Iterative Estimation of Mutual Information with Error Bounds

Michael Vollmer
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
michael.vollmer@kit.edu

Klemens Böhm
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
klemens.boehm@kit.edu

ABSTRACT

Mutual Information (MI) is an established measure for linear and nonlinear dependencies between two variables. Estimating MI is nontrivial and requires notable computation power for high estimation quality. While some estimation techniques allow trading result quality for lower runtimes, this tradeoff is fixed per task and cannot be adjusted. If the available time is unknown in advance or is overestimated, one may need to abort the estimation without any result. Conversely, when there are several estimation tasks, and one wants to budget computation time between them, there currently is no efficient way to adjust it dynamically based on certain targets, e.g., high MI values or MI values close to a constant. In this article, we present an iterative estimator of MI. Our method offers an estimate with low quality near-instantly and improves this estimate in fine grained steps with more computation time. The estimate also converges towards the result of a conventional estimator. We prove that the time complexity for this convergence is only slightly slower than non-iterative estimation. Additionally, with each step our estimator also tightens statistical guarantees regarding the convergence result, i.e., confidence intervals, progressively. These also serve as quality indicators for early estimates and allow to reliably discern between attribute pairs with weak and strong dependencies. Our experiments show that these guarantees can also be used to execute threshold queries faster compared to non-iterative estimation.

1 INTRODUCTION

Motivation. Detecting and quantifying dependencies between variables is an essential task in the database community [10, 13, 20, 30]. Conventional methods such as correlation coefficients and covariance matrices only detect linear or monotonous dependencies. *Mutual Information* (MI) in turn is an index that captures any linear and nonlinear dependency [1, 5]. Probability distributions of the variables in question serve as input to compute the MI. For real-world data however, these distributions are not available. In this case, MI must be estimated based on samples.

Various estimators for MI have been proposed [15, 23, 33], and some offer good results even for small samples [15]. However, continuous variables with an unknown distribution continue to be challenging, since their multivariate distribution is substituted only by a limited sample. A prominent approach for estimation of MI between continuous variables without assumption of the distribution is the nearest-neighbor based method by Kraskov et al. (KSG) [19].

While good estimators are available, they are very rigid in their time requirements and regarding the estimation quality. Once the computation has started, they impose a fixed time requirement and do not yield aby preliminary result if they are terminated

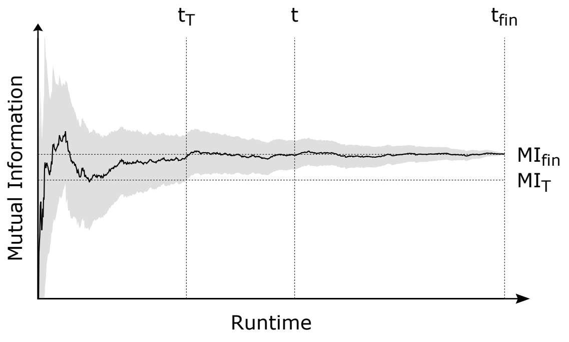

Figure 1: MI estimation with dynamic time allocation.

prematurely. They also are unable to exploit 'easier' queries like whether the MI value is above a certain threshold but instead determined the value. Such features are highly relevant for high-dimensional data and data streams with irregular arrival rate as we showcase with the following two scenarios.

Scenario 1. Consider a modern production plant with smart meters installed on each machine. A first step in data exploration is determining which attributes are strongly dependent. For instance dependencies among currents or energy consumption may offer insights into production sequences. For this first step, a query like "Which pairs of measurements have a MI value above the threshold MI_T?" often suffices. With conventional MI estimators, each pair either induces high computational costs, or results are uncertain because of low estimation quality.

Scenario 2. Think of a database with financial data and its real-time analysis. To maintain a diverse portfolio, it is important to track the relationships between stocks. Because bids and trades happen irregularly, new information and market prices arrive at irregular speed. Thus, it is not known how much time is available to monitor stock relationships in the presence of incoming data. Current MI estimators cannot adapt during execution. They risk not producing a result in time, or estimates are of low quality.

To improve upon these shortcomings, we study estimation of MI with dynamic allocation of computation time. Ideally, such an estimator should not only offer preliminary results, but also indicate its remaining uncertainty. Figure 1 shows exemplary progression over time of such an estimator based on our experiments with real data. The black line indicates the preliminary estimate after a certain runtime, and the gray area shows the (expected) maximum error of the preliminary estimate. To obtain the definitive result $\mathtt{MI}_{\mathtt{fin}}$, a user would require time $\mathtt{t}_{\mathtt{fin}}$. However, he could also stop the estimator as soon as the estimate is above a threshold $\mathtt{MI}_{\mathtt{T}}$ with certainty, or he can use the preliminary result available after time $\mathtt{t}$.

In this work, we focus on iterative estimation of MI in order to offer this functionality. Here, 'iterative' means quickly providing an estimate, but with the option to improve the estimation if there is time left. In other words, improving the estimate with some time available is what we call an *iteration.* At the same time, an iterative estimator can terminate the estimation, i.e., stop iterating, when the result is good enough. For efficiency, it is important that computations from previous iterations remain useful and are not repeated or discarded in a later iteration. So far, efficient iterative estimators for MI do not exist.

Challenges. The most significant feature of an estimator is its quality of estimation. This is even more so for iterative methods because both "preliminary" and "final" estimation quality are important. In other words, the estimate should already be useful after a few iterations, and estimation quality must level up to the one of conventional estimators after many iterations. Ideally, this convergence should happen after a known, finite number of iterations. In this article, we target at respective formal guarantees.

Next, the quality of preliminary estimates is crucial for usability. Determining if a preliminary result is "good enough" or interesting enough to merit additional computation time requires some information on its certainty. The number of iterations alone is insufficient, as the result quality depends on many other factors such as data characteristics, required accuracy and time constraints. Instead, each estimate requires an individual indicator of the uncertainty remaining.

While the time spent to improve the estimate iteratively is committed dynamically, it must of course be used efficiently. Many conventional estimators use data structures that are expensive to build and cheap to use, such as space-partitioning trees [19, 31, 32]. Such an upfront activity is undesirable for an iterative estimator whose first estimate must arrive soon. At the same time, runtime and scalability do remain important characteristics of the estimator. In other words, an iterative estimator must feature guaranteed efficiency for both individual iterations and final estimates.

Contributions. In this article, we present IMIE, our Iterative Mutual Information Estimator. To prove its practical usefulness, we establish several features both formally and experimentally.

Quality of Estimation. In Section 4, we propose a design for IMIE such that estimates converge to the same value as with the KSG. To make early iterations useful, IMIE also offers statistical error bounds for its early estimates. More precisely, an early estimate provides a confidence interval for the final estimate. We describe the specifics and the statistical soundness in Section 4.3.

Complexity. We study the time complexity of initialization and of individual iterations of IMIE. In Section 5 we establish an amortized time complexity for IMIE and the nearest-neighbor search used. This complexity is competitive with existing non-iterative estimators. To be precise, we show that iterating IMIE until convergence is only slightly slower in terms of time complexity than computing the KSG directly with optimal algorithms.

Experimental Validation. We show that IMIE complements the formal guarantees established so far with good actual performance. To do so, we perform extensive experiments using both synthetic and real data sets in Section 6. On the one hand, we show that the concrete runtime and estimation results of IMIE are comparable to the ones of conventional estimation methods. On the other hand, the experiments show the practical benefits of the early results from IMIE. For instance, IMIE finds attribute pairs above a threshold value significantly faster than non-iterative estimators.

2 RELATED WORK

Iterative estimation of MI is interesting from two perspectives. On the one hand, it is methodically interesting, as it can be considered an *anytime algorithm.* On the other hand, it is interesting to consider the benefits it provides over current methods in different settings. Important application scenarios are dependency analysis in high dimensional data and data streams, cf. Scenario 1 and 2.

Anytime Algorithms. Anytime algorithms [36] use available time to increase their result quality. One can obtain a low-quality result after a short time and a better one when waiting longer. In data analysis, anytime algorithms exist for clustering [22], classification [35] and outlier detection [2]. So far however, there is no anytime algorithm to estimate MI. So while there is no direct competitor, IMIE extends the set of tools available as anytime algorithms. Additionally, there has been more general work on the optimal use of available anytime algorithms [11, 18], which may improve the performance of IMIE in larger systems.

MI on Data Streams. Estimating MI on streams has received some attention recently. The MISE framework [14] summarizes a bivariate stream such that the MI for arbitrary time frames can be queried. To this end, MISE offers parameters for the balance between accuracy of older queries and resource requirements both in terms of memory and computation time. In contrast, the DIMID estimator [4] processes a bivariate stream as sliding window for monitoring tasks. This approach provides fast updates between time steps by approximation with random projection. MI estimation in sliding windows has also been the focus of [32]. That paper provides lower bounds for estimates using Equation 5 both in general and for updates in sliding windows. It also features two dynamic data structures, DEMI and ADEMI, to maintain such estimates using either simple or complex algorithms and data structures.

These approaches have limitations. First, they all impose the necessary execution time, i.e., one cannot adapt this time after the start of stream processing. If the rate of new items increases, the estimator may be unable to keep up. If it decreases, the estimator cannot use this time to improve results. Second, the approaches are all focused on bivariate streams. While MI is defined for exactly two variables, the number of attribute pairs grows quadratically in the number of dimensions. In contrast, the only information IMIE maintains on a stream is based on individual dimensions and thus scales linearly with the dimensionality. Third, the approximate results of MISE and DIMID are difficult to use. Their estimation quality is only known on average; this average defines the perceived quality of individual estimates. So if one estimate has a very small error, it is less likely to be appreciated, while the error of a particularly bad estimate may be assumed to be smaller.

Dependencies in High Dimensional Data. Even though MI is defined for exactly two variables, it has many applications with high-dimensional data. Prominent ones are image registration [25], which uses MI between two high-dimensional variables, and feature selection [24], which targets at the MI between attributes and a classification label. But estimating the MI between all pairs of attributes has received little attention, despite being the non-linear equivalent of correlation matrices. [26] uses a different approach, i.e., kernel density estimation, and removes redundant computations that arise when using this estimator for each pair. This approach has a worse computational complexity than a pair-wise application of the KSG estimator, without offering better

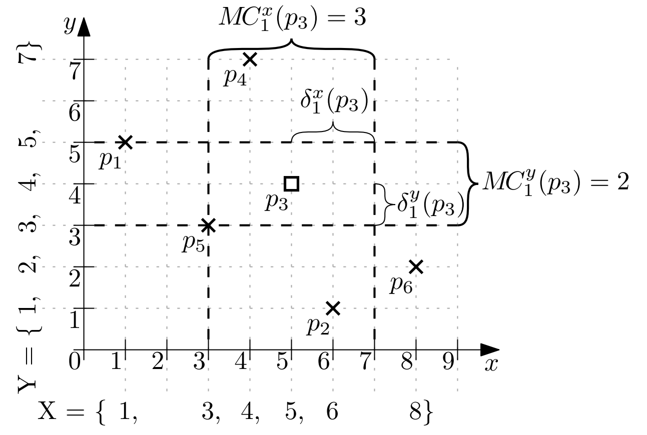

Figure 2: Illustration of terms used for the KSG.

results [15, 23]. While both scale quadratically in the number of attributes, their approach is also quadratic in the number of points. The complexity of the KSG in turn is $\Theta(n \log n)$ [32]. Additionally, it does not expose any parameter to modify the result quality. Consequently, there would not be any benefit of a direct experimental comparison with IMIE.

3 FUNDAMENTALS

We first cover the background of MI and its estimation.

Mutual Information. Shannon has introduced the notion of *entropy* [28] to quantify the expected information gained from observing a value of a random variable. $H(X)$ stands for the entropy of a random variable X. The expected information of observing two random variables X and Y is the *joint entropy* $H(X, Y)$. *Mutual Information* quantifies the amount of information that is shared or redundant between the two variables. It is defined as

$$I(X;Y) = H(X) + H(Y) - H(X;Y). \tag{1}$$

With the definition of entropy for continuous variables [6], the MI of two continuous random variables is

$$I(X;Y) = \int_X \int_Y p_{XY}(x,y)\ \log\left(\frac{p_{XY}(x,y)}{p_X(x)p_Y(y)}\right) dx\, dy, \tag{2}$$

where p_X, p_Y and p_{XY} are the marginal and joint probability density functions of X and Y. The type of logarithm used in Equation 2 determines the unit of measurement. In this work we use the natural logarithm. This means that MI is measured in the *natural unit of information* (nat).

Estimation. One can perceive many sources of data, e.g., smart meters or market prices, as random variables with unknown distribution. Since Equation 2 requires probability density functions, we cannot compute the MI of such sources exactly. Instead, we can only estimate the MI based on available samples. The popular estimator that will serve as foundation of our work is the one by Kraskov, Stögbauer and Grassberger [19], which we call KSG. It is based on the estimator for probability densities by Loftsgaarden and Quesenberry [21], which Kozachenko and Leonenko have studied further in the context of entropy [17]. In the following, we briefly review the terms and computation of the KSG.

Let $P = \{p_1 = (x_{p_1}, y_{p_1}), \dots, p_n = (x_{p_n}, y_{p_n})\} \subseteq \mathbb{R}^2$ be a sample from a random variable with two attributes. Figure 2 illustrates the notions that we define in the following using the sample $P = \{(1,5), (6,1), (5,4), (4,7), (3,3), (8,2)\}$. Let $X = \{x_{p_1}, \dots, x_{p_n}\}$ and $Y = \{y_{p_1}, \dots, y_{p_n}\}$ be the set of values per attribute. For each point $p \in P$, its $k \in \mathbb{N}^+$ nearest neighbors in P using the maximum distance form the set $kNN(p)$. More formally, it is

$$kNN(p) = \underset{S \subseteq (P \setminus \{p\})\ s.t.\ |S|=k}{\arg\min}\ \max_{s \in S} \|p, s\|_\infty, \tag{3}$$

with $\|p, s\|_\infty = \max(|x_p - x_s|, |y_p - y_s|)$. We define the largest distance between x_p and any x-value among the k nearest neighbors of p as $\delta_k^x(p) = \max_{s \in kNN(p)} |x_p - x_s|$. We use this distance $\delta_k^x(p)$ to define the *x-marginal count*

$$MC_k^x(p) = |\{x \in (X \setminus x_p) : |x - x_p| \le \delta_k^x(p)\}|, \tag{4}$$

which is the number of points whose x-value is “close to p”. In Figure 2, vertical dashed lines mark the area of points whose x-values are at least as close as the nearest neighbor of p_3. Since this area contains three points excluding p_3, it is $M_1^x(p_3) = 3$. The distance $\delta_k^y(p)$ and the *y-marginal count* $MC_k^x(p)$ are defined analogously. Note that $\delta_k^x(p)$ and $\delta_k^y(p)$ may differ, which results in differently sized areas for the marginal counts, as seen in Figure 2. Using these counts, the KSG estimate is defined as

$$\widehat{I}(P) = \psi(n) + \psi(k) - \frac{1}{k} - \frac{1}{n} \sum_{i=1}^{n} \psi\left(MC_k^x(p_i)\right) + \psi\left(MC_k^y(p_i)\right), \tag{5}$$

where ψ is the digamma function. This is $\psi(z) = -C + \sum_{t=1}^{z-1} \frac{1}{t}$ for $z \in \mathbb{N}^+$ and $C \approx 0.577$ being the Euler-Mascheroni constant.

While k is a parameter of this estimator, it is generally recommended [15, 16, 19] to use a small k, that is $k \le 10$. Gao et al. [9] have proven that the KSG is a consistent estimator for fixed k, that is, it converges towards the true value with increasing sample size.

4 ITERATIVE ESTIMATION

In this section we present IMIE, our iterative estimator for MI. The core concept of our approach is considering the KSG estimate itself as the mean of a random variable with a finite population. Using subsamples of this population for early estimates offers beneficial properties such as an expected value equal to the KSG estimate and convergence to the KSG for large sample sizes.

We first present IMIE and its underlying data structure as well as the algorithms for the initialization and for subsequent iterations. Then we describe our approach for nearest neighbor search, which is better for iterative algorithms than the standard procedures. Finally, we describe the statistical bounds that IMIE provides with its estimates.

4.1 IMIE

For brevity, we introduce some notation in addition to the one from Section 3. For a point $p \in P$, we define the *pointwise estimate*

$$\Psi(p) = \psi\left(MC_k^x(p)\right) + \psi\left(MC_k^y(p)\right). \tag{6}$$

The set of all pointwise estimates is $\rho = \{\Psi(p_1), \dots, \Psi(p_n)\}$. Seeing ρ as a finite population of size n with mean μ_ρ, Equation 5 can be rewritten as

$$\widehat{I}(P) = \psi(n) + \psi(k) - \frac{1}{k} - \mu_\rho. \tag{7}$$

Using a (random) subsample $\varrho \subseteq \rho$, its mean μ_ϱ is an (unbiased) estimation of μ_ρ. This in turn yields an (unbiased) estimate of $\widehat{I}(P)$,

$$\widehat{I_\varrho}(P) = \psi(n) + \psi(k) - \frac{1}{k} - \mu_\varrho. \tag{8}$$

Data Structure 1: IMIE

```
struct {
    Point[] $P$
    Real $Mean$, $Var$
    Int $k, m$
    Int[] $Order_R$, $Order_x$, $Order_y$
    Real $Offset$
};
```

Algorithm 2: Init (P, k)

```
1  Persist $k$ and $P$                                      $O(n)$
2  $Mean, Var, m \leftarrow 0$                               $O(1)$
3  $Order_R, Order_x, Order_y \leftarrow (0, 1, \dots, |P|-1)$   $O(n)$
4  Sort $Order_x$ and $Order_y$                              $O(n \log n)$
5  $Offset \leftarrow \psi(|P|) + \psi(k) - \frac{1}{k}$       $O(1)$
```

The variance σ_ϱ^2 of our subsample serves as a quality indicator of this approximation, which we further discuss in Section 4.3. The idea of IMIE is to maintain a subsample ϱ and use $\widehat{I_\varrho}(P)$ to estimate $\widehat{I}(P)$. Each iteration then increases the sample size of ϱ by one, to improve the estimate. Starting with an empty set, this means there are exactly $|P|$ iterations before IMIE yields exactly the same result as the KSG, i.e., $\widehat{I_\varrho}(P) = \widehat{I}(P)$.

Data Structure. IMIE uses and stores P and k as well as some additional information listed in Data Structure 1. In the following we use the zero-indexed array notation $P[i] = p_{i+1}$. Contrary to the original data sample P, we do not store ϱ explicitly. Instead we store its mean *Mean*, its variance *Var* and size, which is the number of performed iterations m. To maintain the current variance efficiently, we use the online algorithm by Welford [34]. To ensure that ϱ is a random subsample of ρ, we need to draw without replacement. To this end, IMIE maintains an array of indices $Order_R$, where index i at position j means that $\Psi(p_i)$ is added to ϱ in the j-th iteration. The positions of this array are randomly swapped during iterations to perform the random selection. This enables a fast selection of a random element without replacement in each iteration. In addition, we maintain two arrays $Order_x$ and $Order_y$ containing references to all points in P ordered by their x- and y-value, respectively. For instance, index i at $Order_x[0]$ means that p_i has the smallest x-value in P, i.e., $p_i = \arg\min_{p \in P} x_p$. These ordered arrays are used to find nearest neighbors, as described in Section 4.2. Finally, we store the $Offset = \psi(n) + \psi(k) - \frac{1}{k}$. With this, the (preliminary) MI estimate is available as $\widehat{I_\varrho}(P) = Offset - Mean$.

Methods. We now present the two methods Init and Iterate. See Algorithms 2 and 3, together with amortized time complexities, derived in Section 5. Init ensures the proper state of Data Structure 1 before the first iteration, i.e., preparing all variables assuming that $|\varrho| = 0$. Observe that Init is a straightforward method for the simple case of static data with two attributes. For other scenarios, such as high-dimensional or streaming data, some adjustments to the initialization may be appropriate, as discussed in Section 5.3.

Iterate increases the size of sample ϱ by one. This requires computing $\Psi(p)$ for a random $p \in P$ with $\Psi(p) \notin \varrho$. Iterate consists of three phases. In the first one (Lines 1-3), we select a random point p of P that has not been selected earlier. After

Algorithm 3: Iterate

```
1  $ID \leftarrow$ Draw random integer from $[m, n-1]$            $O(1)$
2  Swap values of $Order_R[m]$ and $Order_R[ID]$             $O(1)$
3  $p \leftarrow P[Order_R[m]]$                              $O(1)$
4  $kNN(p) \leftarrow$ NNSearch$(p)$       (see Algorithm 4) $O(\sqrt{n})$
5  Compute $\delta_k^x(p)$, $\delta_k^y(p)$                     $O(1)$
6  Compute $MC_k^x(p)$, $MC_k^y(p)$                       $O(\log n)$
7  $\Psi(p) \leftarrow \psi(MC_k^x(p)) + \psi(MC_k^y(p))$          $O(1)$
8  $m \leftarrow m + 1$                                    $O(1)$
9  $Diff_{\text{old}} \leftarrow \Psi(p) - Mean$                    $O(1)$
10 $Mean \leftarrow Mean + \frac{Diff_{\text{old}}}{m}$               $O(1)$
11 $Diff_{\text{new}} \leftarrow \Psi(p) - Mean$                    $O(1)$
12 $Var \leftarrow \frac{Var \cdot (m-1) + Diff_{\text{old}} \cdot Diff_{\text{new}}}{m}$   $O(1)$
```

$m - 1$ iterations, we swap the index at position m of $Order_R$ with the index at a random position behind $m - 1$. This ensures that we do not use any index twice, since positions before m are not considered, and that each unused index has the same probability of being selected. This random swap is one step of the Fisher-Yates Shuffle in the version of Durstenfeld [8], which fully randomizes the order of a sequence. The second phase (Lines 4-7) computes $\Psi(p)$ using the ordered lists $Order_x$ and $Order_y$. The last phase (Lines 8-12) performs the online algorithm [34] to maintain mean and variance of a sample, in our case ϱ.

Example 4.1. Disregarding the dashed lines for now, Figure 3 illustrates the state of Data Structure 1 after initialization and before the first iteration. For the first iteration, we draw an integer ID from $\{0, \dots, n-1\}$. Suppose that we drew 5. We swap the content of $Order_R[0]$ and $Order_R[5]$. $Order_R[0]$ now contains 6. This means that this iteration adds $\Psi(p_6)$ to our implicit sample ϱ. We then determine its nearest neighbor $1NN(p_6) = \{p_{15}\}$, the distances $\delta_1^x(p_6)$ and $\delta_1^y(p_6)$ as well as the marginal counts $MC_1^x(p_6) = 1$ and $MC_1^y(p_6) = 3$. The dashed lines in Figure 3 illustrate the area of counted points in x and y-direction, respectively, identically to Figure 2. It follows that $\Psi(p_6) = \psi(1) + \psi(3) = 0.346$. Substituting the appropriate variables, the remaining values are set accordingly, i.e., $m = 0 + 1 = 1$, $Mean = 0 + \frac{0.346}{1} = 0.346$ and $Var = \frac{0 \cdot 0 + 0 \cdot 0.346}{1} = 0$. The second iteration is analogous, drawing $ID = 6$ at random from $\{1, \dots, n-1\}$, thus choosing p_7. Its nearest neighbor is p_8, and the marginal counts are $MC_1^x(p_7) = 1$ and $MC_1^y(p_7) = 6$, cf. the dashed lines in Figure 4. As a result, it is $\Psi(p_7) = \psi(1) + \psi(6) = 1.129$. Analogously to the first iteration, the remaining values are $m = 1 + 1 = 2$, $Mean = 0.346 + \frac{0.783}{2} = 0.738$ and $Var = \frac{0 \cdot 1 + 0.783 \cdot 0.391}{2} = 0.153$. Figure 4 graphs the state of Data Structure 1 after both iterations, and the new MI estimate is $1.164 - 0.738 = 0.426$.

4.2 Nearest-Neighbor Search

A computation-intensive step in Iterate is the computation of nearest neighbors, which also is a key step for static estimation with the KSG. The classic solution [19, 31] is using space-partitioning trees, which are optimal in terms of computational complexity [32]. This efficiency is achieved because the slow tree construction is performed once, and each nearest-neighbor search afterwards is fast. Contrary to the traditional KSG estimation, it is not known beforehand how many nearest-neighbor searches IMIE performs. Constructing such a tree for IMIE would

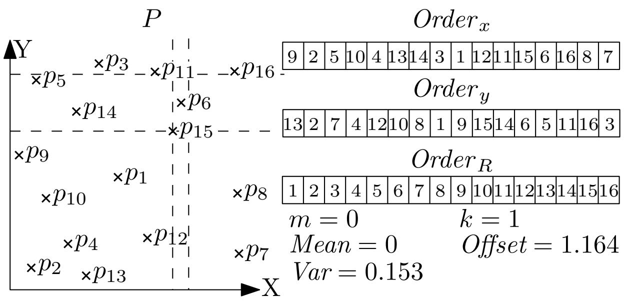

Figure 3: State of IMIE after initialization.

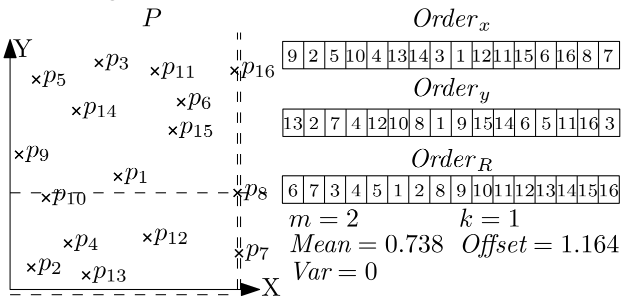

Figure 4: State of IMIE after two iterations ($\Psi(p_6)$ and $\Psi(p_7)$).

not only delay the first estimate, but may also be an inefficient choice overall if only few iterations take place. The opposite, i.e., searching nearest neighbors without any preparation, is a linear search. Each iteration would then require time linear in the number of data points. Since IMIE should offer both fast iterations and preliminary estimates after a short time, our approach is a compromise between these two options. The general idea is to use sorted arrays to perform a "guided" linear search that offers a good amortized time complexity (cf. Section 5). In the following, we elaborate on our NNSearch approach.

Let p be the point whose nearest neighbor we are searching for and q the nearest neighbor we have found so far. Then any point r with $|x_p - x_r| > \|p - q\|_\infty$ cannot be a nearest neighbor with the maximum norm. This means that we only have to consider the interval $[x_p - \|p-q\|_\infty, x_p + \|p-q\|_\infty]$ in the sorted array $Order_x$. When we find a closer point during the search, this interval gets smaller, and fewer points need to be considered. For the y-values, this is analogous. To reduce the number of worst-case scenarios, we perform this search simultaneously in both directions and terminate when either one terminates. See Algorithm 4 for an algorithmic description of NNSearch.

Example 4.2. Figure 5 illustrates an exemplary run of this procedure for $k = 1$. The figure shows four states corresponding to the variables of NNSearch(p) after $0, \ldots, 3$ loops. The query point p is the filled square, and a projection of the points to their x- and y-coordinates is shown at the bottom and the left side, respectively. These projections indicate the order of points in $Order_x$ and $Order_y$, respectively. Each state after the first loop also illustrates the variables of NNSearch. The nearest neighbor found so far is marked with a circle and is labeled *NN*, and the distance $\delta_{\max} = \|p - NN\|_\infty$ is used for the dashed lines that highlight the remaining area of nearest neighbor candidates. Points accessed via $Order_x$ in a previous iteration are marked with a diagonal stripe from the upper left to the lower right. This is done analogously for $Order_y$. Each loop considers the next

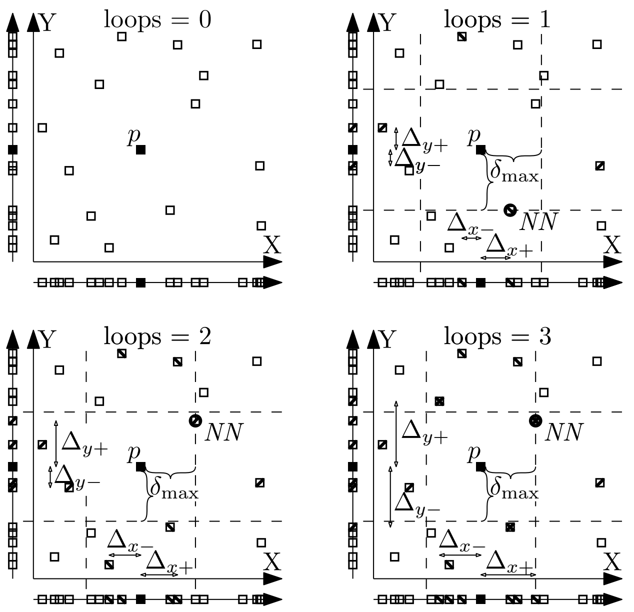

Figure 5: Illustration of Algorithm 4 for each loop.

unmarked point in both directions for both $Order_x$ and $Order_y$. Additionally, the small arrows illustrate the minimal distances $\Delta_{\circ\pm}$ for any further point accessed when iterating over $Order_x$ or $Order_y$ in the respective direction. After the third loop, the arrows of Δ_{y+} and Δ_{y-} both exceed the area of the remaining candidates, represented by the dashed lines. This means that all relevant candidates have been considered via $Order_y$, and that the current nearest neighbor is correct.

4.3 Statistical Quality Indicators

Finally we present statistical guarantees for early estimates by IMIE. Since ϱ is a subsample of ρ, statistical tests with μ_ϱ and σ_ϱ^2 yield statistically significant assertions regarding μ_ρ. Equations 7 and 8 give way to analogous assertions for $\widehat{I}(P)$.

Theorem 4.3 ([27]). *Let ρ be a finite population of size n with mean μ_ρ and a variance σ_ρ^2. When drawing an i.i.d. sample ϱ of size m from ρ, the sample mean μ_ϱ has an expected value of $\mathbb{E}(\mu_\varrho) = \mu_\rho$ and a variance of $\sigma_{\mu_\varrho}^2 = \frac{\sigma_\rho^2}{m}\left(\frac{n-m}{n-1}\right)$.*

Proof. See [27]. □

While the classic version of the Central Limit Theorem is not formulated for finite populations, it has been proven that some variations are applicable, and that μ_ϱ is approximately normally distributed [27]. In other words, drawing a sample of size m with a sample mean μ is as likely as drawing μ from $\mathcal{N}(\mu_\rho, \sigma_{\mu_\varrho})$ with $\sigma_{\mu_\varrho} = \sqrt{\sigma_{\mu_\varrho}^2}$. So we can estimate the probability that a sample mean μ_ϱ is off by more than a specified value $\epsilon > 0$ by using the cumulative distribution function Φ of the standard normal distribution $\mathcal{N}(0,1)$. This is illustrated in Figure 6 and is formally described as

$$\Pr[|\mu_\varrho - \mu_\rho| \geq \epsilon] = 2 \cdot \Phi\left(\frac{-\epsilon}{\sigma_{\mu_\varrho}}\right). \tag{9}$$

Algorithm 4: NNSearch(p)

```
1  $i_x, i_y \leftarrow$ index of $p$ in $Order_x$, $Order_y$, respectively
2  $\Delta_{x+}, \Delta_{x-}, \Delta_{y+}, \Delta_{y-}, loops \leftarrow 0$
3  $\delta_{\max} \leftarrow \infty$
4  $NN \leftarrow \{\}$
5  while $\min(\Delta_{x-}, \Delta_{x+}) < \delta_{\max} \wedge \min(\Delta_{y-}, \Delta_{y+}) < \delta_{\max}$ do
6      $loops \leftarrow loops + 1$
7      if $\Delta_{x+} < \delta_{\max}$ then
8          $\Delta_{x+} \leftarrow |x_p - x_{Data[Order_x[i_x+loops]]}|$
9          UpdateNN($P[Order_x[i_x + loops]]$)
10     if $\Delta_{x-} < \delta_{\max}$ then
11         $\Delta_{x-} \leftarrow |x_p - x_{Data[Order_x[i_x-loops]]}|$
12         UpdateNN($P[Order_x[i_x - loops]]$)
13     if $\Delta_{y+} < \delta_{\max}$ then
14         $\Delta_{y+} \leftarrow |y_p - y_{Data[Order_y[i_y+loops]]}|$
15         UpdateNN($P[Order_y[i_y + loops]]$)
16     if $\Delta_{y-} < \delta_{\max}$ then
17         $\Delta_{y-} \leftarrow |y_p - y_{Data[Order_y[i_y-loops]]}|$
18         UpdateNN($P[Order_y[i_y - loops]]$)
19 return NN

   function UpdateNN($q$)
1  if $\|p - q\|_\infty < \delta_{\max}$ then
2      insert $q$ into $NN$
3      if $|NN| > k$ then
4          remove $\arg\max_{r \in NN} \|r - p\|_\infty$ from $NN$
5      if $|NN| = k$ then
6          $\delta_{\max} \leftarrow \max_{r \in NN} \|r - p\|_\infty$
```

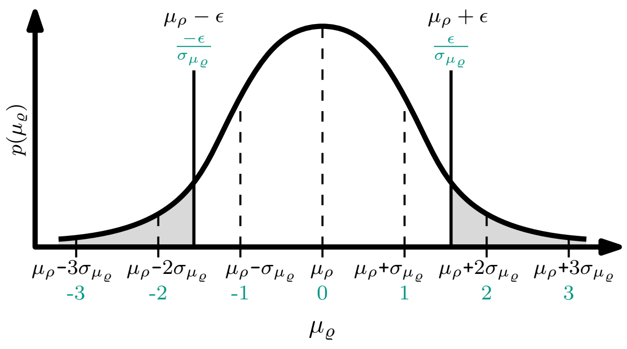

Figure 6: Illustration of the normal distributions $\mathcal{N}(\mu_\rho, \sigma_{\mu_\varrho})$ (upper labels) and $\mathcal{N}(0,1)$ (lower labels).

Alternatively, one can specify a tolerated error probability α and obtain a confidence interval. Let Φ^{-1} be the inverse cumulative distribution function of the standard normal distribution, i.e., $\Phi(\Phi^{-1}(\alpha)) = \alpha$. Then the mean of a sample deviates with probability $1-\alpha$ by at most $|\Phi^{-1}(\frac{\alpha}{2})| \cdot \sigma_{\mu_\varrho}$ from μ_ρ. This is because both tails of the distributions have to be considered. More formally, it is

$$Pr\left[\mu_\varrho - \left|\Phi^{-1}\left(\frac{\alpha}{2}\right)\right| \sigma_{\mu_\varrho} \leq \mu_\rho \leq \mu_\varrho + \left|\Phi^{-1}\left(\frac{\alpha}{2}\right)\right| \sigma_{\mu_\varrho}\right] \approx 1 - \alpha. \tag{10}$$

Lastly, there are two more considerations necessary to obtain these statistical guarantees from IMIE. One is that the variance σ_ρ^2, which is used to determine $\sigma_{\mu_\varrho}^2$ in Theorem 4.3, is not known. Using the approximation $\sigma_\rho^2 \approx \sigma_\varrho^2 \frac{m(n-1)}{(m-1)n}$ yields the unbiased approximation $\sigma_{\mu_\varrho}^2 \approx \frac{\sigma_\varrho^2 (n-m)}{(m-1)n}$, see [27]. The other point is the *multiple testing problem.* The probabilities for errors only hold for individual tests. But when performing multiple tests to obtain a statistically significant result, the chance of an erroneous result in one test is higher. For instance, this occurs when the response to a statistically insignificant test result is to perform another test, evaluating the result without considering the first, inconclusive result. We illustrate this effect with an example.

Example 4.4. Consider an instance of IMIE that has performed some iterations so far. We use the current mean and var to perform a statistical test whether $\hat{I}(P)$ is above a threshold t. We accept an error chance of 10%. Let us assume that the result of the first test is not significant enough, i.e., the probability is less than 90% based on the current sample. We iterate our estimate a few times and perform a second test, which achieves the desired probability of 90%. However, if $\hat{I}(P)$ is below t, the likelihood that a test reports false certainty based on an unlikely sample increases with each sample. For two tests, the probability of obtaining false certainty is then $Pr[\hat{I}(P) < t] = 1 - (1-0.1)^2 = 0.19$.

To account for this problem, we use the correction due to Šidák [29]: To obtain an overall error chance of α, the error chance allowed for the c-th test is $\alpha_{\text{test}} = 1 - (1-\alpha)^{\frac{1}{c}}$.

To summarize this section, we present the full formula for the c-th statistical test whether $\widehat{I}(P)$ is greater than a threshold t, using variables from IMIE.

$$\Pr[\widehat{I}(P) > t] \approx 1 - \left(1 - \Phi\left(\frac{Offset - Mean - t}{\sqrt{\frac{Var \cdot (|P| - m)}{(m-1)|P|}}}\right)\right)^c \tag{11}$$

Since we approximate σ_ρ^2, this equation is not exact. On the other hand, the Šidák-correction is very conservative in our case. Namely, when iterating IMIE, the new sample is a superset of the previous sample. This means that the tests based on these samples are dependent, and that the effect of the multiple testing problem is less pronounced. Ultimately, we do not have any formal result to which degree these effects do cancel each other out. In all our experiments in Section 6 however, the error rate never exceeds the bounds established in this section.

5 TIME COMPLEXITY

Now we derive the time complexity of IMIE. First, we do so for our nearest-neighbor search. We then use this result to derive the complexity for initializing and iterating IMIE. Finally, we discuss potential improvements for specific scenarios.

5.1 Nearest-Neighbor Search

We establish the time complexity of Algorithm 4. Each call of UpdateNN(q) takes time in $O(k)$ to compute the (arg) $\max_{r \in NN}$ $\|r - p\|_\infty$. Additionally, let $\mathbb{I}(p)$ be the number of loops performed by NNSearch(p) before terminating. Then the time complexity is in $O(\log n + \mathbb{I}(p) \cdot k)$. Namely, the only other step that is not an elementary assignment is computing the indices of p in $Order_x$ and $Order_y$ with binary search, in $O(\log n)$. However, $\mathbb{I}(p)$ is linear in n for the worst case. Figure 7 shows such a degenerative case, where all points except for p and q are equally distributed among

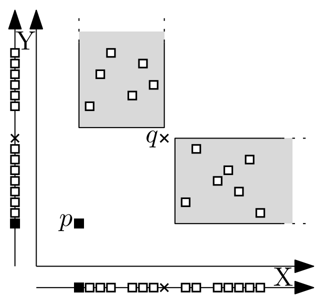

Figure 7: A degenerative case for NNSearch.

the two grey areas. In this case, NNSearch(p) cannot discover the nearest neighbor q via $Order_x$ or $Order_y$ with fewer than $\frac{n-2}{2}$ loops. However, we prove the nontrivial bound $\sum_{p\in P}\mathbb{I}(p) \leq (4\cdot\sqrt{n\cdot k}+1)\cdot n$ below.

To prove this bound, we first introduce some additional notation and properties for the several executions of Algorithm 4. For each point $p \in P$, let $V_x(p)$ and $V_y(p)$ be the set of positions of $Order_x$ and $Order_y$, respectively, accessed by NNSearch(p). Additionally, let Pos_x^p be the position of $Order_x$ containing the reference to a point p, i.e., $P[Order_x[Pos_x^p]] = p$. The set of points that access this position during NNSearch(q) is $R_x(p) = \{q \in P : Pos_x^p \in V_x(q)\}$. Pos_y^p and $R_y(p)$ are defined analogously using $Order_y$ instead of $Order_x$. By definition, it is

$$\sum_{p\in P}|V_x(p)| = \sum_{p\in P}|R_x(p)|, \tag{12}$$

as both count the total number of accesses of $Order_x$ across all searches.

Note that NNSearch(q) for points $q \in R_x(p)$ often performs several loops before accessing Pos_x^p. In particular, there are only two points q^+ and q^- such that NNSearch(q^+) and NNSearch(q^-) access Pos_x^p during their first loop. These two points are the points corresponding to the neighboring positions of Pos_x^p, i.e., $q^+/q^- = P[Order_x[Pos_x^p \pm 1]]$. More specifically, for each $c \in \mathbb{N}_0$, there exist at most two points whose positions are exactly c steps away. This is because $Order_x$ is a linear order of a finite set of elements. As a result, $R_x(p)$ defines a lower bound for $\sum_{q\in R_x(p)}\mathbb{I}(q)$. Formally, for each $p \in P$ it is

$$2\cdot\sum_{i=1}^{\frac{R_x(p)-1}{2}} i \leq 0+0+1+1+\cdots+\left\lfloor\frac{R_x(p)-1}{2}\right\rfloor \leq \sum_{q\in R_x(p)}\mathbb{I}(q). \tag{13}$$

Next, we also consider the properties of $V_y(\cdot)$ and $R_y(\cdot)$. During each loop of a search NNSearch(p), it is $\min(\Delta_{y-},\Delta_{y+}) < \delta_{\max}$. This means that NNSearch(p) accesses at least one new position of $Order_y$ in Line 14 or Line 17. It follows that

$$\mathbb{I}(p) \leq |V_y(p)| \tag{14}$$

and with Equation 13, it is

$$2\cdot\sum_{i=1}^{\frac{R_x(p)-1}{2}} i \leq \sum_{q\in R_x(p)}|V_y(q)|. \tag{15}$$

Now, we use the fact that NNSearch stops accessing new positions in a certain direction when this direction cannot offer a closer nearest neighbor. In the following lemma, we use this pattern to limit the number of points p where NNSearch(p) accesses certain positions of $Order_x$ and $Order_y$. That is, for each combination of a position of $Order_x$ and $Order_y$, there is only a small number of points whose nearest neighbor search accesses both.

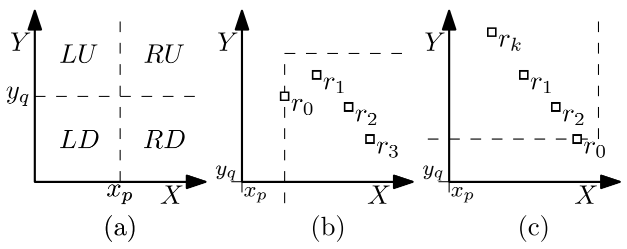

Figure 8: Illustration of arrangements in Claim 5.1. (a) Partitioning of $\mathbb{R}^2$ based on (x_p, y_p). (b),(c) Two cases of layouts of RU.

Lemma 5.1. *For any two points $p, q \in P$, it is $|R_x(p)\cap R_y(q)| \leq 4\cdot k$.*

Proof. We consider a partitioning of $\mathbb{R}^2$ into four axis-aligned quadrants RU, RD, LD and LU centered at (x_p, y_q), as illustrated in Figure 8a. To ensure that any point $r \in P\setminus\{p,q\}$ is in exactly one partition, equalities such as $x_r = x_p$ and $y_r = y_q$ are resolved by their ordering in $order_x$ and $order_y$, respectively. For the sake of contradiction, suppose that there are $k+1$ points $\{r_0,\ldots,r_k\} = R_{RU} \subseteq R_x(p)\cap R_y(q)$ in the area RU. We discern between two cases regarding the arrangement of these points.

In the first case, it is $\max_{r,s\in R_{RU}}|x_r - x_s| \geq \max_{r,s\in R_{RU}}|y_r - y_s|$. That is, the largest difference in x-values among points in RU is at least as large as any difference in y-values among RU. For all r in R_{RU}, it is $Pos_x^r > Pos_x^p$. Without loss of generality, let r_0 be the point closest to p and r_k the furthest from p in $Order_x$, respectively. Formally, $r_0 = \arg\min_{r\in R_{RU}} Pos_x^r$ and $r_k = \arg\max_{r\in R_{RU}} Pos_x^r$. This implies that $|x_{r_k} - x_{r_0}| \geq \|r_k - r\|$ for all $r \in R_{RU}$. As illustrated in Figure 8b, NNSearch(r_k) accesses Pos_x^r for all $r \in R_{RU}\setminus\{r_k\}$ before accessing Pos_x^p. After accessing $Pos_x^{r_0}$ and calling UpdateNN(r_0), it holds for the variables in NNSearch(r_k) that $\delta_{\max} = \Delta_{x-}$. The dashed line in Figure 8b illustrates this. This means that NNSearch(r_k) does not access further positions of $Order_x$ in this direction, and thus there is a contradiction to $r_k \in R_x(p)$.

The second case, $\max_{r,s\in R_{RU}}|x_r - x_s| < \max_{r,s\in R_{RU}}|y_r - y_s|$, is symmetric to the first one using $Order_y$ instead of $Order_x$. With r_0 and r_k being the closest and furthest point from q in $Order_y$, NNSearch(r_k) also accesses all positions corresponding to other points in R_{RU} before Pos_y^q. Analogously, it is $\delta_{\max} = \Delta_{y-}$ after calling UpdateNN(r_0), as illustrated in Figure 8c. Thus NNSearch(r_k) does not access the position Pos_y^q, which contradicts $r_k \in R_y(q)$.

As a result there are at most k points from $R_x(p)\cap R_y(q)$ in RU. By symmetry, the same is true for RD, LD, LU. This yields the lemma. □

Combining this lemma with other equations introduced in this section yields the following limit for the total number of iterations performed by all searches.

Lemma 5.2. *For a set $P \subseteq R^2$ of points, the total number of iterations performed by NNSearch(p) for all $p \in P$ is bounded as $\sum_{p\in P}\mathbb{I}(p) \leq (4\cdot\sqrt{n\cdot k}+1)\cdot n$.*

Proof. Following Lemma 5.1, each position of $Order_y$ is accessed at most $4 \cdot k$ times by searches accessing one specific position of $Order_x$. More formally, with Equation 15, it is for each $p \in P$

$$
\begin{aligned}
4 \cdot k \cdot n &\geq \sum_{q \in R_x(p)} |V_y(q)| \geq 2 \cdot \sum_{i=1}^{\frac{R_x(p)-1}{2}} i \\
&= 2 \cdot \frac{\frac{R_x(p)-1}{2}\left(\frac{R_x(p)-1}{2}+1\right)}{2} \geq \left(\frac{R_x(p)-1}{2}\right)^2 \\
2\sqrt{k \cdot n} &\geq \frac{R_x(p)-1}{2} \\
4 \cdot \sqrt{k \cdot n} &\geq R_x(p) - 1 \qquad (16)
\end{aligned}
$$

Combining Equations 12, 14 and 16 yields

$$\sum_{p \in P} \mathbb{I}(p) \leq \sum_{p \in P} |V_x(p)| = \sum_{p \in P} |R_x(p)| \leq (4 \cdot \sqrt{k \cdot n} + 1) \cdot n. \quad (17)$$

□

Because k is a small constant, the time complexity of performing NNSearch for all points is in $O(n \cdot \sqrt{n})$. So the complexity for each individual search is in amortized $O(\sqrt{n})$.

Theorem 5.3. *NNSearch has an amortized time complexity of* $O(\sqrt{n})$.

5.2 Init and Iterate

We derive the time complexity for initializing and iterating IMIE.

In Init, most operations are assignments, of constant size (Lines 2, 4) or of linear size (Lines 1, 3). The only exception is sorting $Order_x$ and $Order_y$, which is $O(n \log n)$. So the overall runtime of Init is $O(n \log n)$. However, we show in the following section that more efficient variants are possible for scenarios encompassing more than one estimation task. Furthermore, our experiments in Section 6 indicate that the actual runtime of Init often is negligible in comparison to Iterate.

As for the runtime of Iterate, there are only two steps that are not elementary assignments of constant size. One step is computing the marginal counts $MC_k^x(p)$ and $MC_k^y(p)$ (Line 6). It can take place in $O(\log n)$, with binary searches on the sorted arrays as follows. Let i be the smallest integer in $\{0, \ldots, |P|-1\}$ with $x_{P[Order_x[i]]} \geq x_p - \delta_k^x(p)$. Similarly, let j be the largest integer in $\{0, \ldots, |P|-1\}$ with $x_{P[Order_x[j]]} \leq x_p + \delta_k^x(p)$. Because $Order_x$ contains all points sorted by x-coordinate, it is $MC_k^x(p) = j - i$. The other marginal count $MC_k^y(p)$ is available analogously, using $Order_y$, y_p and $\delta_k^y(p)$ instead. The other step is the nearest neighbor search (Line 4), which has an amortized time complexity of $O(\sqrt{n})$ by Theorem 5.3. As a result, Iterate also has an amortized time complexity of $O(\sqrt{n})$. Since P and thus ρ contain n elements, IMIE requires time in $O(n\sqrt{n})$ to reach the final estimate, the one equal to the KSG estimate. This means that IMIE is only slightly slower in reaching the final result than the lower bound $O(n \log n)$ for algorithms without preliminary results [32].

5.3 Scenario-specific Improvements

The initialization procedure presented in Section 4 explains the core concept and properties. Init has been described in a way that is always applicable. However, in many scenarios a user has more than one estimation task based on the same or similar data. Think of estimating the MI for overlapping attribute pairs when searching for strongly dependent attributes. In such a case, it may not be necessary for each instance of IMIE to sort the arrays from scratch, which is the primary computational burden of Init. In this section we present the improvements possible for IMIE in scenarios with high-dimensional data as in Scenario 1 and with streaming data as in Scenario 2. We consider benefits over both the naïve initialization of IMIE as well as the non-iterative estimation.

High Dimensional Data. The number of attribute pairs grows quadratically with the number of attributes. If the data has d attributes, the number of pairs is $\frac{d \cdot (d-1)}{2}$. We now consider using one instance of IMIE for each pair to obtain the pairwise MI estimates. A naïve initialization of these instances would require time in $O(d^2 \cdot n \log n)$. However, we only need to sort the points once per attribute to use the respective sorted arrays for several attribute pairs. This reduces the time complexity for initialization to $O(d \cdot n \log n)$.

Non-iterative estimators for the KSG use two-dimensional space-partitioning trees [19, 31, 32]. This means that each attribute pair requires a different tree, which prohibits a similar improvement. In addition, non-iterative estimators must commit the computation time beforehand. IMIE in contrast can budget computation time between different pairs of attributes, depending on which pairs a user finds interesting, based on the preliminary estimates.

Data streams. With data streams, new data is arriving continuously, and computation time is limited. We consider estimating the current MI using all points whenever new data arrives. This means that most data points remain unchanged. When maintaining up-to-date MI estimates, IMIE can reuse the instance of Data Structure 1 used for the previous estimate instead of another initialization. Considering Data Structure 1, only the adjustment of $Order_x$, $Order_y$ and $Order_R$ does not incur constant costs when a new data point arrives. Adjusting $Order_x$ and $Order_y$ to accommodate new data can take place in $O(\log n)$.[1] Since $Order_R$ is shuffled randomly during the estimation, IMIE can also start off with a (partially) shuffled order and only needs the addition of new indices for new data items. This means that initialization of a new estimator on a data stream can take place in $O(\log n)$ instead of $O(n \log n)$.

Additionally, if the delays between items from the data streams are irregular in length, IMIE automatically offers the best estimate for the time available. Previous work regarding efficient, non-iterative MI estimation on streams [4, 14, 32] imposes a fixed computation time, and there is no easy adjustment if items arrive faster.

Takeaway. While the time complexity of Init may appear prohibitively large in the previous section, we have demonstrated in this section that concrete settings can allow for more efficient solutions. Note that the improvements described are not mutually exclusive. This means that both improvements can be combined when dealing with high-dimensional data in the form of streams. Table 1 summarizes the impact of these techniques on the initialization of IMIE for pairwise MI estimation between d data streams.

[1]From a technical perspective, this time complexity requires $Order_x$ and $Order_y$ to be implemented as binary search trees. For simplicity we keep calling them sorted arrays.

Optimization	Time Complexity
Naïve application	$O(d^2 \cdot n \log n)$
Reuse previous data structure	$O(d^2 \cdot \log n)$
Reuse sorted dimensions	$O(d \cdot n \log n)$
Both	$O(d \cdot \log n)$

Table 1: Impact of optimization techniques for initializing IMIE for pairwise MI of d data streams.

6 EXPERIMENTS

In this section we investigate the performance of IMIE in terms of runtime and estimation quality. We also perform experiments to test the potential benefits from the statistical guarantees and the anytime property of IMIE.

As reference for the performance of IMIE we use the KSG (see Equation 5), because it offers high-quality estimations, and it is the basis of IMIE. To ensure competitive runtime of the KSG we use KD-Trees for its nearest-neighbor search, resulting in the optimal computation complexity of $O(n \log n)$ [32]. As a reference point for faster estimates with lower estimation quality, we use the KSG on subsamples of the data. Since the number of points subsampled can be expressed as a percentile of all points or as an absolute number, we introduce a notation for both. Using a random sample of $p\%$ from all data points to compute the KSG is denoted as *KSG%p*. Subsampling exactly q points at random from all data points to compute the KSG on this subsample is denoted as *KSG@q*.

Setup. All approaches and experiments are implemented in C++ and compiled using the Microsoft® C/C++ Optimizing Compiler Version 19.00. We use the non-commercial ALGLIB[2] implementation of KD-Trees for the KSG. We also use the non-commercial ALGLIB[2] implementation of the cumulative density function of the standard normal distribution Φ and its inverse Φ^{-1} when computing our statistical guarantees. All experiments are conducted on Windows 10 using a single core of an Intel® Core™ i5-6300U processor clocked at 2.4 GHz and 20GB RAM.

6.1 Data

In our experiments we use both synthetic and real-world data. As synthetic data, we use dependent distributions with noise used to compare MI estimators, see [15], uniform distributions used to compare MI with the maximal coefficient, see [16], as well as independent uniform and normal distributions. As real data, we use smart meter readings from an industrial plant (HIPE) [3], recorded smart phone sensors to recognize human activities (HAR) [7], and physical quantities for condition monitoring of hydraulic systems (HYDRAULIC) [12]. As proposed by the inventors of the KSG [19], we prevent duplicate points in real-world data by adding noise with minimal intensity. Beginning with real-world data, we now briefly describe the data specifics.

HIPE. This data set, available online[3], contains high-resolution smart meter data from 10 production machines over 3 months. This data has over 2000 attributes total and over 1.5 million data points. We consider a reduced data set containing the first 1000 data points of the machines "PickAndPlaceUNIT", "ScreenPrinter" and "VacuumPump2" with a grand total of 333 attributes.

[2]ALGLIB (www.alglib.net)

[3]https://www.energystatusdata.kit.edu/hipe.php

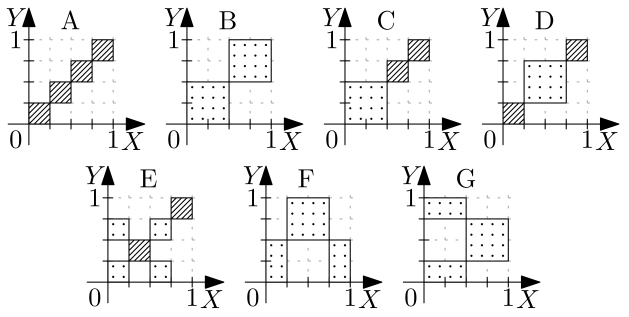

Figure 9: An overview of the uniform distributions used.

HAR. This data set, available at the UCI ML repository[4], features accelerometer and gyroscope sensor readings from smartphones to classify the activity of the human carrying the phone. The data set contains 561 attributes and a total of 5744 data points.

HYDRAULIC. This data set, available at the UCI ML repository[5], features recordings of several physical quantities such as temperature, vibrations and efficiency factors at different sampling rates. For our experiments we use all quantities with a sampling rate of 10 Hz. As a result, each of the 2205 data points has 480 attributes.

For synthetic data, we use the following distributions with known MI values [16, 23]. For distributions with a noise parameter σ_r, we vary σ_r between 0.1 and 1.0.

Linear To construct the point $p_i \in P$, we draw the value x_i from the normal distribution $N(0,1)$. Additionally, we draw some noise r_i from the normal distribution $N(0,\sigma_r)$, where σ_r is the noise parameter of the distribution. This yields the point $p_i = (x_i, x_i + r_i)$.

Quadratic This distribution is generated analogously to the linear distribution, except that the point is $p_i = (x_i, x_i^2 + r_i)$.

Periodic For each point $p_i \in P$, we draw the value x_i from the uniform distribution $U[-\pi,\pi]$. Additionally, we draw some noise r_i from the normal distribution $N(0,\sigma_r)$, where σ_r is the noise parameter. This yields the point $p_i = (x_i, \sin(x_i) + r_i)$.

Uniform The uniform distributions A to G we use are illustrated in Figure 9. Note that the striped areas contain twice as many points as the dotted areas. For these distributions, each striped area with size $0.25 \cdot 0.25$ contains 25% of all points, while dotted areas of the same size contain 12.5% of all points.

Independent Lastly, we use the distributions U_{IND} and N_{IND}, where each point consists of two values drawn independently from $U[0,1]$ and $N(0,1)$, respectively.

6.2 Synthetic Benchmarks

We first evaluate the concrete runtimes of IMIE. While we have established in Section 5 that the time complexity is competitive, actual runtimes may have constant factors that time complexity does not capture. We also look at the estimation quality offered by IMIE after a variable number of iterations. Since the true MI value of real data is unknown, we perform these experiments using

[4]http://archive.ics.uci.edu/ml/datasets/Smartphone+Dataset+for+Human+Activity+Recognition+(HAR)+in+Ambient+Assisted+Living+(AAL)

[5]http://archive.ics.uci.edu/ml/datasets/Condition+monitoring+of+hydraulic+systems

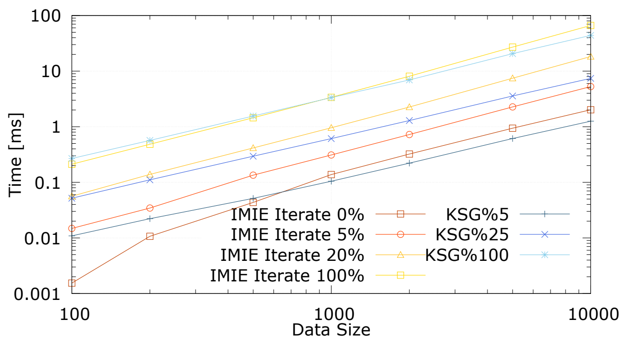

Figure 10: Average runtime depending on the data size for IMIE and subsampling variants.

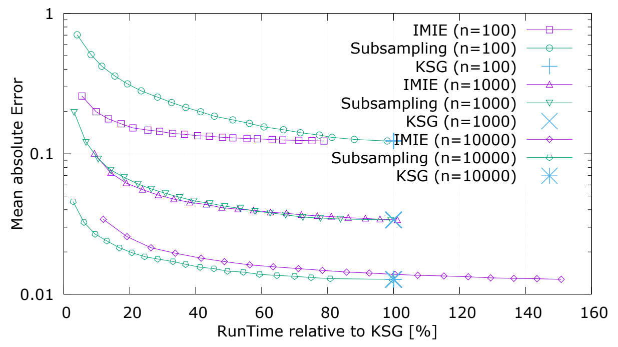

Figure 11: MAE of IMIE and subsampling depending on the runtime relative to KSG for the same data.

synthetic data. Each synthetic data set corresponds to one pair of attributes, for which we produce samples of varying sizes. For each pair, sample size and estimator, we perform 100 estimates and average the runtime and mean absolute error (MAE).

Figure 10 shows the average runtime of IMIE with various numbers of iterations and the KSG with various subsampling settings. Note that the concrete performance of IMIE when iterating until convergence and *KSG%*100 is very similar. This means that computing the exact KSG in the conventional way with a KD-tree and without preliminary results is not faster than using IMIE. Another point to observe is the difference in runtime between IMIE with only the initialization and IMIE that has performed some iterations. Even with only 5% of the iterations, IMIE already consumes more than double the time used for initialization. This shows that the time used for iterations quickly dominates the time required for initialization, even though Init has a high time complexity.

Figure 11 graphs the MAE of subsampling and IMIE depending on the runtime. The plot shows curves per estimator corresponding to a specific sample size, and the time is measured relative to the runtime of the naive KSG estimation for this size. Each point corresponds to the average runtime and absolute error of 100 estimations with the same number of iterations or subsampling size, respectively. In other words, the leftmost point corresponds to subsampling 5% or iterating 5% respectively, while the rightmost point uses all points or iterates until convergence, respectively. The result is that IMIE and KSG with subsampling offer the same time-quality-tradeoff for data of size 1000, with IMIE being somewhat faster for smaller data and somewhat slower for larger data. However, this assumes "optimal" subsampling, in the sense that it is known beforehand which subsampling size is desired. In cases where it is not clear how much time is available or how much time an estimate for a given subsample size takes, this is not given. The time spent finding a good subsampling size is discussed in Section 6.4.

6.3 Statistical Quality Indicators

Next, we investigate the practical relevance of the statistical guarantees. The scenario considered is high-dimensional data. A common information need for high-dimensional data is finding highly dependent attributes. In our experiments we want to know for each of the $\frac{d \cdot (d-1)}{2}$ pairs of attributes whether it is above or below a threshold τ. For IMIE we keep iterating the estimate and perform the test from Equation 11. To be precise, one test is performed for $I(P) > \tau$, and one test is performed for $I(P) < \tau$. To reduce the necessary Šidák-correction for our significance level α_{test} we perform these two tests only every 10 iterations. We start with a minimum sample size of 30 to reduce effects of minimal sample sizes. The exact choice of initial iterations and iterations between tests is arbitrary as long as they are not extreme, e.g., performing statistical tests with sample size one or iterating $\frac{|P|}{4}$ times between tests. Regarding the target significance level, we test different values $\alpha \in \{0.1, 0.05, 0.01, 0\}$. We use fixed percentile subsamples for comparison, i.e., *KSG%5*, *KSG%25* and *KSG%50*.

Figure 12 shows the results for the three real-world data sets with τ varying between 0 and 1. The figure contains two plots per data set. The "Error Rate" shows the number of pairs falsely classified over or under τ as a relative count of all pairs (left axis) and as absolute count (right axis). The "Run Time" shows the total execution time relative to the "naïve" estimation using the KSG (left axis) and as absolute time (right axis). The behavior depending on τ is different per data set. This is because the dependencies in the data are distributed differently. The closer τ is to the actual MI value, the easier it is for an approximate result to be above the threshold while the actual value is below, or vice versa. So it is harder to obtain statistical certainty that the actual value is above or below. To illustrate, the attributes in HIPE are largely independent. This yields MI values close to zero, resulting in high error rates for subsampling approaches and longer execution times for IMIE. Conversely, the attributes of HYDRAULIC are highly dependent. This in turn increases error rates and computation times, for subsampling and IMIE respectively, for higher threshold values.

Nevertheless, there are several common patterns. One is that IMIE does offer better time-quality-tradeoffs than subsampling. I.e., for each subsampling rate there is an α such that IMIE yields fewer errors using less time. A second pattern is that IMIE does adapt to "tough threshold values" by increasing the computation time used. Subsampling in turn makes more false claims. A third interesting pattern is that IMIE with $\alpha = 0$ is almost always faster than the naïve KSG estimation. IMIE can speed up such queries significantly with essentially no risk of error.[6]

6.4 Anytime Experiments

Now we test the performance of IMIE as anytime algorithm. In other words, the available time is not known beforehand. To

[6]Technically there could still be errors due to rounding, numerical evaluation of Φ^{-1} and the approximation in Section 4.3. However, no such error has occurred in any of our experiments.

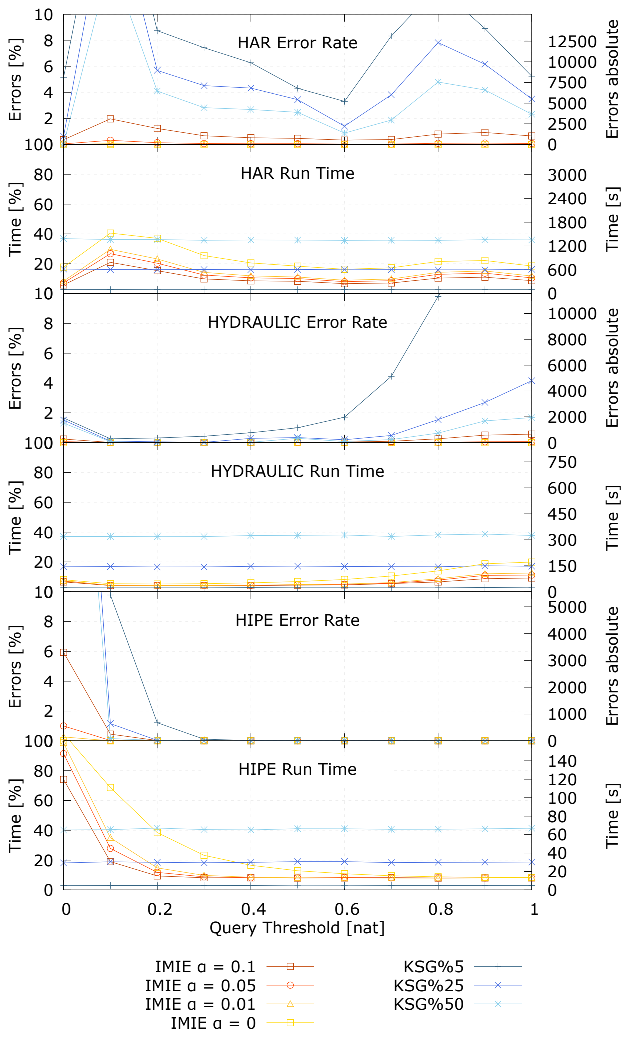

Figure 12: Time and error rate of IMIE and subsampling variants, depending on the chosen threshold τ.

mimic the behaviour of IMIE to improve the estimate with additional time, we also examine two strategies based on subsampling. KSG_{Lin} consecutively computes *KSG%10*, *KSG%20*, . . . , *KSG%100* as long as time is available. We also consider KSG_{Exp}, which computes *KSG@10*, *KSG@20*, *KSG@40*, *KSG@80*, etc. until no time is left.

For this experiment we randomly select 100 pairs of attributes from each real-world data set and estimate MI using IMIE, KSG_{Lin} and KSG_{Exp}. After some time the estimate is interrupted, and the most recent result is used. Since IMIE and subsampling appear most comparable in our synthetic benchmarks for data size n = 1000, we use the first 1000 data points of each attribute pair. Given the small scale of time per estimate (cf. Figure 10), we use 1000 estimators in parallel for each pair. One "iteration" then performs the next computation sequentially for each of these estimators.

Figure 13 shows the mean absolute error compared to a KSG estimate using 1000 points as well as the mean standard deviation of estimates for the same attribute pair. Additionally, for each estimate from IMIE we use the statistical quality indicator to determine the distance ϵ. Additionally, the plot displays the average value ϵ such that our preliminary estimate is wrong

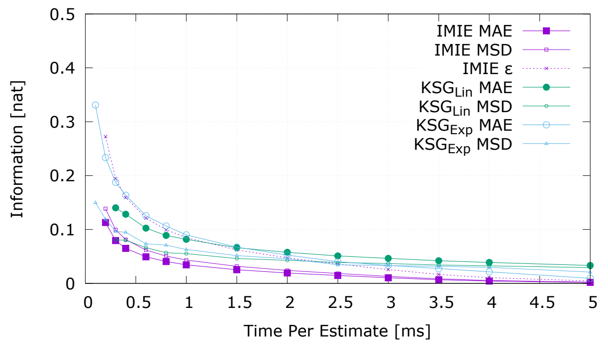

Figure 13: Mean absolute error (MAE) and mean standard deviation (MSD) of anytime approaches as well as the mean ϵ of IMIE.

by at most ϵ with a confidence of 95%. This value is obtained for each estimate using Equation 10. Note that KSG_{Lin} does not consistently produce estimates with time less than 0.3 ms per estimate, and IMIE does not consistently finish the first iteration in 0.1 ms.

A result of this experiment is that IMIE has smaller errors on average than the subsampling approaches, even though they are comparable in Figure 11. This is because the subsampling strategies are not efficient for iterative estimation. Estimates from previous iterations are discarded without further benefit, and iteration steps are less granular. This means that only a part of the overall time available is spent on the estimate that is ultimately presented.

6.5 Discussion

To summarize this section, IMIE offers a time-quality tradeoff similar to the one when estimating the KSG with varying subsampling settings. The time necessary for IMIE to converge towards the KSG result is slightly lower for small data and slightly higher for larger data, compared to the naive KSG estimation. But IMIE also offers preliminary results and achieves this time-quality tradeoff even if the time available is not known beforehand. This means that IMIE offers significant benefits for tasks that use these features, such as threshold queries or irregular data-stream processing, without notable drawbacks for regular tasks.

7 CONCLUSIONS

In this work, we have studied the iterative estimation of Mutual Information (MI). The goal has been to provide an estimator that offers a first estimate quickly and improves the estimation with additional time. It should also use the available time efficiently, even if the time available is not known beforehand. To this end, we have proposed IMIE.

By design, IMIE converges towards the same result as the popular MI estimator (KSG) by Kraskov et al. [19] after sufficiently many iterations. Before convergence, the preliminary results of IMIE also offer helpful statistical quality indicators which one can use to infer information regarding the final estimate, i.e., the KSG result. This can take the form of confidence intervals or the probability of surpassing a certain threshold. In addition to these formal results on estimation quality, we also have studied the time complexity of IMIE both in general and when tailored towards specific use cases. One result is that this complexity when computing the exact KSG estimate is only slightly larger than an optimal implementation to compute the KSG that does not offer any preliminary result.

Using extensive experiments, we have evaluated the practical performance of IMIE in terms of concrete runtimes and quality on real data. Among other results, IMIE remains competitive with its estimation quality per time, even when being compared to approaches without preliminary results. The experiments also demonstrate a significant runtime improvement when searching for attribute pairs with high MI in high-dimensional data.

ACKNOWLEDGMENTS

This work was partially supported by the DFG Research Training Group 2153: "Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation".

REFERENCES

[1] Periklis Andritsos, Renée J Miller, and Panayiotis Tsaparas. 2004. Information-theoretic tools for mining database structure from large data sets. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data.* 731–742.

[2] Ira Assent, Philipp Kranen, Corinna Baldauf, and Thomas Seidl. 2012. Anyout: Anytime outlier detection on streaming data. In *International Conference on Database Systems for Advanced Applications.* 228–242.

[3] Simon Bischof, Holger Trittenbach, Michael Vollmer, Dominik Werle, Thomas Blank, and Klemens Böhm. 2018. HIPE – an Energy-Status-Data Set from Industrial Production. In *Proceedings of ACM e-Energy (e-Energy 2018).* 599–603.

[4] Jonathan Boidol and Andreas Hapfelmeier. 2017. Fast mutual information computation for dependency-monitoring on data streams. In *Proceedings of the Symposium on Applied Computing.* 830–835.

[5] Lei Cao and Elke A Rundensteiner. 2013. High performance stream query processing with correlation-aware partitioning. *Proceedings of the VLDB Endowment* 7, 4 (2013), 265–276.

[6] Thomas M. Cover and Joy A. Thomas. 2006. *Elements of information theory* (2. ed.). 243–256 pages.

[7] Dua Dheeru and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[8] Richard Durstenfeld. 1964. Algorithm 235: Random Permutation. *Commun. ACM* 7, 7 (1964), 420.

[9] Weihao Gao, Sewoong Oh, and Pramod Viswanath. 2017. Demystifying fixed k-nearest neighbor information estimators. In *IEEE International Symposium on Information Theory (ISIT).* 1267–1271.

[10] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *Proceedings of the VLDB Endowment* 1, 1 (2008), 376–390.

[11] Eric A Hansen and Shlomo Zilberstein. 2001. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence* 126, 1-2 (2001), 139–157.

[12] Nikolai Helwig, Eliseo Pignanelli, and Andreas Schütze. 2015. Condition monitoring of a complex hydraulic system using multivariate statistics. In *Instrumentation and Measurement Technology Conference (I2MTC).* 210–215.

[13] Ihab F Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. 2004. CORDS: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data.* 647–658.

[14] Fabian Keller, Emmanuel Müller, and Klemens Böhm. 2015. Estimating mutual information on data streams. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management (SSDBM'15).*

[15] Shiraj Khan, Sharba Bandyopadhyay, Auroop R. Ganguly, Sunil Saigal, David J. Erickson, Vladimir Protopopescu, and George Ostrouchov. 2007. Relative performance of mutual information estimation methods for quantifying the dependence among short and noisy data. *Phys. Rev. E* 76, 2 (2007), 026209.

[16] Justin B Kinney and Gurinder S Atwal. 2014. Equitability, mutual information, and the maximal information coefficient. *Proceedings of the National Academy of Sciences* 111, 9 (2014), 3354–3359.

[17] LF Kozachenko and Nikolai N Leonenko. 1987. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii* 23, 2 (1987), 9–16.

[18] Philipp Kranen and Thomas Seidl. 2009. Harnessing the strengths of anytime algorithms for constant data streams. *Data Mining and Knowledge Discovery* 19, 2 (2009), 245–260.

[19] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. 2004. Estimating mutual information. *Phys. Rev. E* 69, 6 (2004), 066138.

[20] Sebastian Kruse and Felix Naumann. 2018. Efficient discovery of approximate dependencies. *Proceedings of the VLDB Endowment* 11, 7 (2018), 759–772.

[21] Don O Loftsgaarden and Charles P Quesenberry. 1965. A nonparametric estimate of a multivariate density function. *The Annals of Mathematical Statistics* (1965), 1049–1051.

[22] Son T Mai, Xiao He, Jing Feng, Claudia Plant, and Christian Böhm. 2015. Anytime density-based clustering of complex data. *Knowledge and Information Systems* 45, 2 (2015), 319–355.

[23] Angeliki Papana and Dimitris Kugiumtzis. 2009. Evaluation of mutual information estimators for time series. *International Journal of Bifurcation and Chaos* 19, 12 (2009), 4197–4215.

[24] Hanchuan Peng, Fuhui Long, and Chris Ding. 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 8 (2005), 1226–1238.

[25] Josien PW Pluim, JB Antoine Maintz, and Max A Viergever. 2003. Mutual-information-based registration of medical images: a survey. *IEEE Transactions on Medical Imaging* 22, 8 (2003), 986–1004.

[26] Peng Qiu, Andrew J Gentles, and Sylvia K Plevritis. 2009. Fast calculation of pairwise mutual information for gene regulatory network reconstruction. *Computer methods and programs in biomedicine* 94, 2 (2009), 177–180.

[27] John Rice. 2006. *Mathematical statistics and data analysis.* Nelson Education, Chapter Survey Sampling, 199–220.

[28] Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal* 27 (1948), 379–423, 623–656.

[29] Zbyněk Šidák. 1967. Rectangular confidence regions for the means of multivariate normal distributions. *J. Amer. Statist. Assoc.* 62, 318 (1967), 626–633.

[30] Jaroslaw Szlichta, Parke Godfrey, and Jarek Gryz. 2012. Fundamentals of order dependencies. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1220–1231.

[31] Martin Vejmelka and Kateřina Hlaváčková-Schindler. 2007. Mutual information estimation in higher dimensions: A speed-up of a k-nearest neighbor based estimator. In *International Conference on Adaptive and Natural Computing Algorithms (ICANNGA'07).* 790–797.

[32] Michael Vollmer, Ignaz Rutter, and Klemens Böhm. 2018. On Complexity and Efficiency of Mutual Information Estimation on Static and Dynamic Data. In *International Conference on Extending Database Technology (EDBT '18).* 49–60.

[33] Janett Walters-Williams and Yan Li. 2009. Estimation of mutual information: A survey. In *International Conference on Rough Sets and Knowledge Technology (RSKT'08).* 389–396.

[34] BP Welford. 1962. Note on a method for calculating corrected sums of squares and products. *Technometrics* 4, 3 (1962), 419–420.

[35] Ying Yang, Geoff Webb, Kevin Korb, and Kai Ming Ting. 2007. Classifying under computational resource constraints: anytime classification using probabilistic estimators. *Machine Learning* 69, 1 (2007), 35–53.

[36] Shlomo Zilberstein. 1996. Using anytime algorithms in intelligent systems. *AI magazine* 17, 3 (1996), 73–83.