Estimating Mutual Information on Data Streams

Fabian Keller° Emmanuel Müller°• Klemens Böhm°

°Karlsruhe Institute of Technology (KIT), Germany
{fabian.keller, emmanuel.mueller, klemens.boehm}@kit.edu

•University of Antwerp, Belgium
emmanuel.mueller@ua.ac.be

ABSTRACT

Mutual information is a well-established and broadly used concept in information theory. It allows to quantify the mutual dependence between two variables – an essential task in data analysis. For static data, a broad range of techniques addresses the problem of estimating mutual information. However, the assumption of static data is not applicable for today's dynamic data sources such as data streams: In contrast to static approaches, an online estimator must be able to deal with the evolving, changing, and infinite nature of the stream. Furthermore, some tasks require the estimation to be available online while processing the raw data stream. Our proposed solution MISE (Mutual Information Stream Estimation) allows a user to issue mutual information queries in arbitrary time windows. As a key feature, we introduce a novel sampling scheme, which ensures an equal treatment of queries over multiple time scales, e.g., ranging from milliseconds up to decades. We thoroughly analyze the requirements of such a multiscale sampling scheme, and evaluate the resulting quality of MISE in a broad range of experiments.

1. INTRODUCTION

In information theory, mutual information is a ubiquitous measure for the mutual dependence of two random variables. Intuitively, mutual information $I(X,Y)$ is equal to the reduction of uncertainty on one random variable X given knowledge of another variable Y. It is a symmetric measure, i.e., $I(X,Y) = I(Y,X)$. A high mutual information indicates a large reduction of uncertainty, i.e., the variable pair shows a strong mutual dependence. Compared to other dependence measures, like for instance Pearson or Spearman correlation, mutual information is not limited to specific kinds of dependence, e.g., linear or monotonous, but captures every possible type of dependence. Because of these properties, it has a long history in both theory and applications. Proper estimation of mutual information from real-valued data is a non-trivial problem and has been covered in the literature extensively. Amongst the traditional mutual information estimators, Kraskov estimation has emerged as a leading approach [22, 19, 30, 20]. Consequently, we build upon this estimation principle in this work. Traditional estimation algorithms however focus on the case of a fixed, static data sample. The notion of time is not considered explicitly. This is a fundamental difference to data originating from a stream. By its nature, a data stream is evolving and changing over time, is infinite, and comprises multiple time scales. Given these properties, the analysis of data streams has become a challenging task in the database research community.

To illustrate, think of analyzing the mutual dependence in a stream of stock prices. Detecting a mutual dependence of stocks provides important information for financial analysis, investment management, or return prediction. In general, the mutual dependence between stocks fluctuates over time, and one may observe periods of high or low mutual information. Furthermore, the changes in mutual dependence may occur on a broad range of time horizons: In some cases a mutual (in-)dependence lasts for decades, while in other cases a dependence appears and disappears within seconds. An analyst might for instance find a dependence of a pair of stocks in July. This leads to questions like whether the dependence did also exist in June, when it has appeared first, or whether it also exists on different time scales like a yearly time horizon or when looking at hours or minutes. Overall, we make the following key observation: Since the dependencies are dynamic, each analyst may be interested in a different time window. That is, analysts want to estimate mutual information based on an arbitrary window size, and the window may be shifted arbitrarily into the past.

Challenges. This observation has a direct implication when designing a data stream management system (DSMS) that supports mutual information queries: The DSMS must allow a user to explicitly specify the query window boundaries individually for each query. In general, such queries are so-called ad-hoc one-time queries [3], and they are most challenging since this is the most general type of query. Supporting such queries even raises the question: Is it possible at all to answer mutual information queries in *any* window without storing the entire data stream? Naively, one could approach the problem by (1) storing the entire stream and (2) running a static mutual information estimator for every incoming query. Clearly, this naive approach has severe limitations and is not in line with the so-called "streaming model" [3]: First, storing the stream obviously contradicts the idea of stream processing. The second issue affects query

SSDBM '15, June 29 - July 01, 2015, La Jolla, CA, USA

ISBN 978-1-4503-3709-0/15/06…$15.00

DOI: http://dx.doi.org/10.1145/2791347.2791348

performance; we illustrate it using our example scenario: If there are many analysts working simultaneously, the DSMS has to answer many queries as well; the rate of incoming queries may even exceed the data rate of the stream. In such a case, the naive approach will collapse since it has to expensively recompute a mutual information estimate for every single query – even if the windows of two queries have a significant overlap. Therefore, a challenge is to develop a summarization data structure which provides aggregated information that is useful for many queries.

Our Contributions. In this work, we propose the framework MISE (Mutual Information Stream Estimation) which tackles the challenge of answering mutual information queries in arbitrary time windows. To avoid storing the whole data stream, we exploit the *multiscale* nature of time. We illustrate the idea in our example scenario: For financial analyses, time scales can vary significantly, ranging from seconds right up to years or decades. In such analyses, the query window size and the amount the query window is shifted into the past often show a certain relationship. We exploit this by dividing the space of all possible queries into multiscale equivalence classes depending on the ratio of the window size w and the offset o into the past. For instance, the following two queries are equivalent: (I) a query with $w = 1$ second and an offset of $o = 5$ seconds, and (II) a query with $w = 5$ years and an offset of $o = 25$ years. In this work we will tackle the essential question that arises with multiscale equivalence: How can a DSMS answer equivalent queries with equal quality? As a key contribution we provide a solution to this question by deriving the proper sampling distribution out of this requirement. We will see that the common principle of *more detail on more recent data* emerges naturally as a result. Based on the sampling distribution required, we develop two different *multiscale sampling schemes* which have either constant or logarithmic complexity over time. They are the first sampling schemes that inherently provide equal quality over multiple time scales.

As another important contribution, we introduce the notion of a *query anchor*, which is a novel dynamic data structure for mutual information estimation. In a nutshell, a query anchor keeps track of quantities that allow to estimate mutual information according to the Kraskov principle. These quantities include nearest neighbor relationships and counts of data points in the marginal distributions. While the computation of these quantities is straightforward on static data, the challenge with data streams becomes: To obtain an efficient estimation, it is necessary to keep track of all changes in these quantities over time. By proposing the query anchor data structure, we solve this problem and enable an incremental computation of these quantities. Consequently, query anchors provide aggregates that can be used for different queries. This leads to a significant speed-up of query execution time.

Summing up, our contributions to deal with the challenges mentioned are as follows: We deal with

- **the stream's dynamic nature** by design, i.e., by allowing the user to query the stream in arbitrary windows,
- **the stream's infinite** and **multiscale nature**, by introducing the novel multiscale sampling paradigm,
- **a large number of online queries**, by efficient incremental computations within our query anchor data structure.

Furthermore, we provide a detailed analysis of both our multiscale sampling schemes and the query anchor data structure. To complement our analysis, we demonstrate the high quality of MISE in a broad range of experiments, including several real-world scenarios.

2. STATIC ESTIMATION PARADIGMS

Estimation of mutual information on static data has been studied in many publications, including several surveys [28, 30, 19]. Estimators can be categorized according to the underlying formula of the estimation. The first estimation paradigm is based on the integral definition of mutual information:

$$I(X,Y) = \iint p(x,y) \log \frac{p(x,y)}{p_X(x)p_Y(y)}\, dy\, dx \qquad (1)$$

where $p(x,y)$ is the joint probability density function, and marginal distributions are denoted as $p_X(x)$ and $p_Y(y)$. Estimators of this type replace these theoretical functions by density estimates; they are hence called plug-in estimates [28]. A common problem of such estimators is that, since the underlying distributions are unknown, they are prone to underestimating the variability of the distributions based on a finite sample. This leads to a heavily biased estimate of mutual information, which has been studied extensively [27, 29, 11, 13]. Furthermore, to the best of our knowledge, there is no general purpose density estimation technique that allows to query density estimates in arbitrary windows over a data stream. Therefore, we will focus on the second estimation paradigm in this work. Estimators of this kind are based on the entropic definition of mutual information:

$$I(X,Y) = H(X) + H(Y) - H(X,Y) \qquad (2)$$

where $H(.)$ denotes entropy. Therefore, estimating mutual information can be achieved by an estimation of entropy. The Kozachenko-Leonenko-Estimator [21] is a famous non-parametric approach to estimate entropy based on nearest neighbor information. The problem regarding estimation of mutual information is that the errors of estimating the marginal and the joint entropies do not cancel. This was solved by Kraskov et al [22], leading to a mutual information estimator with excellent estimation properties: Comparative studies [19, 30, 20, 22] have shown that the Kraskov estimator (1) shows a very fast convergence, (2) is unbiased in case of independent variables, and (3) shows very low bias in general compared to estimators of the first paradigm. However, it is an open research question how to incorporate its principles into the estimation process for data streams. Our goal is to make these favorable estimation properties available for online processing.

3. RELATED WORK

Analyzing related work shows that certain issues recur. Therefore, we first summarize recurring limitations before analyzing related work in detail.

• FIXEDWINDOW: Many data stream techniques do not allow the user to specify arbitrary query windows. Summarization techniques typically maintain a synopsis aggregated either over the whole stream, a sliding window, or – as generalization of these paradigms – aggregated based on a certain (smooth) time decay function. In either case the scope in time is fixed, i.e., the "query window" is inherently bound to

the aggregate computation. In particular, most techniques focus on keeping track of the *most recent* aggregate value. This prevents the user from querying the aggregate in any window that is strictly in the past.

• LimitedDomain: Many data stream summarization techniques are exclusively designed to operate on data streams consisting of discrete items or integer values within a limited range. Compared to real-valued attributes, the finite attribute domain simplifies any summarization task since it allows to operate on item frequencies, which again allows to make use of various sketching techniques [7]. From an estimation theoretic perspective, the major challenge of estimating mutual information on continuous data is a result of the infiniteness of the attribute domain. Therefore, making any assumptions regarding the domain is not feasible when constructing a general purpose estimator.

• Univariate: Many of the techniques discussed below are designed for summarizing a single univariate stream. In order to leverage them to estimate mutual information, it would be necessary to modify them to the bivariate case. In many cases such a modification is non-trivial or impossible.

• Biased: Many ideas discussed below would result in a mutual information estimator based on Equation 1, and would come with all the issues discussed in Section 2.

Estimation Foundations. We now review stream summarization techniques as proposed in the scientific literature that may serve as a foundation for computing a mutual information estimate. For instance, one might be tempted to leverage techniques which summarize quantiles to estimate mutual information. This problem of summarizing ϵ-approximate quantiles has been solved for both the single pass [17] and the sliding window [2] paradigms. Such an approach would suffer from FixedWindow (inability to specify arbitrary queries) and Biased (due to the binning characteristic of quantiles), and most notably it remains a non-trivial problem to extend the notion of summarizing one dimensional quantiles to the bivariate case (Univariate). Similarly, maintaining histograms as a summary [12, 16] suffers from FixedWindow and Biased as well. Another problem that has been addressed is estimating entropy over data streams [23, 4, 6]. Even if there might be (non-trivial) solutions to issues FixedWindow and Univariate for these techniques, a severe problem remains: The techniques heavily rely on the assumption of a limited attribute domain (LimitedDomain). Overall we can see that all existing summarization techniques are affected by several issues. This highlights the necessity to develop a novel summarization data structure.

Correlation Analysis. Mutual information in the broader context of (pair-wise) dependence measures in general is related to work on online correlation tracking. In particular the so-called all-strong-pairs correlation query problem [31] has been solved for data streams [34, 35]. While issues FixedWindow and LimitedDomain apply for these techniques as well, the major difference is the problem statement itself: Compared to linear binary correlations, mutual information can capture much more complex dependence types. Also note that the problem considered in [14] is entirely different: They investigate pointwise mutual information (PMI), a formalization of the psycholinguistic association score, and try to find word-pairs with the highest PMI in a vocabulary. Technically, there is no connection to our much more general problem.

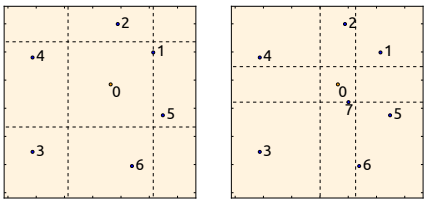
Sampling. The principle of *more detail on recent data* plays an important role in many approaches on data streams. This concept has been applied for instance to the problem of maintaining specific aggregates according to a time-decay weight function [10, 8, 9], and to sampling with a weighted reservoir [15, 1]. However, none of these approaches derives the weight function from quality requirements on the queries. As a major contribution we will derive the particular weight function that is required to ensure the equal treatment of queries over multiple time scales. We will see that this basic requirement results in a unique dynamic weight function, which does not allow a straightforward application of existing weighted sampling schemes.

Nearest Neighbor Querying. Section 4.2 will show that estimating mutual information according to the Kraskov principle requires knowing the nearest neighbors. Thus, our approach is remotely related to work on nearest neighbor (NN) monitoring in spatio-temporal databases. Given a set of objects and a set of query points, continuous k-NN querying addresses the case that both the objects and query points move over time. Techniques like [18] make strong assumptions on the trajectories of the objects, e.g., they must move with a constant velocity. Later work [26, 33, 32] relaxes these assumptions, but does not explicitly consider the appearance/disappearance of objects or queries. To some degree this has been addressed in [25] (by incorporating a sliding window model into continuous queries) and [5] (by allowing objects/queries to expire after a certain time). While these concepts are somewhat similar with what is needed here, fundamental differences remain. We will discuss these differences after we have proposed an exact problem statement in Section 4.1.

4. PROPOSED APPROACH

In what follows, we will structure the presentation of our approach into three steps. First, we introduce a summarization data structure, a so-called query anchor, which is responsible for collecting the dynamics of nearest neighbor relationships in the data stream (Section 4.1). As the next step, we move to the bigger picture, by explaining how the overall algorithm makes use of these query anchors (Section 4.2). We will see that, once we are able to keep track of the changes in nearest neighbors over time, we can extract an online mutual information estimation from the query anchors. Finally we turn to the question of how to solve the challenges introduced by the infinite and multiscale nature of the stream (Section 4.3). Our solution to this problem will exploit the equivalence of multiple time scales for sampling.

Subject of our analysis is data streams formed by a pair of one-dimensional continuous random variables X and Y. We do not make any assumption on the underlying distributions of X and Y. We assume a fixed sampling rate of the data stream, i.e., samples arrive after a fixed time interval. Extending our approach to variable-rate data streams or more than two variables is part of future work. We denote the pair of realizations at time t as $Q_t = \langle X_t, Y_t \rangle$, where X_t and Y_t are the samples at time t. We will refer to subsequences of the data stream with the notation $\mathcal{Q} = \{Q_{t_1}, Q_{t_2}, \ldots\}$. In general a subsequence $\mathcal{Q}$ can be sparse, i.e., does not necessarily contain consecutive data samples. In order to constrain a subsequence to a certain time window starting



t	MC_x	MC_y
0	0	0
1	0	1
2	1	1
3	1	1
4	1	2
5	1	3
6	2	3
7	2	0

$Q_0 = \langle 5.5, 5.6 \rangle, Q_1 = \langle 7.0, 7.0 \rangle, Q_2 = \langle 5.6, 9.0 \rangle, \ldots$

Figure 1: Example showing incremental effects on nearest neighbors, marginal points, and marginal counts

at t_s and ending at t_e, we use the following notation:

$$\mathcal{Q}_{t_e}^{t_s} \equiv \{Q_t \in \mathcal{Q} \mid t_s \leq t \leq t_e\}$$

Regarding time points, our convention is to use the time t_0 to refer to the present time, i.e., the current or most recent time point available.

4.1 Query Anchors

We now introduce the summarization data structure that we use to collect information from the data stream. The computation of a mutual information estimate according to the Kraskov principle requires knowledge of nearest neighbor relationships in both the joint and the marginal spaces. While the computation is straightforward in the case of static data, it becomes a challenge in the online case: Nearest neighbor relationships are no longer static but inherently change over time, making it necessary to incrementally track changes over time. We will model these dynamics in the following.

Definition 1. Distance: We define the distance between two data points Q_t and $Q_{t'}$ according to the maximum norm denoted as:

$$dist(Q_t, Q_{t'}) \equiv max\left(|X_t - X_{t'}|, |Y_t - Y_{t'}|\right)$$

Using the maximum norm is in line with Kraskov and ensures that estimation errors cancel each other out [22].

Definition 2. k Nearest Neighbor Distance: We define the k nearest neighbor distance of Q_t for a subsequence $\mathcal{Q}$ as the distance to the k nearest neighbor of Q_t in $\mathcal{Q}$. The k nearest neighbor is a point $Q_{t^\star} \in \mathcal{Q}$ satisfying:

$$|\{Q_{t'} \in \mathcal{Q} \setminus \{Q_t\} \mid dist(Q_t, Q_{t'}) < dist(Q_t, Q_{t^\star})\}| < k$$

$$|\{Q_{t'} \in \mathcal{Q} \setminus \{Q_t\} \mid dist(Q_t, Q_{t'}) \leq dist(Q_t, Q_{t^\star})\}| \geq k$$

We denote the k nearest neighbor distance as follows:

$$kNND(Q_t, \mathcal{Q}) \equiv dist(Q_t, Q_{t^\star})$$

When the subsequence does not have a length of k, the k nearest neighbor distance is undefined. Please note that for the definition of $kNND$ it is irrelevant whether the k nearest neighbor is unique.

We illustrate these definitions by an example given in Figure 1. It shows how a nearest neighbor relationship can evolve over time. For simplicity we consider the $k = 1$ nearest neighbor. Our point of reference is the point at time $t = 0$, located near the center in the plots. The points are labeled according to their time of occurrence in the stream. The left plot shows the subsequence $\mathcal{Q}_6^0$, i.e., contains all points Q_t with $0 \leq t \leq 6$. We can see that the $k = 1$ nearest neighbor up to time $t = 6$ is the data point Q_1.

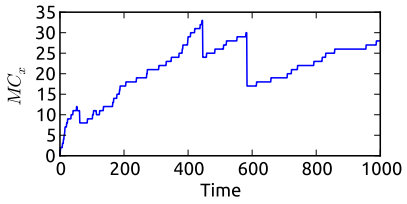


Figure 2: Evolvement of marginal counts over time

The square centered on our point of reference corresponds to $kNND(Q_0, \mathcal{Q}_6^0)$. We illustrate this $kNND$ as dashed slices. The next plot shows the subsequence extended by one data point. This leads to an update of the nearest neighbor, reducing $kNND$ given $\mathcal{Q}_7^0$.

Based on the nearest neighbor information, we now define the notion of marginal points, which plays a key role in the estimation process:

Definition 3. Marginal Points: Given a point of reference Q_t, we call a data point $Q_{t'} \neq Q_t$ an X-marginal point of Q_t if and only if

$$|X_t - X_{t'}| < kNND(Q_t, \mathcal{Q})$$

where $\mathcal{Q}$ is a subsequence containing both Q_t and Q_t'. We define the marginal points w.r.t. Y correspondingly.

We illustrate this notion using Figure 1. Intuitively, marginal points are points that fall into the slices corresponding to the $kNND$ box. For the subsequence $\mathcal{Q}_6^0$ (left plot) and our point of reference Q_0, we identify Q_1, Q_4, and Q_5 as marginal points in Y. With respect to X, only Q_2 and Q_6 are marginal points – Q_1, which defines the k nearest neighbor distance itself, is not included due to the "less than" condition. In the second plot corresponding to subsequence $\mathcal{Q}_7^0$ we observe that some points have lost their marginal point property due to the update of $kNND$. In the Y direction for instance, all three former marginal points are no longer located within the slice.

Definition 4. Marginal Counts: Given a point of reference Q_t and a subsequence $\mathcal{Q}$ containing Q_t, we define the marginal counts as the number of marginal points in the subsequence, i.e.:

$$MC_x(Q_t, \mathcal{Q}) = |\{Q_{t'} \in \mathcal{Q} \mid Q_{t'} \text{ is } X\text{-marginal point of } Q_t\}|$$

Accordingly, we refer to the number of Y-marginal points in $\mathcal{Q}$ as MC_y.

The table on the right in Figure 1 shows the development of the marginal counts over time. In the following discussion we focus on only one dimension (Y w.l.o.g.). When a new data point arrives, there are in general three possibilities: (1) The data point does not fall into the current Y slice, leaving MC_y unchanged. (2) The data point falls into the slice but has no influence on the current $kNND$. This increments MC_y by one. (3) The data point leads to an update of the $kNND$. Note that for $k > 1$, the new point does not have to be the new best nearest neighbor itself; it can take any position within the top-k ranked neighbors. In general this will result in a new distance of the neighbor on rank k. After such a $kNND$-update, the marginal points have to be re-evaluated. In general the decrease in the k nearest neighbor distance means that MC_y may drop to a lower value. Figure 2 shows an exemplary plot of MC_y over

time, summarizing these dynamics of the marginal count: As long as $kNND$ is unchanged, MC_y increases monotonically; updates of $kNND$ lead to sudden drops of MC_y. We will further analyze the growth rate of marginal counts over time in our complexity analysis in Section 5.

In order to handle these dynamics of marginal counts, we now define the notion of a query anchor:

Definition 5. Query Anchor: We define a query anchor as a data structure that precomputes and stores marginal counts. It is associated with a certain data point Q_t, i.e., is located at time t and has knowledge on X_t and Y_t. A query anchor provides

- a method INSERTRIGHT($Q_{t'}$) which adds a data point $Q_{t'}$ in forward time direction, i.e., $t' > t$,
- a method INSERTLEFT($Q_{t'}$) which adds a data point $Q_{t'}$ in backward time direction, i.e., $t' < t$,
- a method QUERY(t_1, t_2) which returns the marginal counts $MC_x(Q_t, \mathcal{Q}_{t_2}^{t_1})$ and $MC_y(Q_t, \mathcal{Q}_{t_2}^{t_1})$, and the total number $N \equiv |\mathcal{Q}_{t_2}^{t_1}|$ of data points that have been shown to the query anchor by its insert operations.

Note that in general the number N can be smaller than the window size $t_2 - t_1$ if the query anchor has only seen a sparse subsequence of the data. Compared to our example from Figure 1, a query anchor differs in the sense that it has to keep track of the marginal counts in both time forward and time backward direction. We will turn to the question of implementing query anchors in Section 5, providing a solution to efficiently store marginal counts in both time directions.

Differences to Continuous k-NN Queries. Having formulated the problem statement, it becomes clear that the problem has fundamental differences to work on continuous k-NN queries:

- There, a continuous query always targets at the *current* state. Here in turn, a query anchor has to evaluate queries w.r.t. *any* time window containing the query anchor.
- Symmetry of time directions: For a query anchor, the notion of time splits into a time forward and backward component. Work on continuous queries does not consider this issue. We will show that it is possible to exploit this symmetry of both time directions in an implementation (Section 5).
- On the other hand, an issue not explicitly studied here, but addressed in related work on spatio-temporal databases is the mobility of objects/queries. Our specific work does not need to take it into account, since both data objects and query points are simply measured values, which cannot change in retrospect.

Overall, these differences highlight that our concept of query anchors is orthogonal to work on continuous k-NN queries.

4.2 MISE Framework

Our query anchor data structure provides an abstraction over the dynamics of marginal counts observed in a data stream. This abstraction allows to formulate the Kraskov estimation principle [22, 21] in the online context:

Definition 6. Mutual information estimate: Given a query anchor for Q_t and a subsequence $\mathcal{Q}_{t_2}^{t_1}$ with $t_1 \leq t \leq t_2$, the mutual information estimate is defined as follows:

$$\hat{I} \;=\; \psi(k) - \psi(MC_x + 1) - \psi(MC_y + 1) + \psi(N) \qquad (3)$$

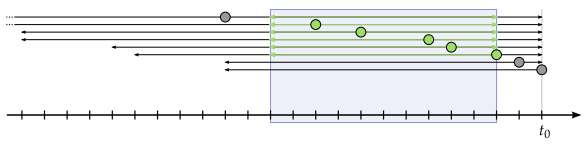


Figure 3: Illustration of MISE

where ψ is the digamma function, N is the length of the subsequence that the query anchor has seen, and MC_x, MC_y are the marginal counts returned by QUERY(t_1, t_2).

For the theoretical background behind Equation 3 we refer to [22]. Briefly sketched, the idea of the Kraskov principle is to formalize the probability that there are $k-1$ objects with a distance lower then $kNND$ and $N-k-1$ objects with a distance exceeding $kNND$. This probability can then be plugged into the integral definition of entropy, leading to a mutual information estimate via Equation 2.

Obviously an estimation based on a single query anchor has a large statistical uncertainty. This statistical error can be reduced significantly by taking the average of the estimates from several query anchors. We will exploit this idea in our MISE framework, which we describe in the following.

The MISE framework (cf. Algorithm 1) provides two operations: (1) an INSERT operation to add data from the stream into the system, and (2) a QUERY operation which retrieves a mutual information value for a certain query window. Internally MISE stores a sample of query anchors. This query anchor sample is modified by a SAMPLING function responsible for the deletion of query anchors. We will discuss the instantiation of this SAMPLING function in Section 4.3 and continue with an explanation of the INSERT and QUERY operations.

The INSERT operation first creates a new query anchor which corresponds to the data point Q_t just received. We then perform a forward and reverse initialization: The forward initialization performs an INSERTRIGHT operation on all existing query anchors for the new element Q_t. In other words, we show the new element to all existing query anchors in the current sample. The reverse initialization on the other hand adds data points corresponding to the existing anchors to the new anchor by using the INSERTLEFT operation. Finally, we add the query anchor to the sample and invoke a SAMPLING function. In general, the SAMPLING function modifies the current anchor sample by deleting certain anchors according to the sampling scheme we will present in Section 4.3. An exemplary result after performing several insert operations is illustrated in Figure 3. Each circle corresponds to a query anchor, and the positioning shows the distribution of the query anchor sample over time. The black arrows indicate how much information was added to a query anchor by either INSERTLEFT or INSERTRIGHT. Note that in reverse time direction (INSERTLEFT) the data points are filled sparsely, i.e., not every data point covered by the arrow was actually inserted into the query anchor. In time forward direction on the other hand, all data points can be inserted. In terms of this illustration, the INSERT operation (1) adds a new query anchor at t_0, (2) extends the arrows of existing anchors by one step to the right, (3) extends the arrow of the new query anchor to the left, up to the position of the oldest query anchor, and (4) modifies the sample.

The QUERY operation first determines the query anchors

that are contained in the query window. We query each anchor in the window for the marginal counts MC_x and MC_y, and the number of data points N that a query anchor has seen in the given window. Overall, we obtain a mutual information estimate from each anchor by Equation 3 and return the arithmetic sample mean of these estimates. Figure 3 illustrates the query operation: The blue shaded area corresponds to an exemplary query window. The green arrows show the query ranges that are used to obtain the marginal counts of the anchors within the query window. Note that for different query windows or a different query anchor distribution it is possible that the green arrows do not extend fully over the query window. In this case the query anchor has only seen a subsample of the whole window, which can still contribute valuable information to the estimation.

Algorithm 1 MISE framework

```
1:  anchors ← {}
2:  procedure INSERT(Q_t)                          ▷ interface to add data
3:      a ← new query anchor at Q_t
4:      for all o ∈ anchors do
5:          o.INSERTRIGHT(Q_t)                     ▷ forward initialization
6:          a.INSERTLEFT(o)                        ▷ reverse initialization
7:      end for
8:      anchors ← anchors ∪ {a}
9:      anchors ← SAMPLING(anchors)                ▷ cf. Section 4.3
10: end procedure
11: function QUERY(t_1, t_2)                       ▷ interface to query MI
12:     inWindow ← {a ∈ anchors | t_1 ≤ a ≤ t_2}
13:     estimates ← ()                             ▷ empty sequence
14:     for all a ∈ inWindow do
15:         MC_x, MC_y, N ← a.QUERY(t_1, t_2)
16:         Î ← ψ(k) − ψ(MC_x + 1) − ψ(MC_y + 1) + ψ(N)
17:         estimates.APPEND(Î)
18:     end for
19:     return MEAN(estimates)
20: end function
```

Estimation Quality. An analytic analysis of the estimation variance and bias would require strong assumptions on the data. Since the overall mutual information estimate is based on taking a sample mean of individual estimates, the standard deviation of MISE can be expressed by the standard deviation of the mean: Assuming that the data distribution is static over the query window leads to a standard deviation of $\sigma = \sigma_{\hat{I}}/\sqrt{M}$, where M is the number of query anchors in the window and $\sigma_{\hat{I}}$ is the standard deviation of the individual estimates. $\sigma_{\hat{I}}$ obviously depends on the distribution of the data. Instead of deriving $\sigma_{\hat{I}}$ only for specific distributions, we focus on a very broad empirical analysis of the estimation characteristics in Section 6, featuring a large number of real-world data streams.

4.3 Multiscale Sampling of Query Anchors

Our goal regarding the SAMPLING function is to exploit the multiscale nature of time. In general, any query window can be specified by its width w and the offset o, which denotes how much the query window is shifted into the past (cf. Figure 4). Intuitively, the motivation behind our multiscale sampling follows the general equivalence of time scales. Think of a query with a window size of 1 second shifted by 1 second into the past, a query with $w = 1$ hour and $o = 1$ hour, or even a query with $w = 1$ year and $o = 1$ year. Though the queries are defined over vastly different time scales, they are structurally equivalent. In many application a user might want to obtain answers of equal quality for these queries. Traditional sampling approaches like sliding window (SW) or reservoir sampling (RS) have significant issues with queries comprising multiple time scales. In general, we expect SW to fail for queries with a large window size in the distant past and RS to fail for queries with a very small window size in the most recent past. The key question is: How is it possible to answer these queries with equal quality? We will see that this simple requirement automatically generalizes for arbitrary o / w values, and that the *more detail on recent data* principle emerges naturally as a result. To formalize equivalence of time scales, we will denote the ratio of the query offset o to the window size w as a unit-free quantity $\Delta \equiv \frac{o}{w}$. Based on this quantity, we can partition the space of all possible queries into equivalence classes.

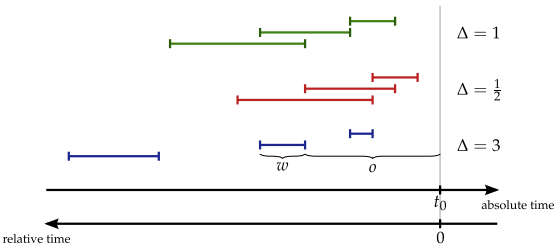


Figure 4: Examples of multiscale equivalence classes

Definition 7. Multiscale Query Equivalence: We define the multiscale query equivalence relation $\triangleq$ between queries A and B by: $A \triangleq B$ iff $\Delta_A = \Delta_B$.

We call the groupings formed by $\triangleq$ multiscale equivalence classes. Based on the multiscale equivalence of queries, we formalize the key idea behind our approach to operate on various time scales:

Definition 8. Multiscale Sampling: A multiscale sampling is a sampling scheme which provides an equal expected number of sampling elements for all queries which belong to the same equivalence class.

Thus, for a multiscale sampling of query anchors the expected number of query anchors in a window is constant for all queries with the same Δ. Figure 4 shows examples of equivalent queries for different Δ values. When the multiscale property is fulfilled, the queries with the same color have the same expected number of query anchors.

We will now propose a novel sampling scheme that fulfills the multiscale property. More specifically, we derive the sample distribution that is required for multiscale sampling. To simplify the presentation, we temporarily assume a continuous time domain and switch to a discrete time in a second step. Since the sample distribution is only defined for $t < t_0$, we will change to a time domain that is relative to t_0 and extends into the past (cf. Figure 4). This allows us to use the notion of probability densities to express the expected number of anchors in a query window. We refer to the probability density of our query anchors as $f(t)$. Thus, we are looking for an $f(t)$ which is a probability density function that satisfies the multiscale property. The expected number of query anchors in a query window

$[o, o+w]$ is equal to the integral $\int_o^{o+w} f(t)dt$ times the total number of query anchors. By using $o = w \cdot \Delta$, we can write the integral bounds as $[w\Delta, w(\Delta+1)]$. Definition 8 requires that, for a fixed Δ, this integral (the expected number of sampling elements) is invariant of the time scale, i.e., it is constant for all w. Thus, $f(t)$ must fulfill:

$$\int_{w\Delta}^{w(\Delta+1)} f(t)dt \stackrel{!}{=} const \tag{4}$$

Lemma 1. *Sampling according to a reciprocal distribution $f(t) = \frac{C}{t}$ fulfills the multiscale property (with appropriate normalization C corresponding to a finite positive support).*

Proof. Equation 4 requires $\frac{d}{dw}\int_{w\Delta}^{w(\Delta+1)} f(t)dt \stackrel{!}{=} 0$. Differentiation under the integral according to the generalized Leibniz integral rule yields:

$$(\Delta+1)\, f\left(w(\Delta+1)\right) \stackrel{!}{=} \Delta\, f\left(w\Delta\right) \tag{5}$$

By plugging $f(t) = \frac{C}{t}$ into Equation 5, one can see that all Δ terms cancel each other out, i.e., the reciprocal distribution satisfies the multiscale property for any $\Delta > 0$. □

We now turn to the question of how to transform this result to a discrete time domain. Obviously the support of a reciprocal distribution is only defined for $t > 0$ due to the singularity at $t = 0$. This directly reflects the general issue of estimation from a very small window size w: The smaller the window size, the larger the necessary density of sample points in order to maintain a sample of a fixed size. In a real-world system there commonly are domain specific constraints on the sampling frequency, i.e., the sampling resolution cannot be arbitrarily high. We deal with this issue by allowing a *saturation* of the discrete distribution in the region where the theoretically necessary sample density exceeds what is physically possible. To formalize the discretization, we will highlight the discretized time domain by using n as a counter of time steps in the past, starting with $n = 1$ as the most recent time point. We discretize the reciprocal distribution at these time points, each resulting in a probability P_n. Each P_n is equal to the probability that our sample contains the query anchor which is n time steps in the past:

$$P_n = \begin{cases} 1 & \text{if } n \leq \alpha \\ \frac{\alpha}{n} & \text{otherwise} \end{cases} \tag{6}$$

The resulting distribution[1] is illustrated in Figure 5. The (negative) x-axis corresponds to the discretized time steps n, and the y-axis shows the probabilities P_n. The parameter α controls the decay of the reciprocal distribution, and it will serve as the parameter to control the overall quality of the sampling. Equation 6 implies that we must keep the $\lfloor\alpha\rfloor$ most recent query anchors with a probability of 1, as a result of the shortage of available sampling points. For older query anchors with $n > \alpha$, the probability to have a certain query anchor in the sample follows the reciprocal function.

So far Equation 6 only specifies the necessary probabilities in the query anchor sample at a fixed time t_0. The essential question now becomes: How do we have to delete existing query anchors when going from one time step to the next in order to maintain an overall distribution according to Equation 6? This requires to convert the sampling probabilities P_n of Equation 6 into an incremental deletion scheme.

[1] The particular shape of this function – a piecewise composition of a reciprocal and a uniform function – makes a direct application of sampling schemes that specify weights in time forward direction [15, 9] impossible.

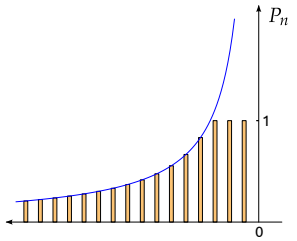


Figure 5: Discretized distribution with saturation

Definition 9. SAMPLING **function:** Based on the notion of *stepwise sampling probabilities*

$$SP_n = \begin{cases} 1 & \text{if } n \leq \alpha \\ \frac{P_n}{P_{n-1}} & \text{otherwise} \end{cases} \tag{7}$$

we define the SAMPLING function as follows: We keep a query anchor of age n with a probability of SP_n.

This means that we generate a random value $rand \in [0, 1]$ for every anchor. If $rand < SP_n$ we keep the anchor; otherwise the anchor is deleted immediately.

Lemma 2. *Modifying the query anchor sample with the* SAMPLING *function results in the sampling probabilities P_n after each time step.*

Proof. In order to show this, we have to link the stepwise sampling probabilities to the probabilities P_n. Intuitively, the stepwise probabilities slide over a certain time point when time evolves. Since the decisions whether to keep a given query anchor are independent, the final probability that a query anchor still exists after k time steps is simply the product of the first k stepwise probabilities. Therefore we have to prove that the product of the stepwise probabilities indeed gives the desired probability P_n, i.e.,

$$P_n \stackrel{?}{=} \prod_{k=1}^{n} SP_k \tag{8}$$

Let n^* be the smallest $n > \alpha$. Obviously, Equation 8 is fulfilled for all $n < n^*$ since both sides are 1. For $n = n^*$ we have $P_{n-1} = 1$, and thus, $SP_n = P_n$, which again satisfies Equation 8 given $\prod_{k=1}^{n-1} SP_n = 1$. For $n > n^*$ we conclude by induction:

$$P_{n+1} \stackrel{!}{=} \prod_{k=1}^{n+1} SP_k = \prod_{k=1}^{n} SP_k \cdot SP_{n+1} = P_n \cdot \frac{P_{n+1}}{P_n} \qquad \square$$

Combining Lemmas 1 and 2 leads us to our final conclusion: It is possible to construct an iterative sampling scheme which always maintains the multiscale property in the query anchor sample.

5. IMPLEMENTATION AND ANALYSIS

In the following we will discuss implementation details of our approach. To ensure repeatability, we provide both pseudo-code and a ready-to-use executable on a supplementary website,[2] and focus on the essentials in the following.

[2] http://www.ipd.kit.edu/~muellere/MISE/

Query Anchor Complexity. An important aspect of our proposed approach is that it is possible to implement query anchors very efficiently. Our solution is based on the fact that a query anchor can treat the forward and backward time directions independently. For both time directions, we can use dynamic arrays to store the marginal points and changes to the set of the k nearest neighbors. The insertion of new data works as follows: INSERTRIGHT first checks whether the new data point leads to a change of the k nearest neighbors in time forward direction. If this is the case, the new set of k nearest neighbors is appended to the dynamic array storing the neighborhood changes. Estimation theory shows that Kraskov estimation in general requires a very low k settings (i.e., $k \leq 4$, cf. Section 6), thus, the $O(\log k)$ complexity of the set operations are negligible. Next, INSERTRIGHT checks whether the new data point is a marginal point in either X or Y, and appends the point to the respective dynamic arrays. INSERTLEFT is implemented accordingly, operating in time backward direction. Overall, an insert operation comes down to extending the dynamic arrays, i.e., the amortized insert complexity is $O(1)$.

The QUERY operation has two substeps: (1) reconstruction of the proper $kNND$ for the given query window and (2) counting of marginal points. Step (1) can be implemented efficiently, since the two dynamic arrays storing the changes of the k nearest neighbor sets in both time directions are sorted by construction. This allows to perform a binary search to retrieve the k nearest neighbor sets in each time direction. To get the $kNND$ over the whole query window, it is simply possible to create the union of both sets and determine the k-th element. Step (2) counts the marginal points which are within the window boundaries and have a marginal distance lower than the just determined $kNND$. Due to the intrinsic sorting of our two-sided insert scheme, a binary search can again solve this efficiently.

Regarding memory complexity, a query anchor obviously requires $O(M)$, where M is the number of marginal points. The question is how the number of marginal points M grows over time. Unfortunately, a respective formal analysis would require assumptions regarding both the data distribution itself and how it changes over time. Even under the assumption of a static data distribution, there is no general result. However, it is possible to derive the general spectrum of possible growth rates. This follows from the findings of extreme value theory [24], which we explain in the following. As illustrated in Figure 1 and 2, there are two opposing effects: On the one hand, if the size of a slice was fixed, the number of marginal points would simply increase linearly, assuming a static data distribution. On the other hand, the width of the slice can only decrease monotonically over time. Thus, the question is how fast the k-NN distance decreases over time. In general the distance distribution of each query anchor is highly individual. Determining the minimum (or the k-th smallest element) of a sample drawn from this distance distribution is a standard problem of extreme value theory [24]. Since the metric is bounded by the minimum distance of zero, the domain of attraction is limited to a specific category, the Type III or Weibull family. However, the convergence rate of the minimum does not have a general result in this category. Hence, the overall growth rate of our marginal count can vary; there are the following cases:

- The minimum distance may decrease $\propto \frac{1}{N}$. For instance, this is the case if the distance distribution is an exponential or uniform distribution [24]. In this case the complexity of marginal counts is $O(1)$, due to the rapid decrease of the slice width.
- For some data distributions the growth of marginal counts is $O(N^{\alpha})$ with $\alpha < 1$. For example, when a query anchor is placed within a uniform distribution, the growth is $O(N^{\frac{1}{2}})$. Due to space constraints we cannot derive this result here. We therefore provide the derivation on our supplementary website.[2]
- For (rare) outlier objects the distance distribution mainly produces large distances, and therefore, the rate of convergence of the minimum is low. This yields the worst case complexity of $O(N)$.

In light of these findings, our expectation for the general case of arbitrary data distributions which may change over time is to obtain a mixture of these three cases. Therefore, we perform a thorough empirical analysis of the growth rate in our evaluation (cf. Section 6.4).

MISE Complexity. Naturally, the most important complexity factor of the MISE framework is the size S of the query anchor sample. The sample size S not only determines the overall memory consumption; it also defines the complexity of the INSERT operation since the INSERT operation has to connect each incoming data sample to the existing query anchors and vice-versa. Therefore the insert complexity is $O(S)$. We can express the expectation value of S after processing T data points as follows:

$$E[S] = \lfloor \alpha \rfloor + \alpha \sum_{k=\lfloor \alpha \rfloor + 1}^{T} \frac{1}{k} = \lfloor \alpha \rfloor + \alpha \left(H_T - H_{\lfloor \alpha \rfloor} \right) \quad (9)$$

where H_i is the i-th harmonic number. Asymptotic expansion of H_T reveals a complexity of $O(\log T)$. We use this result to construct two different versions of our algorithm. The first version MISE_D works with this slowly growing dynamic query anchor sample. For a second version MISE_F, we fix the sample size S and instead operate with slow changes of α over time. This means we modify α in each step by solving Equation 9. This has to be done numerically since the equation has no analytic solutions. In Figure 5 this would correspond to a slight change of the decay rate. Obviously each modification to α introduces a small error since the current query anchors in the sample have been sampled with a probability that has been slightly too large. To account for the accumulation of these slight errors, we delete query anchors with a probability equal to the ratio of the P_n values calculated once with the old and once with the new α. This exactly corrects the error and maintains a proper reciprocal distribution over time. Overall, the two versions of MISE have insert complexities of $O(\log T)$ for MISE_D and $O(1)$ for MISE_F.

6. EXPERIMENTS

The focus of our experimental evaluation is to analyze MISE regarding both performance and estimation quality. In particular, we will analyze how MISE performs overall in a typical online setting (Section 6.1), how we can match the insert frequency to the stream frequency (Section 6.2), and focus on estimation quality in Section 6.3. We are not aware of any direct competitor that supports online mutual information queries. Therefore, we compare our approach to a static Kraskov estimation, which we allowed to use a

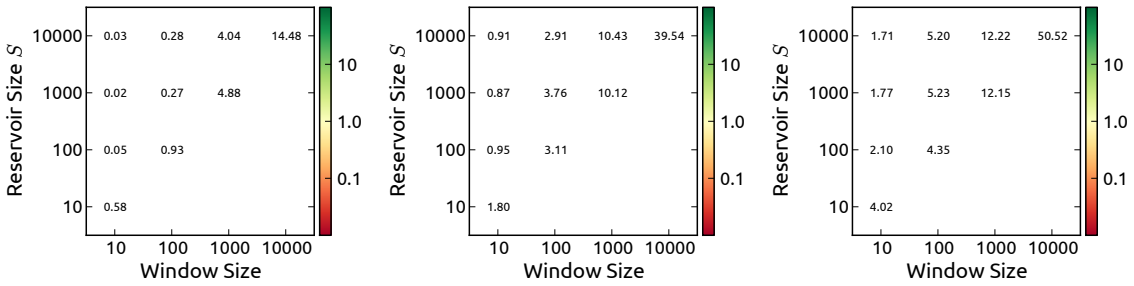


Figure 7: Speed-up of accumulated runtime for query-to-insert-ratios of 0.01 (left), 0.1 (middle), 1.0 (right).

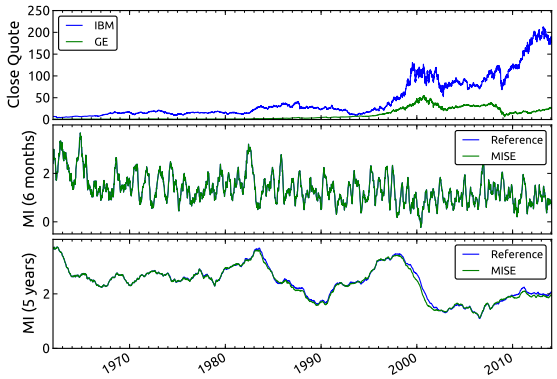


Figure 6: Stock exchange example

(theoretically) infinite data reservoir. By using the same estimation principle based on an infinite reservoir as a ground truth, we can focus on the effects introduced by our finite summarization of the stream. This frees us from reevaluating the properties of Kraskov estimation in general, and we refer to existing studies [19, 30, 20, 22] for details. The guidelines on choosing parameter k obtained in these studies directly apply in our case as well: The best trade-off between the statistical and the systematic estimation error is typically in the range of very low k values (e.g. $k \leq 4$). Since our evaluation scheme measures the relative estimation error we focus on the case that is most challenging: We use $k = 1$ in all our experiments, which maximizes the statistical error of Kraskov estimation and therefore maximizes the effect of using a finite reservoir in MISE.

A relative evaluation also allows us to run our experiments on a broad range of data streams, including a large number of real-world datasets. These data streams contain natural fluctuations of mutual information over time. Figure 6 shows an example of such dynamics found in our real world data: The top plot shows the raw time series themselves; in the example the quotes of the IBM and GE stocks. The middle plot shows mutual information measured using a 6 month sliding window. Compared to the bottom plot, which uses a 5 year sliding window, we can see that mutual information clearly shows different changes over these two different time scales. Such a "running mutual information estimate" also gives the first impression of the potential of MISE: We can see that our estimation of MISE and the reference implementation give almost identical estimation results, i.e., estimation based on the finite summarization of MISE shows no significant difference to estimation from an infinite reservoir. However, as a result of the online processing in MISE, the total time to generate the graphs in Figure 6 were 3.8 minutes for MISE and 112.1 minutes for the reference implementation. In the following we will quantify these performance improvements systematically.

Experimental Setup. We have conducted all experiments on an Ubuntu 12.04 system running on an Intel® i3-550 processor with 8 GB RAM. We have implemented MISE in Scala 2.10 using Oracle JVM 7 as runtime environment.

6.1 Overall Performance

Our reference implementation of static mutual information estimation on a data stream works as follows: The insert operation simply appends a data sample to a theoretically infinite reservoir, while the query operation performs Kraskov estimation on the specified query window using the infinite reservoir. Since this reference approach provides no means of query precomputation, there is obviously a pronounced imbalance between the extremely cheap insert operation and the high complexity of the query. Thus, when comparing MISE to this reference implementation the crucial question is how the number of inserts compares to the number of queries. We express the ratio of queries-to-inserts by $QIR = {}^{\#\text{queries}}/_{\#\text{inserts}}$. Obviously, when there are no queries at all ($QIR = 0$), all query precomputations of MISE are futile and there is nothing to speed up. On the other hand, when there is a large number of queries compared to the number of inserts ($QIR \gg 1$), the benefits of MISE's precomputations can be made arbitrarily high. Therefore, we specifically analyze low QIR values to determine the point where the benefit of MISE begins.

To measure the speed-up we determine the ratio of the total runtimes for MISE and the reference implementation of calculating a "running mutual information estimate" (like shown in Figure 6). This running estimate is performed by inserting and querying the stream with a specific QIR ratio, e.g., for $QIR = 0.1$ we perform a query after every 10 inserts. Regarding the time offset of the queries we set $o = 0$. This means that the queries operate in the region of highest query anchor density, and thus, performance of MISE is worst. The data stream was sampled from a Gaussian distribution ($\rho = 0.1$) with a length of 100000. We started the measurement of the total runtime after the number of processed samples exceeded both the window size and the reservoir size in order to exclude warm-up artifacts.

The results of this experiment are shown in Figure 7. The main factors determining the speed-up are the query window size and the size of the reservoir used by MISE. The latter is determined either by α or S for the dynamic or fixed versions. Due to the more intuitive interpretation of the fixed reservoir size S we focus on this variant. Even though we focus on small QIR ratios, we can see that MISE can lead

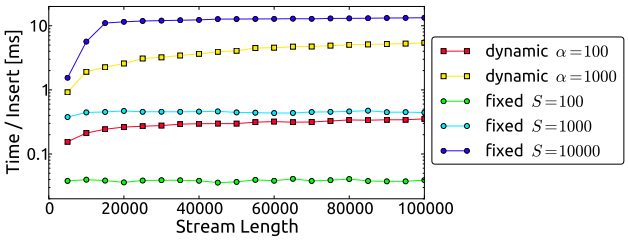


Figure 8: Insert processing times

to drastic speed-ups. We visualize the speed-up threshold of 1.0 where usage of MISE starts to make sense by a yellow color; green and red indicate faster and slower runtimes for MISE respectively. It is interesting to see that a speed-up is even possible for the very low $QIR = 0.01$, where in fact 99% of the precomputations in the inserts were in vain. Overall we can conclude that there is a large potential for speed-ups as a result of our precomputations. Obviously this is especially pronounced for applications where both the window size and offset are free parameters for each query, and thus, having more queries than inserts is usual.

6.2 Scaling with Stream Frequency

The results of the previous experiment can also be interpreted as follows: Since MISE performs parts of the necessary query computations while processing the stream itself, it is possible to tune MISE such that its insert speed perfectly matches the frequency of the stream. This would mean that MISE performs just as much precomputations as possible to keep up with the stream and leads to maximization of the query quality for the given stream frequency (cf. Section 6.3). Thus the essential question becomes: How does the reservoir size of MISE influence the processing speed of the stream? We evaluate this question for both MISE versions, i.e., we analyze the insert speed in dependence of α and S for the dynamic and fixed reservoir versions respectively. Intuitively a higher S or α means higher estimation quality, but a slower insert processing. A user typically might want to set S or α to the largest possible value that still allows to process the given stream frequency.

Figure 8 shows a measurement of the insert times for different α or S values. It shows how the runtime of a single insert (y-axis) changes with the stream length (x-axis). We obtain the runtime of a single insert from the runtime of performing 5000 inserts in a batch. Again we sampled the data stream from a Gaussian distribution with $\rho = 0.1$. For the fixed MISE version we can see that the runtime of a single insert indeed becomes constant once the stream has reached a length corresponding to the fixed reservoir sizes S=100, 1000, 10000. For the dynamic version, the insert time slowly increases over time due to the logarithmic growth of the reservoir. We can see that for typical sizes of the reservoir the corresponding stream processing frequency is in the order of ∼100 Hz (for $S = 10000$) up to ∼20 kHz (for $S = 100$). Thus, despite performing query precomputation while processing the stream, it is possible to operate on very fast streams with sampling periods in the order of a millisecond. Furthermore, there is no dependence of insert performance on the stream length for the fixed MISE version. Thus, there is no degradation over time, which is a key property of efficient stream processing [3].

6.3 Quality

We now want to turn to the question how estimation from a limited reservoir affects the estimation quality. Therefore, we compare MISE to a variant of Kraskov estimation which also operates on a limited data reservoir. We implemented the limited data reservoir based on the two most prominent sampling approaches: Traditional reservoir sampling (RS), and sampling based on a sliding window (SW). We use static Kraskov estimation from an unlimited data reservoir as ground truth for the quality assessment. To facilitate the comparison we focus on the MISE variant with a fixed reservoir size. This allows us to use exactly the same reservoir size for RS, SW, and MISE. To pay attention to the challenge of a stream length being much larger than the reservoir size, we have used a reservoir size of $S = 100$ in the following experiment.

Data. We compiled a set of 26 data streams from various different sources. Our goal was to obtain a very large diversity of different streams, i.e., diversity in distributions and dynamics. Therefore the set contains various streams from different real world datasets plus a small number of synthetic streams. This includes streams of IMU sensors (various combinations of gyrometer, accelerometer, magnetometer streams), climate streams, smart meter streams, stock streams, and electrocardiogram measurements. All features of the data streams are continuous variables, with a floating point precision between 4 and 10 decimal digits. Table 1 shows a summary of all data streams. Preliminary results on quality did not show a strong dependence on the individual data streams. Therefore, we present the aggregated quality over all streams in the following, and provide results on the individual datasets on our supplementary website.[2]This means that we calculate the quality measures discussed below for each query individually and aggregate by taking the average of these measures for all queries obtained from all data streams.

Stream description	Length	# streams used
Pamap (IMU data)	198000	5
Climate data (temperature vs. air pressure)	21124	1
Smart Meter	17568	5
Stock time series	11122	5
Congestive Heart (two ECG measurements)	300000	1
Synth: Static Gaussian $\rho = 0$	∞	1
Synth: Static Gaussian $\rho = 0.95$	∞	1
Synth: Static random mixture of uniform distributions	∞	2
Synth: Dynamic random Gaussian mixture	∞	5

Table 1: Set of data streams

Queries. The queries we perform on our data streams range from a window size of 10 up to 1000. We perform these queries in appropriate steps that avoid an overlap of the query windows to ensure independent query results. We use three different Δ values (0, 1.0, 10.0) to determine the offset of the query window. Using $\Delta = 0$ means that we use an offset $o = 0$ (i.e., we deliberately include the most favorable case for SW), while o follows the multiscale principle in the non-zero cases.

Quality Measure. A first question when performing a certain query on a system with a limited reservoir is whether the system actually has information available for the given window boundaries. Therefore, our first quality measure simply is the percentage of "successful" queries defined as: A query is successful if the query window contains at least a single element, allowing to compute a result. In case the

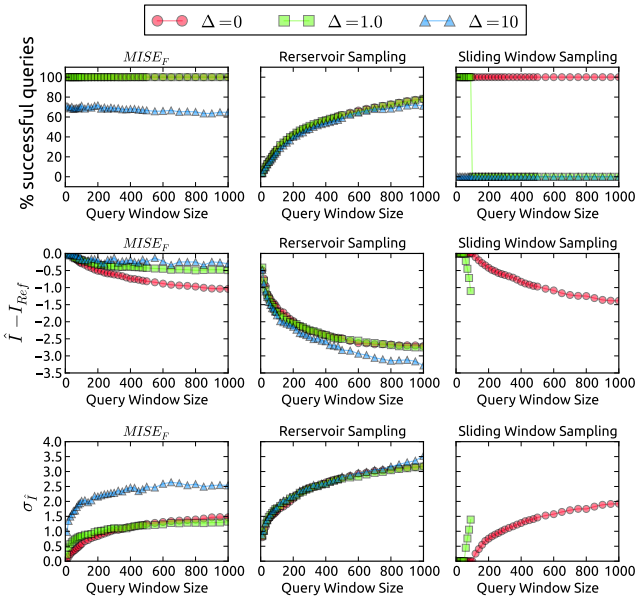


Figure 9: Overall quality results on all data streams

system can answer a query, we are interested in how the limited reservoir influences the estimation in both bias and variance. Therefore, we use the two quality measures ($\hat{I} - I_{Ref}$) and $\sigma_{\hat{I}}$. Here $\hat{I}$ refers to estimation from the limited reservoir, while I_{Ref} is the ground truth obtained from the infinite reservoir; $\sigma_{\hat{I}}$ is the sample standard deviation.

Results. The results of our quality experiment over all data streams are shown in Figure 9. Regarding the success rate of queries we can clearly see the advantage of the MISE sampling: For $\Delta = 0$ and $\Delta = 1.0$, the success rate is 100%. The result shows that the success rate does not depend on the window size. It rather is constant for a given family of queries with a fixed Δ. RS in contrast never achieves a 100% success rate. By the nature of RS, we can clearly see the poor performance for small window sizes (e.g., low success rate, large bias). On the other hand for sliding window sampling, we obtain poor performance for large windows, visible for instance by the sudden drop of the success rate as soon as the offset is larger than the fixed window of size 100. A query with $\Delta = 10$ simply has always been too far into the past and could never be answered. In the second row of Figure 9, we can see that all approaches show a small negative bias, which is a general issue when estimating from very little data. We can see that MISE shows much better bias and variance (third row) compared to Kraskov estimation from limited reservoirs. This is caused by the more flexible placement of query anchors over time and the added information that is used as a result of the online processing. Furthermore the dependence on the query window size is much lower compared to RS or SW sampling. For SW sampling the bias and variance are obviously zero as long as the query window is fully covered by the sampling window. However, we can see that, even in the favorable case of a zero query offset ($\Delta = 0$), estimation quality quickly degrades as soon as the query window size exceeds the one of the sampling window. Overall we can conclude: RS and SW fail either for small or large window sizes respectively; MISE achieves the overall best results, and is most stable w.r.t. shifting a query into the past, as a result of featuring

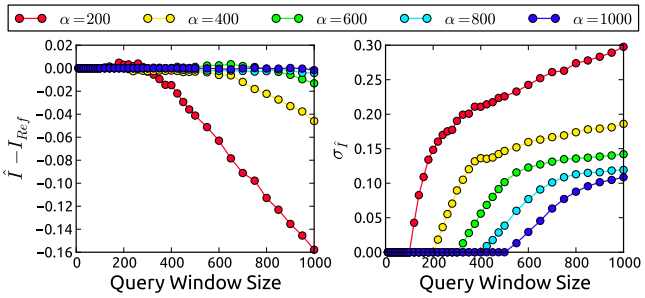


Figure 10: Quality with different reservoir sizes

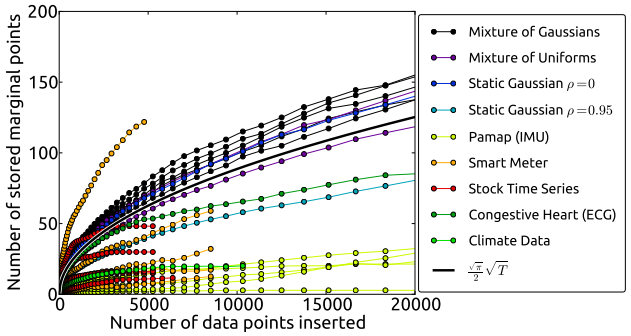


Figure 11: Average marginal counts that have to be stored over time

a multiscale sampling scheme.

In another experiment we want to show that the estimation bias and variance of MISE can be reduced arbitrarily by increasing the reservoir size. For this experiment we use the dynamic version and vary α from 200 to 1000. The queries have $\Delta = 1.0$. See Figure 10 for the results. We can see that the estimation variance becomes almost independent of the window size in the range where query anchor saturation does no longer occur. The takeaway is that the reservoir size gives very fine control over the overall estimation quality. Combined with the results on insert speed this means that bias-free estimates with very low variance are possible without difficulties, while maintaining insert frequencies in the range of 1000 Hz.

6.4 Growth Rate of Marginal Points

In addition to the theoretical results of Section 5 and the supplementary material,[2] we conclude our experiments with an empiric evaluation of the question of how the number of marginal points evolves over time. In the following experiment we determine the empirical growth rate for each of our data streams (cf. Table 1) individually. For each data stream, we randomly pick 1000 data points from the first half as test query anchors. Next, we insert subsequent data points into the test query anchors. Finally, we query all 1000 query anchors for the marginal counts in time forward direction with a varying query window size. Figure 11 shows the average marginal counts depending on this query window size, which corresponds to the number n of inserted data points. As a visual reference we also plot the theoretical result $\frac{\sqrt{\pi}}{2} n^{\frac{1}{2}}$ from our supplementary material[2] for the specific case of a uniform distribution.

Overall, the results in Figure 11 reveal an interesting finding: For most data streams we observe a growth rate of approximately $n^{\sim\frac{1}{2}}$. This result makes sense considering that

the overall spectrum of growth rates for each individual anchor ranges from constant to linear depending on its position in the data distribution (cf. Section 5). Apparently, the averaging of all the individual growth rates seems to yield a similar rate to the one obtained formally for the uniform distribution.

Another remarkable result is the absolute value of the marginal counts itself, indicating that we need an extraordinarily small number of marginal points to represent large time frames: A query anchor with an age of 20000 time units has to store less than 150 marginal points on average. Thus, the ratio of stored marginal points vs inserted points is ∼0.75% after 20000 time units, or ∼0.09% after 1 million steps. This strong compression ratio shows that it is cheap to maintain "old" query anchors, which can explain the very good overall performance of MISE observed in the previous experiments.

7. CONCLUSION

In this work we have proposed a framework that allows a user or data mining algorithm to estimate mutual information on a data stream in arbitrary query windows. To our knowledge, it is the first such estimation technique that incorporates a summarization allowing online query precomputation. Furthermore, we have proposed a novel sampling scheme which provides a solution to the infinite nature of the stream while maintaining information equally over multiple time scales. Given these properties the experiments show that our approach clearly outperforms traditional approaches in the online context.

8. REFERENCES

[1] C. C. Aggarwal. On Biased Reservoir Sampling in the Presence of Stream Evolution. In *VLDB*, 2006.

[2] A. Arasu and G. S. Manku. Approximate Counts and Quantiles over Sliding Windows. In *PODS*, 2004.

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

[4] L. Bhuvanagiri and S. Ganguly. Estimating Entropy over Data Streams. In *Algorithms – ESA*. 2006.

[5] C. Böhm, B. C. Ooi, C. Plant, and Y. Yan. Efficiently Processing Continuous k-NN Queries on Data Streams. In *ICDE*, 2007.

[6] A. Chakrabarti, G. Cormode, and A. McGregor. A Near-optimal Algorithm for Computing the Entropy of a Stream. In *SODA*, 2007.

[7] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Found. Trends Databases*, 4(1-3), 2012.

[8] G. Cormode, F. Korn, and S. Tirthapura. Exponentially Decayed Aggregates on Data Streams. In *ICDE*, 2008.

[9] G. Cormode, V. Shkapenyuk, D. Srivastava, and B. Xu. Forward Decay: A Practical Time Decay Model for Streaming Systems. In *ICDE*, 2009.

[10] G. Cormode, S. Tirthapura, and B. Xu. Time-decaying Sketches for Sensor Data Aggregation. In *SODC*, 2007.

[11] G. Darbellay and I. Vajda. Estimation of the information by an adaptive partitioning of the observation space. *IEEE Transactions on Information Theory*, 45(4), 1999.

[12] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining Stream Statistics over Sliding Windows. In *SODA*, 2002.

[13] C. O. Daub, R. Steuer, J. Selbig, and S. Kloska. Estimating mutual information using B-spline functions - an improved similarity measure for analysing gene expression data. *BMC Bioinformatics*, 5, 2004.

[14] B. V. Durme and A. Lall. Streaming Pointwise Mutual Information. In *Advances in Neural Information Processing Systems 22*. Curran Associates, Inc., 2009.

[15] P. S. Efraimidis and P. G. Spirakis. Weighted Random Sampling with a Reservoir. *Inf. Process. Lett.*, 97(5), 2006.

[16] J. Gama and C. Pinto. Discretization from Data Streams: Applications to Histograms and Data Mining. In *SAC*, 2006.

[17] M. Greenwald and S. Khanna. Space-Efficient Online Computation of Quantile Summaries. In *SIGMOD*, 2001.

[18] G. S. Iwerks, H. Samet, and K. Smith. Continuous K-nearest Neighbor Queries for Continuously Moving Points with Updates. In *VLDB*, 2003.

[19] S. Khan, S. Bandyopadhyay, A. R. Ganguly, S. Saigal, D. J. Erickson, III, V. Protopopescu, and G. Ostrouchov. Relative performance of mutual information estimation methods for quantifying the dependence among short and noisy data. *Physical Review E*, 76(2), 2007.

[20] J. B. Kinney and G. S. Atwal. Equitability, mutual information, and the maximal information coefficient. arXiv e-print 1301.7745, 2013.

[21] L. F. Kozachenko and N. N. Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2), 1987.

[22] A. Kraskov, H. Stögbauer, and P. Grassberger. Estimating mutual information. *Physical Review E*, 69(6), 2004.

[23] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang. Data Streaming Algorithms for Estimating Entropy of Network Traffic. In *ACM SIGMETRICS*, 2006.

[24] M. R. Leadbetter, G. Lindgren, and H. Rootzén. Extremes and related properties of random sequences and processes. 1983.

[25] K. Mouratidis, K. Mouratidis, D. Papadias, and D. Papadias. Continuous Nearest Neighbor Queries over Sliding Windows. *IEEE TKDE*, 19(6), 2007.

[26] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. In *SIGMOD*, 2005.

[27] L. Paninski. Estimation of Entropy and Mutual Information. *Neural Computation*, 15(6), 2003.

[28] S. Panzeri, R. Senatore, M. A. Montemurro, and R. S. Petersen. Correcting for the Sampling Bias Problem in Spike Train Information Measures. *Journal of Neurophysiology*, 98(3), 2007.

[29] T. Schürmann. Bias Analysis in Entropy Estimation. arXiv e-print cond-mat/0403192, 2004.

[30] J. Walters-Williams and Y. Li. Estimation of Mutual Information: A Survey. In *Rough Sets and Knowledge Technology*, number 5589. 2009.

[31] H. Xiong, S. Shekhar, P.-N. Tan, and V. Kumar. Exploiting a Support-based Upper Bound of Pearson's Correlation Coefficient for Efficiently Identifying Strongly Correlated Pairs. In *KDD*, 2004.

[32] X. Xiong, M. Mokbel, and W. Aref. SEA-CNN: scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE*, 2005.

[33] X. Yu, K. Pu, and N. Koudas. Monitoring k-Nearest Neighbor Queries over Moving Objects. In *ICDE*, 2005.

[34] W. Zhou and H. Xiong. Volatile Correlation Computation: A Checkpoint View. In *KDD*, 2008.

[35] W. Zhou and H. Xiong. Checkpoint evolution for volatile correlation computing. *Machine Learning*, 83(1), 2011.