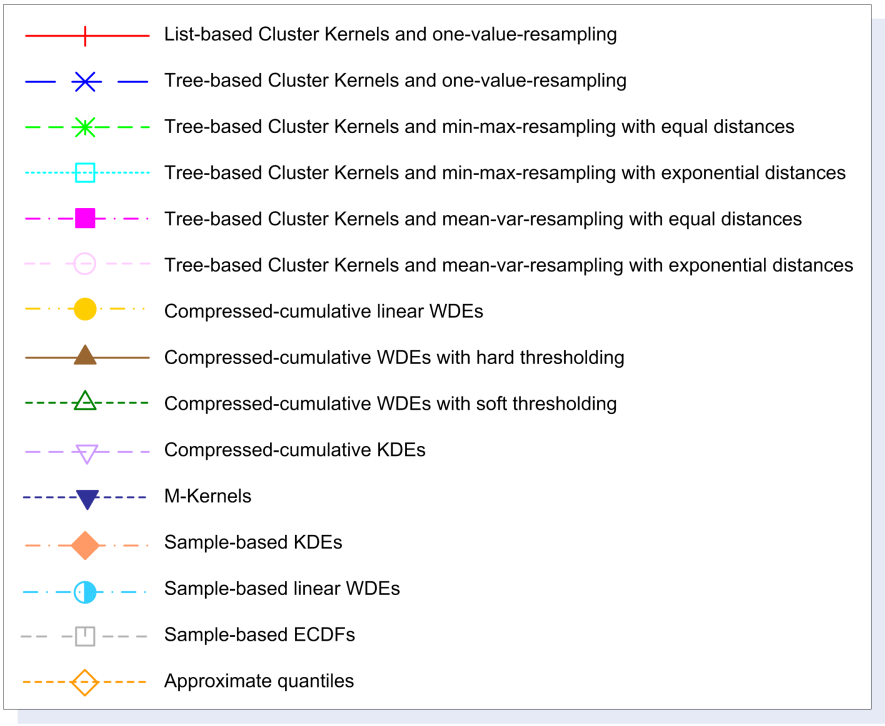


Figure B.1: Line types of the techniques for range query selectivity estimation

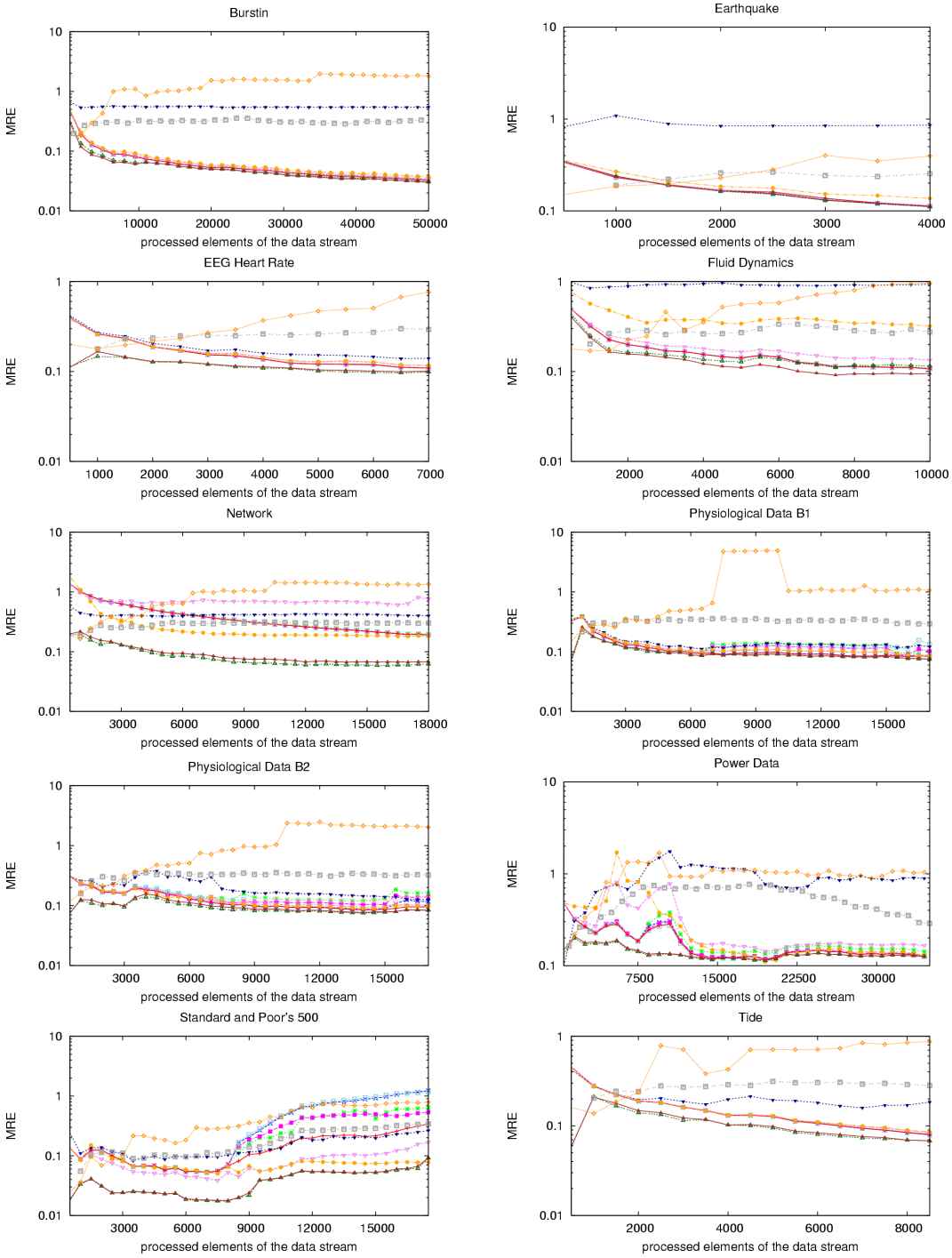


Figure B.2: Results of all techniques for range queries with selectivity 1%

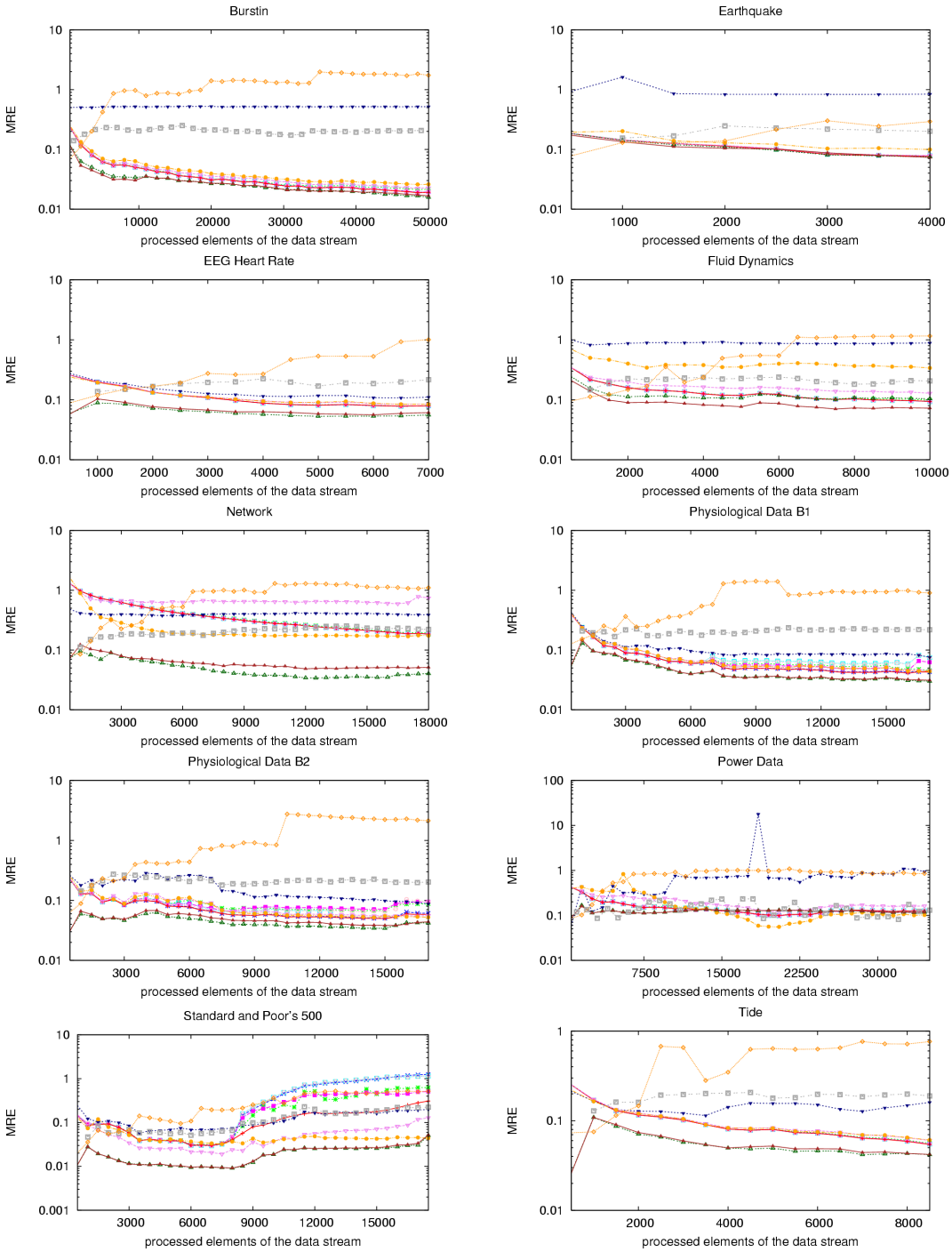


Figure B.3: Results of all techniques for range queries with selectivity 2%

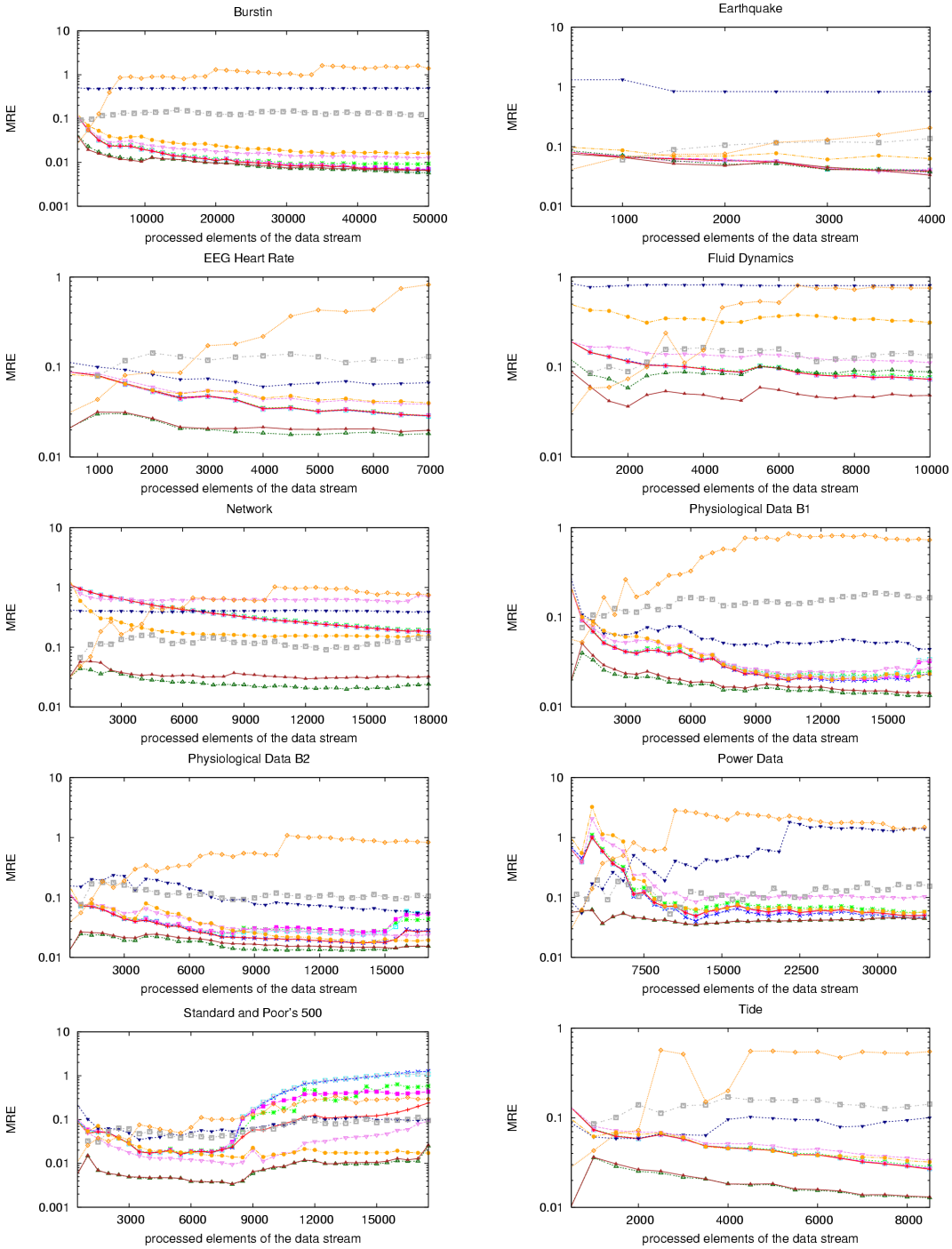


Figure B.4: Results of all techniques for range queries with selectivity 5%

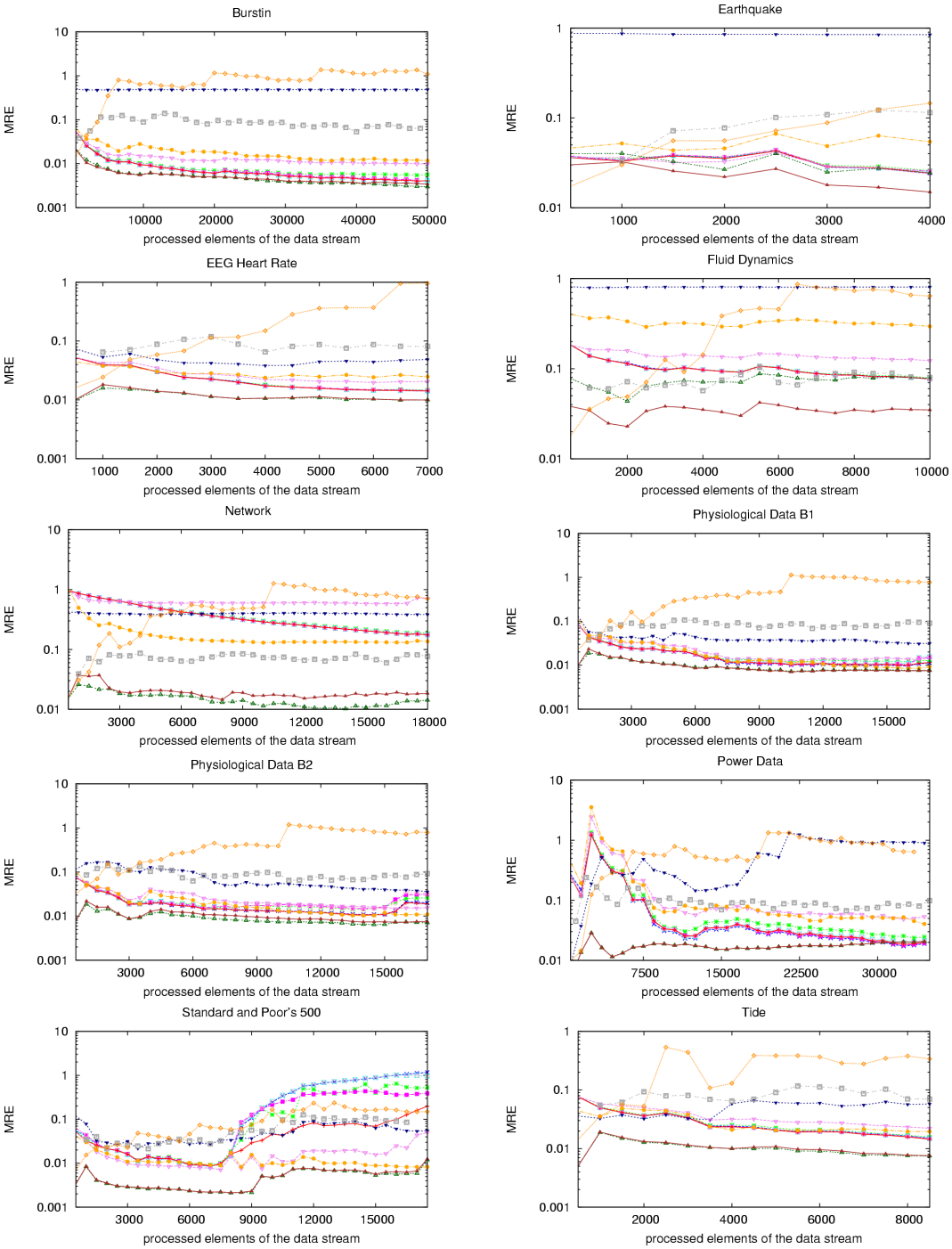


Figure B.5: Results of all techniques for range queries with selectivity 10%

List of Figures

List of Tables

Bibliography

[1] Colt - open source libraries for high performance scientific and technical computing in java. `hoschek.home.cern.ch/hoschek/colt`.

[2] Eclipse. `www.eclipse.org`.

[3] Java. `java.sun.com`.

[4] LHC - the Large Hadron Collider. `lhc.web.cern.ch/lhc/`.

[5] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12(2):120–139, 2003.

[6] C. Aggarwal. A framework for change diagnosis of data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 575–586, 2003.

[7] C. Aggarwal. On Biased Reservoir Sampling in the Presence of Stream Evolution. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 607–618, 2006.

[8] C. Aggarwal. *Data Streams: Models and Algorithms.* Springer, New York, 2007.

[9] C. Aggarwal, J. Han, J. Wang, and P. Yu. A Framework for Clustering Evolving Data Streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 81–92, 2003.

[10] C. Aggarwal, J. Han, J. Wang, and P. Yu. On demand classification of data streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 503–508, 2004.

[11] C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 852–863, 2004.

[12] N. Alon, Y. Matias, and M. Szegedy. The Space Complexity of Approximating the Frequency Moments. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 20–29, 1996.

[13] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(1):19–26, 2003.

[14] A. Arasu, B. Babcock, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *to appear in VLDB Journal*, 2005.

[15] L. Auvil, Y. D. Cai, D. Clutter, J. Han, G. Pape, and M. Welge. MAIDS: Mining Alarming Incidents from Data Streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 919–920, 2004.

[16] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002.

[17] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 234–243, 2003.

[18] S. Ben-David, J. Gehrke, and D. Kifer. Detecting Change in Data Streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 180–191, 2004.

[19] J. Beringer and E. Hüllermeier. Online clustering of parallel data streams. *Data Knowledge and Engineering*, 58(2):180–204, 2006.

[20] P. Billingsley. *Probability and Measure*. John Wiley and Sons, New York, 1995.

[21] B. Blohsfeld. *Online Aggregation über Datenströmen mit Verfahren der mathematischen Statistik in großen Datenbanksystemen*. PhD thesis, University of Marburg, 2002.

[22] B. Blohsfeld, C. Heinz, and B. Seeger. Maintaining Nonparametric Estimators over Data Streams. In *Proceedings of the GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW)*, 2005.

[23] B. Blohsfeld, D. Korus, and B. Seeger. A Comparison of Selectivity Estimators for Range Queries on Metric Attributes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 239–250, 1999.

[24] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying Density-Based Local Outliers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 93–104, 2000.

[25] A. Bulut and A. Singh. SWAT: Hierarchical Stream Summarization in Large Networks. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 303–314, 2003.

[26] Z. Cai, W. Qian, L. Wei, and A. Zhou. M-Kernel Merging: Towards Density Estimation over Data Streams. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 285–292, 2003.

[27] M. Cammert, C. Heinz, J. Krämer, T. Riemenschneider, M. Schwarzkopf, B. Seeger, and A. Zeiss. Stream Processing in Production-to-Business Software. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2006.

[28] M. Cammert, C. Heinz, J. Krämer, and B. Seeger. A Status Report on XXL - a Software Infrastructure for Efficient Query Processing. *IEEE Data Engineering Bulletin*, 26(2):12–18, 2003.

[29] M. Cammert, C. Heinz, J. Krämer, and B. Seeger. Datenströme im Kontext des Verkehrsmanagements. In *GI-Workshop on "Mobilität und Informationssysteme"*, 2003.

[30] M. Cammert, C. Heinz, J. Krämer, and B. Seeger. Anfrageverarbeitung auf Datenströmen. *Datenbankspektrum*, 11:5–13, 2004.

[31] M. Cammert, C. Heinz, J. Krämer, and B. Seeger. Sortierbasierte Joins über Datenströmen. In *Proceedings of the GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW)*, pages 385–404, 2005.

[32] M. Cammert, C. Heinz, J. Krämer, B. Seeger, S. Vaupel, and U. Wolske. Flexible multi-threaded scheduling for continuous queries over data streams. In *Proceedings of the First International Workshop on Scalable Stream Processing Systems*, 2007.

[33] M. Cammert, J. Krämer, B. Seeger, and S. Vaupel. A Cost Model for Adaptive Resource Management in Data Stream Systems. Technical Report 52, Department of Mathematics and Computer Science, University of Marburg, 2006.

[34] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring Streams - A New Class of Data Management Applications. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 215–226, 2002.

[35] T. F. Chan, G. H. Golub, and R. J. LeVeque. Algorithms for Computing the Sample Variance: Analysis and Recommendations. *The American Statistician*, 37(3):242–247, 1983.

[36] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proceedings of the International Conference on Innovative Data Systems Research (CIDR)*, 2003.

[37] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

[38] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 30–39, 2003.

[39] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 379–390, 2000.

[40] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 551–562, 2003.

[41] A. Coman, M. A. Nascimento, and J. Sander. Exploiting redundancy in sensor networks for energy efficient processing of spatiotemporal region queries. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 187–194, 2005.

[42] G. Cormode, M. Garofalakis, and D. Sacharidis. Fast Approximate Wavelet Tracking on Streams. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 4–22, 2006.

[43] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking frequent items dynamically. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 296–306, 2003.

[44] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[45] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A Stream Database for Network Applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 647–651, 2003.

[46] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. The Gigascope Stream Database. *IEEE Data Engineering Bulletin*, 26(1):27–32, 2003.

[47] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proceedings of the ACM-SIAM Symposium on Discrete Mathematics (SODA)*, pages 635–644, 2002.

[48] I. Daubechies. *Ten Lectures on Wavelets.* SIAM, Philadelphia, 1995.

[49] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration.* Academic Press, New York, 1975.

[50] A. Deshpande, C. Guestrin, and S. Madden. Using probabilistic models for data management in acquisitional environments. In *Proceedings of the International Conference on Innovative Data Systems Research (CIDR)*, pages 317–328, 2005.

[51] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 588–599, 2004.

[52] R. A. DeVore and B. J. Lucier. Wavelets. In *Acta Numerica 1*, pages 1–56. Cambridge University Press, 1991.

[53] C. Domeniconi, D. Gunopulos, G. Kollios, and V. J. Tsotras. Approximating Multi-Dimensional Aggregate Range Queries over Real Attributes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 463–474, 2000.

[54] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 106–113, 2001.

[55] P. Domingos and G. Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12, 2003.

[56] G. Dong, J. Han, L. V. S. Lakshmanan, J. Pei, H. Wang, and P. S. Yu. Online mining of changes from data streams: Research problems and preliminary results. In *Proceedings of the ACM SIGMOD Workshop on Management and Processing of Data Streams*, 2003.

[57] D. L. Donoho, I. M. Johnstone, G. Kerkyacharian, and D. Picard. Density estimation by wavelet thresholding. *The Annals of Statistics*, 24(2):508–539, 1996.

[58] W. Fan, Y. Huang, H. Wang, and P. S. Yu. Active mining of data streams. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2004.

[59] U. Fayyad, A. Gray, and P. Smyth. Retrofitting Decision Tree Classifiers Using Kernel Density Estimation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 506–514, 1995.

[60] M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *SIGMOD Record*, 34(2):18–26, 2005.

[61] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pages 406–431, 1993.

[62] V. Ganti, J. Gehrke, and R. Ramakrishnan. DEMON: Mining and Monitoring Evolving Data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(1):50–63, 2001.

[63] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining Data Streams under Block Evolution. *SIGKKD Explorations*, 3(2):1–10, 2002.

[64] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: you only get one look. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2002.

[65] M. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 476–487, 2002.

[66] M. Garofalakis and A. Kumar. Wavelet synopses for general error metrics. *ACM Transactions on Database Systems (TODS)*, 30(4):888–928, 2005.

[67] I. Gijbels, A. Pope, and M. Wand. Automatic forecasting via exponential smoothing: Asymptotic properties. Technical report, Humboldt University Berlin, SFB 373, 1997.

[68] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 389–398, 2002.

[69] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 79–88, 2001.

[70] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. One-Pass Wavelet Decompositions of Data Streams. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(3):541–554, 2003.

[71] L. Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.

[72] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[73] A. Gray and A. Moore. Nonparametric Density Estimation: Toward Computational Tractability. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2003.

[74] M. Greenwald and S. Khanna. Space-Efficient Online Computation of Quantile Summaries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 58–66, 2001.

[75] S. Guha and B. Harb. Wavelet synopsis for data streams: minimizing non-euclidean error. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 88–97, 2005.

[76] S. Guha, P. Indyk, N. Koudas, and N. Thaper. Dynamic multidimensional histograms. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 428–439, 2002.

[77] S. Guha, N. Koudas, and K. Shim. Data-Streams and histograms. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 471–475, 2001.

[78] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams. In *Symposium on Foundations of Computer Science (FOCS)*, pages 359–366, 2000.

[79] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *VLDB Journal*, 14(2):137–154, 2005.

[80] P. Hall, S. N. Lahiri, and Y. K. Truong. On bandwidth choice for density estimation with dependent data. *The Annals of Statistics*, 23(6):2241–2263, 1995.

[81] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 2001.

[82] W. Härdle, G. Kerkyacharian, D. Picard, and A. Tsybakov. *Wavelets, Approximation, and Statistical Applications.* Springer, New York, 1998.

[83] C. Heinz and B. Seeger. Wavelet density estimators over data streams. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 578–579, 2005.

[84] C. Heinz and B. Seeger. Exploring Data Streams with Nonparametric Estimators. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 261–264, 2006.

[85] C. Heinz and B. Seeger. Resource-aware kernel density estimators over streaming data. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 870–871, 2006.

[86] C. Heinz and B. Seeger. Statistical Modeling of Sensor Data and its Application to Outlier Detection. In *GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze"*, 2006.

[87] C. Heinz and B. Seeger. Stream Mining via Density Estimators: A concrete Application. In *Proceedings of the International Conference on Management of Data (COMAD)*, pages 223–226, 2006.

[88] C. Heinz and B. Seeger. Towards Kernel Density Estimation over Streaming Data. In *Proceedings of the International Conference on Management of Data (COMAD)*, pages 91–102, 2006.

[89] C. Heinz and B. Seeger. Adaptive Wavelet Density Estimators over Data Streams. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, 2007.

[90] J. Hellerstein, P. J. Haas, and H. J. Wang. Online Aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 171–182, 1997.

[91] G. Hulten and P. Domingos. Mining high-speed data streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[92] G. Hulten, L. Spencer, and P. Domingos. Mining Time-Changing Data Streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, 2001.

[93] H.-P. Hung and M.-S. Chen. Efficient range-constrained similarity search on wavelet synopses over multiple streams. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 327–336, 2006.

[94] C. Jin, W. Qian, C. Sha, J. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 287–294, 2003.

[95] T. Johnson, S. Muthukrishnan, and I. Rozenbaum. Sampling Algorithms in a Stream Operator. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2005.

[96] H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, and D. Handy. VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2004.

[97] P. Karras and N. Mamoulis. One-pass wavelet synopses for maximum-error metrics. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 421–432, 2005.

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

[99] G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold. Efficient biased sampling for approximate clustering and outlier detection in large data sets. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(5):1170–1187, 2003.

[100] F. Korn, T. Johnson, and H. Jagadish. Range Selectivity Estimation for Continuous Attributes. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 244–253, 1999.

[101] N. Koudas and D. Shrivastava. Data Stream Query Processing: A Tutorial. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2003.

[102] J. Krämer and B. Seeger. A temporal foundation for continuous queries over data streams. In *Proceedings of the International Conference on Management of Data (COMAD)*, pages 70–82, 2003.

[103] J. Krämer and B. Seeger. PIPES - A Public Infrastructure for Processing and Exploring Streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 925–926, 2004.

[104] S. Krishnamurthy, S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. Hellerstein, W. Hong, S. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: An Architectural Status Report. *IEEE Data Engineering Bulletin*, 26(1):11–18, 2003.

[105] C. Lambert, S. Harrington, C. Harvey, and A. Glodjo. Efficient on-line nonparametric kernel density estimation. *Algorithmica*, 25(1):37–57, 1999.

[106] F. Leblanc and A. Juditsky. Computing wavelet density estimator for stochastic processes, 1994.

[107] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. Semantics and Evaluation Techniques for Window Aggregates in Data Streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 311–322, 2005.

[108] T. Li, Q. Li, S. Zhu, and M. Ogihara. A survey on wavelet applications in data mining. *SIGKDD Explorations*, 4(2):49–68, 2002.

[109] X. Lin, H. Lu, J. Xu, and J. X. Yu. Continuously Maintaining Quantile Summaries of the Most Recent N Elements over a Data Stream. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 362–373, 2004.

[110] B. Lindsay, G. Manku, and S. Rajagopalan. Approximate Medians and other Quantiles in One Pass and with Limited Memory. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 426–435, 1998.

[111] H. Liu, J. Srivastava, and S.-Y. Hwang. PSRA: A Data Model for Managing Data in Sensor Networks. *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, 2006.

[112] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 767–778, 2005.

[113] J. S. Marron and M. P. Wand. Exact mean integrated squared error. *The Annals of Statistics*, 20(2):712–736, 1992.

[114] J. M. Martin, K. Mengersen, and C. P. Robert. Bayesian modelling and inference on mixtures of distributions. *Handbook of Statistics*, 25, 2006.

[115] Y. Matias and D. Urieli. Optimal workload-based weighted wavelet synopses. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 368–382, 2005.

[116] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions.* John Wiley and Sons, New York, 1996.

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

[118] S. Muthukrishnan. Data streams: algorithms and applications, 2003.

[119] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[120] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2002.

[121] University of Marburg: The Database Research Group. XXL - eXtensible and fleXible Library. `www.xxl-library.de`.

[122] R. T. Ogden. *Essential Wavelets for Statistical Applications and Data Analysis.* Birkhäuser Boston Inc., Cambridge, MA, USA, 1996.

[123] V. Poosala. *Histogram-Based Estimation Techniques in Database Systems.* PhD thesis, University of Wisconsin, 1997.

[124] C. M. Procopiuc and O. Procopiuc. Density Estimation for Spatial Data Streams. In *Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD)*, pages 109–126, 2005.

[125] M. Rafanelli, A. Bezenchek, and L. Tininini. The aggregate data problem: A system for their definition and management. *SIGMOD Record*, 25(4):8–13, 1996.

[126] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Boston, MA, USA, 1990.

[127] S. Schmidt, T. Legler, S. Schär, and W. Lehner. Robust Real-time Query Processing with QStream. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1299–1302, 2005.

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

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

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

[131] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 187–198, 2006.

[132] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 309–320, 2003.

[133] K. Tribouley. Practical estimation of multivariate densities using wavelet methods. *Statistica Neerlandica*, 49(1):41–62, 1995.

[134] Y.-C. Tu, S. Liu, S. Prabhakar, and B. Yao. Load shedding in stream databases: A control-based approach. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 787–798, 2006.

[135] B. Turlach. Bandwidth Selection in Kernel Density Estimation: A Review, 1993.

[136] J. van den Bercken, B. Blohsfeld, J.-P. Dittrich, J. Krämer, T. Schäfer, M. Schneider, and B. Seeger. XXL-A Library Approach to Supporting Efficient Implementations of Advanced Database Queries. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 39–48, 2001.

[137] M. Vannucci. Nonparametric density estimation using wavelets. Technical report, Duke University, 1998.

[138] B. Vidakovic. *Statistical Modeling by Wavelets.* John Wiley and Sons, New York, 1999.

[139] J. S. Vitter. Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[140] J. von Neumann. Various techniques used in connection with random digits. *Applied Mathematics Series*, 12:36–38, 1951.

[141] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235, 2003.

[142] Y. Wang. Jump and sharp cusp detection by wavelets. *Biometrika*, 82(2):385–397, 1995.

[143] F. Zhao and L. Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, San Francisco, 2004.

[144] W. Zhu, J. Pei, J. Yin, and Y. Xie. Granularity adaptive density estimation and on demand clustering of concept-drifting data streams. In *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery (DaWak)*, pages 322–331, 2006.

[145] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 358–369, 2002.

[146] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 336–345, 2003.

Acknowledgements

First of all, I want to thank my advisor, Prof. Dr. Bernhard Seeger, for his excellent guidance through my phd studies. I really appreciate the patience, support, and encouragement he gave me in the last four years.

I am very grateful to the members of our database research group, in particular Jürgen Krämer, Michael Cammert, Tobias Riemenschneider, and Sonny Vaupel, for the helpful discussions and the pleasant working atmosphere. I also owe a thank to Ben Mills for his careful proofreading of parts of this thesis.

Most of all, I want to thank my sweetheart Christine for her loving support. She always tried to make the process of developing and writing this thesis a comfortable one for me.

Curriculum Vitae

Contact Information

Christoph Heinz
Im Köhlersgrund 20
35041 Marburg, Germany
Email: heinzch@mathematik.uni-marburg.de

Personal Information

Date of Birth: 27.01.1976
Place of Birth: Weilburg, Germany
Citizenship: german

Education

1996–2003	*Studies of Business Mathematics (Wirtschaftsmathematik)* University of Marburg, Germany
1995	*Abitur (higher education entrance qualification)* Fürst-Johann-Ludwig-Schule, Hadamar, Germany

Employment History

2003–2007	*Research Assistant* DFG-funded project "Anfrageverarbeitung aktiver Datenströme" under the direction of Prof. Dr. Bernhard Seeger, Department of Mathematics and Computer Science, University of Marburg, Germany
2001–2003	*Freelancer* System administration, Helaba Trust, Frankfurt, Germany
2000	*Internship* System administration, Dresdner Bank, Frankfurt, Germany

Marburg, May 5, 2007