Time-Efficient Analysis of Complex Dependencies

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Michael Vollmer

aus Freudenstadt

Tag der mündlichen Prüfung:	04.02.2020
Erster Gutachter:	Prof. Dr.-Ing. Klemens Böhm
Zweiter Gutachter:	Prof. Dr. rer. nat. Ignaz Rutter

Acknowledgements

This dissertation represents multiple years' worth of research during which I was supported by many people in various ways. Here I would like to explicitly express my appreciation.

First and foremost I would like to thank my supervisor *Klemens Böhm* for his tireless support both regarding the project as a whole and individual publications. His thorough evaluation of manuscript drafts, pragmatism and the freedom he gave me to pursue my own research certainly improved my writing style and project management skills.

Further, I would like to thank *Ignaz Rutter* for encouraging me to do this project in the first place and accepting the task of reviewing this dissertation. His enthusiasm convinced me how strongly connected different research fields in computer science can be and that application driven tasks often contain formal and theoretical problems in their core.

I am also very grateful to the German Research Foundation (DFG) and their research training group "Energy Status Data" for their financial and structural support of this doctoral project.

During my research, I got the opportunity to work with a lot of people to whom I am thankful. The diverse perspectives and insights regarding both research in general and my research in particular helped me immensely. For this, I would like to thank my co-workers at the chair for systems of information management, my colleagues from the research training group and my collaborators at the University of Waterloo. A special thanks I would like to give to *Jens Willkomm*, *Holger Trittenbach*, *Georg Steinbuss* and *Adrian Englhardt* for the great time that we shared both at work and outside. Your open ears and sound advice for both research problems and the challenges of modern life in general.

Last but not least, I want to thank my friends and family. I like to thank my mother and my siblings for always providing a quiet refuge and encouragement to pursue this degree. I would also like to thank my friends and former fellow students that accompanied my life in Karlsruhe for nearly one decade. From our initial quest to become "mighty students" to this day – and hopefully much longer – they offered countless joyous occasions and outside perspectives whenever needed.

Abstract

The volumes of data that are collected and analyzed in all aspects of life rise steadily and offer numerous challenges and opportunities. When analyzing large volumes of data, two main considerations are the capability to detect relationships within the data and the time efficiency necessary when processing the data. Large data sets in particular often contain dependencies that are less obvious than linear or proportional relationships. Such complex dependencies could be quantified and analyzed with methods from the field of information theory. In this dissertation, we study time-efficient methods to detect, quantify and illustrate complex dependencies. More specifically, we consider three core problems that have applications in the analysis of energy data. The general form of these problems as well as our solutions and results are as follows:

One use case for dependency analysis is real-time monitoring in order to detect abrupt changes, which could indicate faults. In other words, the task is maintaining an up-to-date score of the intensity of a dependency for the most recent data. We quantify this intensity with mutual information, a concept from information theory. The state-of-the-art approach is computing the score from scratch whenever new data becomes available using well-known estimation techniques. On the one hand, we prove that the required time to compute these estimation results scales at least super-linearly with the data size. In detail, the lower bound for the time complexity of these estimates is $\Omega(n \log n)$. This result also implies the lower bound $\Omega(\log)$ for the time complexity to update existing scores for individual data changes. On the other hand, we present two fully dynamic data structures that maintain the aforementioned mutual information scores for a set of points, i.e., the data structures support insertion and deletion of points. We show that our data structures offer better asymptotic runtime than the state-of-the-art, i.e., linear time complexity, when updating a score. For one of the estimation techniques we achieve near-optimal time complexity, i.e., $O(\log n \log \log n)$. Practical evaluation with synthetic and real data validate our formal results and show that our methods are faster by orders of magnitudes.

Another scenario are analysis tasks with dynamic conditions. For instance, one such case are threshold queries, i.e., the question whether the intensity of a dependency is above or below a threshold. This threshold might for instance indicate the difference between normal operation and a suspicious state. Such queries have dynamic requirements on the result quality, because errors in the score are acceptable if they do not change whether the score is above or below the threshold. Then, the required precision, and thus computational effort, depends on the proximity between the score and the threshold. Another example for dynamic conditions are data streams with irregular arrival times, e.g., event-based sensors. Real-time monitoring in this case needs so-called "anytime" algorithms, which may be interrupted at any point in time and still yield results. Even so, result quality of anytime algorithms improves with more time before interruption. Our solution to such dynamic conditions is an iterative estimator for mutual information. Our approach yields a rough estimate as fast as possible and improves the results in time steps with fine granularity, called iterations. An important feature of our approach is that early estimates are accompanied by statistical guarantees towards the still unknown, high precision results. As illustration, this information can determine the probability that the current estimate yields the wrong answer to a threshold-query in comparison to the exact result. Our experiments with real-world data show that our iterative approach answers threshold-queries faster than existing methods, even with a tolerated error rate of 0%.

Finally, we consider not only the intensity of complex dependencies but also explanations for such dependencies that are interpretable by humans. Here, the goal is detection and illustration of relationships between a designated outcome attribute, such as machine labels or product quality, and all other available attributes. In contrast to classification, these explanations should summarize the relationships within given data as detailed as possible instead of targeting more general decision rules. Naturally, human interpretability also requires conciseness. Existing solutions for this task are restricted to categorical attributes and binary outcomes. However, sensor and machine data are often numerical and have multiple different outcomes. Our contribution to this task is an approach that overcomes the those restrictions, i.e., we also summarize numerical data with multiple outcomes. The main challenge of this task is the vast domain size of numerical data, which results in an even higher number of potential intervals and combinations of intervals for these attributes. We experimentally show with real-world data that our approach offers better time efficiency and summary quality in comparison to approaches that we adapted from related tasks.

In summary, this dissertation provides new methods for complex dependency analysis. We present algorithms that provide important tools in light of increasing data volumes and complexity. Our methods can be used to analyze the dependencies directly and serve as preprocessing step in bigger analytic frameworks to find relevant or redundant attributes.

Deutsche Zusammenfassung

Die steigenden Mengen an Daten, die für Analysen verwendet werden, sind Quelle zahlreicher Herausforderungen und Möglichkeiten. Zwei Kernaspekte der Analyse solcher Datenmengen sind die Zeiteffizienz und die Fähigkeit, Zusammenhänge zu erkennen. Gerade in größeren Datenbeständen ergeben sich hierbei oft Abhängigkeiten, die über Proportionalitäten und Linearitäten hinausgehen. Solche komplexen Abhängigkeiten könnten mithilfe der Shannonschen Informationstheorie quantifiziert und damit analysiert werden. Entsprechend beschäftige ich mich in dieser Dissertation mit zeiteffizienten Verfahren um komplexe Abhängigkeiten zu erfassen und darzustellen. Im Speziellen betrachte ich hierbei drei grundlegende Problemstellungen, die häufig in der Analyse von Energie-Zeitreihen wiederzufinden sind. Die entsprechenden allgemeinen Problemstellungen sowie meine Lösungen und Ergebnisse sind wie folgt:

Ein häufiges Szenario ist das Überwachen von Abhängigkeiten in Echtzeit, um abrupte Änderungen und damit potenzielle Störungen zu erkennen. In allgemeiner Form besteht das Problem darin, die aktuelle Intensität der Abhängigkeit, hier quantifiziert durch Transinformation (engl. „Mutual Information“), bei Änderungen der Daten zu bestimmen. Der bisherige Stand der Technik für dieses Problem ist eine komplette Neuberechnung mit gängigen Schätzmethoden. Einerseits beweise ich eine superlineare untere Schranke für die Zeitkomplexität dieser Schätzmethoden, die ebenfalls eine untere Schranke für Aktualisierungen des Schätzwerts bei Datenänderung impliziert. Andererseits entwickle ich eine Datenstruktur für Punkte, die die Transinformation der enthaltenen Punkte bestimmt und voll dynamisch ist, d.h. Einfügen und Entfernen von Punkten unterstützt. Diese Datenstruktur hat bessere asymptotische Laufzeit als bestehende Methoden und erreicht für eines der Schätzverfahren nahezu optimale Zeitkomplexität für Aktualisierungen. Experimente mit realen Daten belegen die formalen Ergebnisse und zeigen Beschleunigungen um Größenordnungen.

Ein anderes Szenario stellen Analyseaufgaben mit dynamischen Anforderungen dar. Hierbei sind beispielsweise Schwellenwert-Anfragen gemeint, d.h. die Frage, ob die Intensität der Abhängigkeit über oder unter einem Schwellenwert liegt. Die dynamische Natur dieser Anfrage liegt darin, dass eine gewisse Unsicherheit bezüglich

der Schätzung erlaubt ist, solange es mit statistischer Sicherheit keinen Unterschied bezüglich des Schwellenwerts gibt. Entsprechend ist die nötige Präzision und damit der Rechenaufwand der Schätzung abhängig davon, wie nahe der Schwellenwert ist. Ein anderes Beispiel dynamischer Anforderungen ist die Echtzeit-Überwachung von unregelmäßigen Datenströmen, beispielsweise von ereignisgetriebenen Sensoren. Dieses Szenario benötigt sogenannte "Anytime"-Algorithmen, die jederzeit unterbrochen werden können und dennoch ein Ergebnis liefern; dabei führt mehr Zeit zu höherer Ergebnisqualität. Meine Lösung für dynamische Anforderungen ist ein iterativer Schätzer für Transinformation. Dieser liefert nach kürzester Zeit eine Schätzung, die in feingranularen Zeitschritten, genannt Iterationen, verbessert werden kann. Die Besonderheit meiner Lösung ist dabei, dass frühzeitige Schätzungen zusätzliche zu dem unsicheren Schätzwert auch statistische Garantien bezüglich dem Endergebnis bieten. Hiermit ist es beispielsweise möglich, die Fehlerwahrscheinlichkeit einer unsicheren Schätzung bezüglich einer Schwellenwert-Anfrage zu bestimmen. Meine Experimente mit realen Daten zeigen, dass mein iteratives Verfahren Schwellenwert-Anfragen, selbst mit 0% Fehlerwahrscheinlichkeit, grundsätzlich schneller beantworten kann als bestehende Verfahren.

Schließlich habe ich mich neben der Intensität von komplexen Abhängigkeiten auch mit verständlichen Erklärungen solcher komplexen Abhängigkeiten beschäftigt. Hier ist das Ziel, die Zusammenhänge zwischen einem Klassen- oder Ergebnisattribut, beispielsweise Maschinentyp oder Produktqualität, und den weiteren gemessenen Attributen zu erkennen und darzustellen. Im Gegensatz zur Klassifikation geht es dabei insbesondere um eine möglichst spezifische Zusammenfassung des Datenbestands anstelle von allgemeingültigen Entscheidungsregeln. Bisherige Lösungen für dieses Problem waren der signifikanten Beschränkung ausgesetzt, lediglich kategorische Attribute und binäre Ergebnisse zu unterstützen. Sensor- und Maschinendaten sind jedoch oft numerisch und haben mehrere unterschiedliche Ergebnisklassen. Mein Beitrag zu diesem Problem ist ein Verfahren, das Erklärungen auch für numerische Daten und mehrwertige Ergebnisse erstellt. Die großen Domänen von numerischen Daten sowie die Kombinationsmöglichkeiten innerhalb eines Attributs für Intervalle und zwischen unterschiedlichen Attributen sind dabei die zu bewältigenden Herausforderungen. Mithilfe von realen Daten zeige ich experimentell, dass mein Verfahren bessere Zeiteffizienz und Erklärungsqualität hat als andere Verfahren für verwandten Probleme.

Zusammenfassend bietet diese Dissertation neue Verfahren als Werkzeug zur Analyse von Abhängigkeiten angesichts steigender Datenmengen und Komplexität. Die entwickelten Verfahren können dabei sowohl direkt zur Auswertung eingesetzt werden als auch als Vorverarbeitung für weiterführende Verarbeitung und Analysen. Beispielsweise wird die Intensität der Abhängigkeit genutzt um möglichst relevante oder unabhängige Attribute auszuwählen.

Contents

Chapter 1

Introduction

Collecting and processing data has become a significant aspect of personal lives and industries alike. The quick access of information through internet search engines is not only based on the vast information from web pages but also individual person-based optimization [FG08, SG05]. Non-intrusively recording human behavior through digital systems is also used to improve route planning [L^+08], product design [JLJL16] and advertisement [CH12]. Industrial machines also include more and more sensors that are continuously monitored. This data is not only useful for the machine producers to improve the design, but also operators of the machines. Detailed machine information may save costs by detecting machines in bad condition before they break and induce costly interruptions to the production [Has11].

Extracting insights from the available data is a broad task that is tackled by countless researchers using a vast variety of techniques [MR05]. Generally speaking, this knowledge discovery usually formalizes some kind of regularity or pattern within the data such as probability distributions for individual quantities or varying similarities between objects. While the knowledge found is usually imperfect, e.g., only true to certain degree or for a limited portion of the data, it remains useful for various tasks such as predicting future data. One form of this knowledge are dependencies between different measured quantities, which is particularly relevant in light of the increasing number of monitored and recorded devices. These dependencies are not primarily focused on the values measured, but rather on the questions whether the values of different quantities appear to affect each other. The motivation to find dependencies is that they show which measured quantities are linked to each other and thus further the understanding of the underlying system producing the data.

Example 1.1. *Consider the stock market, which provides an estimated value for companies through frequent trades of small shares. Grouping companies by region and combining their values then enables higher level perspectives, where the combined value serves as* index *for the group. Figure 1.1a graphs the change of value*

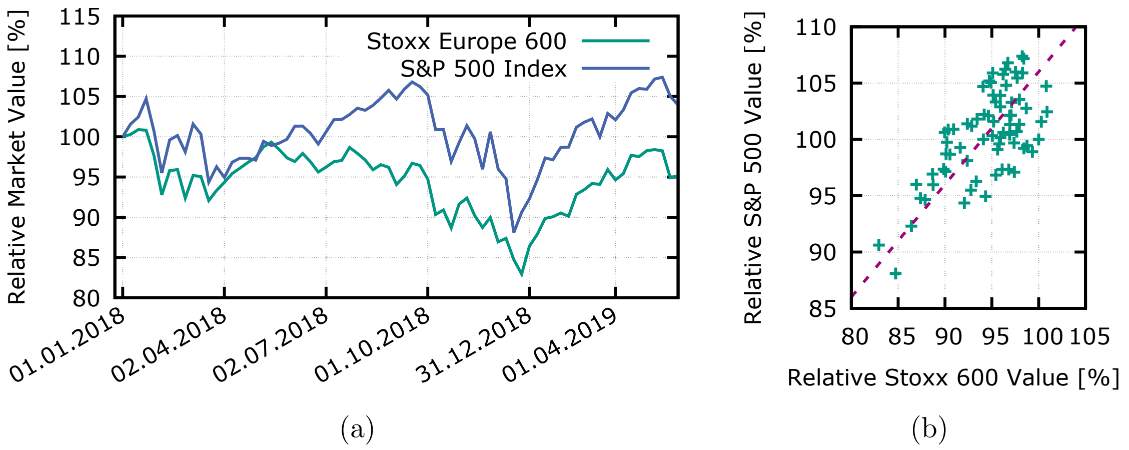


Figure 1.1: Relative stock prices of the American S&P and the European Stoxx index during 2018 and the beginning of 2019. (a) Development of both stock indices over time. (b) Scatter plot of the two stock indices with each point representing one week and a dashed line to illustrate the linear relationship.

for American companies, the "S&P 500 Index", and European companies, the "Stoxx Europe 600", for 2018 and early 2019. While there are some discrepancies, both curves exhibit similar behavior like a drop at the end of 2018 and subsequent recovery. An alternative illustration of this data that neglects the exact sequence is shown in Figure 1.1b, where each dot represents the same week for both stock index values. While not perfectly aligned, there is also a clear trend indicated: Those weeks with comparatively high S&P Index values tend to also have Stoxx Europe values and vice versa. This means that the data shows some dependency between the two stock indices. After all, this observation is not overly surprising as the current economy is strongly linked in the sense that many American companies have European companies as suppliers and vice versa. At this point it is important to note, that the dependency indicates that these indices are somewhat linked, even without any financial knowledge or explanation for the phenomenon. Similarly, if there is a significant change in this dependency, e.g., the discrepancy between curves increases, this hints at a change in the system producing the data such as stricter trading policies that result in economies that are more autonomous.

The dependency shown in this example is the well-known case of *linear correlation*, also called just *correlation*. In this case, there is a linear relationship between the two attributes in the sense that the data can be roughly approximated with a non-axis-aligned line. Depending on the sign of slope of this line or "trend", the data is correlated positively, cf. Figure 1.1b, or negatively, also known as anti-correlated. This denotes whether relatively higher values in one attribute occur with relatively

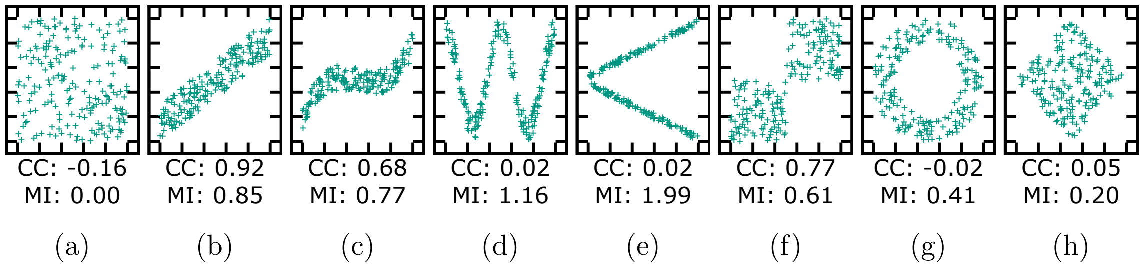


Figure 1.2: Various forms and strengths dependencies with their Pearson correlation coefficient (CC) and mutual information (MI) score: (a) Independent, (b) Linear, (c) Cubic, (d) Sinusoidal, (e) Split-beam, (f) Checkers, (g) Donut, (h) Diamond.

higher or relatively lower values of the other attribute. If the line is axis-aligned or the noise of individual points overshadows the line, the data is uncorrelated. In consequence, the more clearly data fits such a line, the stronger the correlation.

In this thesis, we consider *dependencies* as any data that is not statistically independent. Thus, data without dependencies is generally also uncorrelated, the inverse is not true. Figure 1.2 shows eight scatter plots of data with different relationships between two attributes. Note that seven plots, i.e., all except (a), show some dependency pattern while only three plots are correlated as indicated by a high Pearson correlation coefficient [Ric06] (CC). We call dependencies *complex* if they are not linear relationships , i.e., they are not identified by Pearson correlation coefficient. While this is a straightforward definition, it is only a binary label that does not distinguish between clearer dependencies such as Figure 1.2d and data with more randomness such as Figure 1.2h. To this end, we use the concept of mutual information from information theory to quantify the intensity of (complex) dependencies, as it has the desired property of scoring zero if and only if the data is independent. Returning to Figure 1.2, all plots except (a), which shows independent data, has a mutual information (MI) score above zero.

Another form of complex dependencies arises by examining multiple attributes. Relationships between more than two attributes may or may not be visible when examining smaller attribute sets. For instance, solar power generation depends on both the season and weather, but even without the season, there is a relationship between weather and generated solar power. Alternatively, consider an “either-or” scenario, which corresponds to the logical XOR operator, represented by a lamp connected to two light switches as illustrated in Figure 1.3. If either switch is flipped, the light is on, but if both or neither switches are flipped, the light is off. As a result, there is no dependency when observing only one switch and the light, as one can observe “light on” and “light off” for both position of that switch. However, there is a ternary relationship since knowledge of both switches determines the light status

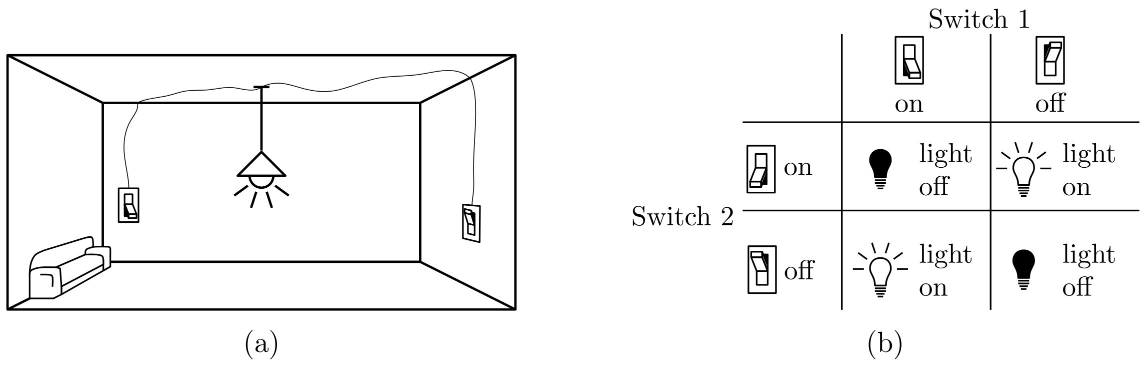


Figure 1.3: (a) An exemplary room with two independent switches connected to the same light. (b) Table showing the effect of switch positions on the light status.

and similarly the position of one switch and the light status suffice to determine the position of the other switch. Unfortunately, there is no standardized measure to detect or quantify such dependencies. While mutual information is a very good measure for (complex) relationships between two attributes, it is not defined for more attributes and so far, there is no consensus on a proper generalization. This lack of a formal measure is one indicator of the remaining challenge to define, describe and interpret dependencies between multiple attributes.

So far, we discussed the notion of complex dependencies on a conceptual level. The environment for practical applications, however, is less ideal. In general, challenges arise due to measured real-world data instead of well-defined probability distributions. As a result, there is a limited number of measurements, there are errors in the data and there is no ground truth. As illustration, 1.2b offers a much clearer relationship than Figure 1.1b. This means that some interpretation and estimation is necessary when working with complex dependencies. Mutual information is no exception to this case: It is properly defined on probability distributions and thus requires estimation when working with real-world data, which usually has an unknown distribution.

Returning to the opening scenario, the implications of rising data volumes are interesting. On the one hand, more available data allows for analysis of weaker and more intricate dependencies. That is, the more measurements are available the better one can distinct between dependencies and measurement error or coincides. Thus, complex dependency analysis for real-world applications becomes more relevant as it is easier to detect such dependencies. On the other hand, algorithms that analyze complex dependencies are also non-trivial themselves and often computationally expensive, e.g., their runtime scales super-linearly with the data size. This is prohibitive for time-restrictive tasks such as real-time monitoring or data exploration, where one considers many potential dependencies. While it would be possible to improve the computational burden by using only some of the available data, one might miss the relevant dependencies due to decreasing result quality. In consequence, it is necessary to improve the time-efficiency in a way that does not impact the result quality for the corresponding analysis task.

In summary, methods to analyze complex dependencies require large data volumes for reliable results on real data, yet the same methods are challenged by large data volumes due to their scalability. Additionally, complex dependencies are harder to describe and interpret, particularly when multiple attributes are involved. Reducing both the computational and mental burden necessary to consider complex dependencies when analyzing data might enable analysts to extract more knowledge from the data available. This might enable plant managers to incorporate complex dependencies as part of their monitoring system or help researchers to gain more insights from an experiment.

1.1 Contributions

In this dissertation, we introduce algorithms to facilitate more widespread analysis of complex dependencies. Our contributions include practical algorithms for dependency analysis as well as formal results and algorithms for more general geometric problems. In particular, our contributions are as follows:

Complexity of Nearest-Neighbor Based Mutual Information Estimation. We prove a lower bound for the computational complexity of the popular nearest-neighbor based mutual information estimation [Eva08, KSG04]. This bound proves the asymptotic optimality of current computation algorithms and enables the derivation of lower bounds for variants of the estimation tasks. In consequence, this contribution provides a point of reference for optimal efficiency.

Fully Dynamic Data Structure and Stream Processing. We present two data structures that contain a set of data points and the corresponding mutual information estimate. These data structures are fully dynamic by supporting insertion and deletion of points with time-efficient computation of the updated mutual information estimate. In detail, we achieve sub-linear time complexity for these updates, which is only a logarithmic factor slower than the optimal time complexity. To achieve this result, we formally proof the limited effect of appearing and disappearing individual points on geometric constellations like the nearest neighbors of other points. For real-time monitoring of dependencies within data streams, our data structures reduce the run times by orders of magnitudes. However, the generic nature of the data structures allows for other use cases as well.

Iterative Estimation. We present an iterative method to estimate mutual information that provides a rough estimate almost immediately and improves this estimate with additional computation time in small increments, called iterations. Additionally, estimates are accompanied by statistical guarantees towards the accurate, nearest-neighbor based estimate. For instance, for a rough estimate one could determine the probability that the difference to the accurate estimate is at most 0.1 without computing the accurate estimate itself. Our iterative method enables mutual information estimation as anytime algorithm where the available computation time may be unknown beforehand. Additionally, our experiments show that our method answers some queries, e.g., whether the mutual information score is above or below a threshold, faster even when no errors are tolerated. To achieve these results, we establish the statistical background enabling the guarantees and present a new lightweight algorithm for nearest-neighbor search. This nearest-neighbor search is necessary since it is unknown beforehand how much time is available or necessary to answer individual queries. As a result, it is necessary to minimize the overhead and front-loaded computations in case few iterations are performed without falling behind if many iterations are performed.

Explaining Relationships with Multiple Numerical Attributes. We build a framework to produce human-interpretable summaries of dependencies based on existing formalism that is limited to discrete attributes. In addition to the semantically meaningful inclusion of ordinal and numerical attributes into the formal framework, we also develop an algorithm to produce such summaries. Notably, numerical attributes are not discretized as preprocessing, but rather the full domain of values is used to produce the most informative summaries. As part of this process we use a novel and lightweight data structure that is tailored to solve special cases of the geometric problem “range sum query”.

1.2 Thesis Outline

The remaining chapters of this dissertation are structured as follows. In the next chapter, Chapter 2, we briefly recapitulate relevant terms and techniques from both information theory. Chapter 3 reviews work that is related to this dissertation. Our contributions are then contained in the subsequent two chapters: In Chapter 4 we present our results on the estimation of mutual information. Then, Chapter 5 shows our work on explanation and summarization of relationships for human interpreters. Finally, Chapter 6 summarizes our findings and illustrates the further potential both for immediate applications and future research.

Chapter 2

Fundamentals of Information Theory

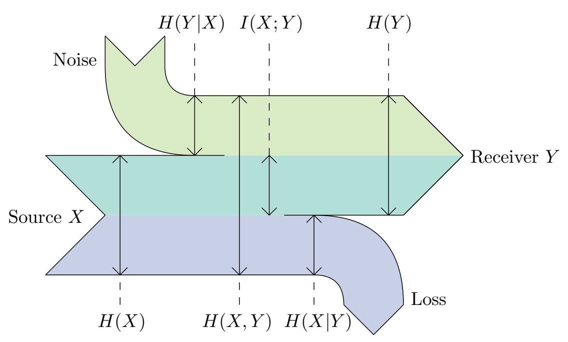
Here, we recapitulate the aspects and techniques of information theory relevant to this dissertation. For completeness, we start with a brief historical motivation of Shannon entropy as measure of information. Then, we introduce the concepts of mutual information as a measure of dependence and maximum entropy models as accurate representation of unknown distributions based on limited knowledge. While these concepts have theoretic origins, we also recapitulate methods for good approximations for practical use with real data.

2.1 Shannon Entropy

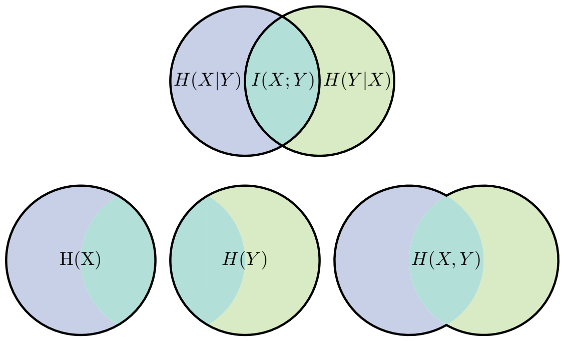
Claude Shannon introduced the theoretical study and formalization of information in the context of telecommunication and signal processing[Sha48]. To study the limitations of information transmission, a formalization of information is necessary. Shannon defined the information of a message to relate to its probability of occurring, as an expected message offers little insight. For instance, a daily message whether the earth was hit by an asteroid today will almost always be negative and thus offer no insights as this coincides with our expectations. Conversely, the very unlikely positive message has a high information content as it strongly contrasts our previous state of knowledge. Formally, the information content of a message x from a source X by Shannon is

$$I(x) = -\log(p_X(x)), \tag{2.1}$$

where $p_X(x)$ is the probability that X sends x. The base of the logarithm determines the unit for the information, i.e., a binary logarithm measures information in bits and a natural logarithm measures the information in nats. For instance, a message with probability 0.5, like the probability of a coin toss resulting in "heads", has an information content of 0.693 nats or 1 bit, as it distinguishes exactly two states. The expected information of a message from X, i.e., the weighted average, is called



(a)



(b)

Figure 2.1: Illustration of the relationship between various measures of information content in relation to the transmission-scenario (a) and as Venn-diagram (b).

the entropy $H(X)$ of X. With $\mathbb{X}$ being the set of all messages X can send, this is formally

$$H(X) = \sum_{x\in\mathbb{X}} -\log(p_X(x)) \cdot p_X(x) = -\sum_{x\in\mathbb{X}} p_X(x)\log(p_X(x)). \tag{2.2}$$

Next, consider a receiver Y that receives a message y for each message x sent by X. In addition to the entropy $H(Y)$ based on p_Y which represents the average information received by Y, this also gives way to further interpretations when considering both X and Y. Figure 2.1 illustrates the relationship of the following definitions. Using conditional distributions, i.e., $p(y|x)$ being the probability that Y received y when X sent x, enables additional quantifications. Specifically, the so-called conditional entropies

$$H(X|Y) = \sum_{y\in\mathbb{Y}} p(y) \sum_{x\in\mathbb{X}} -\log(p(x|y)) \cdot p(x|y) \quad \text{and} \tag{2.3}$$

$$H(Y|X) = \sum_{x\in\mathbb{X}} p(x) \sum_{y\in\mathbb{Y}} -\log(p(y|x)) \cdot p(y|x) \tag{2.4}$$

represent the loss of information and noise, respectively. The loss of information $H(X|Y)$ models the information sent by X but not received by Y. The noise $H(Y|X)$ quantifies the information received by Y that was not sent by X. Lastly, the joint distribution $p(x, y)$ of messages both sent and received gives way to the total entropy $H(X, Y)$ and mutual information $I(X; Y)$, which quantifies the information transmitted from X to Y. Formally, these are

$$H(X,Y) = -\sum_{y\in\mathbb{Y},x\in\mathbb{X}} \log(p(x,y)) \cdot p(x,y) \quad \text{and} \tag{2.5}$$

$$I(Y;X) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{2.6}$$

$$= H(X) + H(Y) - H(X,Y) \tag{2.7}$$

$$= \sum_{y\in\mathbb{Y},x\in\mathbb{X}} p(x,y) \log\left(\frac{p(x,y)}{p(x)\cdot p(y)}\right). \tag{2.8}$$

Translating these definitions to our use cases in data analysis, X and Y are (sets of) attributes and $\mathbb{X}$ and $\mathbb{Y}$ are the possible (combinations of) values. The entries in the data set then provide an empirical estimate for the (conditional) probabilities of the value combinations. For instance, the probability $p(x|y)$ is the fraction of of entries with value(s) x for attribute(s) X among all entries with value(s) y for attribute(s) Y. That is, each entry is considered as a message and the data set serves as estimate of the distribution. While this transfer loses the semantic relationship between a source and receiver, the general concept of information sharing remains. That is, independent attributes share no information, which can be easily seen by substituting $p(x,y)$ with $p(x)\cdot p(y)$ in Equation 2.8. Conversely, the more dependent X and Y are, the higher is their shared information. For instance, if the value(s) of Y can be determined from the value(s) of x, the conditional entropy $H(Y|X)$ is zero and the shared information $I(X;Y)$ is equal to $H(Y)$ by Equation 2.6.

However, contrary to messages from a specified set, attribute values may also be numerical from an infinite domain. Particularly attributes with real values may have unique values for each entry and follow a continuous distribution instead of a discrete one. As a result the probability $p(x)$ of messages is not a set that sums up to one, but a probability density function $f(x)$ for all messages that integrates up to one. Overall, there are equivalent definitions for the different concepts of information theory for this case, which substitute the sum of probabilities with integrals over probability densities. The result is called differential information theory and has the following definitions [CT06].

$$H(X) = -\int_x f(x)\log(f(x))\, dx \tag{2.9}$$

$$H(X|Y) = -\int_y f(y) \int_x f(x|y)\log(f(x|y))\, dy \tag{2.10}$$

$$H(X,Y) = -\int_x\int_y f(x,y)\log(f(x,y))\, dy\, dx \tag{2.11}$$

$$I(X;Y) = \int_x\int_y f(x,y)\log\left(\frac{f(x,y)}{f(x)\cdot f(y)}\right)\, dy\, dx \tag{2.12}$$

Differential information theory is a powerful tool for dependency analysis, as mutual information quantifies any kind of dependency between two attributes. However, some challenges arise with practical scenarios and real data. While discrete probabilities are straightforward to estimate from a sample, estimating the probability density of a continuous distribution of numerical data is an ongoing field of study [DL01, Gra18].

2.2 Entropy Estimation

Estimating probability densities is an important task [Sil86, WJ95] that gained attention in statistics before the concept of entropy and information theory [WW39]. Particularly important are methods that make no assumptions on the characteristics of the distribution, which are called *non-parametric* methods. Despite this name, these methods may still use parameters, however they are not dependent on specific types of distributions.

Usually, non-parametric estimation techniques use the values and proximity of the entries to each other as foundation to estimate the density for all possible values. The general idea is to find the probability density function that is likely to yield a sample identical to the data available without fitting the function too specifically. Working with continuous distributions, let $X \subseteq \mathbb{R}^d$ be the combination of d real valued attributes in the available data. For a combination of values $x \in \mathbb{R}^d$, a common approach to estimate the density $f(x)$ is using a fixed area around x to count entries inside and potentially weight them by distance. With the exact area definition and weighting scheme being variable, this general approach is called kernel density estimation [Sil86, WJ95]. In entropy estimation, however, an alternative approach gained much popularity. First introduced by Loftsgaarden and Quesenberry [LQ65], one determines a number k and evaluates how much space around x is necessary to contain k points of the sample. That is, the distance $\epsilon_X^k(x)$ of x to the kth nearest neighbor in X determines the estimated density. Formally, it is

$$\hat{f}(x) = \frac{k}{|X|} \cdot \frac{1}{V(d, dist) \cdot (\epsilon_X^k(x))^d}, \tag{2.13}$$

where $V(d, dist)$ is the volume of an d-dimensional unit ball using distance function $dist$. In the same work, it is also proven that Equation 2.13 is a consistent estimator, i.e., the estimated value converges towards the true value with increasing sample size. While they used the euclidean distance as default, it is explicitly stressed that any distance function is equally usable. Naturally, this estimate could be used with Equation 2.9 and approximation techniques for the integration to estimate $H(X)$. However, a far simpler approach is to use the fact that $H(X)$ is the expected value of $-log(f(x))$, which means the average over all entries $x \in X$ can also serve as estimator. Formally, this estimator is

$$\hat{H}(X) = -\frac{1}{|X|}\sum_{x\in X}\log\left(\frac{k}{|X|}\cdot\frac{1}{V(d,dist)\cdot(\epsilon_X^k(s))^d}\right) \tag{2.14}$$

$$= -\frac{1}{|X|}\sum_{x\in X}\big(\log(k)-\log(|X|)-log(V(d,dist))-d\cdot log(\epsilon_X^k(x))\big) \tag{2.15}$$

$$= -\log(k)+\log(|X|)+\log(V(d,dist))+\frac{d}{|X|}\sum_{x\in X}log(\epsilon_X^k(x)). \tag{2.16}$$

Originally, Loftsgaarden and Quesenberry suggested [LQ65] that the choice of k scales with $|X|$ for the density estimation Equation 2.13. Through the works of Kozachenko and Leonenko [KL87], as well as Singh et al. [SMH$^+$03] a correction was introduced that yielded the estimator

$$\widehat{H_{KL}}(X) = -\psi(k)+\psi(|X|)+\log(V(d,dist))+\frac{d}{|X|}\sum_{x\in X}\log(\epsilon_X^k(x)), \tag{2.17}$$

where ψ is the digamma function, i.e., $\psi(x) = \left(\sum_{m=1}^{x-1}\frac{1}{m}\right)-\gamma$ for $x \geq 1$ and $\gamma \approx 0.577$ being the Euler-Mascheroni constant. This estimator is consistent for any fixed k and mostly attributed to Kozachenko and Leonenko in the literature [Eva08, GOV18, KMB15, KSG04]. Furthermore, we follow the literature [Eva08, KSG04] and use the *maximum norm* $\|(x_1,\dots,x_d),(x'_1,\dots,x'_d)\|_\infty = \max_i(|x_i - x'_i|)$, which is also called called L_∞ metric or Chebyshev-distance. This results in $V(d,dist) = 2^d$.

Next, let $P = \{p_1 = (x_1,y_1),\dots,(x_n,y_n)\} \subseteq \mathbb{R}^2$ be the combinations of two attributes X and Y available in the data. Figure 2.2a illustrates the following terms with an exemplary set $P = \{(1,5.5),(8,1),(5,4),(3.5,7),(2,2),(8,1)\}$. Next, X and Y are the sets of individual values in the data for the attribute, i.e., $X = \{x_1,\dots,x_n\}$ and $Y = \{y_1,\dots,y_n\}$. Analogue to previous notation, $\epsilon_P^k(p_i)$, $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ is the distance of p_i, x_i and y_i to its k-th nearest neighbor in P, X and Y, respectively. Substituting the entropy values in Equation 2.7 with $\widehat{H_{KL}}$ yields the estimator

$$\begin{aligned}\widehat{I_{3KL}}(P) =& -\psi(k)+\psi(n)+\log(2)+\frac{1}{n}\sum_{i=1}^{n}\log(\epsilon_X^k(x_i))\\ & -\psi(k)+\psi(n)+\log(2)+\frac{1}{n}\sum_{i=1}^{n}\log(\epsilon_Y^k(y_i))\\ & +\psi(k)-\psi(n)-\log(2^2)-\frac{2}{n}\sum_{i=1}^{n}\log(\epsilon_P^k(p_i)) \qquad (2.18)\\ =&\psi(n)-\psi(k)+\frac{1}{n}\sum_{i=1}^{n}\log\left(\frac{\epsilon_X^k(x_i)\cdot\epsilon_Y^k(y_i)}{\left(\epsilon_P^k(p_i)\right)^2}\right), \qquad (2.19)\end{aligned}$$

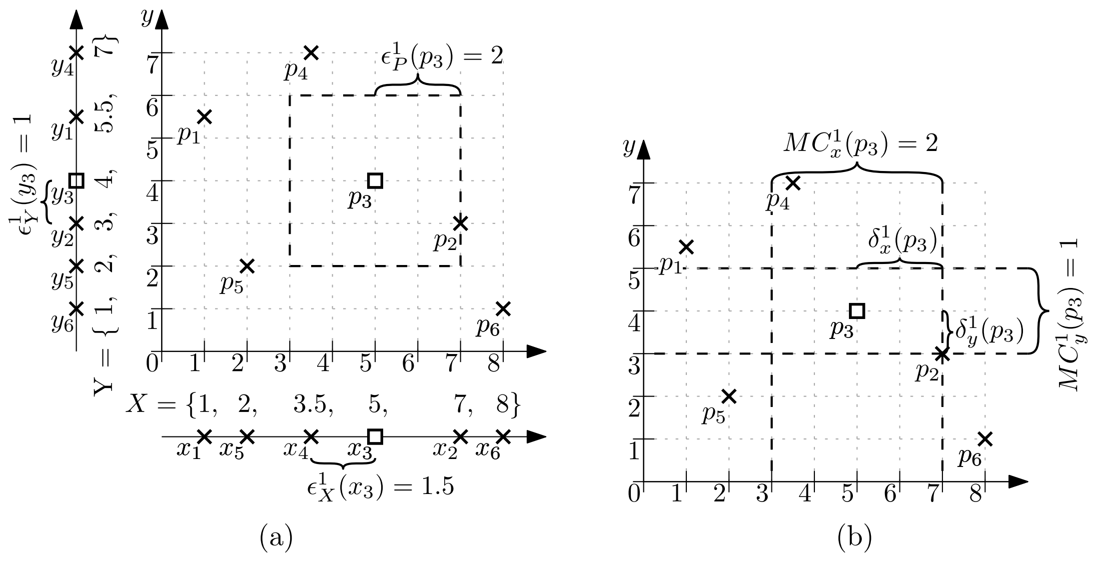


Figure 2.2: Illustration of the notation used for mutual information estimation with the 3KL estimator (a) and the KSG estimator (b), respectively.

called the *3KL* estimator for mutual information. Because each term in Equation 2.7 is substituted with a consistent estimator, the 3KL estimator is also consistent.

Later, Kraskov et al. [KSG04] argued argued that the errors in estimation for $H(X)$, $H(Y)$ and $H(X, Y)$ are unlikely to cancel out and rather lead to a larger overall error. They also proposed a different approach to use the entropy estimator in Equation 2.16 for mutual information estimation. While the 3KL estimator uses the same k when estimating $H(X)$, $H(Y)$ and $H(X, Y)$ to obtain a compact formula, Kraskov et al. adjust k for every point such that the logarithmic term is 0. The idea is to make the distances $\epsilon_P^k(p_i), \epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ of a point to its nearest neighbors in X, Y and P comparable. To achieve this, the parameter k for $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ has to be set accordingly and may be different for each point. Specifically, Kraskov et al. proposed two version, where the first worked with identical distances and the second one worked with "identical neighbors". Because the approach with identical distances used the KL estimator properly only for either $H(X)$ or $H(Y)$, the second version was created. While both version are very similar and thus all results and techniques in this dissertation apply to both version, we only introduce and discuss the second version to avoid ambiguity and redundancy. In this version, illustrated by Figure 2.2b, each nearest neighbor p_j of p_i in P should result in x_j being a nearest neighbor of x_i in X and y_j being a nearest neighbor of y_i in Y. For each point $p_i \in P$, its $k \in \mathbb{N}^+$ nearest neighbors in P using the maximum distance form the set $kNN(p_i)$. More formally, it is

$$kNN_P(p_i) = \underset{S \subseteq (P \setminus \{p\}) \ s.t. \ |S|=k}{\arg\min} \ \max_{p_j \in S} \|p_i, p_j\|_\infty, \tag{2.20}$$

with $\|p, s\|_\infty = \max(|x_i - x_j|, |y_i - y_j|)$. We define the largest distance between x_i and any x-value among the k nearest neighbors of p_i as $\delta_x^k(p_i) = \max_{p_j \in kNN(p_i)} |x_i - x_j|$. We use this distance $\delta_x^k(p_i)$ to define the *x-marginal count*

$$MC_x^k(p_i) = |\{x \in (X \setminus \{x_i\}) : |x - x_i| \leq \delta_x^k(p_i)\}|, \tag{2.21}$$

which is the number of points whose x-value is "close to p". Specifically, the x-marginal count is the smallest number of nearest neighbors for x such that each of the k nearest neighbor of p is also a nearest neighbor of x, i.e.,

$$p_i \in kNN_P(p) \Rightarrow x_i \in \overbrace{k_{p_i}}^{MC_x^k(p_i)} NN_X(x_i) \tag{2.22}$$

holds. In Figure 2.2b, 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 one point excluding p_3, it is $MC_x^1(p_3) = 1$. The distance $\delta_y^k(p_i)$ and the *y-marginal count* $MC_y^k(p_i)$ are defined analogously. Note that $\delta_x^k(p_i)$ and $\delta_y^k(p_i)$ may differ, which results in differently sized areas for the marginal counts, as seen in Figure 2.2b. These marginal counts are then used as substitute for k when estimating $H(X)$ and $H(Y)$. Defering the full derivation to [KSG04], the KSG estimator, named after its inventors inventors Kraskov, Stögbauer and Grassberger, is

$$\widehat{I_{KSG}}(P) = \psi(n) + \psi(k) - \frac{1}{k} - \frac{1}{n} \sum_{i=1}^{n} \psi\left(MC_x^k(p_i)\right) + \psi\left(MC_y^k(p_i)\right). \tag{2.23}$$

While the original introduction of the KSG estimator offered no formal guarantees, empirical studies showed benefits over the 3KL estimator and entirely different techniques. Recently, Gao et al. [GOV18] studied the formal properties of the KSG estiamtor and proved that it is a consistent estimator for fixed k.

We conclude this section with a few additional remarks on nearest-neighbor based estimation of mutual information. While both the 3KL and KSG have k as open parameter, they are consistent estimators for any fixed choice and the literature recommends small choices, e.g., $k \leq 10$ [KA14, KBG$^+$07, KSG04]. As a result, we consider k to be a constant for asymptotic considerations. Furthermore, other "non-parametric" methods to estimate mutual information like kernel density estimation or binning have parameters that are harder to choose [DL01, KBG$^+$07, PK09]. Additionally, nearest-neighbor based estimation consistently yields the best or among the best results for non-parametric methods [KA14, KBG$^+$07, PK09, WL09], resulting in the widespread use and popularity of the KSG estimator [AYL$^+$11, KBHJ08, QGP10, SHR$^+$07]. As a result, we consider the 3KL and KSG as default methods for mutual information estimation and the foundation for our techniques.

2.3 Maximum Entropy Models

Naturally, entropy as formalization of information content offers many possibilities in data analysis beyond quantifying dependency strength. Another use of entropy in this dissertation is the representation of knowledge. In this section, we motivate and present a technique that produces distributions based on limited information. More precisely, the *maximum entropy model* is an distribution that is a representation of incomplete knowledge about a unknown distribution. This concept is relevant to the task of dependency summarization. To evaluate how well a summary describes the dependency, one might compare the maximum entropy model corresponding to the summary with the empirical distribution. In the following, we give a brief motivation and formalization for this method and recapitulate a well-known computation algorithm.

The underlying problem for maximum entropy models is that some information on a dependency is known in the form of a limited number of *hints*. Given two discrete attributes X and Y, each hint defines a selection of value combinations from X and Y and the overall probability of this selection. For instance, let X and Y by two six-sided dice, then one hint could be that the probability of dice X showing less than a four and dice Y showing an even number is 25%. As a result, the hints representing the available, incomplete knowledge may overlap arbitrarily and some value combination may not be covered by any hint. Nevertheless, many applications demand a unified model or distribution for formal analysis, such as analytical biology [PAS06]. In most cases, the available hints are insufficient to exactly determine the distribution. This means that the challenge lies with the choice from a potentially infinite number of distributions that conform to the hints available.

A popular approach to tackle the problem of choice is commonly known by the term "Occam's Razor" [Laz10]. Plainly speaking, the concept states that the best hypothesis is the one with the fewest assumptions that still yields the same predictions. Transferring this notion to distribution modeling, Jayne [Jay57] argued that the "hypothesis with the fewest assumptions" translates well to the distribution with highest entropy. This understanding is called the *principle of maximum entropy*. Among all distributions that conform to the available hints, the one with the highest entropy is called the *maximum entropy model* and introduces the fewest additional assumptions by this argumentation.

Next, we formalize the maximum entropy model and follow up with a brief example. Let X and Y be discrete attributes with the domain $\mathbb{X}$ and $\mathbb{Y}$, respectively. Additionally, let $p(x)$, $p(y)$, and $p(x,y)$ be the probability of X, Y and their joint distribution to take the specific value, respectively. Lastly, let $R = ((\mathbb{Z}_1, r_1), \ldots, (\mathbb{Z}_m, r_m))$ be the collection of hints with $\mathbb{Z}_i \subseteq \mathbb{X} \times \mathbb{Y}$, $r_i \in \mathbb{R}$ and $\sum_{(x,y)\in\mathbb{Z}_i} p(x,y) = r_i$.

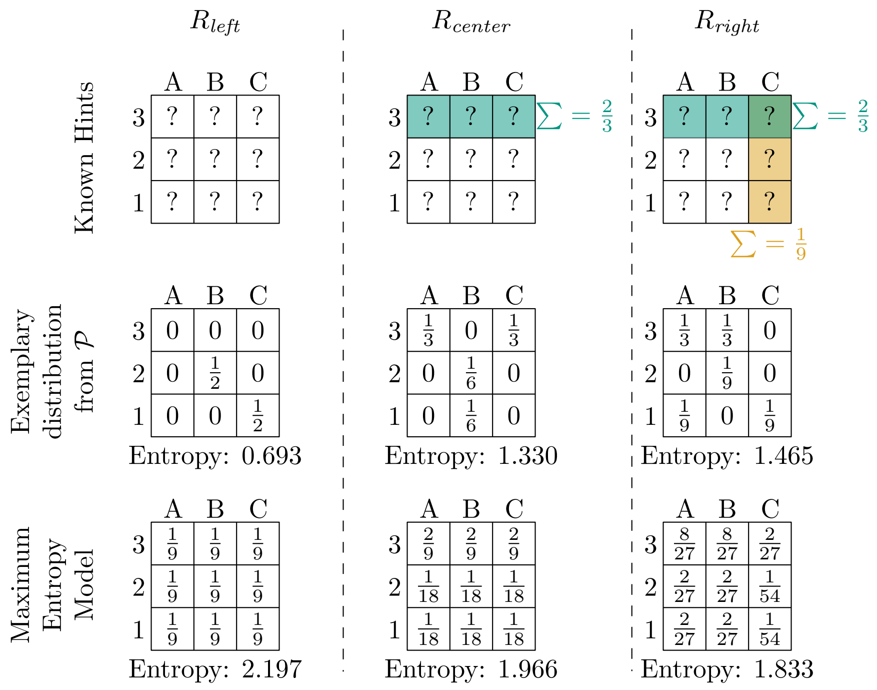


Figure 2.3: Illustration of Example 2.1 for maximum entropy models.

Then the set of all probability distributions that conform to all hints is

$$\begin{aligned}\mathcal{P} = \{f : \mathbb{X} \times \mathbb{Y} \to \mathbb{R} \mid \forall (x,y) \in \mathbb{X} \times \mathbb{Y} \colon f(x,y) \geq 0 \\ \wedge \sum_{(x,y)\in\mathbb{X}\times\mathbb{Y}} f(x,y) = 1 \\ \wedge \forall (\mathbb{Z}_i, r_i) \in R \colon \sum_{(x,y)\in\mathbb{Z}_i} f(x,y) = r_i\}\end{aligned} \tag{2.24}$$

and the maximum entropy model for the joint probability distribution is

$$\arg\max_{f\in\mathcal{P}} H(f) = \arg\max_{f\in\mathcal{P}} \left(- \sum_{y\in\mathbb{Y}, x\in\mathbb{X}} \log(f(x,y)) \cdot f(x,y) \right) \tag{2.25}$$

Example 2.1. *As a short example, we consider the attributes $\mathbb{X} = \{A, B, C\}$ and $\mathbb{Y} = \{1, 2, 3\}$. Then the set $\mathcal{P}$ and maximum entropy model depends on the set R of hints. Figure 2.3 illustrates three sets for R with an exemplary distribution from $\mathcal{P}$ and the maximum entropy model. The left column shows $R_{\text{left}} = \emptyset$, the center column is $R_{\text{center}} = \{(\{(3,A),(3,B),(3,C)\}, \frac{2}{3})\}$ and the right column represents $R_{\text{right}} = \{(\{(3,A),(3,B),(3,C)\}, \frac{2}{3}), (\{(1,C),(2,C),(3,C)\}, \frac{1}{9})\}$. For each column,* *the top row illustrates the corresponding set R, the middle row shows an arbitrary distribution conforming to R and the bottom shows the respective maximum entropy model. Note that the entropy of the maximum entropy model decreases with more hints in R. The reason is that additional hints in R enforce more structure for conforming distributions, i.e., they have frequent, less informative values. For instance, when comparing the bottom left distribution with the bottom center distribution, the combinations with $C = 3$ are more frequent and the other combinations are less frequent. Considering entropy, cf. Equation 2.5, these items are more frequent and less informative which outweighs the increased information content of the now less frequent items.*

When analyzing dependencies, however, one might be more interested in the interactions between X and Y and less interested in the probabilities of individual combinations. Then, the conditional distribution $f(y|x)$ of Y conditioned on X might be more insightful. By the same arguments as before a maximum entropy model for the conditional distribution is well defined and insightful as shown by its use in natural language processing [BPP96, NLM99, PLV02] and outcome explanations [GAG+14, MTV11, Sar01]. To be specific, the necessary modification for the formal definition of the maximum entropy model of $f(y|x)$ essentially substitutes $f(x, y)$ by $p(x) \cdot f(y|x)$. Note that this means that $f(y|x)$ is a collection of probability distributions, one distribution for each $x \in \mathbb{X}$. The resulting formal definitions are the set of potential conditional distributions

$$\begin{aligned}\mathcal{P}' = \{f : \mathbb{X} \times \mathbb{Y} \to \mathbb{R} \mid \forall (x, y) \in \mathbb{X} \times \mathbb{Y}\colon\ & f(y|x) \geq 0 \\ \wedge \forall x \in \mathbb{X} & \sum_{y \in \mathbb{Y}} f(y|x) = 1 \\ \wedge \forall (\mathbb{Z}_i, r_i) \in R\colon & \sum_{(x,y) \in \mathbb{Z}_i} p(x) \cdot f(y|x) = r_i\}\end{aligned} \tag{2.26}$$

and the maximum entropy model for the conditional probability distributions is

$$\arg\max_{f \in \mathcal{P}'} H(f) = \arg\max_{f \in \mathcal{P}'} \left(\sum_{x \in \mathbb{X}} p(x) \left(- \sum_{y \in \mathbb{Y}} \log(f(y|x)) \cdot f(y|x) \right) \right) \tag{2.27}$$

Next, it remains to determine the maximum entropy model algorithmically. This is non-trivial because $\mathcal{P}'$ has many and potentially an infinite number of elements. It has been shown that the optimaztion problem in Equation 2.27 with the constraints in Equation 2.26 has an unconstrained dual problem that optimizes real-valued *multipliers*. Deferring the details to Berger et al. [Ber97], this alternative form of the optimization problem is

$$\max_{f \in \mathcal{P}'} H(f) = \max_{(\lambda_1, \ldots, \lambda_m) \in \mathbb{R}^m} \left(\sum_{x \in \mathbb{X}} p(x) \left(- \sum_{y \in \mathbb{Y}} \log(f_\wedge(y|x)) \cdot f_\wedge(y|x) \right) \right) \tag{2.28}$$

with

$$f_\wedge(y|x) = \frac{e^{\sum_{\{i\in\mathbb{N}:(x,y)\in\mathbb{Z}_i\}}\lambda_i}}{\sum_{y'\in\mathbb{Y}} e^{\sum_{\{i\in\mathbb{N}:(x,y')\in\mathbb{Z}_i\}}\lambda_i}}. \tag{2.29}$$

While there is still no easy analytical solution to the unconstrained optimization problem, it permits various numerical approximations. Berger et al. [BPP96, Ber97] provided an algorithm exactly for Equation 2.28, which is called *improved iterative scaling*. The core concept of this method is iteratively adjusting the multipliers λ_i such that the modeled distribution matches one hint at a time. While we defer the detailed derivation and proof of convergence [BPP96, PPL97], the equation used to adjust individual multipliers is

$$g(\Delta_i) := r_i - \sum_{(x,y)\in\mathbb{Z}_i} p(x)\cdot f_\wedge(y|x)\cdot e^{R_\#(x,y)\cdot\Delta_i} = 0, \tag{2.30}$$

where $R_\#(x,y) = |\{(\mathbb{Z}_j, r_j) \in R\colon (x,y)\in\mathbb{Z}_j\}|$ is the number of hints that include the combination (x,y). To be sufficiently practical, analytical solutions to Equation 2.30 require $R_\#(x,y)$ to be constant. Since this is not true in general, the proposed alternative is using Newton's Method [AS72, BPP96], that is, using the recurrence $\Delta_i^{j+1} = \Delta_i^j - \frac{g(\Delta_i^j)}{g'(\Delta_i^j)}$ to obtain better approximates for Δ_i in sequence. This is done consecutively and repeatedly for each multiplier until all multipliers converge (up to a choosable precision ϵ). Algorithm 2.1 summarizes the steps of improved iterative scaling.

Algorithm 2.1: Improved Iterative Scaling

```
1  $\lambda_1, \ldots, \lambda_m \leftarrow 0$
2  **repeat**
3      $\Delta_1, \ldots, \Delta_m \leftarrow 0$
4      **for** $i \leftarrow 1$ **to** $m$ **do**
5          **if** $R_\#(x,y)$ *constant for all* $x \in \mathbb{X}_i, y \in \mathbb{Y}_i$ **then**
6              $\Delta_i \leftarrow$ Solution to $r_i = \sum_{(x,y)\in\mathbb{Z}_i} p(x) \cdot f_\wedge(y|x) \cdot e^{R_\#(x,y)\cdot\Delta_i}$
7          **else**
8              $\Delta_i \leftarrow 0$
9              **repeat**
10                 $\Delta' \leftarrow \Delta_i$
11                 $\Delta_i \leftarrow \Delta' - \frac{r_i - \sum_{(x,y)\in\mathbb{Z}_i} p(x)\cdot f_\wedge(y|x)\cdot e^{R_\#(x,y)\cdot\Delta'}}{\sum_{(x,y)\in\mathbb{Z}_i} p(x)\cdot f_\wedge(y|x)\cdot e^{R_\#(x,y)\cdot\Delta'}\cdot R_\#(x,y)}$
12             **until** $|\Delta_i - \Delta'| < \epsilon$;
13         $\lambda_i \leftarrow \lambda_i + \Delta_i$
14 **until** $\sum_{i=1}^{m} |\Delta_i| < \epsilon$;
```

Chapter 3

Related Work

The topic of complex dependencies has many facets and allows for many perspectives and aspects to focus on. In this chapter, we provide a brief overview on related work. For this purpose, we consider different scopes and goals of this dissertation and put them into context with prior and competing work. We start with a short discussion of dependency measurements and the position of mutual information. Then, we focus on more specific aspects of this thesis. That is, we briefly discuss related fields to our contributions and introduce prior work and competing methods.

3.1 Quantifying Dependencies

At first glance, one might expect that an optimal dependency measure, i.e., unlimited by sample size and computational power, would be straightforward and easy to define. However, for given data or distributions it is unclear which portions represent some kind of dependency and which portions are noise. If the studied dependency is well defined, the separation is easier and allows for clear quantification of the noise involved. An examples for such cases is the Pearson correlation coefficient [Ric06] for linear dependencies. With the objective to quantify arbitrary dependencies, it becomes unclear how one could define the noise such that it is comparable across different dependency patterns. Remembering the illustration of various dependency types in Figure 1.2, there are no definitive criteria to determine whether one sample has a stronger dependency than another. As a result, there are various concepts one may use as foundation for a general dependency measures.

From a purely statistical perspective *copulas*, which are multivariate distribution where the marginal distributions are uniformly distributed, provide an interesting starting point. By Sklar's Theorem [DFS13, Skl59] any multivariate distribution can be formulated as combination of the marginal distributions and a copula. For a joint distribution of multiple attributes, a corresponding copula then encapsulates the interaction between the attributes, because the behavior of individual attributes is “factored out”. This enables simple measures for dependencies that use distances between the copula and a independent, multivariate uniform distribution [PGS12]. Other dependency measures based on copulas may be less intuitive. For instance, after transforming available data based on an estimated copula, the “Randomized Dependence Coefficient” [LHS13] uses correlation across random non-linear projections and “Copula Index for Detecting Dependence and Monotonicity” [KM19] evaluates piece-wise monotonicity for different regions of the data.

Another starting point utilizes the implication of independent attributes that the distribution of one attribute does not change if it is conditioned on the other attribute. A practical implementation of this notion is “Contrast” [KMB12, Kel15], which selects a subset of the data based on random ranges on some attribute(s). Then, a statistical test is used to determine whether the distribution of another attribute is different between the original data and the subset. This process is repeated multiple times with different random ranges and the contrast measure is the average significance of the test. This approach was further developed with “Monte Carlo Dependency Estimation” [FB19], which compares two complementary subsets instead and generalizes the statistical test used.

Other methods to quantify dependencies, i.e., being zero if and only if the data is independent, include the “Maximal Information Coefficient” [RRF$^+$11] and “Distance Correlation” [SRB$^+$07]. The maximum information coefficient is loosely based on mutual information, although a stronger emphasis is put on structural discretization of the data. That is, a large number of grids is used to discretize the data with mutual information being used to determine how well individual grids represent the structure of the data. In contrast, distance correlation utilizes distances within a sample both regarding including all marginal distances within the data to provide a measure with a similar overall structure as Pearson’s correlation coefficient [Ric06]. Although it is less intuitive that this measure captures arbitrary dependencies, the authors offer extensive statistical analysis.

Overall, however, there is no consensus yet regarding the evaluation and comparison between different measures. Rényi [Rén59] proposed a set of rules that dependency measures should satisfy, and mutual information does so when it is normalized [Bel62]. However, many measures have not been tested against these rules. Additionally, Rényi offers no optimization goal to compare different measures. Another proposition to evaluate dependency measures is “Equitability” [RRF$^+$11], which demands similar scores for different functional dependencies with similar noise. Unfortunately, it remains up to debate what formalization of this notion is proper and whether any dependency measure can satisfy the criterion [KA14, MMM14]. In summary, entropy and mutual information are far from the only options when quantifying dependencies. However, due to its semantic meaning, widespread usage [AYL$^+$11, KBHJ08, QGP10, SHR$^+$07] and comparatively long history, which permitted a high volume of studies and analysis [KA14, KBG$^+$07, PK09, WL09], mutual information remains a very compelling choice.

3.2 Efficient Mutual Information Estimation

While there exist different approaches to estimating mutual information, this work focuses on the nearest-neighbor based approaches introduced in Section 2.2. This is because they offer good estimation quality [KA14, KBG$^+$07, KSG04, PK09] and are widely used [HSPVB07, PQB05]. Note however, that comparative studies focus primarily on the quality of estimation without particular assessment of the computation time. So far, there has been little research regarding the computational complexity of mutual information estimation in general and the KSG and 3KL methods in particular. Several proposals to compute the KSG appear in the original KSG article [KSG04] with the claimed time complexity $O(n)$ for their fastest, so-called "box-assisted" algorithm on smooth distributions. Vejmelka et al. [VH07] compare their own approach with the box-assisted algorithm and cite [Sch95] for different conditions for a linear runtime of the box-assisted algorithm. In the end, the best universal time complexity of their presented algorithms is $O(n \log n)$. The same complexity is given for the algorithm computing the 3KL by Evans [Eva08]. As part of this thesis we show that this limit is not a coincidence, i.e., we prove that no algorithm computing these estimators can have a time complexity lower than $O(n \log n)$.

3.3 Data Streams

One of our contributions is a fully dynamic data structure to estimation mutual information. While there are various applications working with dynamic sets of data, a prime example that is gaining more and more prominence are data streams. Data streams are data sources that yield new data over time possibly without end. Because data streams grow over time, and memory and storage is limited, it is impossible to store all data points. This means that information is lost over time. Nevertheless, there exist space-efficient estimators for entropy of discrete distributions on streams. With Equation 2.7, entropy estimators can also be used to estimate mutual information, but with accumulating error. The estimator of Chakrabarti et al. [CCM07] provides multiplicative approximations of entropy on insert-only streams. In contrast, the estimator by Harvey et al. [HNO08] offers multiplicative and additive approximations of entropy on streams with insertions and deletions, but requires knowledge about the maximum length of the stream. However, both estimators are restricted to discrete distributions. Estimating mutual information on discrete distributions is easier, because the relative count of points is a good estimator for the probability. Estimating the density of continuous distributions requires more elaborate techniques and remains an ongoing field of research [Gra18, Sil86]. Some works exist for estimation of probabilities on data streams [CHM12, ZCWQ03], however, we are not aware of any work that extends these techniques to entropy or mutual information estimation.

Instead, the few works on the estimation of mutual information on data streams exclusively use the nearest-neighbor based estimation introduced in Chapter 2. The "Mutual Information Stream Estimator" (MISE) framework [KMB15] offers estimates of mutual information between continuous variables for any time interval on data streams. To achieve these results without storing all data points, Equation 2.23 is not computed exactly but rather approximated. The framework provides a sampling scheme such that results are more precise for more recent time intervals. Additionally, queries with equal ratio between interval size and the offset to the current time provide roughly the same quality of estimation. An improvement to this approach was recently proposed as "Point-Based Approach" (PBA) [ABR19], which removes some estimation bias and simplifies the algorithmic structure. However, a relatively common task for processing data streams is monitoring of recent events. This scenario essentially requires accurate estimates in a sliding window using a fixed number of the most recent data points. To provide accurate estimates for such cases, MISE and PBA require both unlimited storage and time that is superlinear in the size of the sliding window. For such scenarios DIMID [BH17] is an approach that provides approximate results to Equation 2.19 with constant storage and time logarithmic in the size of the sliding window. As approximation technique, DIMID uses random projections to transform the joint distribution of two attributes into a third distribution with a single dimension.

Overall, existing works build upon the nearest-neighbor based estimation, however, the results are only approximations that cannot claim the same estimation quality of the KSG or 3KL. In this thesis, in turn, we present fully dynamic data structures that yield exactly the KSG and 3KL estimation results. Thus, the updates of a sliding window can be easily realized as insertions and deletions within that structure. As the state-of-the-art for KSG estimation on sliding windows is computation from scratch for each step [KBHJ08, QGP10], we provide better time complexity by offering liner-time updates. For 3KL estimation, we even offer time in $O(\log n \log \log n)$ for updating an estimate according to a sliding window.

3.4 Anytime Algorithms

Another contribution of ours is an iterative estimator of mutual information, which provides the anytime property. Anytime algorithms [Zil96] 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 [MHF$^+$15], classification [YWKT07] and outlier detection [AKBS12]. The very recent and previously mentioned "Monte Carlo Dependency Estimation" [FB19] is another dependency measure with anytime capability. So far, however, there is no anytime algorithm to estimate mutual information, which remains the widely accepted approach for general dependency estimation. In consequence, the anytime algorithm for mutual information estimation proposed in this thesis extends the set of tools available for anytime computations. Additionally, there has been more general work on the optimal use of available anytime algorithms [HZ01, KS09], which may improve the performance of our proposed method in larger systems.

3.5 Dependency Summarization

Another important aspect of this thesis are explanations for dependencies, i.e., compact, human-readable summaries of concrete dependencies. As dependency measures offer only a single number, it is often unclear what kind of relationship these attributes actually have. Commonly [GAG$^+$14, MTV11, Sar01], such summaries describe multiple portions of the data where stronger claims on the relationship are possible. To simplify the task somewhat and improve interpretability of the summaries, one attribute or combination of attributes is designated as *outcome* beforehand. That is, the summaries are built with the intent of summarizing the influence of all other attributes on this outcome. For the task of selecting these portions of the data used for the summary, various approaches exist. We categorize the available approaches into the following three categories.

Maximum-Entropy Models A self-evident starting point is using the conditional distribution $P(\mathit{outcome}|\mathit{attributes})$ as formalization of the dependency between the outcome and the remaining attributes. This means, the better a summary describes this distribution, the more informative it is. A popular way [GAG$^+$14, MTV11, Sar01] to judge this description quality is comparing the maximum entropy model (cf. Section 2.3) for a summary with the empirical distribution provided by the data. The task of building a summary is then equivalent to the search for the set of hints, usually of a fixed size to ensure conciseness, that yield the entropy model with the smallest distance to the empirical distribution. However, the task to find the optimal set for a given size is NP-hard [GAG$^+$14].

While there exist multiple heuristics, the most recent and best performing approach is explanation tables [GAG$^+$14, FGS17, GFG$^+$18]. The core idea is that relevant hints for the model usually cover large portions of the data. Thus, when drawing a small to moderate sample from the data, most relevant hints should cover at least one of the data points in the sample.

So far, all approaches to produce summaries based on maximum entropy models only support nominal attributes. That is, for numerical values these approaches consider all values entirely distinct irrespective of the differences. Our proposed method to summarize dependencies extends the capabilities of these existing methods to support also ordinal and numerical values. That is, we propose a framework that uses summaries for discrete attributes to build summaries that also support ordinal and nominal attributes. While our method is compatible with any approach that uses maximum entropy models, in this dissertation we use explanation tables as they currently offer the best results [GAG$^+$14, GFG$^+$18].

Subgroup Discovery A strongly related field is subgroup discovery, where the goal are subsets of data that have a very different distribution of outcomes compared to the entire data set [Atz15, Hel16, HCGdJ11, VL18]. The parallel to our summarization problem is that these interesting subgroups correspond to different relationships between the attributes and the outcome. Thus, a collection of interesting subgroups is also a summary of the dependency. However, each subgroup is individually assigned an interestingness score without considering other subgroups. This leads to subgroups that are interesting on their own but redundant and therefore not informative as part of a larger collection. We compare our approach to summarize dependencies experimentally with two state-of-the-art subgroup discovery techniques that also support ordinal and numerical attributes.

The first approach, *MergeSD* [GR09], introduces pruning techniques to reduce the number of candidate subgroups. The interestingness measure used for subgroups is the difference in likelihood of a true outcome between the overall data and the subgroup, weighted by subgroup size. They show an upper bound for this measure under "refinement" of subgroups. That is, for each subgroup, characterized as a pattern, they determine the maximum interestingness for any pattern matching a subset of this subgroup. This upper bound is then used in a depth-first search to prune the pattern space.

The second method is *Diverse Subgroup Set Discovery* (DSSD) [vLK12], which aims to find non-redundant subgroups. DSSD uses a three-phase approach to ensure subgroup diversity. First, it uses a greedy best-first search that maintains a fixed number of candidates to find a large set of relevant subsets. Next, DSSD diversifies the candidate set by pruning dominated subgroups. Here, a subgroup dominates another subgroup if it is formed by a subset of the conditions and has higher interestingness. Finally, the best subgroups from this diversified set are selected. However, this approach may still select similar subgroups, if they are not strictly dominated by others, e.g., through slightly different conditions or higher interestingness.

Classification Instead of summarizing the relationship between an outcome and the remaining attributes, classification uses these relationships to predict the outcome. The representation of the dependency is then encoded in the model of the classifier, which may or may not take a form that is interpretable for humans. This means that the goals are not identical but somewhat similar and comparable to summarization. Classifiers are judged by their correctness on data outside the initial “training” set instead of their ability to summarize existing data. Additionally, human interpretability is often sacrificed for accuracy through feature engineering [GRD$^+$11], ensembles [Rok09] or neural networks [KSH12, Zha00]. Rule based classifiers such as decision trees [Qui93] are an exception as human-interpretable classification methods that partition the data into non-overlapping sets. Interpretable Decision Sets [LBL16] are another exception, which allow a minimal amount of overlap to reduce the number of decision boundaries. However, these approaches are not concise, even after applying optimizations such as tree or rule pruning [GFG$^+$18]. This means that the number of rules used is often overwhelming for human interpretation. So far, there already exists evidence that decision trees and interpretable decision sets produce less informative summaries than explanation tables [GAG$^+$14, GFG$^+$18], which form the basis of our approach. Nevertheless, we include decision trees as a competing approach when evaluating our proposed approach to produce summaries.

Chapter 4

Estimating Mutual Information

This chapter discusses our advances in regards to efficient quantification of complex dependencies. As discussed in the previous chapter, we consider mutual information as the most relevant measure and nearest-neighbor based estimation the most compelling approach. We improve the understanding of this approach and its usability through the following aspects. First we consider the computational complexity of the estimation and prove a lower bound for the problem, which shows that current computation techniques are asymptotically optimal. This means that new algorithms can improve the computation time only by constant factors but not its scalability with the size of the data. Then, we turn to estimation of mutual information on dynamic data, i.e., data sets that change over time. Formally, we consider dynamic data as a set that allows insertion and deletion of individual items. Sliding windows, which are one of the most well-known cases of dynamic data, then correspond to one insertion and one deletion for each time step. We establish the complexity of updating estimates according to changes in the data and present new algorithms to compute these updates. Finally, we consider the scenario of dynamic estimation of mutual information in the sense that the estimator itself offers a dynamic time-quality-tradeoff. Specifically, we introduce an iterative estimator that quickly offers a rough approximation of the nearest-neighbor based estimate and could use additional computation time to improve this approximation. Additional computation time is used in fine-grained time-steps called *iterations*. This gives the iterative estimator the anytime property as one could interrupt the estimation at any point in time and obtain the results from the latest finished iteration. Additionally, this estimator offers statistical guarantees for its approximations such that users can evaluate whether a given estimation quality is sufficient or additional time should be used.

4.1 Computational Complexity

This section is previously published in a mostly identical form at the International Conference on Extending Database Technology (EDBT '18) in [VRB18]. There are only some adjustments to fit the form and consistency for this dissertation.

In this section we present our first contribution, the lower bounds for computing estimates using the KSG and 3KL. As a brief reminder, the corresponding equations are (cf. Section 2.2):

$$\widehat{I_{3KL}}(P) = \psi(n) - \psi(k) + \frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{\epsilon_X^k(x_i)\cdot\epsilon_Y^k(y_i)}{\left(\epsilon_P^k(p_i)\right)^2}\right), \tag{2.19 revisited}$$

$$\widehat{I_{KSG}}(P) = \psi(n) + \psi(k) - \frac{1}{k} - \frac{1}{n}\sum_{i=1}^{n} \psi\left(MC_x^k(p_i)\right) + \psi\left(MC_y^k(p_i)\right). \tag{2.23 revisited}$$

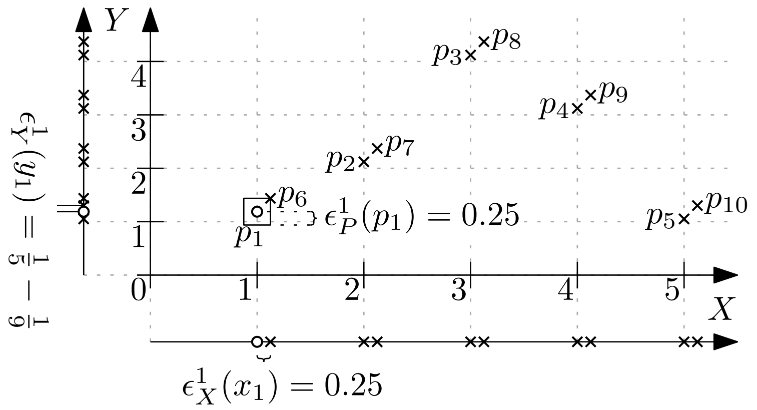
All existing approaches to compute the 3KL and KSG follow the original description in the sense that they first compute the nearest neighbors of all points. In the case of the KSG, the marginal counts MC_x^k and MC_x^k are computed afterwards. However, it is not known if this is the only approach to compute $\widehat{I_{3KL}}(P)$ and $\widehat{I_{KSG}}(P)$, or if it is computationally optimal. For instance, there could be a different formula for either of these estimators that does not require explicit computation of the nearest neighbors. Consequently, the complexity of computing the 3KL and KSG estimates can only be based on the result and not on intermediate steps such as determining the nearest neighbors. The problems whose complexities we want to study in general, i.e., without confinement to specific algorithms, are the following ones.

Problem 4.1 (3KL-Estimation)**.** *For a set $P \subseteq \mathbb{R}^2$ of points, determine $\widehat{I_{3KL}}(P)$.*

Problem 4.2 (KSG-Estimation)**.** *For a set $P \subseteq \mathbb{R}^2$ of points, determine $\widehat{I_{KSG}}(P)$.*

In the following, we show the complexity of Problem 4.1. By reducing a problem with known complexity to 3KL-Estimation, we prove that it has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model [Ben83]. We use the algebraic computation tree model because it allows us to prove bounds without assuming any statistical properties of the data. This is important because we want general-purpose estimation of mutual information. If knowledge regarding the data or its distribution was known, it could be used to model the density function in Equation 2.19.

Theorem 4.3. *The problem* 3KL-Estimation *has time complexity* $\Omega(n \log n)$.



Y
X
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
$\epsilon_P^1(p_1) = 0.25$
$\epsilon_Y^1(y_1) = \frac{1}{5} - \frac{1}{6}$
$\epsilon_X^1(x_1) = 0.25$
0
1
2
3
4
5

Figure 4.1: An illustration of the construction of P in Theorem 4.3.

Proof. The proof is by reduction from the problem IntegerElementDistinctness: Given a multiset $A = \{a_1, \ldots, a_n\}$ of integers, are there two indices $i \neq j$ such that $a_i = a_j$ are duplicates. The problem IntegerElementDistinctness has a known lower bound of $\Omega(n \log n)$ in the algebraic computation tree model [LR91]. For an instance A of IntegerElementDistinctness, we construct an instance of 3KL-Estimation P as follows. For $a_i \in A$, the set P contains two points $p_i = (i, a_i + \frac{1}{4+i})$ and $p_{n+i} = p_i + (0.25, 0.25)$. Note that p_i and p_{n+i} are closer than any other pair, because i and a_i are integers. Additionally, we add the offset $\frac{1}{4+i}$ to the y-coordinates, because duplicates in A would otherwise lead to a nearest-neighbor distance of 0 and thus $\log(0)$ in Equation 2.19. Figure 4.1 features an example for this construction for the IntegerElementDistinctness instance $A = \{1, 2, 4, 3, 1\}$. The point p_1 is highlighted as circle and its nearest-neighbor distances are highlighted.

Claim 4.4. *A contains a duplicate if and only if* $\widehat{I_{3KL}}(P) \neq \psi(|P|) - \psi(1)$ *for* $k = 1$.

Subproof. Let $i \neq j \in \{1, \ldots, n\}$ be two integers. By construction of P, it follows that $|x_i - x_j| = |x_{n+i} - x_{n+j}| \geq 1$. Additionally, it is $|x_i - x_{n+i}| = |y_i - y_{n+i}| = 0.25$. Using the reverse triangle inequality, it is $|x_i - x_{n+j}| \geq |x_i - x_j| - |x_j - x_{n+j}| \geq 0.75$. This holds for any $i \neq j$, which means that p_i is the nearest neighbor of p_{n+i} and vice versa, because we use the maximum norm. As a consequence the nearest-neighbor distances are $\epsilon_P^1(p) = 0.25$ for all $p \in P$ and $\epsilon_X^1(x) = 0.25$ for all $x \in X$. Note that this means that the nearest-neighbor distances in P and X are independent of the existence of duplicates in A.

If A does not contain any duplicates, it follows that $|y_i - y_j| = |y_{n+i} - y_{n+j}| \geq \frac{4}{5}$, since A only contains integers and the difference between $\frac{1}{4+i}$ and $\frac{1}{4+j}$ is less than $\frac{1}{5}$. By the same arguments as above it follows that $|y_i - y_{n+j}| \geq 0.55$ and that $\epsilon_Y^1(y) = 0.25$ for all $y \in Y$. We can then use these values in Equation 2.19, which yields:

$$\widehat{I_{3KL}}(P) = \psi(|P|) - \psi(1) + \frac{1}{|P|}\sum_{m=1}^{|P|}\log\left(\frac{0.25 \cdot 0.25}{(0.25)^2}\right) = \psi(|P|) - \psi(1). \tag{4.1}$$

Conversely, if A contains the duplicates $a_i = a_j$, it is $|y_i - y_j| = |\frac{1}{4+i} - \frac{1}{4+j}| \leq 0.2$ and $|y_i - y_{n+j}| \geq |0.25 - \frac{1}{4+j}|$ and thus $\epsilon^1_Y(y_i) \leq 0.2$. Additionally, because of $j \neq i$ it also is $\epsilon^1_Y(y_i) > 0$. Also, because the maximum norm is used for nearest neighbors, it is $\log(\frac{\epsilon^1_X(x_m)\cdot\epsilon^1_Y(y_m)}{(\epsilon^1_P(p_m))^2}) \leq 0$ for all $m \in \{1, \ldots, |P|\}$. It follows that

$$\log\left(\frac{\epsilon^1_Y(y_i) \cdot \epsilon^1_X(x_i))}{(\epsilon^1_P(p_i))^2}\right) < 0 \Rightarrow \frac{1}{|P|}\sum_{m=1}^{|P|}\log\left(\frac{\epsilon^1_X(x_m) \cdot \epsilon^1_Y(y_m)}{\epsilon^1_P(p_m))^2}\right) < 0 \tag{4.2}$$

and analogously to Equation 4.1 we obtain $\widehat{I_{3KL}}(P) < \psi(|P|) - \psi(1)$. This concludes the subproof. ■

It is clear that P can be constructed in time $O(|A|)$, which means $|P| \in O(|A|)$. After computing $\widehat{I_{3KL}}(P)$, the result is only compared to a sum over $|P|$ numbers, because $\psi(|P|) - \psi(1) = \sum_{m=1}^{|P|-1}\frac{1}{m}$ by definition of the digamma function. Note that this reduction works analogously for any fixed $k > 0$ by placing $k - 1$ points evenly spaced on the diagonal between each pair p_i and p_{n+i}. Because k is fixed, the size of P increases only by a constant factor. Therefore, the complexity of the reduction is in $O(n)$. This means that determining $\widehat{I_{3KL}}(P)$ has a lower bound of $\Omega(n \log n)$. □

This lower bound matches the running time of the algorithm presented by Evans [Eva08] to solve 3KL-Estimation. Consequently, this algorithms is already asymptotically optimal, and the lower bound is tight.

Corollary 4.5. *The computational complexity of* 3KL-Estimation *is* $\Theta(n \log n)$.

We use the same approach to prove a lower bound for KSG-Estimation. With the algorithms presented by Vejmelka et al. [VH07] this lower bound is tight as well.

Theorem 4.6. *The problem* KSG-Estimation *has a time complexity in* $\Omega(n \log n)$.

Proof. Similarly to the Proof of Theorem 4.3, we reduce the problem to IntegerElementDistinctness. For any instance A of IntegerElementDistinctness, we construct an instance of KSG-Estimation P as follows. For $a_i \in A$, the set P contains two points $p_i = (i, a_i)$ and $p_{n+i} = (i + 0.25, a_i + 0.25)$. We use 0.25 because it means that this pair of points is closer than any other pair, because i and a_i are integers. Figure 4.2 features an example for this construction for the IntegerElementDistinctness instance $A = \{1, 4, 2, 5, 3, 2\}$. The dashed lines in the figure illustrate the areas of the marginal counts $MC^1_x(p_3)$ and $MC^1_y(p_3)$.

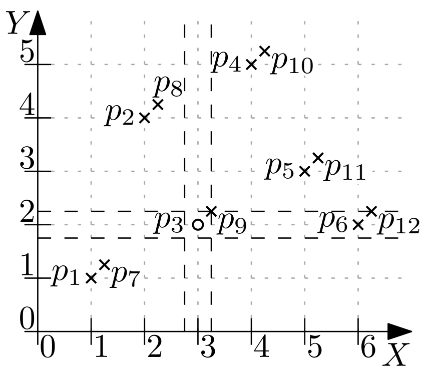


Figure 4.2: An illustration of the construction of P in Theorem 4.6.

Claim 4.7. *It is* $\widehat{I_{KSG}}(P) \neq \sum_{m=1}^{|P|-1}(\frac{1}{m}) - 1$ *for* $k = 1$ *if and only if* A *contains a duplicate.*

Subproof. Let $i \neq j \in \{1, \dots, n\}$ be two integers. By construction of P, it follows that $|x_i - x_j| = |x_{n+i} - x_{n+j}| \geq 1$. Additionally, it is $|x_i - x_{n+i}| = |y_i - y_{n+i}| = 0.25$. Using the reverse triangle inequality, it is $|x_i - x_{n+j}| \geq |x_i - x_j| - |x_j - x_{n+j}| \geq 0.75$. This holds for any $i \neq j$, which means that p_i is the nearest neighbor of p_{n+i} and vice versa, because we use the maximum norm. As a consequence, the marginal counts $MC_x^1(p)$ are 1 for all $p \in P$, independent of the existence of duplicates in A.

If A does not contain any duplicates, it follows that $|y_i - y_j| = |y_{n+i} - y_{n+j}| \geq 1$, since A only contains integers. By the same arguments as above it follows that $|y_i - y_{n+j}| \geq 0.75$ and that $MC_y^1(p) = 1$ for all $p \in P$. We can then use these values in Equation 2.23 and because of $\psi(x) = \sum_{m=1}^{x-1}(\frac{1}{m}) - \gamma$ it is:

$$\widehat{I_{KSG}}(P) = \psi(1) + \psi(|P|) - \frac{1}{1} - \frac{1}{|P|}\sum_{m=1}^{|P|}\psi(1) + \psi(1) = \sum_{m=1}^{|P|-1}\left(\frac{1}{m}\right) - 1 \tag{4.3}$$

Conversely, if A contains the duplicates $a_i = a_j$, it follows that $|y_i - y_j| = 0$ and $|y_i - y_{n+j}| = 0.25$ and thus $MC_y^1(p_i) \geq 3$. Because of $\psi(x+1) = \psi(x) + \frac{1}{x} > \psi(x)$ for all $x \geq 0$, it is, analogous to Equation 4.3, $\widehat{I_{KSG}}(P) < \sum_{m=1}^{|P|-1}(\frac{1}{m}) - 1$. This concludes the subproof. ■

It is clear that P can be constructed in time $O(|A|)$, which means $|P| \in O(|A|)$. Note that this reduction works analogously for any fixed $k > 0$ by placing $k - 1$ points evenly spaced on the diagonal between each pair p_i and p_{n+i}. Because k is fixed, the size of P increases only by a constant factor. After computing $\widehat{I_{KSG}}(P)$, the result is only compared to a sum over $|P|$ numbers. Therefore the complexity of the reduction is in $O(n)$. This means that determining $\widehat{I_{KSG}}(P)$ has a lower bound of $\Omega(n \log n)$. □

Corollary 4.8. *The computational complexity of* KSG-Estimation *is* $\Theta(n \log n)$.

To summarize this section, we have proven a lower bound for the well-known nearest-neighbor based mutual information estimators. As this lower bound matches the run-time of existing computation algorithms, the bound is tight and the algorithms are asymptotically optimal. In the next section we expand this investigation of complexity and solutions from estimation for a fixed data set to estimation with dynamic data.

4.2 Mutual Information Estimation on Changing Data

This section is previously published in a mostly identical form at the International Conference on Extending Database Technology (EDBT '18) in [VRB18]. There are only some adjustments to fit the form and consistency for this dissertation.

Now, we investigate the challenges and opportunities to estimate mutual information on dynamic data. The distinctive feature of dynamic data is that the data changes over time. For a set P of points, all changes can be modeled using insertion of new points and deletion of existing points. For instance, moving a point from (x, y) to (x', y') can be modeled with one deletion of (x, y) and one insertion of (x', y'). To maintain an estimate of mutual information with the 3KL or KSG, we need to adjust the estimate according to such insertions or deletions. We see this as a problem for a dynamic data structure and thus allow storage of some auxiliary information about P, noted as *state* S_P of a dynamic data structure. The formal problem is then:

Problem 4.9 (3KL-Update). *Let* $P \subseteq \mathbb{R}^2$ *be a set of points,* S_P *the state for* P *and* $p \in \mathbb{R}^2$ *a point. Determine* $\widehat{I_{3KL}}(P \cup \{p\})$ *and* $S_{P\cup\{p\}}$ *if* p *is inserted and* $\widehat{I_{3KL}}(P \setminus \{p\})$ *and* $S_{P\setminus\{p\}}$ *if* p *is deleted using only* S_P *and* p.

Problem 4.10 (KSG-Update). *Let* $P \subseteq \mathbb{R}^2$ *be a set of points,* S_P *the state for* P *and* $p \in \mathbb{R}^2$ *a point. Determine* $\widehat{I_{KSG}}(P \cup \{p\})$ *and* $S_{P\cup\{p\}}$ *if* p *is inserted and* $\widehat{I_{KSG}}(P \setminus \{p\})$ *and* $S_{P\setminus\{p\}}$ *if* p *is deleted using only* S_P *and* p.

Because these problems can be used to solve 3KL-Estimation and KSG-Estimation, respectively, we can use the previous results to infer lower bounds for their time complexities. If we start with an empty set P and incrementally insert n points, the total time required cannot generally be asymptotically faster than $\Omega(n \log n)$ by Theorem 4.3 and Theorem 4.6. Because this includes n insertions, the time complexity of individual insertions is in $\Omega(\log n)$.

Corollary 4.11. *The problem* 3KL-UPDATE *has a time complexity in* $\Omega(\log n)$.

Corollary 4.12. *The problem* KSG-UPDATE *has a time complexity in* $\Omega(\log n)$.

These lower bounds represent both challenge and opportunity, since they are much lower than the lower bounds for static estimation. On the one hand this indicates that there is a lot of potential for asymptotic improvements. On the other hand, it is very challenging to provide algorithms with logarithmic runtime, that is, it is hard to achieve asymptotically optimal algorithms. This is particularly true because these bounds might not be tight. So there could exist another, asymptotically higher lower bound. In the following, we present two data structures for these tasks, evaluate their time complexities and compare them to the lower bounds presented here.

4.2.1 Updating Estimates

Naturally, the simplest solution to KSG-UPDATE and 3KL-UPDATE is storing exactly P in S_P and computing $\widehat{I_{3KL}}(\cdot)$ and $\widehat{I_{KSG}}(\cdot)$, respectively, with every change. The result from the previous section is that any such approach would require time in $\Omega(n \log n)$ for 3KL-UPDATE and KSG-UPDATE. In the following we show that this is not optimal and present a more efficient solution.

We propose the data structure DEMI (Dynamic Estimation of Mutual Information) that focuses on updating an estimate of the 3KL or KSG for a single insertion or deletion. First, we present how this data structure works with 3KL estimates in detail. Afterwards, we describe the differences when maintaining a KSG estimate. For the 3KL estimate, we first describe the changes that can occur by inserting or deleting a point. Then we describe which information our data structure stores and how it determines the changes in the 3KL estimate efficiently. Lastly, we evaluate the space complexity of our data structure as well as the time complexity of adding or deleting a point.

Updating 3KL Estimates

Let $P = \{p_1 = (x_1, y_1), \dots, p_n = (x_n, y_n)\} \subseteq \mathbb{R}^2$ be the set of points in our sample and let $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_n\}$ be the set of values per attribute, respectively. For a new point $p_{n+1} = (x_{n+1}, y_{n+1}) \in \mathbb{R}^2$, let $P' = P \cup \{p_{n+1}\}$, $X' = X \cup \{x_{n+1}\}$ and $Y' = Y \cup \{y_{n+1}\}$ be the sets including p_{n+1}, x_{n+1} and y_{n+1}, respectively. Considering Equation 2.19, the change from $\widehat{I_{3KL}}(P)$ to $\widehat{I_{3KL}}(P')$ consists of three partial changes:

(1) $\psi(n)$ increases to $\psi(n+1) = \psi(n) + \frac{1}{n}$,

Data Structure 4.1: DEMI

```
struct {
    real $x, y$
    real $\epsilon_P^k, \epsilon_X^k, \epsilon_Y^k$
} DemiPoint;

struct {
    DemiPoint[ ] $P_D$
    BST<DemiPoint*> $T_x, T_y$
    real $base, sum$
} state;
```

(2) the arithmetic mean includes $n + 1$ logarithms instead of n,

(3) and the nearest-neighbor distances $\epsilon_P^k(p_i)$, $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ may change for any $i \in \{1, \ldots, n\}$.

While Change (1) is trivial, Change (2) requires the computation of $\epsilon_{P'}^k(p_{n+1})$, $\epsilon_{X'}^k(x_{n+1})$ and $\epsilon_{Y'}^k(y_{n+1})$. However, Change (3) could require the re-evaluation of all nearest-neighbor distances. Clearly, these changes apply analogously if p_1 is removed from P instead of inserting p_{n+1}. Following these observations, we propose a dynamic data structure that determines these changes efficiently and evaluate its computational complexity.

Overview Our data structure, DEMI, is given in Data Structure 4.1. For each point $p_i \in P$ of our sample, we store its attributes x_i, y_i and k-th nearest-neighbor distances $\epsilon_P^k(p_i)$, $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ as a *DemiPoint*. In addition, we store references to all DemiPoints, ordered by the x-component and y-component of the point, in binary search trees (BST) T_x and T_y, respectively. Using self-balancing BSTs like red-black-trees, we can insert, delete and search items in logarithmic time. Additionally, we also maintain the values $\mathit{base} = \psi(|P|) - \psi(k)$ and $\mathit{sum} = \sum_{i=1}^{n} \log\left(\frac{\epsilon_X^k(x_i)\cdot\epsilon_Y^k(y_i)}{(\epsilon_P^k(p_i))^2}\right)$. The collection of all stored data is the state S_P of our data structure for the sample P. Because we store a constant amount of information per point, the space complexity of DEMI is $\Theta(n)$. Given State S_P, one can query the 3KL estimate on the set P in constant time as $\widehat{I_{3KL}} = \mathit{base} + \frac{\mathit{sum}}{|P|}$. However, this data structure requires adjustment of S_P after every change of P.

Insertion Algorithm To insert a point p_{n+1} into a state S_P, illustrated in Algorithm 4.2, we distinguish two phases of the update. First (Lines `1-6`), we add p_{n+1}

Algorithm 4.2: INSERT(S_P, p_{n+1})

```
1  Compute $\epsilon^k_{P'}(p_{n+1})$                                                    $O(n)$
2  Compute $\epsilon^k_{X'}(x_{n+1})$ and $\epsilon^k_{Y'}(y_{n+1})$                   $O(k \cdot \log n)$
3  Insert $p_{n+1}$ into $P_D$                                                          $O(1)$
4  Reference $p_{n+1}$ in $T_x$, $T_y$                                                  $O(\log n)$
5  $base \leftarrow base + \frac{1}{n}$                                                 $O(1)$
6  $sum \leftarrow sum + \log\left(\frac{\epsilon^k_{X'}(x_{n+1}) \cdot \epsilon^k_{Y'}(y_{n+1})}{(\epsilon^k_{P'}(p_{n+1}))^2}\right)$   $O(1)$
7  $A \leftarrow \{p_i \in P\colon \max(|x_i - x_{n+1}|, |y_i - y_{n+1}|) < \epsilon^k_P(p_i)\}$   $O(n)$
8  $B \leftarrow \{p_i \in P\colon |x_i - x_{n+1}| < \epsilon^k_X(x_i)\}$              $O(n)$
9  $C \leftarrow \{p_i \in P\colon |y_i - y_{n+1}| < \epsilon^k_Y(y_i)\}$              $O(n)$
10 forall $p_i \in A$ do
11     Compute $\epsilon^k_{P'}(p_i)$                                                   $O(|A| \cdot n)$
12     $sum \leftarrow sum + \log((\epsilon^k_P(p_i))^2) - \log((\epsilon^k_{P'}(p_i))^2)$   $O(|A|)$
13 forall $p_i \in B$ do
14     Compute $\epsilon^k_{X'}(x_i)$                                                   $O(|B| \cdot k \cdot \log n)$
15     $sum \leftarrow sum - \log(\epsilon^k_X(x_i)) + \log(\epsilon^k_{X'}(x_i))$      $O(|B|)$
16 forall $p_i \in C$ do
17     Compute $\epsilon^k_{Y'}(y_i)$                                                   $O(|C| \cdot k \cdot \log n)$
18     $sum \leftarrow sum - \log(\epsilon^k_Y(y_i)) + \log(\epsilon^k_{Y'}(y_i))$      $O(|C|)$
```

as a DemiPoint to P_D and update *base* and *sum* accordingly. Second (Lines 7-18), we determine which nearest-neighbor distances change and adjust *sum* according to the changes. We now describe these steps in more detail, together with the computational complexity of elementary operations, to allow for an easier evaluation. We discuss possible improvements in Section 4.2.2.

To add p_{n+1} to S_P, we first compute its k-th nearest neighbor in P' by linear search and derive the k-th nearest-neighbor distance $\epsilon^k_{P'}(p_{n+1})$ ($O(n)$, Line 1). To determine the k-th nearest-neighbor distances $\epsilon^k_{X'}(x_{n+1})$ and $\epsilon^k_{Y'}(y_{n+1})$ we can use the binary search tree and evaluate the distance to the next k and preceding k elements ($O(k \cdot \log n)$, Line 2). With this information we construct the DemiPoint for p_{n+1} and insert it into P_D ($O(1)$, Line 3). References to this point are then inserted into T_x and T_y ($O(\log n)$, Line 4). Then, we add the appropriate terms to *base* and *sum* ($O(1)$, Lines 6 and 7), respectively.

Next, we find all previous nearest-neighbor distances that changed, by linear search. For each $i \in \{1, \dots, n\}$ we test whether p_{n+1}, x_{n+1} and y_{n+1} is closer than $\epsilon^k_P(p_i), \epsilon^k_X(x_i)$ and $\epsilon^k_Y(y_i)$, respectively. This takes time in $O(n)$ and yields the sets A, B and C (Lines 7-9), respectively. For each point $p_i \in A$ we compute $\epsilon^k_{P'}(p_i)$

analogously to $\epsilon^k_{P'}(p_{n+1})$, which takes $O(n)$ each. Then we adjust *sum* accordingly ($O(1)$, Line 12). The sets A and B are handled in an analogous way, using $\epsilon^k_{X'}(x_i)$ and $\epsilon^k_{Y'}(y_i)$, respectively, instead (Lines 13-18). Note that these distances can be computed in time $O(k \cdot \log n)$ each, instead of $O(n)$, analogous to $\epsilon^k_{X'}(x_{n+1})$ and $\epsilon^k_{Y'}(y_{n+1})$.

Computational Complexity The total runtime for inserting a point into our structure therefore is in $O(k \cdot n + |A| \cdot n + (|B| + |C|) \cdot k \cdot \log n)$. In the following theorem we show that $|A|$,$|B|$ and $|C|$ are in $O(k)$, because there are at most $8 \cdot k$ points for which p_{n+1} is one of the k nearest neighbors. Consequently, our insertion time is in $O(k \cdot n + k^2 \cdot \log n)$. Since k is suggested to be a small constant, e.g., less than 10, in the literature, we can assume k to be constant. This means that an insertion is in $O(n)$. This results in the total time complexity of $O(n)$. Because deleting a point changes the estimate analogously, we can use an analogous algorithm with the same complexity, i.e., $O(n)$.

Theorem 4.13. *Let $P \subseteq \mathbb{R}^2$ be a set of points. For any point $p \in P$ there exist at most $8k$ points $q \in P$ such that p is one of the k nearest neighbors of q using the maximum norm.*

Proof. Let $p = (x, y) \in P$ be a point and $Q \subseteq P$ be the set of points such that for each point $q \in Q$, p is one of the k nearest neighbors of q. We separate Q into eight sets based on their relative location to p, as illustrated in Figure 4.3a. There are four axis-aligned rays $Q_L, Q_R, Q_U, Q_D \subseteq Q$ centered at p such that points on any of these rays share one component with p and differ in the other one. Additionally, there are four quadrants $Q_{RU}, Q_{LU}, Q_{LD}, Q_{RD} \subseteq Q$ centered at p excluding the axis-aligned rays. Because p cannot be its own nearest neighbor, these eight sets partition Q. To prove the lemma we proceed to show that each of these eight sets contains at most k points.

Let $r = (x_r, y_r)$ be the most distant point to p in the axis-aligned ray Q_R, that is, $|x - x_r| = \max_{(x_i,y_i)\in Q_R} |x - x_i|$. Then all other points in Q_R are on the line between p and r and thus closer to r than p. This means that Q_R cannot contain more than k points, because p would not be a nearest neighbor of r otherwise. By symmetry, this result also holds for the sets Q_L, Q_U, Q_D.

Similarly, Let $r = (x_r, y_r)$ be the most distant point to p in the quadrant Q_{RU} and let Δ be that distance. More formally, it is

$$\Delta = \max(|x - x_r|, |y - y_r|) = \max_{(x_i,y_i)\in Q_{RU}} \max(|x - x_i|, |y - y_i|). \tag{4.4}$$

An exemplary illustration can be found in Figure 4.3b with the set $Q_{RU} = \{s, r\}$. For any other point $q_i = (x_i, y_i) \in Q_{RU}$ with $q_i \neq r$ it is $x < x_i \leq x + \Delta$ and

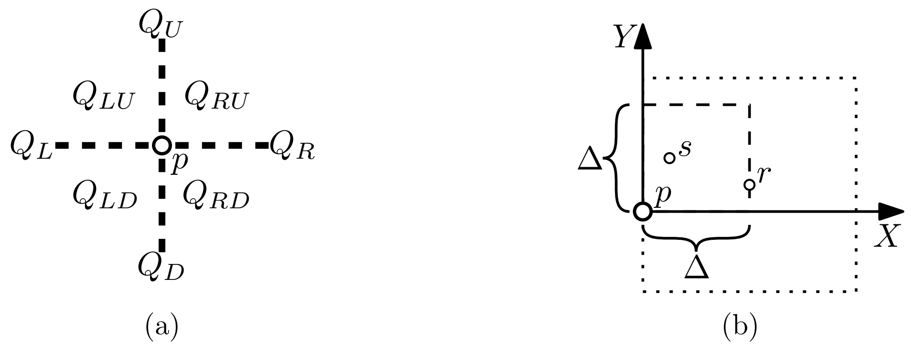


Figure 4.3: (a) The partioning of Q in Theorem 4.13 and (b) an example for Q_{RU}.

$y < y_i \leq y + \Delta$, because r is the point most distant to p. Figure 4.3b illustrates this by delimiting the area in which all points of Q_{RU} lie with dashed lines. Because of $x_r > x$ and $y_r > y$, it follows that $|x_r - x_i| < \Delta$ and $|y_r - y_i| < \Delta$. This means that q_i is a nearest neighbor of r. Figure 4.3b shows this by highlighting the area of nearest neighbors of r with dotted lines. Analogously to the axis-aligned rays, Q_{RU} cannot contain more than k points, because p would not be a nearest neighbor of r otherwise. By symmetry, this result also holds for the sets Q_{LU}, Q_{LD}, Q_{RD}. $\square$

As context for the update time of $O(n)$, Theorem 4.3 proves that any algorithm requires time in $\Omega(n \log n)$ to compute the 3KL from scratch. As a result, updating an estimate using DEMI is asymptotically faster than recomputing it, independently of the method used. In Section 4.2.2 we show how the time for updates on the 3KL can be improved even further. However, we first discuss how we use the same approach to update KSG estimates.

Updating KSG Estimates

Now, we describe how we achieve the same results, that is linear space and linear time for updates, using KSG estimates instead of 3KL estimates. As with the 3KL, we decompose the KSG estimate into $\widehat{I_{KSG}} = base + \frac{sum}{|P|}$. Comparing Equation 2.19 and Equation 2.23, it follows that *base* and *sum* need to maintain different values when maintaining 3KL or KSG estimates. The different equation for *base*, that is $base = \psi(|P|) + \psi(k) - \frac{1}{k}$ instead of $base = \psi(|P|) - \psi(k)$, does not have any influence on the overall procedure. However, the change from $sum = \sum_{i=1}^{n} \log\left(\frac{\epsilon_X^k(x_i) \cdot \epsilon_Y^k(y_i)}{(\epsilon_P^k(p_i))^2}\right)$ to $sum = -\sum_{i=1}^{n} \psi(MC_x^k(p_i)) + \psi(MC_y^k(p_i))$ has stronger implications. Most notably, we do not require explicit nearest-neighbor distances per point but need marginal counts. We need to update a marginal count $MC_x^k(p_i)$ if and only if the nearest neighbors of p_i in P changes, or a point (x, y) with $|x - x_i| \leq \delta_x^k(p_i)$ is in-

Data Structure 4.3: DEMI-KSG

```
struct {
    real $x, y$
    real $\epsilon_P^k, \delta_x^k, \delta_y^k$
    int $MC_x^k, MC_y^k$
} $DemiPointKSG$;

struct {
    DemiPointKSG[ ] $P_D$
    BST<DemiPointKSG*> $T_x, T_y$
    real $base, sum$
} $state$;
```

serted or deleted. As a consequence, per point p_i we do not store $\epsilon_X^k(x_i)$ and $\epsilon_Y^k(y_i)$ but the distances to the furthest x- and y-values among the k nearest neighbors in P, i.e., $\delta_x^k(p_i)$ and $\delta_y^k(p_i)$. Additionally we track the marginal counts $MC_x^k(p_i)$ and $MC_y^k(p_i)$. These slight changes are displayed in Data Structure 4.3. Furthermore, this means that we still store a constant amount of information per point, and the space complexity of the data structure remains $\Theta(n)$.

Updating the data structure follows the same principles as before, that is, we include the new point into the data structure and evaluate its impact on other marginal counts afterwards. In the following we describe the changes in specific steps between the update algorithm for 3KL estimates and KSG estimates, that is, Algorithm 4.2 and Algorithm 4.4.

Tracking marginal counts, instead of nearest-neighbor distances, per attribute allows for faster updates, because the counts only need increments and decrements ($O(1)$ each, Lines `16` and `18`), instead of recomputation. However, a change of nearest neighbors does also invalidate the marginal counts and requires computing them and correct adjustment of *sum* (Lines `11-14`). Computing marginal counts from scratch can be done with linear search ($O(n)$ each, Lines `2` and `13`).

Regarding the time complexity of Algorithm 4.4, it is important to note that B and C are not sets of points with changed nearest neighbors. As a consequence, only the size of A has an upper bound of $8 \cdot k$ by Theorem 4.13. In the worst case, B and C contain all points, that is, $|B| \leq n$ and $|C| \leq n$. The total time complexity therefore is $O(n + |A| \cdot n) = O(k \cdot n)$. As before, k is taken as constant, which yields the time complexity $O(n)$. This is asymptotically faster than recomputing the estimate by Theorem 4.6.

```
Algorithm 4.4: Insert-KSG($S_P, p_{n+1}$)
1  Compute $\delta_x^k(p_{n+1}), \delta_y^k(p_{n+1})$ and $\epsilon_{P'}^k(p_{n+1})$          $O(n)$
2  Compute $MC_x^k(p_{n+1})$ and $MC_y^k(p_{n+1})$                                     $O(n)$
3  Insert $p_{n+1}$ into $P_D$                                                        $O(1)$
4  Reference $p_{n+1}$ in $T_x$, $T_y$                                                $O(\log n)$
5  $base \leftarrow base + \frac{1}{n}$                                               $O(1)$
6  $sum \leftarrow sum - \psi(MC_x^k(p_{n+1})) - \psi(MC_y^k(p_{n+1}))$                $O(1)$
7  $A \leftarrow \{p_i \in P\colon \max(|x_i - x_{n+1}|, |y_i - y_{n+1}|) < \epsilon_P^k(p_i)\}$   $O(n)$
8  $B \leftarrow \{p_i \in P\colon |x_i - x_{n+1}| < \delta_x^k(p_i)\}$                $O(n)$
9  $C \leftarrow \{p_i \in P\colon |y_i - y_{n+1}| < \delta_y^k(p_i)\}$                $O(n)$
10 forall $p_i \in A$ do
11     $sum \leftarrow sum + \psi(MC_x^k(p_i)) + \psi(MC_y^k(p_i))$                    $O(|A|)$
12     Compute $\delta_x^k(p_i), \delta_y^k(p_i)$ and $\epsilon_{P'}^k(p_i)$          $O(|A| \cdot n)$
13     Compute $MC_x^k(p_i)$ and $MC_y^k(p_i)$                                         $O(|A| \cdot n)$
14     $sum \leftarrow sum - \psi(MC_x^k(p_i)) - \psi(MC_y^k(p_i))$                    $O(|A|)$
15 forall $p_i \in B$ do
16     $sum \leftarrow sum - \frac{1}{MC_x^k(p_i)}$                                    $O(|B|)$
17     $MC_x^k(p_i) \leftarrow MC_x^k(p_i) + 1$                                        $O(|B|)$
18 forall $p_i \in C$ do
19     $sum \leftarrow sum - \frac{1}{MC_y^k(p_i)}$                                    $O(|C|)$
20     $MC_y^k(p_i) \leftarrow MC_y^k(p_i) + 1$                                        $O(|C|)$
```

4.2.2 Polylogarithmic Updates

Because DEMI relies only on simple algorithms like linear search and binary search trees during insertions and deletions, faster solutions might exist. In this section we determine which parts of our insertion algorithm have a high computational cost and present solutions for these tasks. There are two factors that lead to the linear time complexity of Algorithm 4.2.

1. Computing the nearest neighbors with linear search

2. Finding the points whose nearest neighbors changed by linear search

Computing the Nearest Neighbors Computing the k nearest neighbors of a point is a classic problem of computational geometry, which has received a lot of research. While there exist many solutions, most of them are built for static data and are not compatible with the incremental changes in dynamic data. But there also exist solutions that allow for insertions and deletions. Chan [Cha06] proposed a dynamic data structure that computes nearest neighbors in two-dimensional spaces with sub-linear times for insertion, deletion and queries. However, the computational complexity of deletions is $O(\log^6 n)$, which is quite high. Kapoor and Smid [KS96] provide an alternative based on dynamic range trees [Wil85]. With dynamic fractional cascading [MN90] the time complexities for insertions, deletions and querying the nearest neighbor of a point are in $O(\log n \log \log n)$. To query two nearest neighbors, we can query one nearest neighbor, delete this point from the tree, query the new nearest neighbor and insert the deleted point. Querying the k nearest neighbors can thus easily be achieved through a sequence of k queries, $k-1$ deletions, and $k-1$ insertions, with total time in $O(k \cdot \log n \log \log n)$.

Finding Points with Changed Nearest Neighbors Finding all points whose nearest neighbors have changed is also a geometric problem, that is, finding the reverse nearest neighbors of the inserted or deleted point. For each point $p = (x, y)$ with nearest-neighbor distance ϵ, all nearest neighbors of p (using the maximum norm) are within the square $[x-\epsilon, x+\epsilon] \times [y-\epsilon, y+\epsilon] \subseteq \mathbb{R}^2$. To find all points whose nearest neighbors contain a point p', the task is to determine which squares contain p'. One data structure to solve this problem is the segment tree by Bentley [Ben77]. The technique of dynamic fractional cascading is also applicable for segment trees [MN90] and yields the time complexities for insertions and deletions in $O(\log n \log \log n)$. Queries require time in $O(\log n \log \log n + m)$, where m is the number of squares returned.

Improving DEMI To achieve sub-linear time complexity for updates, we integrate a two-dimensional dynamic range tree and a two-dimensional dynamic segment tree into DEMI. We call this the augmented version of DEMI (ADEMI). The insertion algorithm is nearly identical to Algorithm 4.2, except for changes in time complexities and insertions and deletions to the integrated tree structures. The full data structure and insert algorithm are listed in Data Structure 4.5 and Algorithm 4.6. For brevity, we only mention the changes relative to Algorithm 4.2 here, indicating line numbers of the corresponding lines in both algorithms as `Line [Algorithm 4.2]/[Algorithm 4.6]` Using the dynamic range tree, Line `1/1` requires only time in $O(k \cdot \log n \log \log n)$, and Line `11/14` requires time in $O(|A| \cdot k \cdot \log n \log \log n)$. Using the dynamic segment tree, Line `7/9` can be done in time $O(\log n \log \log n + |A|)$. Additionally, B and C can only contain elements that are at most k positions before and after x_{n+1} and y_{n+1} in T_x and T_y, respectively. Consequently, Lines `8-9/10-11` can also be done using the binary search trees in time $O(k \cdot \log n)$.

Data Structure 4.5: ADEMI

```
struct {
    real $x, y$
    real $\epsilon^k_P, \epsilon^k_X, \epsilon^k_Y$
} $DemiPoint$;

struct {
    DemiPoint[ ] $P_D$
    BST<DemiPoint*> $T_x, T_y$
    real $base, sum$
    2D dynamic range tree $T_{range}$
    2D dynamic segment tree $T_{seg}$
} $state$;
```

Algorithm 4.6: ADEMI-Insert(S_P, p_{n+1})

```
1  Compute $\epsilon^k_{P'}(p_{n+1})$                                                      $O(k \cdot \log n \log\log n)$
2  Compute $\epsilon^k_{X'}(x_{n+1})$ and $\epsilon^k_{Y'}(y_{n+1})$                       $O(\log n)$
3  Insert $p_{n+1}$ into $P_D$                                                             $O(1)$
4  Reference $p_{n+1}$ in $T_x$, $T_y$                                                     $O(\log n)$
5  Insert $p_{n+1}$ into $T_{range}$                                                       $O(\log n \log\log n)$
6  Insert $square(p_{n+1}, P')$ into $T_{seg}$                                             $O(\log n \log\log n)$
7  $base \leftarrow base + \frac{1}{n}$                                                    $O(1)$
8  $sum \leftarrow sum + \log\left(\frac{\epsilon^k_{X'}(x_{n+1}) \cdot \epsilon^k_{Y'}(y_{n+1})}{(\epsilon^k_{P'}(p_{n+1}))^2}\right)$   $O(1)$
9  $A \leftarrow \{p_i \in P\colon \max(|x_i - x_{n+1}|, |y_i - y_{n+1}|) < \epsilon^k_P(p_i)\}$
                                                                                           $O(\log n \log\log n + |A|)$
10 $B \leftarrow \{p_i \in P\colon |x_i - x_{n+1}| < \epsilon^k_X(x_i)\}$                  $O(k \cdot \log n)$
11 $C \leftarrow \{p_i \in P\colon |y_i - y_{n+1}| < \epsilon^k_Y(y_i)\}$                  $O(k \cdot \log n)$
12 forall $p_i \in A$ do
13     Delete $square(p_i, P)$ from $T_{seg}$                                              $O(|A| \cdot \log n \log\log n)$
14     Compute $\epsilon^k_{P'}(p_i)$                                                      $O(|A| \cdot k \cdot \log n \log\log n)$
15     Insert $square(p_i, P')$ into $T_{seg}$                                             $O(|A| \cdot \log n \log\log n)$
16     $sum \leftarrow sum + \log((\epsilon^k_P(p_i))^2) - \log((\epsilon^k_{P'}(p_i))^2)$  $O(|A|)$
17 forall $p_i \in B$ do
18     Compute $\epsilon^k_{X'}(x_i)$                                                      $O(|B| \cdot k \cdot \log n)$
19     $sum \leftarrow sum - \log(\epsilon^k_X(x_i)) + \log(\epsilon^k_{X'}(x_i))$         $O(|B|)$
20 forall $p_i \in C$ do
21     Compute $\epsilon^k_{Y'}(y_i)$                                                      $O(|C| \cdot k \cdot \log n)$
22     $sum \leftarrow sum - \log(\epsilon^k_Y(y_i)) + \log(\epsilon^k_{Y'}(y_i))$         $O(|C|)$
```

Additionally, we need to maintain the integrated tree structures. Specifically, we insert p_{n+1} into the dynamic range tree in `Line -/5` and insert the square of its nearest neighbors, that is,

$$\begin{aligned} \mathit{square}(p_{n+1}, P') = [&x_{n+1} - \epsilon^k_{P'}(p_{n+1}), x_{n+1} + \epsilon^k_{P'}(p_{n+1})] \\ &\times [y_{n+1} - \epsilon^k_{P'}(p_{n+1}), y_{n+1} + \epsilon^k_{P'}(p_{n+1})], \end{aligned} \tag{4.5}$$

into the dynamic segment tree in `Line -/6`. This square corresponds exactly with the area of nearest neighbors, which is illustrated with dashed lines during the introduction of the 3KL in Figure 2.2a. Both insertions require time in $O(\log n \log\log n)$. Finally, for each point $p_i \in A$ we delete its old square $\mathit{square}(p_i, P)$ from the dynamic segment tree in `Line -/13` and insert the new square $\mathit{square}(p_i, P')$ in `Line -/15`. This requires time in $O(|A| \cdot \log n \log\log n)$.

For an overview of the new time complexity, the complexity per operation of the insertion algorithm for ADEMI are shown in Algorithm 4.6. Because it is $|A|, |B|, |C| \in O(k)$, the total time complexity of an insertion is $O(k^2 \cdot \log n \cdot \log\log n)$. As before, k can be assumed to be a small constant, which leads to an insertion time of $O(\log n \log\log n)$. Deleting a point is completely analogous to insertions in (A)DEMI, and the used tree structures have the same complexity for insertions and deletions. Consequently, deletions in ADEMI also have a deletion time of $O(\log n \log\log n)$. Since the time complexity of queries is in $O(1)$, ADEMI solves problem 3KL-Update in time $O(\log n \log\log n)$. This means that ADEMI is nearly optimal, since its time complexity is only a factor $\log\log n$ higher than the lower bound from Corollary 4.12.

The drawback of ADEMI is an increased space complexity. The two-dimensional range tree and segment tree both have the space complexity $O(n \log n)$. Additionally, the improvements to the time complexity cannot be used when maintaining KSG estimates. This is because the number of points whose marginal counts change during an update has no bound lower than n. Also, the impact of incrementing or decrementing a marginal count on the overall estimate depends on the current count, which can be any value between k and n. As a consequence, it remains unclear whether any dynamic data structure can solve KSG-Update in sub-linear time, or whether there exists a stronger lower bound.

4.2.3 Experiments

In this section we empirically validate the estimation quality and time efficiency of our approach. To this end, we use data with known mutual information values and show that the 3KL converges to these values even with small samples. We also do so for the KSG. For brevity we only present the results for $k = 4$, since this value offers good rates of convergence for both the KSG and 3KL and follows the general recommendation of small values for k. Additionally, we compare the runtimes for maintaining 3KL estimates using ADEMI, DEMI and repeated estimation from scratch (REFS). For REFS we compute the 3KL, i.e., Equation 2.19, repeatedly with a state-of-the-art static approach [Eva08], i.e., using sorting and space-partitioning trees for nearest-neighbor searches. While we have already proven a clear hierarchy regarding their asymptotic scalability, the complexity classes neglect constant factors. So it remains interesting how their concrete runtimes compare.

Setup All approaches are implemented in C++ and compiled using the Gnu Compiler (version 5.4) with optimization (-O3) enabled. We use the non-commercial ALGLIB[1] implementation of KD-Trees as space-partitioning trees in REFS. We conduct all experiments on Ubuntu 16.04.2 LTS using a single core of an AMD Opteron™ Processor 6212 clocked at 2.6 GHz and 128GB RAM.

Data

For our evaluation, we use synthetic and real data sets. In particular, we use the dependent distributions with noise used for comparing mutual information estimators [KBG+07]. These distributions have a noise parameter σ_r, which we vary from 0.1 to 1.0. Thus, we use 10 distributions for each of these dependency types. Additionally, we use the uniform distributions used to compare mutual information with the maximal information coefficient [KA14] as well as independent uniform and normal distributions. As real data sets, we use sensor data of randomly charged and discharged batteries [BKD14] and time series of household power consumption [DG17b]. Monitoring mutual information on such data could be useful to monitor the condition of battery cells for maintenance or to infer knowledge about the behavior of the households inhabitants. In the following, we briefly describe the different distributions and data sets.

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 $p_i = (x_i, x_i^2 + r_i)$.

[1]ALGLIB (www.alglib.net), Sergey Bochkanov

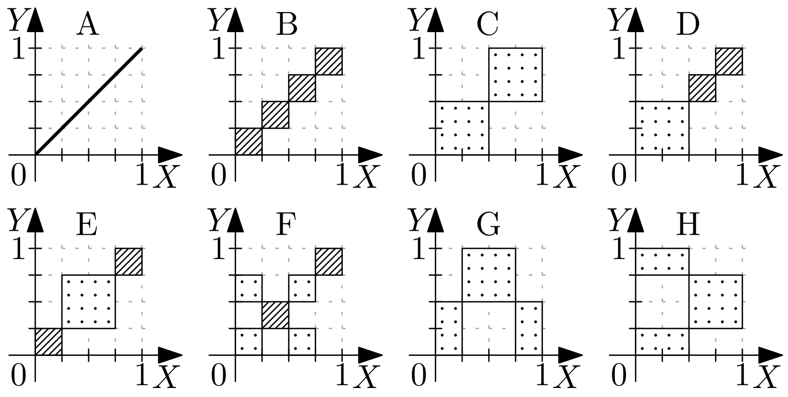


Figure 4.4: An overview of the uniform distributions used.

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 $p_i = (x_i, \sin(x_i) + r_i)$.

Chaotic This distribution uses the classical Hénon Map, that is,

$$\begin{aligned} h_{x_{i+1}} &= 1 - \alpha \cdot h_{x_i}^2 + h_{y_i} \\ h_{y_{i+1}} &= \beta \cdot h_{x_i}, \end{aligned}$$

with $\alpha = 1.4, \beta = 0.3$ and $(h_{x_0}, h_{y_0}) = (0, 0)$. For a point p_i we additionally independently draw noise r_{xi}, r_{yi} from the distribution $N(0, \sigma_r)$, where σ_r is the noise parameter. Each point $p_i \in P$ is then $p_i = (h_{x_i} + r_{xi}, h_{y_i} + r_{yi})$.

Uniform The uniform distributions A to H we use are illustrated in Figure 4.4. 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. The distribution A simply draws values v_i from U[0,1] and constructs the points $p_i = (v_i, v_i)$.

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

Battery Data This data set, available at the NASA Prognostics Center of Excellence [BKD14], monitors voltage, current and temperature of battery cells during random loads. We use the data corresponding to battery cell "RW9" and use each combination of the attributes as bivariate sample.

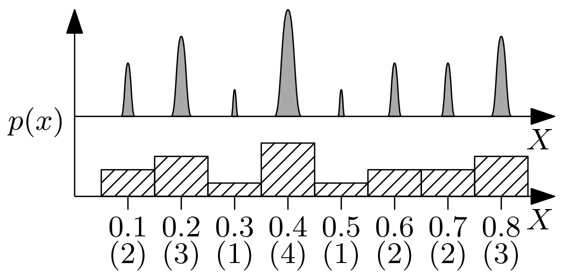


Figure 4.5: Avoiding duplicates in a sample by adding minimal noise (top) or filling the missing precision uniformly (bottom).

Power Consumption This data set, available at the UCI Machine Learning Repository [DG17b], monitors the power consumption of a household in France. We use each combination of *global active power*, *global reactive power* and *voltage* as a bivariate sample.

Data Precision

The nearest-neighbor based entropy estimator, and by consequence the 3KL and KSG, expects samples from continuous distributions and require samples without duplicate values. Because of the limited precision of the battery and the power consumption data, we add noise to the sample. Kraskov et al. also have observed this issue and recommend the addition of low intensity noise, e.g., a normal distribution with variance 10^{-10}, to eliminate duplicate points [KSG04]. However, we think that filling the missing precision with uniform noise is a better compensation for rounded or imprecise data. Figure 4.5 illustrates both approaches with the number of duplicates per value of an imprecise data set in parentheses. For our experiments we use the second approach.

Quality of Estimation

To evaluate the quality of estimation, we use all data sets with well defined mutual information values. That is, all synthetic data sets except the chaotic distributions, whose probability densities are unknown, and the uniform distribution A, whose mutual information is infinite. We use these distributions to evaluate the consistency and the rate of convergence of the KSG, 3KL and the estimator used by DIMID [BH17]. Specifically, we are interested in the difference between the estimated mutual information and true value for the distribution as well as the variance of estimates for samples of the same distribution. Since the behavior has turned out to be very homogeneous across the different distributions, we restrict our presentation to selected results.

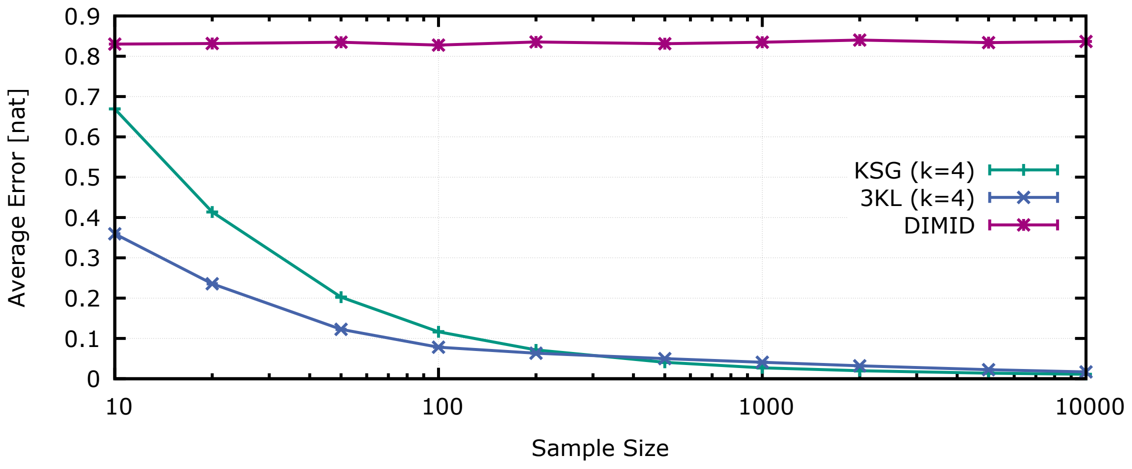


Figure 4.6: Average difference of estimates to true mutual information values depending on sample size.

Development with sample size For each distribution we created samples with sample sizes between 10 and 10000 and 1000 repeats per size. Figure 4.6 graphs the average difference between the estimate and the true mutual information value of the respective distribution. Additionally, Figure 4.7 shows the standard deviation of estimates of the same distribution and sample size, averaged across all distributions. One can see in these diagrams, that both the 3KL and the KSG converge quickly to the true values and have only small variance. In contrast, the approximate estimator in DIMID has a strong variance and difference. We think the reason is the random projection used by that estimator. It may retain enough information such that estimates are comparable to each other, as shown in their work [BH17]. However, we think that the projection loses too much information regarding the joint probability density to allow for good mutual information estimates.

Different dependency types We also studied whether the quality of estimation changes for different dependency types. As we have seen in the previous paragraph, both the 3KL and KSG are very consistent even with moderate sample sizes. As a result we use a small sample size, i.e., 100, to highlight differences. Figure 4.8 shows the average estimation error and standard deviation of estimates using 3KL, KSG and DIMID for each dependency type. While the variance of both KSG and 3KL are comparable for all dependency types, the difference to the true value is imbalanced for the KSG but not the 3KL. Unfortunately, we do not have any explanation for this difference. As before, we notice strong differences between the DIMID approximation and the results of KSG and 3KL.

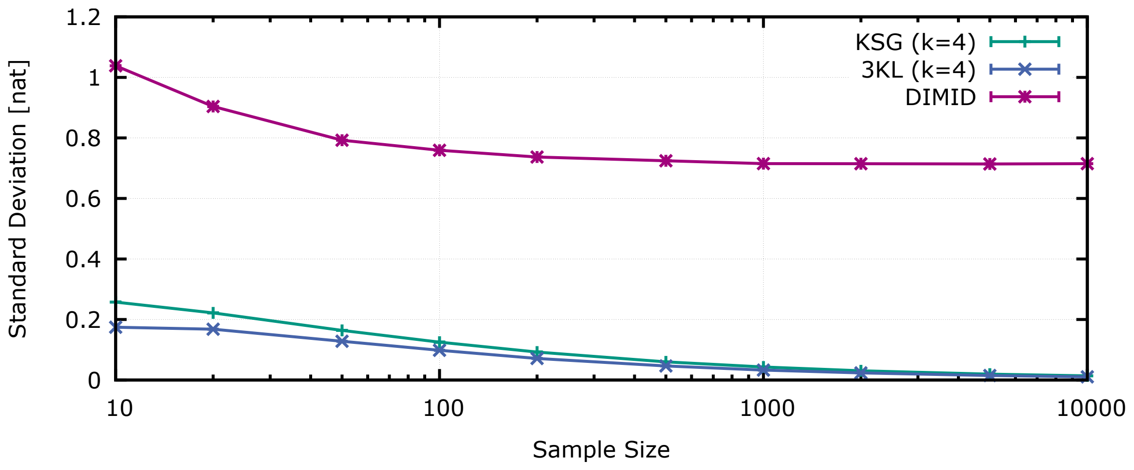


Figure 4.7: Standard deviation of estimates depending on sample size.

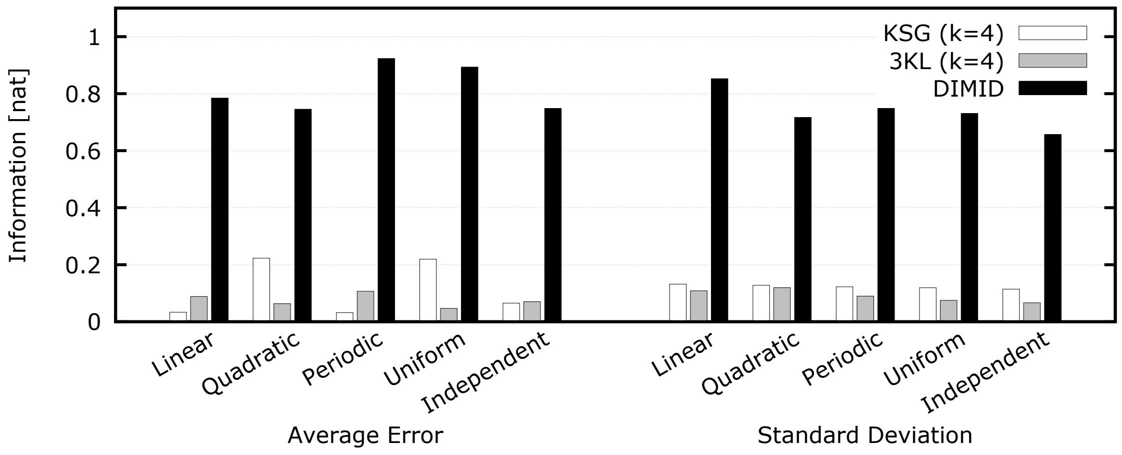


Figure 4.8: Average difference (left) and standard deviation (right) of estimates to true mutual information values on distribution type.

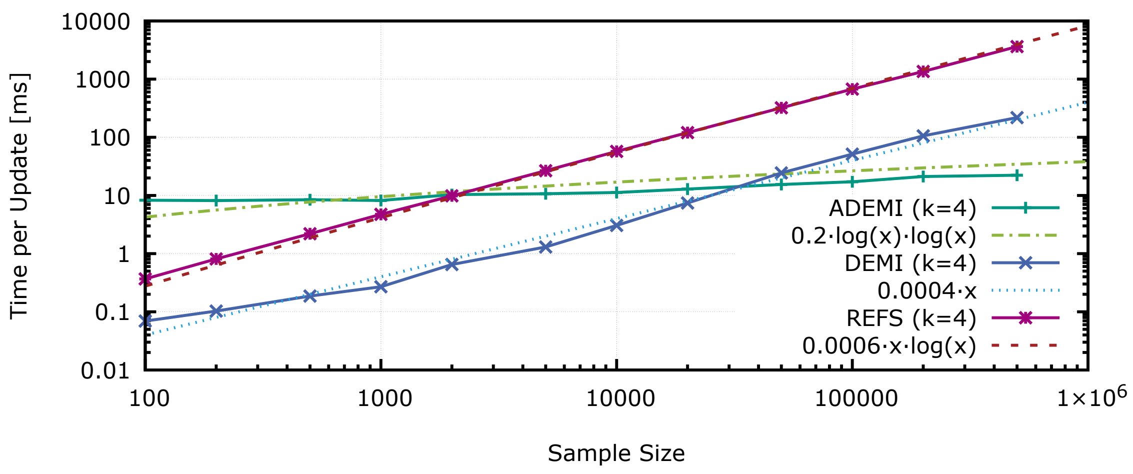


Figure 4.9: Average time for an update depending on sample size for the synthetic distributions.

Runtime Analysis

We have benchmarked runtimes of our data structures for all data sets. Because we are not aware of any competitor that offers good mutual information estimates on dynamic data, we compare our performance to naïve recomputation of the estimate when an update occurs. We compare the runtime to maintain 3KL estimates using DEMI and ADEMI as well as repeated recomputation (REFS). We use a slight simplification of the ADEMI trees, compared to the description in Section 4.2.2. Specifically, we did not implement dynamic fractional cascading and relied only on the technique of Willard [Wil85] for insertion and deletion of nodes. The reason is that dynamic fractional cascading provides a small asymptotic benefit, i.e., reducing a factor $\log n$ to $\log\log n$, but requires a lot of overhead. As a result, the structure labeled ADEMI in this section has insertion and deletion time in $O(\log^2 n)$ instead of $O(\log n \log\log n)$.

By design, both DEMI and ADEMI require only constant time for querying the current mutual information value, but require more time to update the data structure during insertions and deletions. The repeated static estimation REFS has inverse properties, i.e., constant time insertions and deletions but expensive queries. To provide a good overview we use the task of monitoring the mutual information of a changing data set of fixed size. That is, each update consists of deleting one point, inserting a different point and querying the current mutual information estimate. For these experiments we averaged the time required per update using 1000 updates per distribution and sample size.

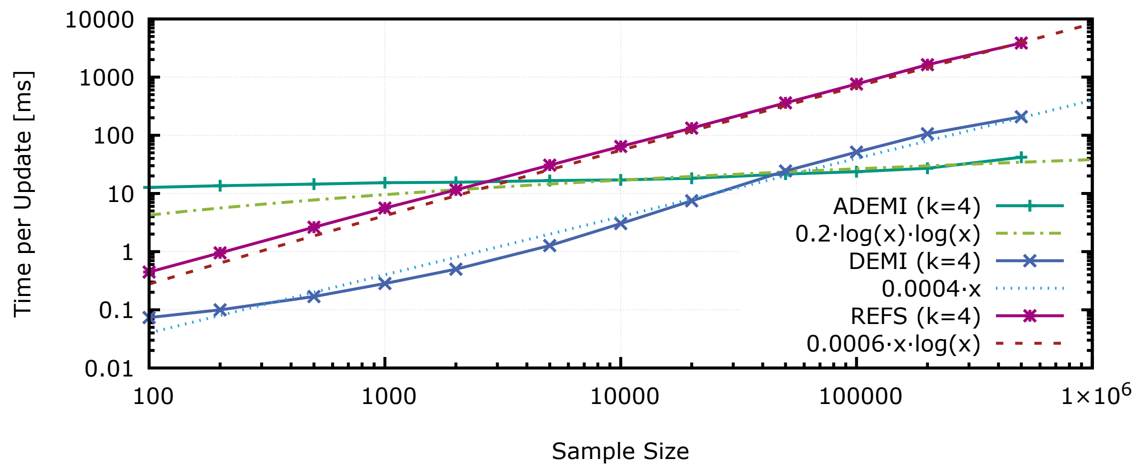


Figure 4.10: Average time for an update depending on sample size for the used real data set.

Figure 4.9 shows the average update time required across all synthetic distributions per sample size. The same graph based on the real data sets instead of the synthetic distributions is Figure 4.10. As expected, the time complexity of each approach translates directly to asymptotic scaling with sample sizes, that is, steepness of the curve in the double log plot. To highlight this, the graphs include different asymptotic functions with dashed and dotted lines. An interesting result is that ADEMI has by far the worst performance for small windows and by far the best performance for large ones. Our explanation is as follows: The maintenance of the range trees and even more so the segment trees is expensive, even if it scales favorably. For instance, when inserting a square into a two-dimensional segment tree, $8 \cdot (1 + \log n)$ nodes are created in the tree. This is a lot even for small n but does not increase significantly for large n.

Discussion

To summarize our experiments, we confirmed the estimation quality of 3KL and KSG across all dependency types tested. Additionally, we compared the performance of DEMI, ADEMI and REFS both on synthetic and real data. As expected, DEMI consistently outperforms REFS. The evaluation of ADEMI depends on the context and application. While it is slow for small window sizes, it barely slows down for larger sizes. On the one hand, this means that it is often recommendable to use DEMI if the data size is small. On the other hand, ADEMI can be used for very data-intensive tasks such as monitoring high-throughput streams. A problem with stream monitoring often is the multiplicative cost of high temporal resolutions: A stream with frequent items permits less time to process a new item, and a window with fixed time length contains more items. This leads to increased time to process a new item. As we have seen, the second factor is nearly negligible when using ADEMI.

Overall our experiments show that our techniques reduce the computation time for maintaining mutual information estimates on sliding windows by orders of magnitudes. As result, using mutual information for real-time monitoring becomes realistically feasible. For instance, if we consider a data stream with a frequency of 50 Hz, which is the frequency of the European power grid, real time processing cannot take longer than 20ms per update. Then, REFS can handle sliding windows up to roughly 4000 points (80 seconds worth of data) while DEMI and ADEMI can handle up to 50000 (16 mintues) and 200000 (70 minutes), respectively.

4.3 Iterative Mutual Information Estimation

This section is previously published in a mostly identical form at the International Conference on Extending Database Technology (EDBT '19) in [VB19]. There are only some adjustments to fit the form and consistency for this dissertation.

Now, we turn our attention from mutual information estimation on dynamic data to dynamic estimation. While the 3KL and KSG offer good estimation results, 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 any preliminary result if they are terminated prematurely. They also are unable to exploit 'easier' queries like whether the mutual information value is above a certain threshold but instead determined the value. These characteristics are also present with any other estimation technique for mutual information. However, 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 4.14. *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 mutual information value above the threshold MI_T?" often suffices. With conventional mutual information estimators, each pair either induces high computational costs, or results are uncertain because of low estimation quality.*

Scenario 4.15. *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 mutual information 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 mutual information with dynamic allocation of computation time. Ideally, such an estimator does not only offer preliminary results, but also indicate its remaining uncertainty. Figure 4.11 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

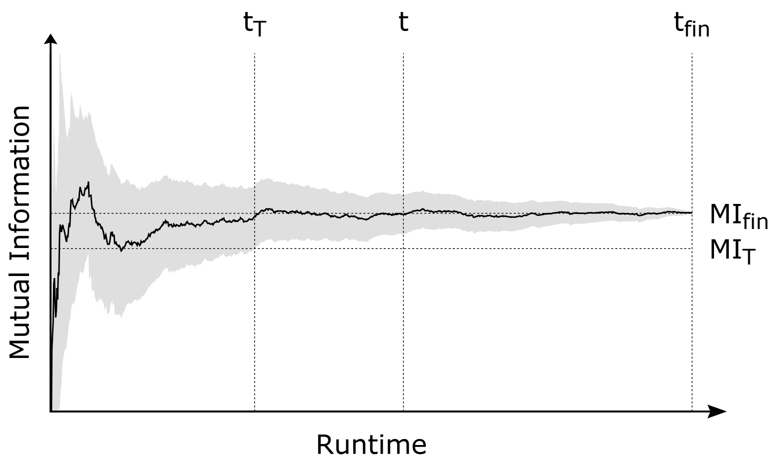


Figure 4.11: MI estimation with dynamic time allocation.

a threshold $\texttt{MI}_\texttt{T}$ with certainty, or he can use the preliminary result available at an arbitrary time $\texttt{t}$.

In this section, 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 mutual information do not exist.

The outline for the presentation of our iterative mutual information estimator (IMIE) is as follows. First we describe the core concept for IMIE, 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 conventional methods. Afterwards, we describe the statistical bounds that IMIE provides with its estimates. Due to our unconventional approach for the nearest-neighbor search and the dynamic nature of the estimation task, we present an extensive analysis of the time complexity with IMIE. Finally, we present the quality of estimation and practical run time of IMIE through extensive experiments with synthetic and real data.

4.3.1 Core Principles

The core concept of our approach lies with the fact that both the KSG and 3KL estimates consists of a fixed value and an the average of something that is computed for each point in the data. As a brief reminder, the equations (cf. Section 2.2) are

$$\widehat{I_{3KL}}(P) = \psi(n) - \psi(k) + \frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{\epsilon_X^k(x_i)\cdot\epsilon_Y^k(y_i)}{\left(\epsilon_P^k(p_i)\right)^2}\right), \tag{2.19 revisited}$$

$$\widehat{I_{KSG}}(P) = \psi(n) + \psi(k) - \frac{1}{k} - \frac{1}{n}\sum_{i=1}^{n} \psi\left(MC_x^k(p_i)\right) + \psi\left(MC_y^k(p_i)\right). \tag{2.23 revisited}$$

This means that that up to some offset that depends only on n and k, the estimates are exactly the arithmetic means of sets of numbers. While it is computationally expensive to compute each exact mean, which we have shown in Theorem 4.3 and Theorem 4.6, the idea is that we may not need the exact mean. Specifically, if we compute this mean not over all n indices but only some of them, we can use statistical properties. disregarding the offset for a brief moment, the exact estimates are means of random variables with finite populations. Then the mean of a subsample of such a population has an expected value equal to the exact mean and converges to the true mean with increasing subsample size. For brevity, we focus all further explanations and formalism to the KSG estimates. This is because all the following techniques work analogously and the KSG is the more well-known method.

To define IMIE, we introduce some notation in addition to the one from Section 2.2. For a point $p \in P$, we define the *pointwise estimate*

$$\Psi(p) = \psi\left(MC_x^k(p)\right) + \psi\left(MC_y^k(p)\right). \tag{4.6}$$

The set of all pointwise estimates is $\rho = \{\Psi(p_1), \ldots, \Psi(p_n)\}$. Seeing ρ as a finite population of size n with mean μ_ρ, Equation 2.23 can be rewritten as

$$\widehat{I_{KSG}}(P) = \psi(n) + \psi(k) - \frac{1}{k} - \mu_\rho. \tag{4.7}$$

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 $\widehat{I_{KSG}}(P)$,

$$\widehat{I}_\varrho(P) = \psi(n) + \psi(k) - \frac{1}{k} - \mu_\varrho. \tag{4.8}$$

The variance σ_ϱ^2 of our subsample serves as a quality indicator of this approximation, which we further discuss in Section 4.3.3. The idea of IMIE is to maintain a subsample ϱ and use $\widehat{I}_\varrho(P)$ to estimate $\widehat{I_{KSG}}(P)$. 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., $\widehat{I}_\varrho(P) = \widehat{I_{KSG}}(P)$.

Data Structure 4.7: IMIE

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

Algorithm 4.8: Init(P, k)

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

4.3.2 IMIE Data Structure

IMIE uses and stores P and k as well as some additional information listed in Data Structure 4.7. 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 [Wel62]. 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 we describe later in this section. 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 outlined in Algorithms 4.8 and 4.9. For brevity, these algorithms already show amortized time complexities per operation, which we derive and discuss properly in Section 4.3.4. Init ensures the proper state of Data Structure 4.7 before the first iteration, i.e.,

Algorithm 4.9: 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 4.10) $O(\sqrt{n})$
5  Compute $\delta_x^k(p)$, $\delta_y^k(p)$                               $O(1)$
6  Compute $MC_x^k(p)$, $MC_y^k(p)$                                       $O(\log n)$
7  $\Psi(p) \leftarrow \psi\big(MC_x^k(p)\big) + \psi\big(MC_y^k(p)\big)$  $O(1)$
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 $Var \leftarrow \frac{Var\cdot(m-1)+Diff_{\text{old}}\cdot Diff_{\text{new}}}{m}$   $O(1)$
```

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 we discuss in Section 4.3.4.

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. In the (m-1)-th iteration, we swap the index at position m of $Order_R$ with the index at a random position after $m-1$. 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 [Dur64], 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 [Wel62] to maintain mean and variance of a sample, in our case ϱ.

Example 4.16. *Disregarding the dashed lines for now, Figure 4.12 illustrates the state of Data Structure 4.7 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* $\text{Order}_R[0]$ *and* $\text{Order}_R[5]$. $\text{Order}_R[0]$ *now contains 5. This means that this iteration adds* $\Psi(P[5]) = \Psi(p_6)$ *to our implicit sample* ϱ. *We then determine its nearest neighbor* $1NN(p_6) = \{p_{15}\}$, *the distances* $\delta_x^1(p_6)$ *and* $\delta_y^1(p_6)$ *as well as the marginal counts* $MC_x^1(p_6) = 1$ *and* $MC_y^1(p_6) = 3$. *The dashed lines in Figure 4.12 illustrate the area of counted points in x and y-direction, respectively. It follows that* $\Psi(p_6) = \psi(1) + \psi(3) = 0.346$. *Substituting the appropriate variables, the*

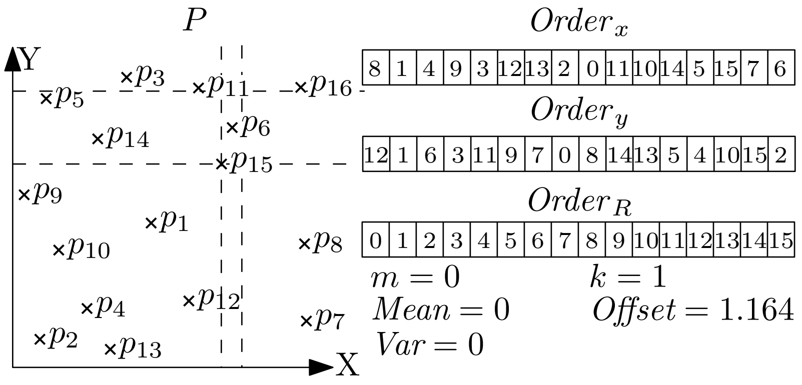


Figure 4.12: State of IMIE after initialization.

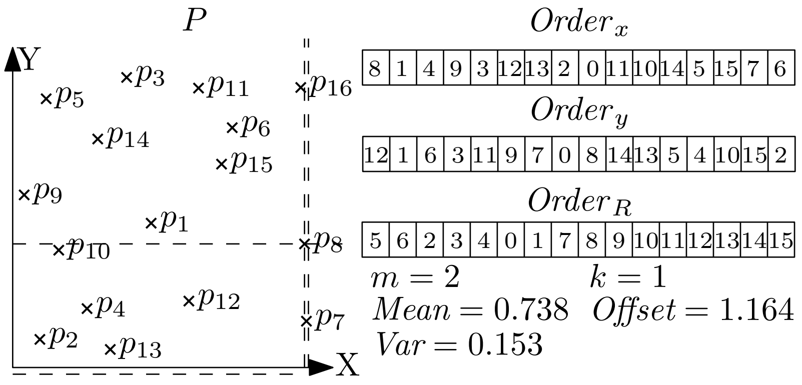


Figure 4.13: State of IMIE after two iterations ($\Psi(p_6)$ and $\Psi(p_7)$).

remaining values are set accordingly, i.e., $m = 0 + 1 = 1$, $\text{Mean} = 0 + \frac{0.346}{1} = 0.346$ *and* $\text{Var} = \frac{0 \cdot 0 + 0 \cdot 0.346}{1} = 0$. *The second iteration is analogous, drawing* $ID = 6$ *at random from* $\{1, \ldots, n-1\}$*, thus choosing* p_7*. Its nearest neighbor is* p_8*, and the marginal counts are* $MC_x^1(p_7) = 1$ *and* $MC_y^1(p_7) = 6$*, cf. the dashed lines in Figure 4.13. As a result, it is* $\Psi(p_7) = \psi(1) + \psi(6) = 1.129$*. Analogously to the first iteration, the remaining values are* $m = 1 + 1 = 2$, $\text{Mean} = 0.346 + \frac{0.783}{2} = 0.738$ *and* $\text{Var} = \frac{0 \cdot 1 + 0.783 \cdot 0.391}{2} = 0.153$*. Figure 4.13 graphs the state of Data Structure 4.7 after both iterations, and the new MI estimate is* $1.164 - 0.738 = 0.426$*.*

Lightweight 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 [KSG04, VH07] is using space-partitioning trees, which are optimal in terms of computational complexity by Corollary 4.8. 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 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 4.3.4). In the following, we elaborate on our approach, NNSearch.

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 4.10 for an algorithmic description of NNSearch.

Example 4.17. *Figure 4.14 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 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.*

Algorithm 4.10: 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          $\Delta_{x+} \leftarrow |x_p - x_{P[Order_x[i_x + loops]]}|$
9          UpdateNN($P[Order_x[i_x + loops]]$)
10     if $\Delta_{x-} < \delta_{\max}$ then
11         $\Delta_{x-} \leftarrow |x_p - x_{P[Order_x[i_x - loops]]}|$
12         UpdateNN($P[Order_x[i_x - loops]]$)
13     if $\Delta_{y+} < \delta_{\max}$ then
14         $\Delta_{y+} \leftarrow |y_p - y_{P[Order_y[i_y + loops]]}|$
15         UpdateNN($P[Order_y[i_y + loops]]$)
16     if $\Delta_{y-} < \delta_{\max}$ then
17         $\Delta_{y-} \leftarrow |y_p - y_{P[Order_y[i_y - loops]]}|$
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          $\delta_{\max} \leftarrow \max_{r \in NN} \|r - p\|_\infty$
```

4.3.3 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 4.7 and 4.8 give way to analogous assertions for $\widehat{I_{KSG}}(P)$.

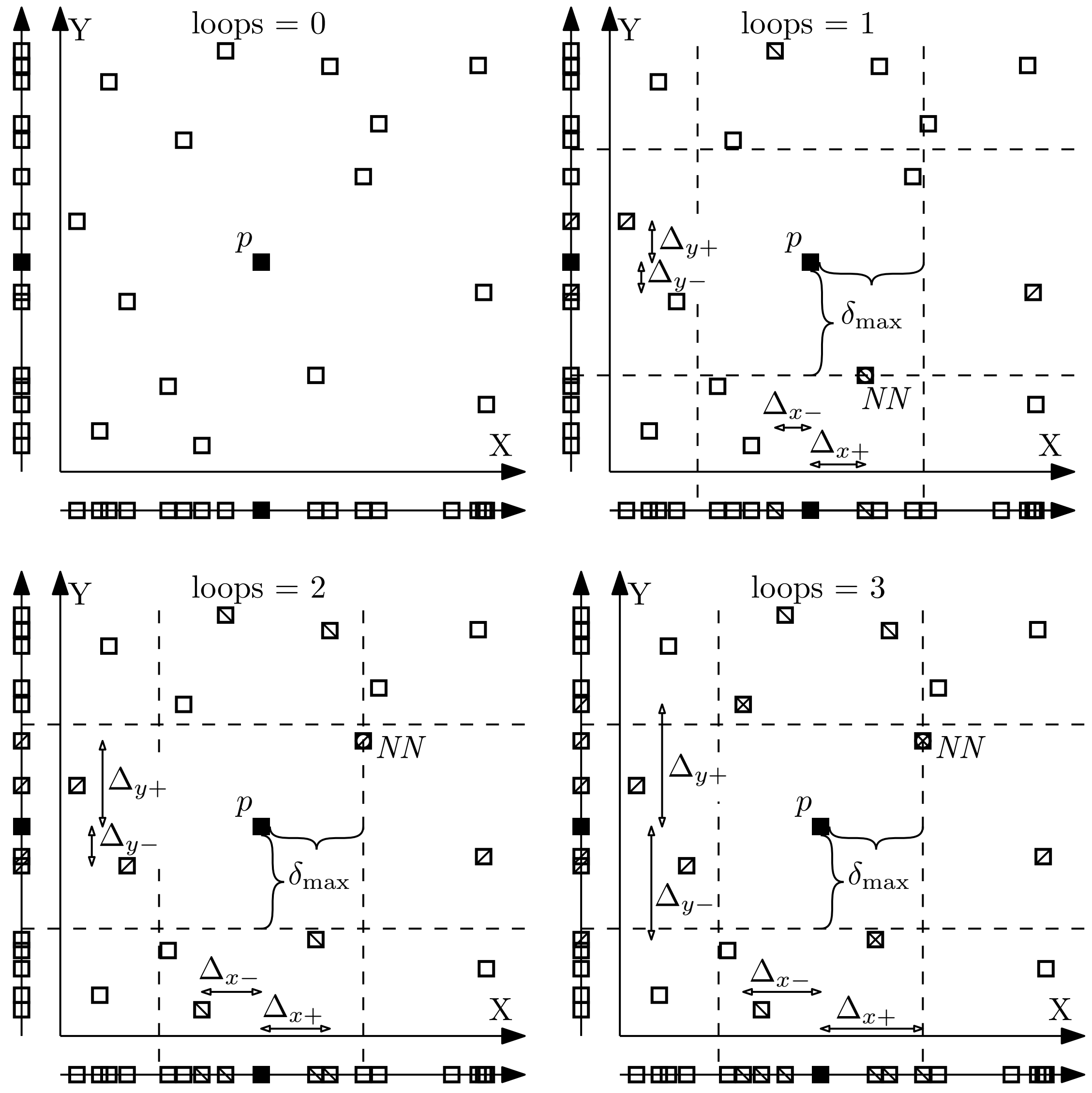


Figure 4.14: Illustration of Algorithm 4.10 for each loop.

Theorem 4.18 ([Ric06])**.** *Let ρ be a finite population of size n with mean μ_ρ and a variance σ^2_ρ. 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 $\sigma^2_{\mu_\varrho} = \frac{\sigma^2_\rho}{m}\left(\frac{n-m}{n-1}\right)$.*

Proof. See [Ric06]. □

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 [Ric06]. In other words, drawing a sample

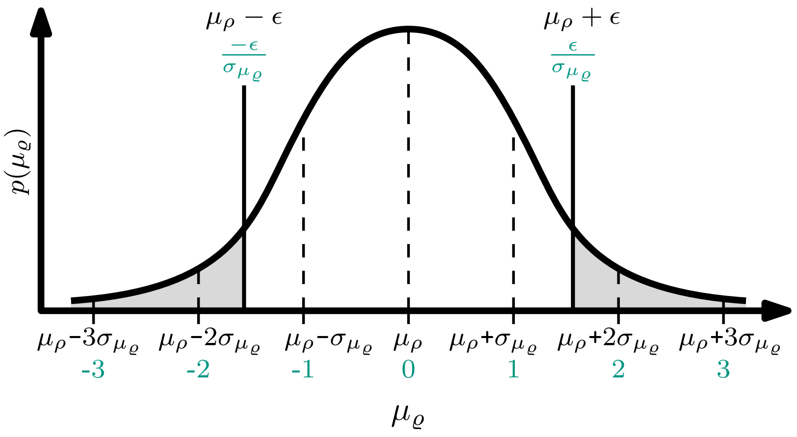


Figure 4.15: Illustration of the normal distributions $\mathcal{N}(\mu_\rho, \sigma_{\mu_\varrho})$ (upper labels) and $\mathcal{N}(0, 1)$ (lower labels).

of size m with a sample mean μ is as likely as drawing μ from $\mathcal{N}(\mu_\rho, \sigma_{\mu_\varrho})$. 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 4.15 and is formally described as

$$\Pr[|\mu_\varrho - \mu_\rho| \geq \epsilon] = 2 \cdot \Phi\left(\frac{-\epsilon}{\sigma_{\mu_\varrho}}\right). \tag{4.9}$$

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

$$Pr\left[\mu_\varrho - \left|\Phi^{-1}\left(\frac{\alpha}{2}\right)\right| \sigma_{\mu_\varrho} \leq \mu_\rho \leq \mu_\varrho + \left|\Phi^{-1}\left(\frac{\alpha}{2}\right)\right| \sigma_{\mu_\varrho}\right] = 1 - \alpha. \tag{4.10}$$

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 4.18, 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 [Ric06]. 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 4.19. *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 $\widehat{I_{KSG}}(P)$ 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 $\widehat{I_{KSG}}(P)$ 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 $Pr[\widehat{I_{KSG}}(P) < t] = 1 - (1 - 0.1)^2 = 0.19$.*

To account for this problem, we use the correction due to Šidák [Šid67]: 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 $\widehat{I_{KSG}}(P)$ is greater than a threshold t, using variables from IMIE.

$$\Pr[\widehat{I_{KSG}}(P) > t] \approx 1 - \left(1 - \Phi\left(\frac{\mathit{Offset} - \mathit{Mean} - t}{\sqrt{\frac{\mathit{Var}\cdot(|P|-m)}{(m-1)|P|}}}\right)\right)^{c} \tag{4.11}$$

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 4.2.3 however, the error rate never exceeds the bounds established in this section.

4.3.4 Complexity Analysis

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.

Nearest-Neighbor Search

We establish the time complexity of Algorithm 4.10. 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

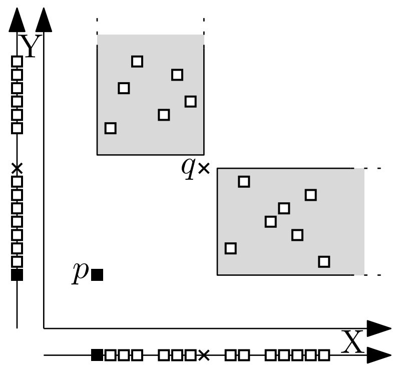


Figure 4.16: A degenerative case for NNSearch.

with binary search, in $O(\log n)$. However, $\mathbb{I}(p)$ is linear in n for the worst case. Figure 4.16 shows such a degenerative case, where all points except for p and q are equally distributed among 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 4.10. 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^p_x be the position of $Order_x$ containing the reference to a point p, i.e., $P[Order_x[Pos^p_x]] = p$. The set of points that access this position during NNSEARCH($\cdot$) is $R_x(p) = \{q \in P : Pos^p_x \in V_x(q)\}$. Pos^p_y and $R_y(p)$ are defined analogously using $Order_y$ instead of $Order_x$. By definition, it is

$$\sum_{p\in P} |V_x(p)| = \sum_{p\in P} |R_x(p)|, \tag{4.12}$$

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^p_x. In particular, there are only two points q^+ and q^- such that NNSEARCH(q^+) and NNSEARCH(q^-) access Pos^p_x during their first loop. These two points are the points corresponding to the neighboring positions of Pos^p_x, i.e., $q^+/q^- = P[Order_x[Pos^p_x \pm 1]]$. 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

$$2 \cdot \sum_{i=1}^{\frac{|R_x(p)|-1}{2}} i \leq 0 + 0 + 1 + 1 + \cdots + \left\lfloor \frac{|R_x(p)| - 1}{2} \right\rfloor \leq \sum_{q\in R_x(p)} \mathbb{I}(q). \tag{4.13}$$

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

$$\mathbb{I}(p) \leq |V_y(p)| \tag{4.14}$$

and with Equation 4.13, it is

$$2 \cdot \sum_{i=1}^{\frac{R_x(p)-1}{2}} i \leq \sum_{q \in R_x(p)} |V_y(q)|. \tag{4.15}$$

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.

Lemma 4.20. *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 4.17a. 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, we assume $\max_{r,s \in R_{RU}} |x_r - x_s| \geq \max_{r,s \in R_{RU}} |y_r - y_s|$. 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 4.17b, 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 4.17b 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 $r_k \in R_x(p)$.

Conversely, in the the second case it is $\max_{r,s \in R_{RU}} |x_r - x_s| < \max_{r,s \in R_{RU}} |y_r - y_s|$. This 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

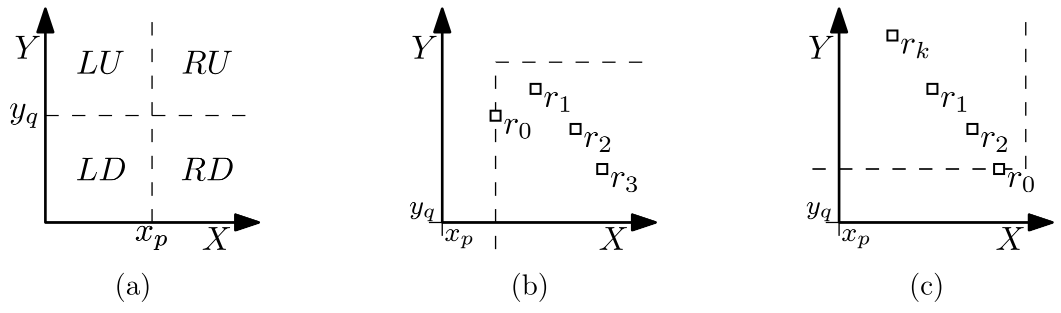


Figure 4.17: Illustration of arrangements in Lemma 4.20. (a) Partitioning of $\mathbb{R}^2$ based on (x_p, y_p). (b),(c) Two cases of layouts of RU.

all positions corresponding to other points in R_{RU} before Pos^q_y. Analogously, it is $\delta_{\max} = \Delta_{y-}$ after calling UPDATENN(r_0), as illustrated in Figure 4.17c. Thus NNSEARCH(r_k) does not access the position Pos^q_y, 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 bound for the total number of iterations performed by all searches.

Lemma 4.21. *For a set $P \subseteq \mathbb{R}^2$ 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 4.20, each position of $Order_y$ is accessed at most $4 \cdot k$ times by searches accessing one specific position of $Order_x$. Then for any $p \in P$, the total number of accesses to all n cells of $Order_y$ by NNSEARCH(q) with $q \in R_x(p)$ is at most $4 \cdot k \cdot n$. More formally, with Equation 4.15, it is for each $p \in P$

$$\begin{aligned}
4 \cdot k \cdot n &\geq \sum_{q \in R_x(p)} |V_y(q)| \geq 2 \cdot \sum_{i=1}^{\frac{|R_x(p)|-1}{2}} i \\
&= 2 \cdot \frac{\frac{|R_x(p)|-1}{2}\left(\frac{|R_x(p)|-1}{2} + 1\right)}{2} \geq \left(\frac{|R_x(p)| - 1}{2}\right)^2 \\
2\sqrt{k \cdot n} &\geq \frac{|R_x(p)| - 1}{2} \\
4 \cdot \sqrt{k \cdot n} &\geq |R_x(p)| - 1 \qquad (4.16)
\end{aligned}$$

Combining Equations 4.12, 4.14 and 4.16 yields

$$\sum_{p\in P} \mathbb{I}(p) \leq \sum_{p\in P} |V_x(p)| = \sum_{p\in P} |R_x(p)| \leq (4 \cdot \sqrt{k \cdot n} + 1) \cdot n. \qquad (4.17)$$

□

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

Theorem 4.22. NNSearch *has an amortized time complexity of* $O(\sqrt{n})$.

Data Structure

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 4.3.5 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 $MC_x^k(p)$ and $MC_y^k(p)$ (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 $x_{P[Order_x[i]]} \geq x_p - \delta_x^k(p)$. Similarly, let j be the largest integer in $\{0, \ldots, |P| - 1\}$ with $x_{P[Order_x[j]]} \leq x_p + \delta_x^k(p)$. Because $Order_x$ contains all points sorted by x-coordinate, it is $MC_x^k(p) = j - i$. The other marginal count $MC_y^k(p)$ is available analogously, using $Order_y, y_p$ and $\delta_y^k(p)$ instead. The other step is the nearest-neighbor search (Line `4`), which has an amortized time complexity of $O(\sqrt{n})$ by Theorem 4.22. 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 moderately slower in reaching the final result than the lower bound $\Omega(n \log n)$ for algorithms without preliminary results, cf. Theorem 4.6.

Application-Dependent Considerations

The initialization procedure Init presented in Section 4.3.2 explains the core concept and properties. Init has been defined 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 mutual information 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 4.14 and with streaming data as in Scenario 4.15. 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 mutual information 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 [KSG04, VH07]. 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 mutual information using all points whenever new data arrives. This means that most data points remain unchanged. When maintaining up-to-date mutual information estimates, IMIE can reuse the instance of Data Structure 4.7 used for the previous estimate instead of another initialization. Considering Data Structure 4.7, 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 $O(\log n)$.[2] 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 mutual information estimation on streams [ABR19, BH17, KMB15] as well as our methods DEMI and ADEMI impose a fixed computation time. This means they cannot easily adapt if items arrive faster.

Takeaway While the time complexity of INIT may appear prohibitively large in its basic form, we have demonstrated here 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 4.1 summarizes the impact

[2]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 4.1: Impact of optimization techniques for initializing IMIE for pairwise mutual information of d data streams.

of these techniques on the initialization of IMIE for pairwise mutual information estimation between d data streams.

4.3.5 Experimental Evaluation

Now, 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 2.23), 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 $O(n \log n)$. 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[3] implementation of KD-Trees for the KSG. We also use the non-commercial ALGLIB[3] 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.

[3]ALGLIB (`www.alglib.net`)

Data

In our experiments we use both synthetic and real-world data. As synthetic data, we use dependent distributions with noise used to compare mutual information estimators, see [KBG$^+$07], uniform distributions used to compare mutual information with the maximal coefficient, see [KA14], as well as independent uniform and normal distributions. These are the same synthetic distributions used when evaluating DEMI and ADEMI in Section 4.2.3. For distributions with a noise parameter σ_r, we vary σ_r between 0.1 and 1.0. As real data, we use smart meter readings from an industrial plant (HIPE) [BTV$^+$18], recorded smart phone sensors to recognize human activities (HAR) [DG17a], and physical quantities for condition monitoring of hydraulic systems (HYDRAULIC) [HPS15]. As proposed by the inventors of the KSG [KSG04], we prevent duplicate points in real-world data by adding noise with minimal intensity. In the following we provide some detail to these data sets.

HIPE This data set, available online[4], 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.

HAR This data set, available at the UCI ML repository[5], 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[6], 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.

Synthetic Benchmarks

We first evaluate the concrete runtimes of IMIE. While we have established in Section 4.3.4 that the time complexity is competitive, actual runtimes may have con-

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

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

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

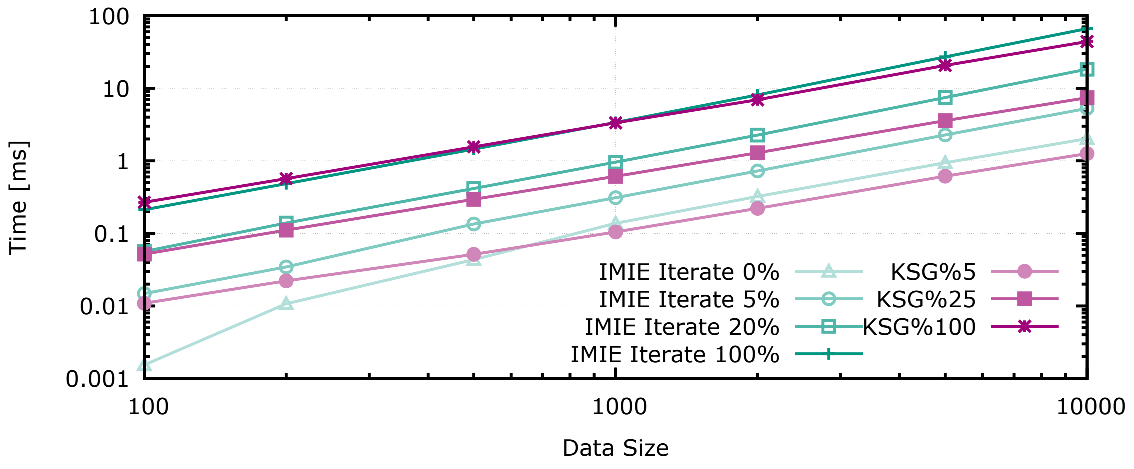


Figure 4.18: Average runtime depending on the data size for IMIE and subsampling variants.

stant 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 mutual information value of real data is unknown, we perform these experiments using 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 4.18 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 generally 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 higher time complexity.

Figure 4.19 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

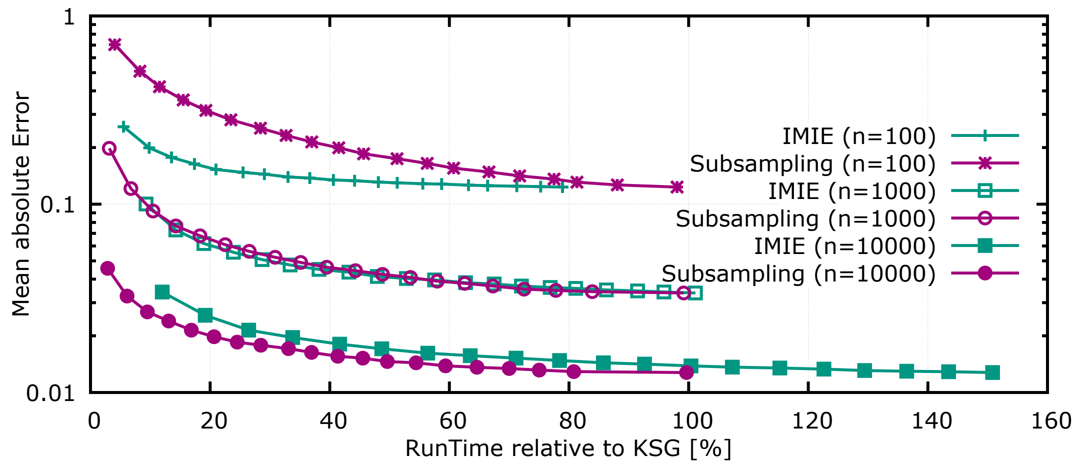


Figure 4.19: MAE of IMIE and subsampling depending on the runtime relative to KSG for the same data.

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 later in this section.

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 4.11. To be precise, one test is performed for $\widehat{I_{KSG}}(P) > \tau$, and one test is performed for $\widehat{I_{KSG}}(P) < \tau$. 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*.

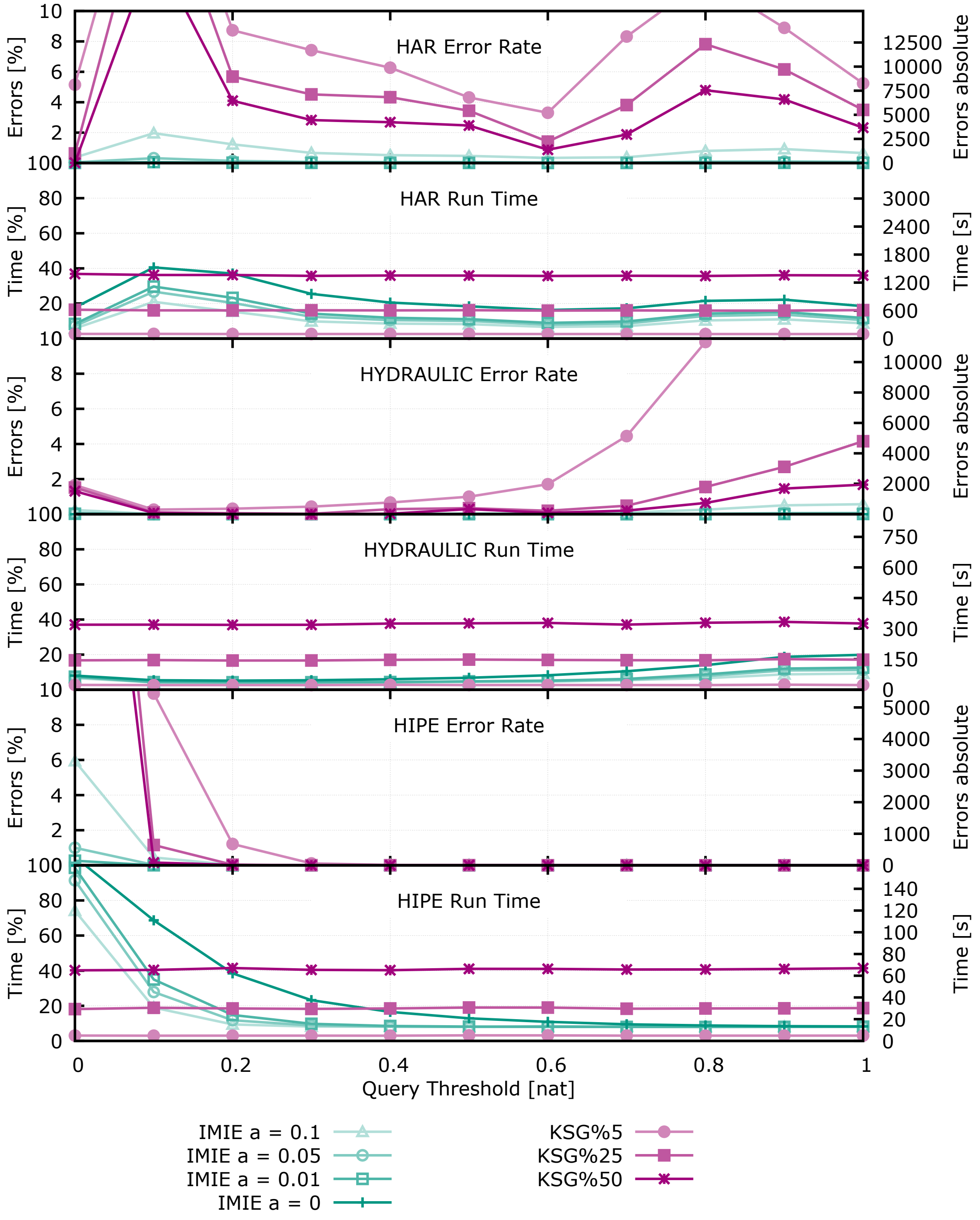


Figure 4.20: Time and error rate of IMIE and subsampling variants, depending on the chosen threshold τ.

Figure 4.20 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 mutual information 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 mutual information 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.[7]

Anytime Experiments

Now we test the performance of IMIE as anytime algorithm. In other words, the available time is not known beforehand. To 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 mutual information 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 4.18), we use 1000 estimators in parallel for each pair. One “iteration” then performs the next computation sequentially for each of these estimators.

[7]Technically there could still be errors due to rounding, numerical evaluation of Φ^{-1} and the approximation in Section 4.3.3. However, no such error has occurred in any of our experiments.

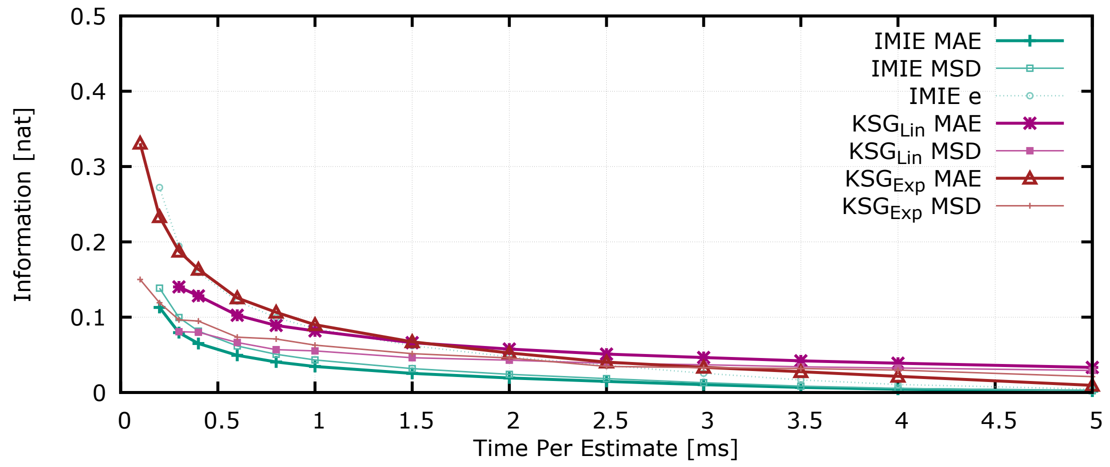


Figure 4.21: Mean absolute error (MAE) and mean standard deviation (MSD) of anytime approaches as well as the mean ϵ of IMIE.

Figure 4.21 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 by at most ϵ with a confidence of 95%. This value is obtained for each estimate using Equation 4.10. 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 4.19. 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.

Discussion

To summarize our experiments, 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.

4.4 Summary

Mutual information is a well-known measure to quantify complex dependencies. In this chapter, we studied the efficiency of mutual information estimation using the popular nearest-neighbor based estimation methods. Our results are threefold:

First, we established a lower bound for the computational complexity to compute these estimators. Using reductions to the IntegerElementDistinctenss problem, we have proven the lower bound $\Omega(n \log n,)$ for both the KSG as well as the 3KL estimator in the algebraic computation tree model. Since existing methods for computation achieve exactly this time complexity, they are asymptotically optimal and our lower bound is tight. In consequence, no new algorithm can be asymptotically faster without yielding different estimation results or assuming stronger computation models.

Second, we considered the specific scenario for mutual information estimation where an estimate should be maintained for a changing data set. As formalization, we considered this task as a problem to compute the new mutual information estimate when data is inserted or deleted. Additionally, some information may be stored between estimates. Even so, the consequence of our lower bound for regular estimation is that computing such updated estimates has a computational complexity of at least $\Omega(\log n)$. For this task, we then presented two dynamic data structures DEMI and ADEMI that maintain 3KL and KSG estimates. We have proven that both data structures require asymptotically less time to update their estimate than the lower bound to recompute it. Additionally, for maintenance of 3KL estimates, the time complexity of ADEMI is near optimal with a time complexity in $O(\log n \log \log n)$. Our experiments validate the formal time complexities for DEMI and ADEMI and show that DEMI is always at least a factor of five, usually an order of magnitude, faster than conventional recomputation. For ADEMI, our experiments have shown that the superior scalability imposes some fixed additional computations. Thus, ADEMI is the slowest approach for very small data sets and the fastest approach for large data sets.

Third, we considered iterative estimation of mutual information. 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, which exploits the structure of nearest-neighbor based estimation formulas for statistical approximations. While we focused on the KSG because it is more well-known, this approach would work analogously with the 3KL. Our approach converges towards the same result as the nearest-neighbor based estimator 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 the time complexity to compute the exact KSG estimate is $O(n\sqrt{n})$, which is only slightly larger than the lower bound to compute the KSG without any preliminary results. Based on experiments with synthetic and real data, we have shown that 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 mutual information in high-dimensional data.

These results offer additional insights for a better understanding of the computational aspects of nearest-neighbor based mutual information estimation. We offered both lower bounds for exact computation as well as new approaches that offer faster computation under specific circumstances such as dynamic data or results with small error chances. In consequence, our algorithms improve the feasibility to use mutual information in real-time applications.

Chapter 5

Explaining Dependencies

This chapter is previously published in a mostly identical form at the International Conference on Scientific and Statistical Database Management (SSDBM'19) in [VGBS19]. There are only some adjustments to fit the form and consistency for this dissertation.

Now, we consider dependencies from a different perspective. Instead of searching and quantifying dependencies, the focus of this chapter lies with understanding certain dependencies. To this end, we consider one attribute as designated *outcome* and the remaining attributes as the *explanatory* attributes. A common data analysis task is to determine how a chosen outcome attribute is affected by the explanatory attributes. That is, the question is not only whether there is a dependency and how strong it is, but also how this dependency manifests. Before building complex models and making data-driven decisions, data scientists therefore are often interested in exploring and summarizing the data. This motivates the problem of constructing human interpretable summaries for the effect of the explanatory attributes on the outcome attribute in a given dataset.

Prior work proposed *explanation tables* [GAG$^+$14, GFG$^+$18, FGS17] to solve this problem in a concise, interpretable and informative way, with informativeness defined as the ability to capture the distribution of the outcome attribute. However, only discrete explanatory attributes were supported. In contrast, large parts of the data collected and analyzed today, e.g., by the Internet of Things [AIM10] and by smart infrastructure [CNW$^+$12, FMXY12], are ordinal or numeric. Therefore, we argue that data summarization is particularly compelling and timely with ordinal and numeric explanatory attributes, such as time, temperature or locations. Numeric domains can be very large, which makes summarization techniques critical, but also technically challenging. To fill this gap and address these challenges, we propose a method for Lightweight Extraction of Numeric Summaries (LENS) in this chapter. The following example highlights the need and benefit of our approach.

Table 5.1: A fragment of a building `Occupancy` dataset

Day	Hour	Temp.	Humid.	Light	CO_2	Occupied
Mon	14	23.7	26.27	585.2	749.2	True
Mon	18	22.39	25	0	805.5	False
Wed	10	23.39	25.6	738	1042	True
Thu	15	22.5	27	469	1063	True
Fri	02	21	25	0	440	False
Sun	13	20.5	28.7	265	426.7	False
Tue	11	22.2	27.6	535.7	1137	False
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Example 5.1. *Consider the `Occupancy` dataset [CF16], an excerpt of which is shown in Table 5.1. The dataset includes sensor measurements from a smart building (time, temperature, humidity, light level, carbon dioxide level) as well as an outcome attribute denoting whether a given room was occupied at the time. Suppose a data scientist wants to understand how the sensor measurements inform occupancy status. Table 5.2 shows the corresponding explanation table [GAG+14]. Each row is a pattern that represents a subset of the data matching the given values of the explanatory attributes, with "*" matching all values. Each pattern also includes the count of matching records and the fraction of records within this subset having a true outcome. The first row indicates that 23 percent of all records correspond to occupied rooms. The second row, chosen to provide the most additional information about the distribution of the outcome attribute, suggests that rooms are not occupied when the light level is zero. The next two rows similarly suggest that rooms are not occupied on Saturdays and Sundays. In contrast, LENS produces the summary shown in Table 5.3. The use of value ranges allows LENS summaries to capture patterns such as weekdays ("Mon-Fri") or evening hours ("15-23"), which would have to be pieced together from multiple rows of an explanation table. For example, the second row indicates that 4911 records, i.e., 23 percent of the data, correspond to occupied rooms on weekdays when the light level is high, which would require many separate rows without ranges. Subsequent rows identify additional subsets whose outcome distribution diverges from the expectation such as occupied rooms during evening hours with lights on (which, again, would require a separate pattern for each hour without ranges).*

Table 5.2: An explanation table for the `Occupancy` dataset

Day	Hour	Light	CO_2	Count	Occupied	Correct
$*$	$*$	$*$	$*$	20560	True	23.1%
$*$	$*$	0	$*$	12772	True	0.01%
Sun	$*$	$*$	$*$	2880	True	0.00%
Sat	$*$	$*$	$*$	2880	True	0.00%
Thu	13	$*$	$*$	122	True	13.85%
Thu	14	$*$	$*$	118	True	41.48%

Table 5.3: An informative summary for the `Occupancy` dataset created by LENS

Day	Hour	Light	Count	Occupied	Correct
$*$	$*$	$*$	20560	True	23.1%
Mon–Fri	$*$	356–1697	4911	True	96.27%
Mon–Tue	$*$	428–536	2025	True	99.41%
$*$	15–23	429–576	1522	True	99.67%
Thu–Fri	9	$*$	240	True	99.58%

5.1 Fundamentals and Formalization

We now formalize informative summaries as a generalization of explanation tables [GAG$^+$14, GFG$^+$18, FGS17] to ordered attributes, patterns with intervals, and multi-valued outcomes. We then present our problem statement and a greedy framework for building informative summaries. The context for this task is somewhat different compared with mutual information so far. Most notably, this task considers an arbitrary number of attributes and has a stronger emphasis on the potential that the data is a multiset, i.e., the data contains duplicate tuples. As a result, we use some notation that is different from the previous chapter for the same high-level concepts. To avoid confusion, the following section introduces all notation used both for known and novel concepts, with Table 5.4 summarizing the notation.

5.1.1 Informative Summaries

Let $A_1, \ldots, A_d$ be a set of d explanatory attributes with data space $\mathcal{A} = A_1 \times \cdots \times A_d$ and let O be an *outcome* attribute. For each attribute domain $A_i = \{a_i^1, \ldots, a_i^{|A_i|}\}$, if A_i is ordered, we assume that a_i^j is before a_i^k in the ordering if and only if $j < k$. We assume finite domains, i.e., we restrict the domains of real values to the values present in the data. A dataset is a multiset of tuples over $\mathcal{A} \times O$. We represent it as a function $f : \mathcal{A} \times O \to \mathbb{N}$ that counts the number of tuples for specific

Table 5.4: Notation

Symbol	Meaning
A_i	Explanatory attribute
d	Number of explanatory attributes
$\mathcal{A}$	Data space formed by $A_1 \times \cdots \times A_d$
t	A tuple of explanatory attributes ($t \in \mathcal{A}$)
O	Outcome attribute
$o \in O$	An outcome value
$f(t, o)$	A dataset represented by tuple counts
$f(o)$	Count of tuples with outcome o
$f(t)$	Count of tuples t disregarding outcome
n	Total number of tuples
p	A pattern
$supp(p)$	Number of tuples matching p
$cnt(p, o)$	Number of tuples matching p with outcome o
$r(p, o)$	A rule formed by a pattern p and outcome o
S	An informative summary as set of rules
$P(o\|t)$	Probability for a tuple t to have outcome o
$P_S(o\|t)$	Maximum entropy estimate of $P(o\|t)$ by S
$g_S(p)$	Maximal gain by including any rule with p

combinations of attribute and outcome values. The total number of tuples in the dataset is $n = \sum_{t\in\mathcal{A}} \sum_{o\in O} f(t, o)$. For brevity, we overload f to sum over the omitted parameter, that is, $f(o) = \sum_{t\in\mathcal{A}} f(t, o)$ and $f(t) = \sum_{o\in O} f(t, o)$.

Patterns are compact specifications of subsets of $\mathcal{A}$. For each attribute A_i, a pattern p specifies a closed interval $[p_i, q_i]$ with $p_i, q_i \in \{1, \ldots, |A_i|\}$. Note that for attribute A_i without ordering, the only meaningful intervals are single values, $[p_i, p_i]$, and all values, $[1, |A_i|]$. A tuple $t = (t_1, \ldots, t_d) \in \mathcal{A}$ *matches* a pattern $p = ([p_1, q_1], \ldots, [p_d, q_d])$ if $a_i^{p_i} \leq t_i \leq a_i^{q_i}$ for all $i \in \{1, \ldots, d\}$. We also write $t \asymp p$ for "t matches p". Next, the *support* of p is the count of matching tuples regardless of their outcome: $supp(p) = \sum_{t\in\mathcal{A}\asymp p} f(t)$. Additionally, for each outcome value $o \in O$, $cnt(p, o) = \sum_{t\in\mathcal{A}\asymp p} f(t, o)$ is the count of matching tuples with this outcome. A rule $r(p, o)$ is a combination of a pattern p, outcome o, pattern support $supp(p)$ and outcome percentage $cnt(p, o)/supp(p)$. An informative summary S is a set of such rules.

We use the following notation for patterns. Intervals $[p_i, q_i]$ that cover all values, i.e., $p_i = 1$ and $q_i = |A_i|$, are called *wildcards* and are represented by "$*$". If an interval covers exactly one value, i.e. $p_i = q_i$, it is a *constant*, represented by $a_i^{p_i}$.

All other intervals include both endpoints: $[a_i^{p_i}, a_i^{q_i}]$. Additionally, we say a pattern is *simple* if it consists of only constants or wildcards.

Let $P(o|t)$ be the conditional distribution of outcome value o for a given combination of explanatory attribute values t. For a given dataset, we can calculate $P(o|t)$ empirically from f. Formally, for all $o \in O$ and $t \in \mathcal{A}$, $P(o|t) = \frac{f(t,o)}{f(t)}$ if $f(t) > 0$ and 0 otherwise. While the set of all these probability distributions is informative, it is too large for human interpretation, i.e., it is not concise. We consider informative summaries as compact representations of these conditional probabilities.

Following previous work on data summarization [GAG+14, FGS17, MTV11, Sar01], we use information theoretic methods to quantify informativeness. Let $\mathcal{P}_S = \{P_S(o|t) : o \in O, t \in \mathcal{A}\}$ be a model of the conditional probabilities based on the information contained in a summary S. By the maximum-entropy principle (cf. Section 2.3), the distribution with the fewest additional assumptions is the one with the highest entropy. In general, this is the most uniform distribution that agrees with the rules in S, which we called hints in Section 2.3. Formally, this is $\mathcal{P}_S$ that maximizes

$$H(\mathcal{P}_S) = -\sum_{t \in \mathcal{A}} \sum_{o \in O} \frac{f(t)}{n} P_S(o|t) \log \big(P_S(o|t)\big) \tag{5.1}$$

with the constraints that $0 \leq P_S(o|t) \leq 1$, and for all rules $r(p, o) \in S$, that $cnt(p, o) = \sum_{t \asymp p} f(t) \cdot P_S(o|t)$. Computing $\mathcal{P}_S$ is non-trivial as it has no closed form and requires numeric methods such as *improved iterative scaling*, which we detailed in Chapter 2 with Algorithm 2.1. To give some additional intuition on maximum entropy models in practice, the following example calculates $\mathcal{P}_S$ for an easier case which can be solved without improved iterative scaling.

Example 5.2. *Consider the `Occupancy` dataset from Table 5.1 and an informative summary S containing the first two patterns from Table 5.2, call them p_1 and p_2. Since p_1 matches all tuples, the maximum-entropy model for $\mathcal{P}_S$ distinguishes only between tuples that match p_2 and those that do not. In consequence, we have to find $x_1 = P_S(true|t)$ for $t \asymp p_2$ and $x_2 = P_S(true|t')$ for $t' \not\asymp p_2$. Since the outcome is binary, the probabilities are complementary, i.e., $P_S(false|t) = 1 - P_S(true|t)$ and $P_S(false|t') = 1 - P_S(true|t')$. Based on the values we know from S, the restrictions are $20560 \cdot 0.231 = 12772 \cdot x_1 + 7788 \cdot x_2$ and $12772 \cdot 0.01 = 12772 \cdot x_1$, which yields $x_1 = 0.01$ and $x_2 = 0.59$. Naturally, this becomes harder when there are more probabilities to determine than constraints, as induced by overlapping patterns.*

We use the Kullback-Leibler (KL) Divergence between $P(o|t)$ and $P_S(o|t)$ to measure the accuracy of S in estimating $P(o|t)$. For each tuple t, this divergence is defined as

$$KL_t\left(P||P_S\right) = \sum_{o \in O} P(o|t) \log \left(\frac{P(o|t)}{P_S(o|t)}\right). \tag{5.2}$$

Since this yields one divergence value per tuple, we aggregate the divergences and weigh them by tuple frequency to obtain the total error of a summary S:

$$div\left(P||P_S\right) = \sum_{t \in \mathcal{A}} f(t) KL_t \left(P||P_S\right). \tag{5.3}$$

The informativeness of a summary is measured as the reduction in error compared to only knowing the overall outcome distribution in the entire dataset. That is, we compare the divergence of the maximum-entropy estimate to a baseline model that uses only the total outcome distribution. For all $o \in O$ and $t \in \mathcal{A}$, the baseline is $P_B(o|t) = \frac{f(o)}{n}$. The information gain based on an informative summary S is then

$$gain(S) = div\left(P||P_B\right) - div\left(P||P_S\right). \tag{5.4}$$

Our formal problem statement is as follows:

Problem 5.3 (Informative Summarization Problem). *Given a dataset represented as function f counting entries for each attribute and outcome combination and a desired number of rules s, compute an informative summary S with $|S| = s$ that maximizes* gain(S).

5.1.2 Constructing Informative Summaries

Since informative summaries generalize explanation tables, the NP-hardness to construct optimal explanation tables [GAG+14] extends to optimal informative summaries. As a result, exact solutions are infeasible assuming $P \neq NP$. Instead, a common approach [GAG+14, FGS17, MTV11] for this and similar problems is a greedy approach, as shown in Algorithm 5.1. Starting with an empty summary S, the rules are selected one at a time until the desired number of rules s is reached, i.e., $|S| = s$[1]. For each selection, improved iterative scaling is performed to obtain the current model $\mathcal{P}_S$. Then, a set of candidate rules R is considered and a rule $r(p, o) \in R$ that maximizes $gain(S \cup \{r(p, o)\})$ is selected.

Since R can be very large, even if only simple patterns are considered, one way to speed up Algorithm 5.1 is to *prune* R. For example, the Flashlight [GAG+14] and SIRUM [FGS17] techniques perform sample-based pruning as follows. In each iteration of the while loop, a random sample is drawn from the dataset, and R is populated with only those simple patterns that match at least one sampled tuple. The intuition is that informative patterns are likely to have high support and therefore should match at least one sampled tuple.

Even after pruning the candidate rule set, Algorithm 5.1 may be too expensive because line 5 requires gain calculations for each candidate. For efficiency, prior work

[1] In the unlikely case that no rule yields additional gain, we stop the algorithm early.

Algorithm 5.1: Greedy Construction of Informative Summaries

```
1 $S \leftarrow \emptyset$
2 while $|S| < s$ do
3     $\mathcal{P}_S \leftarrow$ Improved Iterative Scaling($S$)
4     $R \leftarrow$ candidate rule set
5     $r(p, o) \leftarrow \arg\max_{q \in R} gain(S \cup \{q\})$
6     $S \leftarrow S \cup \{r(p, o)\}$
7 return $S$
```

uses the following approximation for binary outcomes [GAG+14, MTV11, Sar01]. Let $r(p, o)$ be a candidate rule and $S^+ = S \cup \{r(p, o)\}$. It is assumed that $P_{S^+}(o|t) \approx \frac{cnt(p,o)}{supp(p)}$ for $t \asymp p$ and $P_{S^+}(o|t) \approx P_S(o|t)$ otherwise. That is, when computing gain, it is assumed that adding $r(p, o)$ only refines the outcome distribution estimates for tuples matching p, without requiring recomputing the estimates for any other tuples (which would require improved iterative scaling). Additionally, in the current model based on S without the new candidate, all tuples matching p are given the same probability $P_S(o|t) \approx \frac{\sum_{t \asymp p} f(t) \cdot P_S(o|t)}{supp(p)}$. With this approximation, the improvement in gain by adding rule $r(p, o)$ is [GAG+14]:

$$\begin{aligned} gain(S^+) - gain(S) \approx\ & cnt(p, o) \cdot \log \left(\frac{cnt(p, o)}{\sum_{t \asymp p} f(t) \cdot P_S(o|t)} \right) \\ & + (supp(p) - cnt(p, o)) \cdot \log \left(\frac{supp(p) - cnt(p, o)}{\sum_{t \asymp p} f(t) \cdot (1 - P_S(o|t))} \right) \end{aligned} \tag{5.5}$$

Call the two possible outcome values o and o'. Note that the second term above accounts for the gain due to refining the probability estimates for the complementary outcome value o' because $supp(p) - cnt(p, o) = cnt(p, o')$ and $1 - P_S(o|t) = P_S(o'|t)$.

We now extend the above approximation to non-binary outcomes. A rule $r(p, o)$ describes the frequency of one particular outcome value o within its matching tuples. With binary outcomes, we can immediately infer the frequency of the other value o'. With non-binary outcomes, the question is how to adjust the probability estimates due to $r(p, o)$ for *all* the other outcome values besides o. Formally, Equation 5.5 becomes

$$gain(S^+) - gain(S) \approx cnt(p,o) \log \left(\frac{cnt(p,o)}{\sum_{t \asymp p} f(t) \cdot P_S(o|t)} \right) \\ + \sum_{o' \in O \setminus \{o\}} cnt(p,o') \log \left(\frac{\sum_{t \asymp p} f(t) \cdot P_{S^+}(o'|t)}{\sum_{t \asymp p} f(t) \cdot P_S(o'|t)} \right). \tag{5.6}$$

To approximate P_{S^+} without improved iterative scaling, we again appeal to the maximum-entropy principle. When computing gain due to $r(p,o)$, we adjust the probabilities $P(o'|t)$ for each $o' \neq o$ proportionally to the adjustment for $P(o|t)$. This approximates the necessary adjustments to P_{S^+} without introducing any assumptions beyond $r(p,o)$. Formally this is

$$P_{S^+}(o'|t) \approx P_S(o'|t) \frac{1 - P_{S^+}(o|t)}{1 - P_S(o|t)} = P_S(o'|t) \frac{supp(p) - cnt(p,o)}{supp(p) - \sum_{t' \asymp p} f(t') \cdot P_S(o|t')}. \tag{5.7}$$

With this equation, we can now substitute P_{S^+} in Equation 5.6. Cancelling $\sum_{t \asymp p} f(t) \cdot P_S(o'|t)$ from the fraction within the logarithm of the sum yields as approximation

$$gain(S^+) - gain(S) \approx cnt(p,o) \log \left(\frac{cnt(p,o)}{\sum_{t \asymp p} f(t) \cdot P_S(o|t)} \right) \\ + \sum_{o' \in O \setminus \{o\}} cnt(p,o') \log \left(\frac{supp(p) - cnt(p,o)}{supp(p) - \sum_{t' \asymp p} f(t') \cdot P_S(o|t')} \right). \tag{5.8}$$

Note that with non-binary outcomes, we need to make two decisions when seeking the next rule with the highest gain: its pattern p and its outcome value o. We use $g_S(p)$ to denote the highest gain of any rule with pattern p over all the possible outcome values, i.e., $g_S(p) = \max_{o \in O} gain(S \cup \{r(p,o)\}) - gain(S)$.

Finally, we note that some existing techniques that use similar information-theoretic methods [FGS17, Sar01] are compatible with numeric outcomes. The idea is to scale each outcome value by the sum of all the outcomes, which means that the scaled outcomes add up to one and can be thought of as a probability distribution. We omit the details and remark that this transformation is compatible with our summarization method for numeric explanatory attributes.

5.2 LENS Approach

We now present the **L**ightweight **E**xtraction of **N**umeric **S**ummaries (LENS) approach for informative summaries with numeric attributes. We motivate our approach and present an overview, followed by a discussion of candidate rule generation and how we speed up the gain computation of the selected candidates. We end with a discussion of the computational complexity of LENS.

5.2.1 Motivation and Overview

Numeric and ordinal explanatory attributes produce a much larger pattern space that cannot be handled effectively by existing methods. For example, a straightforward extension of the sample-based pruning method used by Flashlight [GAG$^+$14] and SIRUM [FGS17] is to consider all patterns with all possible intervals that match at least one sampled tuple. However, when we tested this extension on the `Occupancy` dataset, this modified version of Flashlight could not produce a single rule within 3 hours. This means that the candidate rule space is too large even after sample-based pruning.

Another straightforward optimization that is used in many interval pattern mining techniques is to discretize or bin numeric domains. However, this leads to information loss, and minimizing the impact of this loss requires careful selection of the discretizing technique depending on both the application and the data [BKB00, GLS$^+$13]. Alternatively, one can discretize the data manually. However, this requires domain knowledge for meaningful intervals, and even then, it may not be clear which intervals are informative. For example, in the `Occupancy` dataset, one would have to know that days of the week can be binned into weekdays and weekends. But even then, we would lose interesting patterns if, e.g., the building was occupied differently at the beginning of a workweek than at the end. Since informative summaries are meant to provide insights for users who may not be familiar with the data, requiring domain knowledge beforehand defeats their purpose. This means that we need a solution that is data driven and does not perform any discretization apriori.

Figure 5.1 summarizes the ideas behind LENS. Instead of abandoning existing methods completely, we leverage their strengths (finding simple informative patterns) while avoiding their weaknesses (inability to scale to large ordered domains). In each iteration, LENS first uses an existing method (shown at the top of the figure) to find top k *simple* informative patterns. Notably, LENS is compatible with any greedy method for finding simple informative patterns such as Flashlight, SIRUM or SURPRISE [Sar01]. LENS then "grows" the presumably informative constants in the ordered attributes of these patterns in a principled way to form informative intervals. Furthermore, the intervals considered by LENS are deterministic for each underlying simple pattern, and therefore benefit from pre-computations. To do this, we present a data structure called *Sparse Cumulative Cube* (SCC) in Section 5.2.3 that stores intermediate sums required by Equation 5.8 to estimate gain.

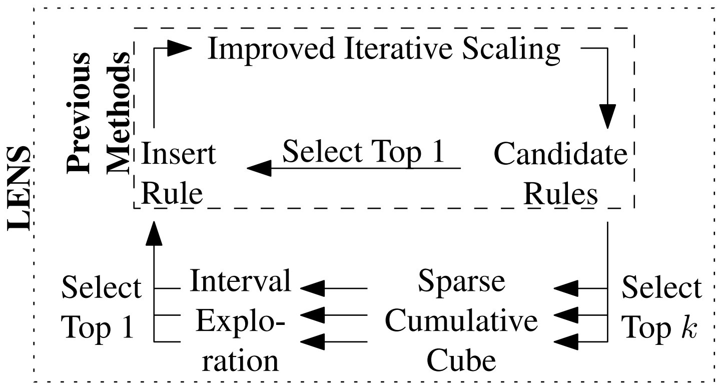


Figure 5.1: Summary of LENS

Algorithm 5.2 details the operations of LENS. Starting with an empty summary (Line `1`), LENS performs improved iterative scaling (Line `3`) and selects the next best rule (Lines `3-14`) s times (Line `2`). For each rule, LENS first obtains k simple patterns using some existing method (Line `5`). For each of these patterns (Line `6`), LENS explores patterns with intervals over ordered attributes (Line `10`), as we describe in the following section, with q being the best pattern in terms of gain. To speed up the gain computation, we build sparse cumulative cubes (Lines `8-9`), which we describe in Section 5.2.3. While q is the most informative pattern based on a particular simple pattern p, r_{best} stores the rule with highest gain across all patterns (Lines `11-13`), which is added to the summary (Line `14`). We restrict the search to simple patterns with fewer than 5 constants (Line `7`) for interpretability and efficiency. Patterns with many constants or intervals are harder to interpret by users [LBL16], and, as we discuss in Section 5.2.3, the size of the sparse cumulative cube grows exponentially with the number of constants.

5.2.2 Interval Exploration

This section describes how LENS creates intervals based on a simple pattern p. Intervals are created only over ordered attributes that have a constant in p; unordered attributes or ordered attributes with a wildcard in p are not considered. However, the number of possible intervals over the considered attributes can still be prohibitively large. Take an ordered attribute A_i and a constant $a_i \in A_i$. There are up to $\frac{|A_i|}{2}$ values before and after a^i in the ordering as valid interval endpoints. This gives up to $\left(\frac{|A_i|}{2}\right)^2$ intervals on A_i containing a_i. If a simple pattern has more than one constant, the set of potential patterns is the Cartesian product of the intervals for each attribute.

To reduce the set of candidate rules, we consider exponentially increasing intervals. For an attribute A_i with ordered values $a_i^1, \ldots, a_i^{|A_i|}$ and a constant p_i occurring

Algorithm 5.2: LENS

```
Input: Data $f$, summary size $s$, number of simple patterns $k$
Output: Informative summary $S$
1  $S \leftarrow \emptyset$
2  while $|S| < s$ do
3      $\mathcal{P}_S \leftarrow$ IMPROVED ITERATIVE SCALING($S$)
4      $maxGain \leftarrow 0$
5      $Candidates \leftarrow$ top $k$ simple patterns
6      foreach $p = ([p_1, q_1], \dots, [p_d, q_d]) \in$ Candidates do
7          if $|\{i \in \{1, \dots, d\} : p_i = q_i\}| < 5$ then
8              foreach $o \in O$ do
9                  Build $\mathcal{C}_o$ and $\mathcal{C}_o^S$                cf. Section 5.2.3
10             $q \leftarrow$ IntervalExploration($p$)                   cf. Algorithm 5.3
11             if $g_S(q) >$ maxGain then
12                 $r_{best} \leftarrow \arg\max_{r \in \{r(q,o) : o \in O\}} gain(S \cup \{r\})$
13                 $maxGain \leftarrow gain(S \cup \{r_{best}\})$
14     $S \leftarrow S \cup \{r_{best}\}$
15 return $S$
```

in a simple pattern, we consider starting points $p_i, p_i-1, p_i-3, p_i-7, \dots$ and ending points $p_i, p_i+1, p_i+3, p_i+7, \dots$, bounded by 1 and $|A_i|$, respectively. Note that these exponential steps are based on the rank order of attribute values and not the values themselves, meaning that all pairs of consecutive values are treated as equidistant. Also, note that the considered intervals are more fine grained close to p_i. This is desirable because p_i was chosen to be an informative constant in the simple pattern, so values close to p_i could also be informative. Additionally, for any interval $[p'_i, q'_i]$ on A_i containing a_i, we consider starting and ending points that are similar to p'_i and q'_i in distance to a_i. This exponential strategy reduces the number of intervals over an attribute A_i to at most $\left(\log_2\left(\frac{|A_i|}{2}\right)\right)^2$. Overall, there are up to $\Pi_{i=1}^{d}\left(\log_2\left(\frac{|A_i|}{2}\right)\right)^2$ patterns with intervals for each simple pattern. Note that our approach for interval selection is adaptive to the data and to the rules included into the summary so far as it is centered around constants from a pattern that is informative in this context. This is different than other interval selection schemes such as those using quantiles, which are similar to apriori discretization.

To further reduce the candidate search space, we use a greedy breadth-first search (BFS) that prunes some intervals along the way. We say that a pattern p is *incremented* if it is extended by moving its starting or ending point by one position in the allowed set of positions listed above. For any pattern p' resulting from an in-

Algorithm 5.3: `IntervalExploration`

```
   Input: Simple pattern $([p_1, q_1], \ldots, [p_d, q_d])$
1  $List \leftarrow \{(([p_1, q_1], \ldots, [p_d, q_d]), 0)\}$
2  $bestGain \leftarrow 0$
3  while List $\neq \emptyset$ do
4      $NextList \leftarrow \emptyset$
5      foreach $(([p'_1, q'_1], \ldots, [p'_d, q'_d]), \text{prevGain}) \in$ List do
6          $thisGain \leftarrow g_S(([p'_1, q'_1], \ldots, [p'_d, q'_d]))$
7          if thisGain > bestGain then
8              $bestGain \leftarrow thisGain$
9              $bestPattern \leftarrow ([p'_1, q'_1], \ldots, [p'_d, q'_d])$
10         if thisGain > prevGain then
11             forall $1 \leq i \leq d$ do
12                 if $p'_i > 1$ then
13                     $nextLim \leftarrow \max(1, p'_i - (2 \cdot |p_i - p'_i|) + 1)$
14                     $nextPat \leftarrow ([p'_1, q'_1], \ldots, [nextLim, q'_i], \ldots, [p'_d, q'_d])$
15                     $NextList.insert(nextPat, thisGain)$
16                 if $q'_i < |A_i|$ then
17                     $nextLim \leftarrow \min(|A_i|, q'_i + (2 \cdot |q_i - q'_i| + 1))$
18                     $nextPat \leftarrow ([p'_1, q'_1], \ldots, [p'_i, nextLim], \ldots, [p'_d, q'_d])$
19                     $NextList.insert(nextPat, thisGain)$
20     $List \leftarrow NextList$
21 return bestPattern
```

crement of p, we further explore the increments of p' only if p' has higher gain than p. Starting with p, the BFS considers patterns with the same number of increments from p during each iteration. As a result, duplicate patterns during the search could only occur within one iteration of the BFS. Keeping track of the candidates in each iteration via a hash table therefore prevents redundant computations during the interval exploration of a simple pattern p without explicitly listing all the previously considered patterns.

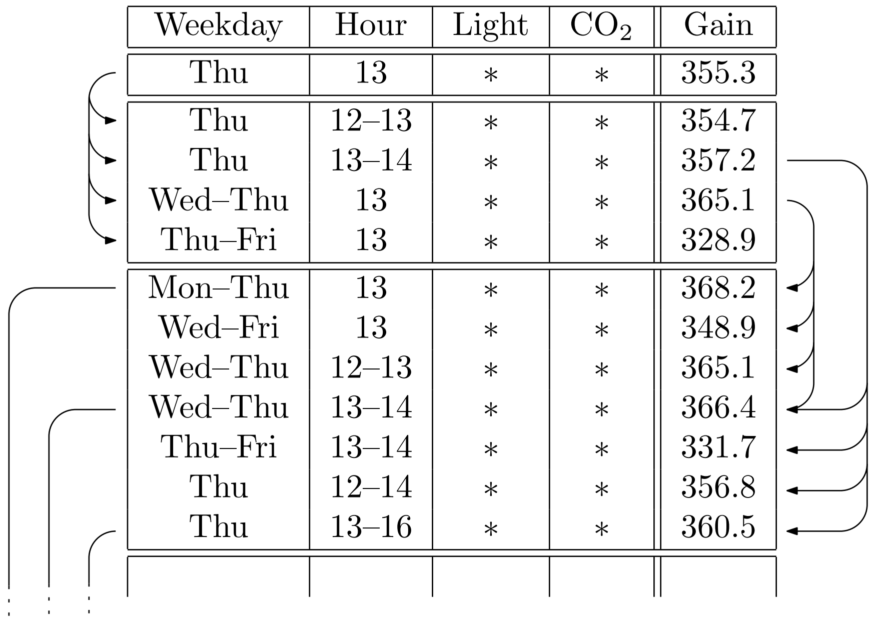
Algorithm 5.3 shows the interval exploration for one simple pattern, assuming for simplicity that all d attributes are ordered. Unordered attributes do not change; that is, they keep the same constant or wildcard as in the original simple pattern. For multiple simple patterns, the algorithm runs separately for each one, as seen in Algorithm 5.2. *List* is the set of patterns considered during the current iteration of the BFS, together with the gain thresholds used to test whether the increment producing the pattern increases gain. Starting with the simple pattern itself and zero as the threshold (Line `1`), the algorithm considers all patterns in *List* (Line `5`) and evaluates their gain (Line `6`). If it is the highest gaining pattern so far, we store it as *bestPattern* (Lines `7-9`). If the gain is higher than the accompanying threshold in *List*, the increment producing this pattern improves gain, and we include further increments in the next BFS iteration (Lines `10-19`). Specifically, for each attribute (Line `11`), we determine the next smaller starting point (Line `13`) if it is not the smallest value, i.e. 1 (Line `12`). The new pattern obtained by replacing this starting point (Line `14`) is then inserted with the gain of the current pattern into *nextList*. *nextList* collects patterns for the next BFS iteration (Line `20`). Analogously, a pattern with the next ending point, if possible, is inserted (Lines `16-19`). Note that attributes with wildcards in the original simple pattern are never altered because their intervals already span from 1 to $|A_i|$. Finally, the algorithm returns the best pattern (Line `21`) when there is no pattern left for another BFS iteration (Line 3). While this termination can happen early if no increment has resulted in a higher gain, there are at most $\sum_{i=1}^{d} 2 \cdot \log_2 \left(\frac{|A_i|}{2}\right)$ increments to any pattern before it consists entirely of wildcards. Similarly, since no pattern is considered twice, the inner loop (Lines `6-19`) is executed at most $\Pi_{i=1}^{d} \left(\log_2 \left(\frac{|A_i|}{2}\right)\right)^2$ times.

We illustrate our interval search with an example and the accompanying Figure 5.2. Note that the gain values depend on the current model $\mathcal{P}_S$ and are just used for illustrative purposes. The figure shows each iteration of the BFS as one block of patterns. Incrementing a pattern for the next BFS iteration is displayed as an arrow from the original pattern towards the incremented one. For brevity, we refer to increments as $p'_i \to X$. This means that the pattern is the same except for the starting or an ending point p'_i of one of the attributes being replaced by X.

Example 5.4. *Consider the `Occupancy` dataset illustrated in Table 5.1 and the simple pattern* $p = (Thu, 13, *, *)$. *In our formal notation, this pattern is written as* $([p_1, q_1], [p_2, q_2], [p_3, q_3], [p_4, q_4]) = ([4, 4], [13, 13], [1, |Light|], [1, |CO_2|])$. *Additionally, let its gain be* $g_S(p) = 355.3$. *Since this pattern already covers the full range for Light and* CO_2*, the available increments are* $p'_1 \to p_1 - 1$, $q'_1 \to q_1 + 1$, $p'_2 \to p_2 - 1$ *and* $q'_2 \to q_2 + 1$. *As shown in Figure 5.2, only* $q'_2 \to 14$ *and* $p'_1 \to 3$ *result in higher gain, of* 357.2 *and* 365.1 *respectively. Next, we consider the patterns obtained through increments of these two patterns. We then check which of these patterns increase the gain over their predecessor. This process is repeated until no improvements are found or all the intervals become wildcards.*

5.2.3 Efficient Gain Estimation

Even though we have reduced the number of patterns to consider, it is inefficient to evaluate the gain of each candidate, cf. Equation 5.8, by a separate linear scan of



Weekday	Hour	Light	CO_2	Gain
Thu	13	$*$	$*$	355.3
Thu	12–13	$*$	$*$	354.7
Thu	13–14	$*$	$*$	357.2
Wed–Thu	13	$*$	$*$	365.1
Thu–Fri	13	$*$	$*$	328.9
Mon–Thu	13	$*$	$*$	368.2
Wed–Fri	13	$*$	$*$	348.9
Wed–Thu	12–13	$*$	$*$	365.1
Wed–Thu	13–14	$*$	$*$	366.4
Thu–Fri	13–14	$*$	$*$	331.7
Thu	12–14	$*$	$*$	356.8
Thu	13–16	$*$	$*$	360.5

Figure 5.2: BFS Pattern Exploration

the data. We introduce a data structure to quickly provide $cnt(p, o)$ and $\sum_{t \preceq p} f(t) \cdot P_S(o|t)$ for all outcomes $o \in O$. With $f : \mathcal{A} \times O \to \mathbb{N}$ for ordered attributes being integers in a $(d+1)-$dimensional space, $cnt(p, o)$ becomes a range-sum query. Similarly, the expected number of tuples $\sum_{t \preceq p} f(t) \cdot P_S(o|t)$ based on the current summary S is a range-sum query on similar data with $f_S(t, o) = f(t) \cdot P_S(o|t)$.

Example 5.5. *Consider an ordered attribute $A = (a^1, \ldots, a^{15})$. One way to count the number of records matching pattern $[a^3, a^8]$ is to sequentially scan the dataset. However, if we precompute an array C where $C[i]$ stores the count of records with $A \leq a^i$, then the number of records matching our pattern is simply $C[8] - C[2]$.*

There exists a data structure for such range-sum queries called *Prefix-Sum Array* [HAMS97]. It precomputes cumulative sums across all data dimensions and answers range-sum queries in time $\mathcal{O}(2^{d+1})$, i.e., independently of n and of attribute-domain sizes. However, these cumulative sums are stored for all attribute combinations, leading to excessive memory consumption. In our case, this data structure has $|O| \cdot \Pi_{1 \leq i \leq d} |A_i|$ sums, and we need two data structures, one for f and one for f_S. Since numeric attributes can have very large domains, this memory footprint is too large, even with few attributes. Even for the smallest real-world dataset that we use in our experiments, it is $\Pi_{1 \leq i \leq d} |A_i| \geq 10^{19}$, i.e., at least 10^{10} gigabytes of memory.

Our solution is to create compact versions of this data structure for each simple pattern tailored to our pattern exploration scheme. That is, they are built for

$$
\begin{array}{lccccccccccccc}
A_1 = (& a_1^1, & a_1^2, & a_1^3, & a_1^4, & a_1^5, & a_1^6, & a_1^7, & a_1^8, & a_1^9, & a_1^{10}, & a_1^{11}, & a_1^{12}, & a_1^{13}) \\
 & [& & & & [& & [& [\,] &] & &] & &] \\
B_1(8) = (& & & & 4, & & 6, & 7, & 8, & 9, & & 11, & & 13)
\end{array}
$$

Figure 5.3: Aggregation thresholds for attribute $A_1 = (a_1^1, \ldots, a_1^{13})$ and constant $p_1 = 8$. The brackets represent starting and ending points considered by Algorithm 5.3 during interval exploration.

each simple pattern LENS explores and used only to speed up gain evaluation of increments from that simple pattern, which is Line 6 in Algorithm 5.3. We use two properties of our exploration scheme, namely that wildcards are never reduced to smaller intervals and that intervals grow exponentially. The first property allows us to ignore attributes with a wildcard in the simple pattern in the sense that we only store the sum of all values of these attributes. Second, the predefined interval limits indicate which subset sums will be required. For instance, if a simple pattern p has a constant value p_i for an attribute A_i, our exploration does not consider any pattern with an ending point of $p_i + 2$. Since prefix-sum arrays store cumulative sums, this aggregation is implicit and only the following upper bounds of aggregatable intervals are relevant

$$
\begin{aligned}
B_i(p_i) = (&p_i - 2^{\lfloor \log(p_i - 1) \rfloor}, \ldots, p_i - 8, p_i - 4, p_i - 2, p_i - 1, \\
&p_i, p_i + 1, p_i + 3, p_i + 7, \ldots, |A_i|)
\end{aligned} \tag{5.9}
$$

as illustrated in Figure 5.3. Note that these upper bounds are equal to the interval ending points we consider during pattern exploration, while they are shifted by one for the starting points. This is because range-sum queries on these cumulative sums subtract the sum *up to* the lower bound from the sum at the upper bound. As a result, we reduce both the dimensionality of our data structure and the domain size per dimension.

Additionally, note that we use this data structure to compute $cnt(p, o)$ and $\sum_{t \preceq p} f(t) \cdot P_S(o|t)$, respectively, which means that each query covers exactly one outcome value. Therefore, we can build the data structure separately for each outcome value instead of treating the outcome value as an additional dimension. In contrast to the previous optimizations, this does not reduce memory consumption as we need the same number of cells, which are simply spread across multiple instances. However, since the time complexity of range-sum queries is exponential in the dimensionality of the data structure, this optimization halves the query time.

To improve readability, and without loss of generality, suppose that the attributes are ordered so that the m attributes where a simple pattern $p = ([p_1, q_1], \ldots, [p_d, q_d])$ uses a constant are at the front. That is, there exists an integer $1 \leq m \leq d$ with $p_i = q_i$ for $i \leq m$ and $p_i \neq q_i$ for $i > m$. Our *sparse cumulative cube* (SCC) is an m-dimensional array, where each cell with $i_1, \ldots, i_m$ in $B_1(p_1), \ldots, B_m(p_m)$, respectively, is defined as

$$\mathcal{C}_o[i_1]\cdots[i_m] = \sum_{j_1=1}^{i_1} \cdots \sum_{j_m=1}^{i_m} \sum_{j_{m+1}=1}^{|A_{m+1}|} \cdots \sum_{j_d=1}^{|A_d|} f((a_1^{j_1}, \ldots, a_d^{j_d}), o) \tag{5.10}$$

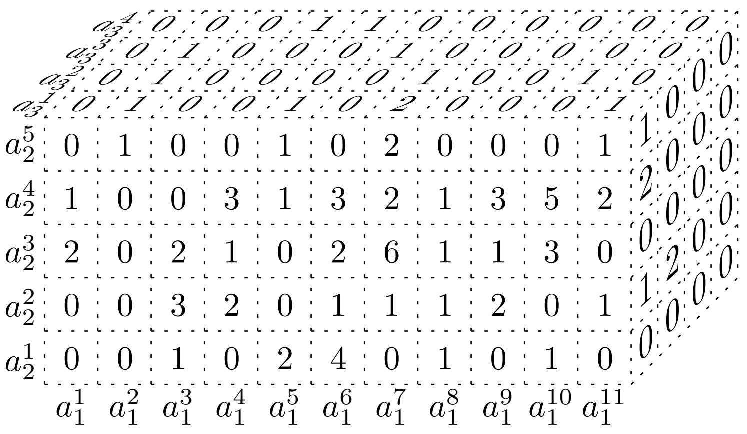
Analogously to [HAMS97], a range-sum query, in our case $cnt(p, o)$, is

$$cnt(p, o) = \sum_{i_1 \in \{p_1-1, q_1\}} \cdots \sum_{i_m \in \{p_m-1, q_m\}} \mathcal{C}_o[i_1]\cdots[i_m] \cdot (-1)^{\sum_{j=1}^{m} \mathbb{I}(i_j, p_j-1)} \tag{5.11}$$

where $\mathbb{I}(i_j, p_j - 1)$ is the identity function, i.e., it is 1 if $i_j = p_j - 1$ and 0 otherwise. Note that $\mathcal{C}_o$ only includes range sums for the first m attributes (with constants in the simple pattern) and implicitly aggregates all values of the remaining attributes (with wildcards in the simple pattern). Similarly, we build a SCC $\mathcal{C}_o^S$ by substituting $f(t, o)$ with $f_S(t, o) = f(t) \cdot P_S(o|t)$ in Equation 5.10. This allows us to compute $\sum_{t \preceq p} f(t) \cdot P_S(o|t)$. Note that both of our SCCs can be constructed in a single pass over the data for each simple pattern considered for interval exploration: the interval endpoints are known beforehand because they depend on the constants in the simple pattern.

Example 5.6. *Consider a dataset with the three attributes $A_1 = (a_1^1, \ldots, a_1^{11})$, $A_2 = (a_2^1, \ldots, a_2^5)$ and $A_3 = (a_3^1, \ldots, a_3^4)$, and two outcomes o and o'. Figure 5.4a shows the tuple counts f for one fixed outcome o as a cube. Additionally, let $p = ([8, 8], [4, 4], [1, 4])$ be a simple pattern, i.e., $(8, 4, *)$. All patterns considered in `IntervalExploration`(p) therefore cover the full range of A_3. This means that the SCC is two dimensional to accommodate queries with different ranges for A_1 and A_2. To provide all the relevant numbers for this example, Figure 5.4b shows the counts of items with outcome o summed up over different values for A_3, which may not be visible in (a). As 8 and 4 are the constants of p, the relevant aggregation thresholds for our data structures are $B_1(8) = (4, 6, 7, 8, 9, 10)$ and $B_2(4) = (2, 3, 4, 5)$. The SCC $\mathcal{C}_o$ is thus a $6 \times 4-$matrix with accumulated item counts up to the thresholds $B_1(8)$ and $B_2(4)$ as shown in Figure 5.4c.*

A pattern that might be checked during our exploration is $p' = ([5, 11], [3, 4], [1, 4])$. Without $\mathcal{C}_o$, evaluating $cnt(p', o)$ would require a linear scan of all entries of f or at least summation of 40 cells if f is stored with random access like the cube in



$$
\begin{array}{c|ccccccccccc}
a_3^4 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_3^3 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
a_3^2 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
a_3^1 & 0 & 1 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 & 1
\end{array}
$$

$$
\begin{array}{c|ccccccccccc|ccc}
a_2^5 & 0 & 1 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
a_2^4 & 1 & 0 & 0 & 3 & 1 & 3 & 2 & 1 & 3 & 5 & 2 & 0 & 0 & 0 \\
a_2^3 & 2 & 0 & 2 & 1 & 0 & 2 & 6 & 1 & 1 & 3 & 0 & 0 & 0 & 0 \\
a_2^2 & 0 & 0 & 3 & 2 & 0 & 1 & 1 & 1 & 2 & 0 & 1 & 2 & 0 & 0 \\
a_2^1 & 0 & 0 & 1 & 0 & 2 & 4 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
 & a_1^1 & a_1^2 & a_1^3 & a_1^4 & a_1^5 & a_1^6 & a_1^7 & a_1^8 & a_1^9 & a_1^{10} & a_1^{11} & & &
\end{array}
$$

(a)

$$
\begin{array}{c|ccccccccccc}
a_2^5 & 0 & 3 & 0 & 1 & 2 & 1 & 3 & 0 & 0 & 1 & 1 \\
a_2^4 & 1 & 0 & 0 & 3 & 1 & 4 & 2 & 8 & 5 & 6 & 2 \\
a_2^3 & 2 & 2 & 4 & 1 & 0 & 2 & 7 & 1 & 1 & 3 & 0 \\
a_2^2 & 0 & 1 & 4 & 2 & 0 & 1 & 5 & 2 & 2 & 1 & 3 \\
a_2^1 & 1 & 0 & 1 & 0 & 2 & 4 & 0 & 1 & 0 & 1 & 0 \\
\hline
 & a_1^1 & a_1^2 & a_1^3 & a_1^4 & a_1^5 & a_1^6 & a_1^7 & a_1^8 & a_1^9 & a_1^{10} & a_1^{11}
\end{array}
$$

(b)

$$
\begin{array}{c|cccccc}
a_2^5 & 26 & 43 & 60 & 72 & 80 & 98 \\
a_2^4 & 22 & 36 & 50 & 62 & 70 & 86 \\
a_2^3 & 18 & 27 & 39 & 43 & 46 & 54 \\
a_2^2 & 9 & 16 & 21 & 24 & 26 & 31 \\
 & & & & & & \\
\hline
 & a_1^4 & a_1^6 & a_1^7 & a_1^8 & a_1^9 & a_1^{11}
\end{array}
$$

(c)

Figure 5.4: (a) Illustration of f as cube displaying the count of tuples with outcome o. (b) Projection of A_3, i.e. summation, to a 2-dimensional matrix. (c) Sparse cumulative cube $\mathcal{C}_o$.

Figure 5.4a. With $\mathcal{C}_o$, we can determine cnt(p', o) with 4 cells by Equation 5.11:

$$\begin{aligned} cnt(p,o) &= \sum_{a_i \in \{4,11\}} \sum_{a_2 \in \{2,4\}} \mathcal{C}_o[a_1][a_2] \cdot (-1)^{\mathbb{I}(a_1,a_1^4)+\mathbb{I}(a_2,a_2^2)} \\ &= \mathcal{C}_o[4][2] \cdot (-1)^2 + \mathcal{C}_o[11][2] \cdot (-1)^1 \\ &+ \mathcal{C}_o[4][4] \cdot (-1)^1 + \mathcal{C}_o[11][4] \cdot (-1)^0 \\ &= 9 - 31 - 22 + 86 = 42 \end{aligned}$$

5.2.4 Time Complexity

We now study the worst-case time complexity of LENS. Since LENS can use any method for simple pattern generation and since improved iterative scaling runs until convergence without bounded complexity, our analysis focuses on the complexity of our contribution, which is Lines `6-14` in Algorithm 5.2.

In Lines `8-9`, we build two SCCs for each outcome, which repeat for each of the k simple patterns and s rules generated. The size of an SCC depends on the domain size of the attributes, which is generally only bounded by n since every data record could have a different value. As LENS limits the maximum number of constants of simple patterns to 5 in Line `7`, each SCC has a maximal size of $(2 \cdot \log(\frac{n}{2}))^5$. Since a single scan of the data suffices to build an SCC, each construction takes time $\mathcal{O}(n + \log^5(n)) = \mathcal{O}(n)$.

Next, the worst case for our interval exploration scheme is that there is no pruning. As a result, for each simple pattern, all patterns obtainable through our increments would be considered once. For each attribute A_i, we would consider up to $2 \cdot \log_2(|A_i|)$ intervals. Since the attribute domain sizes can still only be bounded by n, up to $((\log_2(\frac{n}{2}))^2)^5 \leq \log_2^{10}(n)$ patterns with intervals would be considered for each simple pattern. Note that the only operation in Algorithm 5.3 that is not an elementary assignment or set insertion is the computation of gain in Line `6`. By Equation 5.8, gain can be computed for a pattern p with $cnt(p, o)$ and $\sum_{t \asymp p} f(t) \cdot P_S(o|t)$ for each $o \in O$. These intermediate values can be computed each using 2^5 cells of our SSCs by Equation 5.11. In total, this step can be performed in time $\mathcal{O}(1)$, and thus Line `10` in Algorithm 5.2 takes time $\mathcal{O}(\log_2^{10}(n))$ to build an informative summary with s rules.

The remaining lines in Algorithm 5.2, except for the simple pattern generation and improved iterative scaling, consist of assignments and gain computations, which take constant time with our SCCs. Let $\mathcal{T}_{IS}$ and $\mathcal{T}_{SP}$ be the time complexity of improved iterative scaling and the method used to select simple patterns. The total complexity of LENS is then $\mathcal{O}(s \cdot (\mathcal{T}_{IS} + \mathcal{T}_{SP} + k \cdot n))$. Since our pattern exploration scales linearly, it is unlikely to dominate the total complexity of LENS.

5.3 Experiments

In this section, we experimentally evaluate the performance of LENS on real datasets. We analyze the informativeness and runtime of LENS against related methods, parameter choices and scalability.

5.3.1 Setup

We implemented LENS in C++ and compiled it using the Microsoft® C/C++ Optimizing Compiler Version 19.00. We use Flashlight [GAG+14] as the simple pattern generation subroutine. 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. Unless otherwise noted, all presented results are arithmetic means over 100 runs. We use the following datasets.

`Occupancy` measures room temperature, humidity, lighting and CO_2 concentration, and includes a binary outcome indicating whether the room was occupied. This dataset is described in [CF16] and is available in the UCI repository[2]. Splitting the available timestamps into "Day of the week" and "Hour of the day", this dataset has 7 attributes and 20560 entries.

`Wine` described in [CCA+09] and available in the UCI repository[3], measures chemical wine properties and includes a quality score as an outcome. Merging the red wine and white wine parts of the data and using colour as an additional attribute yields 6497 entries with 12 attributes. Although there are 10 scores as outcome, we discretize them into "good" (scores 6 and higher) and "bad" to enable a comparison with subgroup discovery approaches.

`Electricity` presents data from the Australian New South Wales energy market. Each of the 45312 entries has the rising or falling price trend as outcome and 7 attributes such as energy price and demand in new South Wales and the neighboring Victoria region, excluding dates. The data is in the openML repository[4].

`Gas` measures the effect of two stimuli, bananas and wine, on an array of 8 gas sensors. The data is provided by Huerta et al. [HMF+16] and is available in the UCI repository[5]. There are 928991 entries with 10 attributes after adding

[2] `https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+`

[3] `http://archive.ics.uci.edu/ml/datasets/Wine+Quality`

[4] `https://www.openml.org/d/151`

[5] `http://archive.ics.uci.edu/ml/datasets/Gas+sensors+for+home+activity+monitoring`

Table 5.5: Dataset characteristics

Name	Rows	Columns	Outcomes	Data space size
`Occupancy`	20560	7	2	$1.9 \cdot 10^{19}$
`Wine`	6497	12	2	$1.1 \cdot 10^{25}$
`Electricity`	45312	7	2	$1.4 \cdot 10^{20}$
`Gas`	928991	10	3	$8.3 \cdot 10^{53}$
`Appliance`	19735	25	2	$5.7 \cdot 10^{79}$

temperature and humidity to the gas sensors, and one of three outcomes (no stimulus, wine and banana).

`Appliance` measures power consumption of household appliances with temperature and humidity per room as well as local weather measurements. The data was described in [CFD17] and is available via Github[6]. The outcome in our experiments is the total power consumption, discretized into two values of equal frequency. Each entry has 25 other measurement attributes.

Table 5.5 contains statistics for these data sets. Most of our experiments use `Occupancy`, `Electricity` and `Wine` because they have similar size and all competing approaches produce summaries in reasonable time. These datasets represent tall (`Electricity`), wide (`Wine`) and balanced (`Occupancy`) data. To evaluate scalability, we use `Gas` and `Appliance` as these datasets have many rows and columns, respectively.

5.3.2 Parameter Sensitivity

Before comparing LENS to its competitors, we explore parameter choices and sensitivity. Our approach has two parameters: the sample size FLS for Flashlight's sample-based pruning, and the number k of simple patterns for exploration by LENS. To establish sensitivity, we evaluate the impact on gain and runtime for informative summaries of size 20 for our datasets, with the other parameter being fixed to $FLS = 8$ or $k = 16$, respectively.

The impact of the sample size FLS is graphed in Figure 5.5 with a logarithmic x-axis. These graphs show that larger samples increase the runtime without a clear benefit in terms of informativeness. These results are in line with the conclusion of the authors of Flashlight that small samples suffice to find the highest gaining (simple) patterns [GAG+14].

For the number k of explored simple patterns for each rule, the impact is shown in Figure 5.6 with a logarithmic x-axis. As one might expect, k represents a classic

[6]`https://github.com/LuisM78/Appliances-energy-prediction-data/`

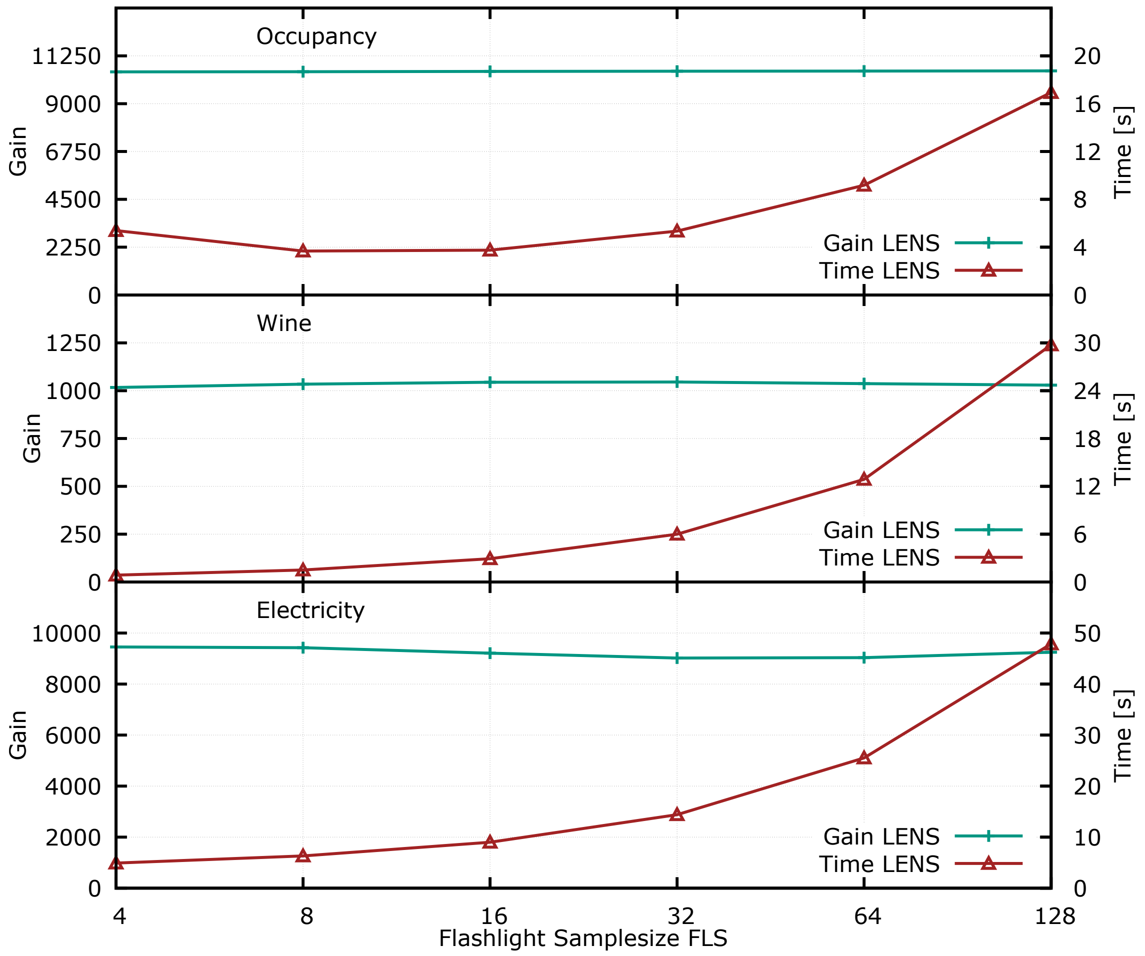


Figure 5.5: Impact of FLS on runtime of LENS and informativeness for `Occupancy` (top), `Wine` (middle) and `Electricity` (bottom)

time-quality tradeoff, as it directly controls how many (non-simple) patterns are considered. Based on these experiments, runtime increases significantly for large k while the gain stagnates.

Our conclusion regarding parameter sensitivity is that moderate choices for k and FLS suffice, as larger values increase runtime without a significant gain improvement. Specifically, we use $FLS = 8$ and $k = 16$ in all further experiments.

5.3.3 Competing Approaches

As a brief reminder from Section 3.5, we consider methods for Subgroup Discovery and Decision Trees as related approaches and include them as competitors. Since neither Subgroup Discovery nor Decision Trees are intended for informative sum-

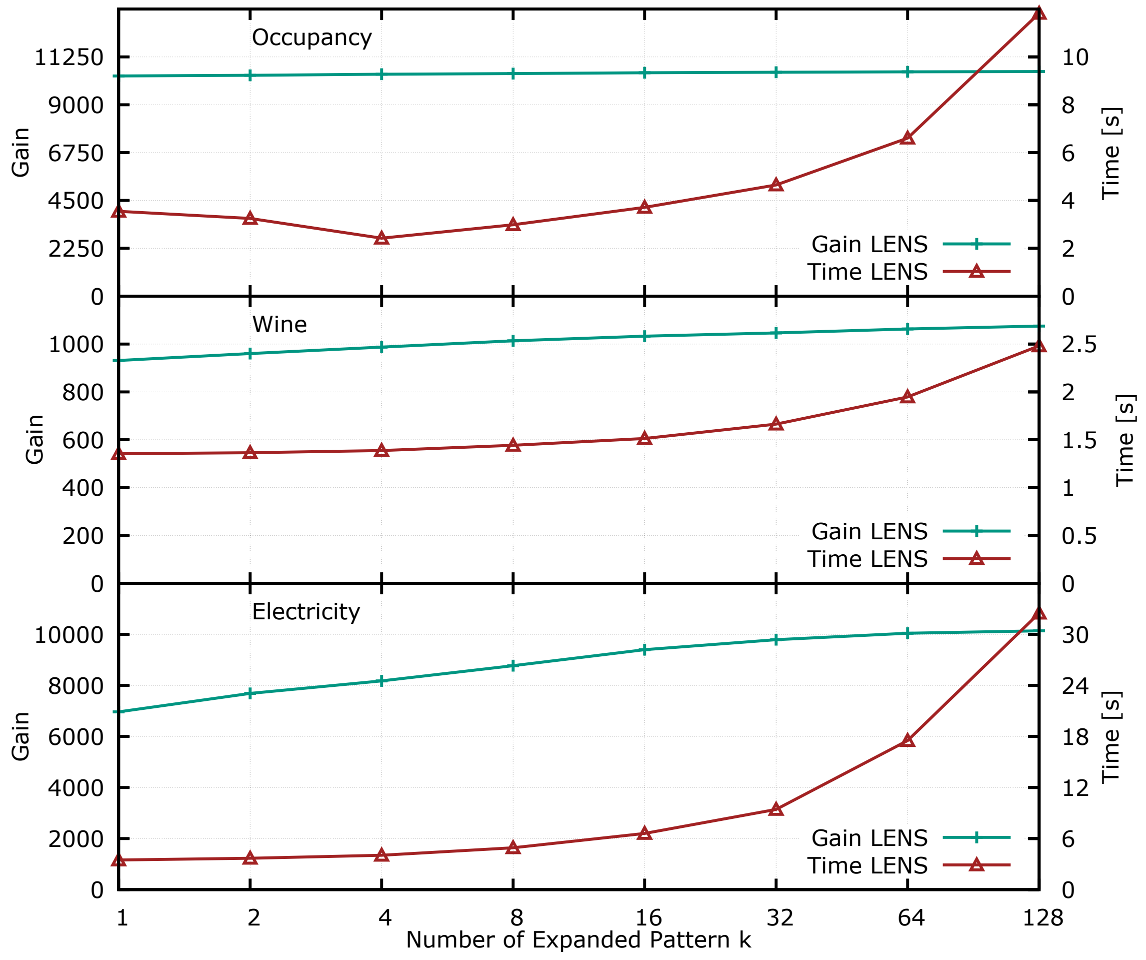


Figure 5.6: Impact of k on runtime of LENS and informativeness for the `Occupancy` (top), `Wine` (middle) and `Electricity` (bottom) data sets.

marization, we now describe how we adapted these methods to produce informative summaries comparable to those produced by LENS.

Subgroup discovery algorithms produce subgroups without a specified outcome value or model for the outcome distribution. To compare their informativeness with LENS, we use datasets with a binary outcome. For each subgroup, a comparable summary contains one rule with the subgroup as the pattern and an arbitrary outcome value (recall that for binary outcomes, we can easily infer the distribution of the other outcome value). We then use improved iterative scaling, as in LENS, to compute the maximum-entropy estimates. However, the time for improved iterative scaling is not included in the reported runtime of subgroup discovery methods as it is only necessary for our calculation of gain for comparison with LENS.

As mentioned in Section 3.5, we use DSSD [vLK12] and mergeSD [GR09] due to their focus on subgroup diversity and numeric data, respectively. While we use the official C++ implementation of DSSD[7], we reimplemented mergeSD as no C++ implementation was available. Due to the abundance of parameters for DSSD, we use mostly default settings, e.g., *beamWidth*=100, *qualMeasure*=WKL, *beamStrategy*=cover and *minCoverage*=10. The parameters *maxDepth* (maximum number of non-wildcard intervals of a pattern), *floatNumSplits* (number of bins for discretization) and *beamVarWidth* (dynamic adaption of beam width) all appear to show time-quality-tradeoffs. MergeSD also uses the parameters *maxDepth* and *floatNumSplits* but no other parameters. We found that the parameter choices mostly affect runtime but not gain in our experiments. As such, we use the following parameters with low runtime. For DSSD we use *maxDepth=1* and *floatNumSplits=10* and *beamVarWidth=false*, and for MergeSD we use *maxDepth=1* and *floatNumSplits=5*. We suspect that gain does not increase because these methods focus on different quality criteria, not informativeness.

For decision trees, each leaf is equivalent to a rule. For each leaf, we represent the collection of decisions leading to this leaf as pattern and the dominant class of the leaf as the outcome. Different than production rules from decision trees [Qui87], we also consider the precision of the dominant class for the data represented by the leaf. Since these leaves form a partition of the data and the outcome is binary, the decision tree therefore provides a well defined model for $P(o|t)$. This enables a comparison of informativeness, where the summary size is the number of leaves in the decision tree. For this comparison, we use the C++ implementation[8] of "Yet another Decision Tree builder" (YaDT) [Rug04] version 2.2.0 with several optimizations to reduce runtime and memory consumption. However, a problem with decision tree algorithms, including YaDT, is that we cannot build a tree of a particular size. For instance, using the default parameter values, the decision trees have 51, 477 and 1427 leaves for the `Occupancy`, `Wine` and `Electricity` data, respectively. This means that we have to perform a parameter search for every requested summary size and dataset in order to find an appropriate tree. As a result, we perform a grid search on the parameters *confidence* $\in \{0.01, 0.04, 0.16, 0.64\}$, *minCasesToSplit* $\in \{2, 8, 32, \ldots, 2048\}$ and *maxDepth* $\in \{1, 2, 4, 8\}$. As in previous work [GAG+14], the reported time is the total time for the grid search.

In contrast, explanation tables are informative summaries by design. We used our C++ implementation of Flashlight [GAG+14] that is also used as part of LENS. For fairness, we also use $FLS = 8$ when evaluating stand-alone Flashlight.

Finally, note that some approaches, such as mergeSD, assume that the global distribution of outcomes is known and only present subgroups whose distributions

[7] Available at `http://www.patternsthatmatter.org/software.php#dssd`

[8] Available at `http://pages.di.unipi.it/ruggieri/software.html`

are significantly different. Others, like Flashlight, explicitly report the global distribution through an all-wildcard first rule (recall Table 5.2). To level the playing field, each informative summary and subgroup collection is allowed to contain an all-wildcard rule or subgroup without counting it towards the summary size.

5.3.4 Conciseness and Efficiency

We use a single plot that shows runtime and gain of summaries of various sizes for each method. In Figure 5.7, each datapoint represents the average gain and runtime for summaries of a certain size, noted by a small text label, for each method. Note that YaDT cannot produce decision trees of size one, so the corresponding datapoints are missing. In this case, the corresponding datapoints are missing. Overall, the figure shows that LENS is the best overall choice in terms of gain per unit time.

Next, we zoom in on the time required to build informative summaries of certain sizes. For up to 10 rules, LENS is a close second to unmodified Flashlight, whose explanation tables contain no intervals. Since LENS uses Flashlight as a subroutine, LENS cannot be faster if both are using the same sample size FLS. Beyond 10 rules, DSSD is usually faster, as its runtime does not depend on the summary size. In contrast, MergeSD and YaDT are significantly slower.

Figure 5.7 also shows that LENS provides the highest gain per summary size with the exception of very small summaries on the `Electricity` dataset. Subgroup discovery algorithms are very inconsistent across datasets: while there is sometimes significant gain between summaries of size 1 and 2 and no gain from rules 11–20, it may also be the other way around. On the `Occupancy` and `Wine` datasets, this phenomenon manifests itself differently for MergeSD and DSSD. The reason for this erratic behavior is most likely the different optimization goals of subgroup discovery which may or may not coincide with information gain. As a final note, LENS shows highest consistency of improvement of gain as the number of rules increases.

5.3.5 Scalability

To evaluate how well LENS scales with the number of columns, we use the `Appliance` dataset. For any number d of attributes, we use the first d columns in the dataset. Figure 5.8 shows the runtime to build informative summaries of size 20 on a logarithmic scale depending on the number of columns, averaged over 10 runs. Runtime increases superlinearly with the number of columns for both Flashlight and LENS. Given that the runtime of Flashlight gets closer to the runtime of LENS as the number of columns increases, we conclude that Flashlight dominates the runtime of LENS. This means that any speedup to the process of selecting simple patterns would directly speed up LENS, be it an improvement to Flashlight like SIRUM [FGS17],

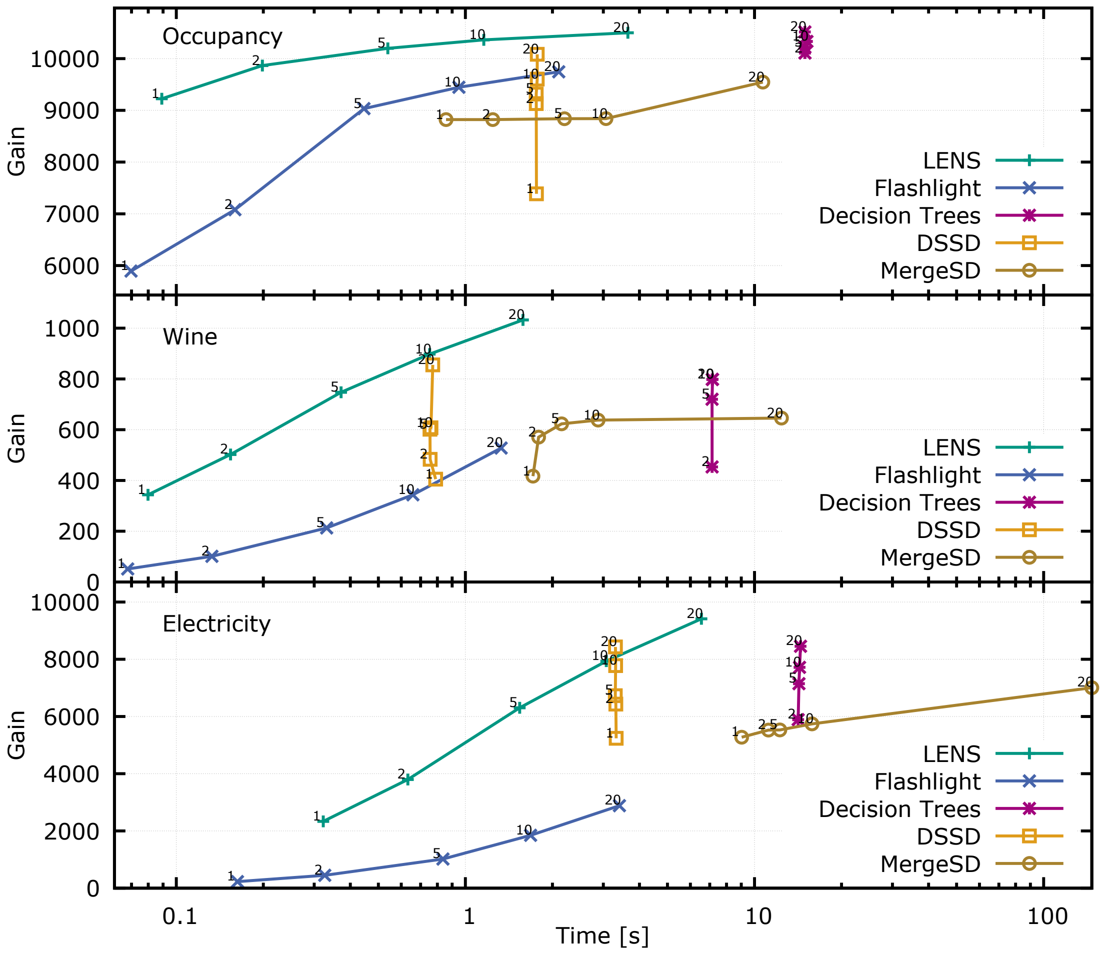


Figure 5.7: Runtime and gain for all methods and various summary sizes on `Occupancy` (top), `Wine` (middle) and `Electricity` (bottom)

or an entirely different method. Given this result, we defer the issue of speeding up simple pattern generation to future work.

For the `Gas` dataset, Figure 5.9 shows the row scalability of LENS and Flashlight as a double log plot. Here, we create smaller datasets by randomly sampling from `Gas`, to obtain unbiased subsets of the original attribute domains. LENS and Flashlight scale roughly linearly with the number of rows. While LENS requires nearly an order of magnitude more time than Flashlight for a smaller number of rows, it requires only twice as much time as Flashlight on the full dataset. This observation is also consistent with Figure 5.7. One explanation for the relatively higher runtimes of LENS on small data is that small samples contained a number of columns with very few duplicates. As a result, many simple patterns matched only one tuple and therefore many patterns were pruned away during sample-based

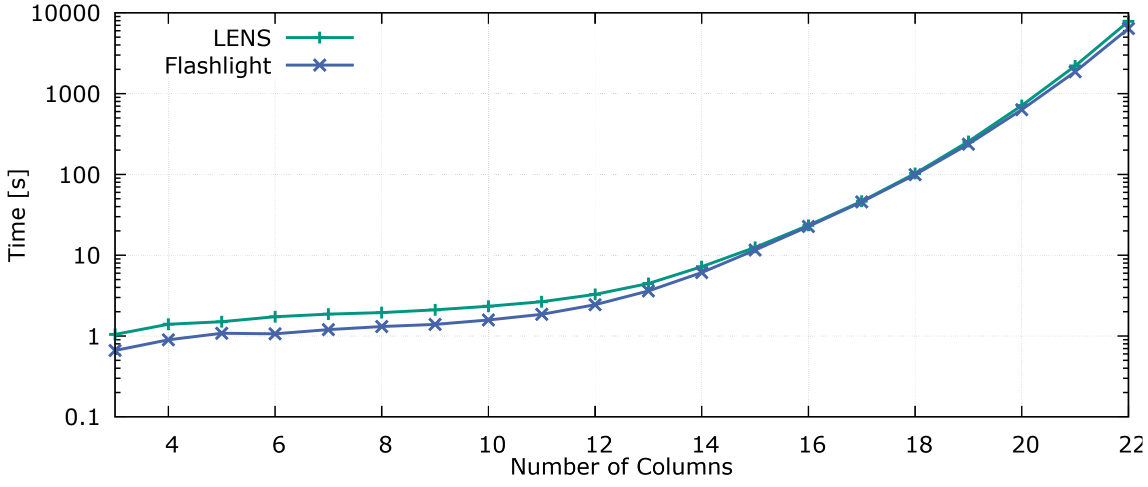


Figure 5.8: Scalability of LENS and Flashlight with the number of attributes using `Appliance`

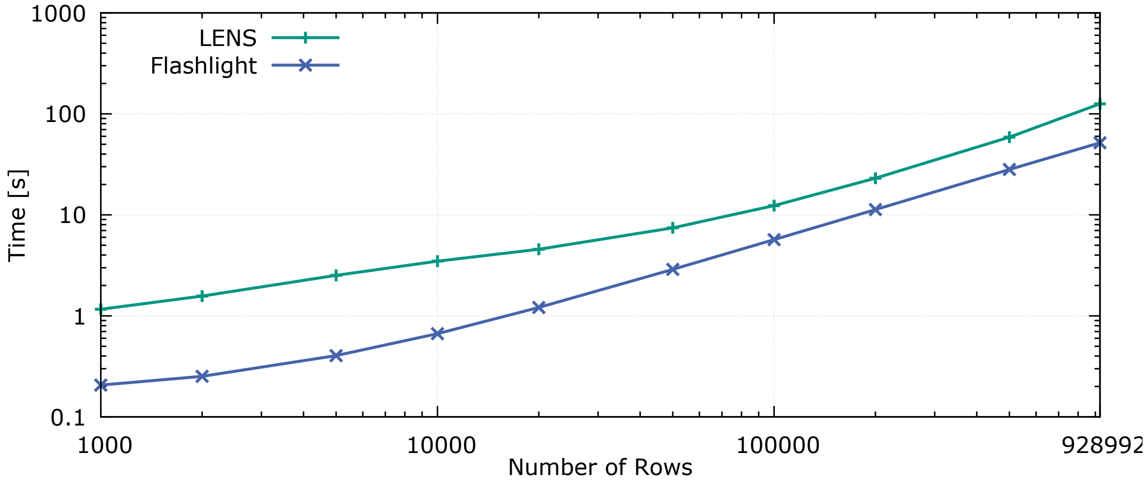


Figure 5.9: Scalability of LENS and Flashlight depending on the number of rows using `Gas`

pruning. In contrast, LENS considered patterns with various intervals that matched at least one sampled tuple.

5.3.6 Summary of Results

Our main experimental findings are that LENS generally produces the most informative summaries of a given desired size and excels at informativeness per unit of runtime. In terms of scalability, LENS scales linearly with the number of rows and exponentially with the number of columns in the data. However, this exponential scaling is due to using Flashlight as subroutine and could be improved with existing parallel techniques for Flashlight [FGS17] or with different algorithms for simple pattern selection. Finally, we find that the choices for the parameters sample size FLS and number of candidates k influence the runtime significantly, while FLS does not impact gain and k influences gain only moderately. As a result, small values, i.e., $FLS = 8$ and $k = 16$, lead to low runtimes without sacrificing information gain.

5.4 Summary

In this chapter, we considered the task of explaining dependencies through human interpretable summaries. Specifically, for a specified outcome attribute these summaries describe the relationship between the outcome and the remaining, explanatory attributes. The informativeness of a summary is determined with information theory using the maximum entropy principle to construct a distribution from the summary. The better the distribution constructed from a summary approximates the empirical distribution of the data, the more informative is the summary.

Our contribution to this task is the additional functionality of handling ordinal and numerical explanatory attributes as well as non-binary categorical outcomes. To this end, we formalized the problem such that summaries may use intervals to capture ranges of values. The resulting challenge is that numerical domains are large and thus the number of potential intervals and number of combinations of intervals from different attributes are much larger. Our solution for this problem is LENS, which is a heuristic approach using existing work to generate summaries without intervals and expanding the chosen, informative constants. Additionally, we introduced a data structure tailored to LENS that allows our approach to evaluate the benefit of many potential additions to the summary with a single pass over the data.

Finally, we experimentally evaluated LENS and showed the benefit of considering ordinal data when summarizing real data. Additionally, LENS outperforms other related approaches that could be used as summarization methods in terms of conciseness and time-efficiency. Our experiments also indicate that moderate parameter choices suffice and that LENS scales linearly with the number of rows in the data.

Chapter 6

Conclusions and Outlook

Analyzing and understanding relationships is an important and challenging task in data analysis. As the volume of available data rises, more intricate and non-linear dependencies become noticeable. Unfortunately, algorithms that work with such complex dependencies are often somewhat complicated and scale sub-optimally with the data volume. As a result, the very same data volume that enables the analysis of complex dependencies might be the main obstacle by inducing high computational costs. In this dissertation, we considered limitations and opportunities in regards to time-efficient dependency analysis with two particular aspects as focus. For one, we studied the detection and quantification of dependencies as practical application for mutual information estimation. As second aspect, we considered human-interpretable explanations for dependencies as a way to illustrate and understand them. In this chapter, we briefly recapitulate our results with an outlook towards future work.

6.1 Summary

In this dissertation, we used mutual information as a measure for arbitrary dependencies. Mutual information is a compelling candidate as it is zero if and only if the data is independent, it has a semantic meaning to its score and it has been used for this purpose for decades. As the most well-known estimator for continuous variables, we specialized on the nearest-neighbor based estimation techniques 3KL [Eva08] and KSG [KSG04]. While previous work considered the time complexity of their proposed algorithms to compute the estimate [Eva08, KSG04, VH07], we are the first to evaluate the complexity of the problem itself. As our first contribution, we have proven the lower bound of $\Omega(n \log n)$ to compute either the 3KL or the KSG, which fits the complexity of the algorithms proposed so far. As a result, the existing algorithms are asymptotically optimal and any further asymptotic improvements require specific scenarios or assumptions.

One such scenario is the maintenance of mutual information on dynamic data. While we considered arbitrary insertions and deletions as possible changes, a specific case with practical relevance are sliding windows. Here, the goal is an up-to-date estimate of the mutual information for the most recent data points. That is, each change is exactly the insertion of a new point and the deletion of the least recently inserted point. As consequence of our lower bound for regular estimation, updating estimates for individual points takes at least time in $\Omega(\log n)$. Our observation for efficient updates is, that the impact of individual insertions and deletions on the estimation equations is relatively limited. While mathematically easy to define, correcting these changes is algorithmically challenging. This is because of the nearest-neighbor search that is necessary and the reverse nearest-neighbor search to find those points whose nearest neighbors have changed due to an insertion or deletion. The first data structure that we proposed, DEMI, utilizes only linear and binary search for this task and achieves a linear time complexity for insertions and deletions. With our second data structure, ADEMI, we substituted the linear searches in the algorithms. Specifically, we included two augmented search trees, i.e., a two-dimensional range tree and a two-dimensional segment tree. With these more complex data structures and a technique called fractional cascading, ADEMI is able to process updates to 3KL estimates with time in $O(\log n \log\log n)$. Our experiments showed that our data structures improve the computation time for updating estimation in sliding window by orders of magnitude.

Another option to improve the time efficiency of mutual information estimates are approximations. Our proposed method, IMIE, offers two additional benefits over naïve approximation approaches such as subsampling. First, IMIE offers anytime capability, i.e., IMIE offers a very rough estimate as quickly as possible and improves upon this result until it is interrupted or converges to the accurate result. This means that IMIE offers a dynamic time-quality tradeoff in the sense that a user can decide at runtime how much time they are willing to commit to this estimate. Second, in addition to approximate results, IMIE also provides statistical quality indicators that empower the user to make informed decision whether the current approximation suffices. These quality indicators enable statistical tests to determine the probability that the current approximation is more than some value above the exact KSG or 3KL estiamte. This is particularly useful if a user is interested in certain thresholds and estimates can be concluded prematurely because the accurate estimate remains below that threshold with sufficient probability. In our experiments, IMIE was faster than conventional estimation even if the permitted chance of error was 0%.

Lastly, we considered the task of explaining the dependencies in an interpretable manner. Specifically, we consider the task of summarizing the relationship between a designated *outcome* attribute and the remaining *explanatory* attributes. The requirements for such a summary are that they are concise, in a readable form and informative, which we determined through information theory. While there exist solutions for this problem, they are limited to categorical attributes. Our approach to this task, LENS, builds upon this existing work and supports ordinal and numerical attributes by including intervals instead of specific values as part of the summary. Finding these intervals is the main challenge for this task as numerical attributes may have big domains with even larger numbers of possible intervals. LENS is a heuristic that considers exponentially growing intervals originating at single values that are considered informative. To maintain time efficiency, we presented a data structure that prevents repeated scans of the data to evaluate the benefit of candidates for the summary. While there exist no other approaches for exactly this task, we used approaches from related tasks and adapted them as competing approaches. In our experiments, LENS outperforms these approaches both in time-efficiency and conciseness. LENS has also further proven its practical relevance with [VET+19], where we used the summaries to create interpretable machine descriptions.

Overall, this dissertation provides multiple new algorithms and data structures that make complex dependencies more accessible. DEMI and ADEMI reduce the computational burden for real-time mutual information monitoring by orders of magnitudes. IMIE enables anytime estimation of mutual information where users can assess the quality of early estimates, which is often multiple times faster than conventional estimation even with negligible error. LENS summarizes multivariate dependencies with numerical data faster and more concise than existing related approaches. Additionally, we have proven some new characteristics of nearest-neighbor based mutual information estimation and nearest neighbors. Finally, the insights and techniques of this dissertation may be the catalyst for new research. We present a small selection of possible direction in the following section.

6.2 Future Research Directions

Computational Complexity of Mutual Information Estimation While we provided a tight lower bound for 3KL and KSG estimation, open questions remain. For updating estimates, our solution nearly reaches the lower bound for the 3KL. Regarding the KSG, however, there remains a large gap between our algorithm and our lower bound for the problem. Notable, this gap exists because the limiting factor in computing the updates is not based on nearest neighbors which is otherwise the dominating challenge of these estimation techniques. Instead, the marginal counts are affected stronger by changes. This begs the question, whether it might be possible to prove formally that updating the KSG is harder than the 3KL, even though they have the same complexity for estimation on static data.

Another interesting consideration on the issue of complexity is, whether the bound can be extended to certain approximations. As a reminder, our constructions to reduce the problem *IntegerElementDistinctness* to the KSG and 3KL estimation were comparatively simple. Therefore, a more elaborate construction could also show the hardness of achieving certain approximation quality to the KSG or 3KL. If such results exist, the hardness might be expanded from the particular estimation method to the task of approximating mutual information in general with the results of Gao et al. [GOV18] on the convergence behavior of nearest-neighbor based estimation.

New Statistical Anytime Algorithms Recalling IMIE, one might notice that the only aspect of the KSG and 3KL that we utilize for the iterative construction is that they average over something that they compute for each point. This means that other algorithms might use the same construction, if the main burden of their computation also relies on the average or sum of something computed for each point. However, there are two things to consider. First, if the value computed for each point does not depend on the entire data set, the approximation may be identical to naïve subsampling. Second, the discrepancy in computational burden to compute this value for one point and computing this value for all points should not be too large. This is already noticeable with IMIE, where conventional estimation uses specialized data structures that find nearest neighbors must faster than linear search and faster than our lightweight search.

Depending on the task, it might be attractive to combine our iterative construction with techniques from *database cracking* [IKM07, SJD13], which tackle exactly the problem of building efficient data structures without front-loading the computational burden.

Mutual Information with Higher Dimensionality While there is no consensus how mutual information should be generalized for more than two attributes, the two variables could be multivariate themselves. For two bivariate variables, the KSG performs its nearest-neighbor search in the joint four-dimensional space and counts marginal counts based on rectangles in the two-dimensional marginal spaces [KSG04]. While it is easy to see how some of our results generalize to this case, for others it is unclear. For instance, DEMI relies on linear search and is barely affected by higher dimensions while the augmented trees for ADEMI scale unfavorable with increased dimensionality [dBCvKO08]. In contrast, it is unclear whether the lightweight search used by IMIE performs better or worse with increased dimensionality and how strong the impact of two-dimensional areas for marginal counts is.

Bibliography

[ABR19] Vadim Arzamasov, Klemens Böhm, and Ignaz Rutter. Minimizing bias in estimation of mutual information from data streams. In *International Conference on Scientific and Statistical Database Management SSDBM'19*, pages 1–12, 2019. doi:10.1145/3335783.3335796.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010. doi:10.1016/j.comnet.2010.05.010.

[AKBS12] Ira Assent, Philipp Kranen, Corinna Baldauf, and Thomas Seidl. Anyout: Anytime outlier detection on streaming data. In *Database Systems for Advanced Applications DASFAA'12*, pages 228–242, 2012. doi:10.1007/978-3-642-29038-1_18.

[AS72] Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, volume 55. Courier Corporation, tenth edition, 1972.

[Atz15] Martin Atzmueller. Subgroup discovery. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 5(1):35–49, 2015. doi:10.1002/widm.1144.

[AYL$^+$11] Fatemeh Amiri, Mohammad Mahdi Rezaei Yousefi, Caro Lucas, Azadeh Shakery, and Nasser Yazdani. Mutual information-based feature selection for intrusion detection systems. *J. Network and Computer Applications*, 34(4):1184–1199, 2011. doi:10.1016/j.jnca.2011.01.002.

[Bel62] C. B. Bell. Mutual information and maximal correlation as measures of dependence. *The Annals of Mathematical Statistics*, 33(2):587–595, 1962. doi:10.1214/aoms/1177704583.

[Ben77] Jon Louis Bentley. Algorithms for klee's rectangle problems. Technical report, Carnegie Mellon University, 1977.

[Ben83] Michael Ben-Or. Lower bounds for algebraic computation trees (preliminary report). In *ACM Symposium on Theory of Computing STOC'83*, pages 80–86, 1983. doi:10.1145/800061.808735.

[Ber97] Adam L. Berger. The improved iterative scaling algorithm: A gentle introduction. *Unpublished manuscript*, 1997.

[BH17] Jonathan Boidol and Andreas Hapfelmeier. Fast mutual information computation for dependency-monitoring on data streams. In *Symposium on Applied Computing, SAC'17*, pages 830–835, 2017. doi:10.1145/3019612.3019669.

[BKB00] Ivan Bruha, Pavel Kralik, and Petr Berka. Genetic learner: Discretization and fuzzification of numerical attributes. *Intell. Data Anal.*, 4(5):445–460, 2000. URL `http://content.iospress.com/articles/intelligent-data-analysis/ida00031`.

[BKD14] Brian Bole, Chetan Kulkarni, and Matthew Daigle. Adaptation of an electrochemistry-based li-ion battery model to account for deterioration observed under randomized use. 09 2014. URL `https://ti.arc.nasa.gov/c/25/`.

[BPP96] Adam L. Berger, Stephen Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[BTV$^+$18] Simon Bischof, Holger Trittenbach, Michael Vollmer, Dominik Werle, Thomas Blank, and Klemens Böhm. HIPE: an energy-status-data set from industrial production. In *International Conference on Future Energy Systems, e-Energy'18*, pages 599–603, 2018. doi:10.1145/3208903.3210278.

[CCA$^+$09] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009. doi:10.1016/j.dss.2009.05.016.

[CCM07] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. A near-optimal algorithm for computing the entropy of a stream. In *ACM-SIAM Symposium on Discrete Algorithms SODA'07*, pages 328–335, 2007. URL `http://dl.acm.org/citation.cfm?id=1283383.1283418`.

[CF16] Luis M. Candanedo and Véronique Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. *Energy and Buildings*, 112:28 – 39, 2016. doi:https://doi.org/10.1016/j.enbuild.2015.11.071.

[CFD17] Luis M. Candanedo, Véronique Feldheim, and Dominique Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings*, 140:81 – 97, 2017. doi:https://doi.org/10.1016/j.enbuild.2017.01.083.

[CH12] Peng-Ting Chen and Hsin-Pei Hsieh. Personalized mobile advertising: Its key attributes, trends, and social impact. *Technological Forecasting and Social Change*, 79(3):543 – 557, 2012. doi:https://doi.org/10.1016/j.techfore.2011.08.011.

[Cha06] Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *ACM-SIAM Symposium on Discrete Algorithms SODA'06*, pages 1196–1202, 2006. URL `http://dl.acm.org/citation.cfm?id=1109557.1109689`.

[CHM12] Yuan Cao, Haibo He, and Hong Man. SOMKE: kernel density estimation over data streams by sequences of self-organizing maps. *IEEE Trans. Neural Netw. Learning Syst.*, 23(8):1254–1268, 2012. doi:10.1109/TNNLS.2012.2201167.

[CNW$^+$12] Hafedh Chourabi, Taewoo Nam, Shawn Walker, José Ramón Gil-García, Sehl Mellouli, Karine Nahon, Theresa A. Pardo, and Hans Jochen Scholl. Understanding smart cities: An integrative framework. In *Hawaii International Conference on Systems Science (HICSS'12*, pages 2289–2297, 2012. doi:10.1109/HICSS.2012.615.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006. URL `http://www.elementsofinformationtheory.com/`.

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. URL `http://www.worldcat.org/oclc/227584184`.

[DFS13] Fabrizio Durante, Juan Fernández-Sánchez, and Carlo Sempi. A topological proof of sklar's theorem. *Appl. Math. Lett.*, 26(9):945–948, 2013. doi:10.1016/j.aml.2013.04.005.

[DG17a] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[DG17b] Dheeru Dua and Casey Graff. UCI machine learning repository - individual household electric power consumption data set, 2017. URL `https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption`.

[DL01] Luc Devroye and Gábor Lugosi. *Combinatorial methods in density estimation*. Springer series in statistics. Springer, 2001.

[Dur64] Richard Durstenfeld. Algorithm 235: Random permutation. *Commun. ACM*, 7(7):420, 1964. doi:10.1145/364520.364540.

[Eva08] Dafydd Evans. A computationally efficient estimator for mutual information. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 464, pages 1203–1215. The Royal Society, 2008. doi:10.1098/rspa.2007.0196.

[FB19] Edouard Fouché and Klemens Böhm. Monte carlo dependency estimation. In *International Conference on Scientific and Statistical Database Management, SSDBM'19*, pages 13–24, 2019. doi:10.1145/3335783.3335795.

[FG08] Paolo Ferragina and Antonio Gulli. A personalized search engine based on web-snippet hierarchical clustering. *Softw., Pract. Exper.*, 38(2):189–225, 2008. doi:10.1002/spe.829.

[FGS17] Guoyao Feng, Lukasz Golab, and Divesh Srivastava. Scalable informative rule mining. In *IEEE International Conference on Data Engineering, ICDE'17*, pages 437–448, 2017. doi:10.1109/ICDE.2017.101.

[FMXY12] X. Fang, S. Misra, G. Xue, and D. Yang. Smart grid — the new and improved power grid: A survey. *IEEE Communications Surveys Tutorials*, 14(4):944–980, 2012.

[GAG$^+$14] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. Interpretable and informative explanations of outcomes. *PVLDB*, 8(1):61–72, 2014. doi:10.14778/2735461.2735467.

[GFG$^+$18] Kareem El Gebaly, Guoyao Feng, Lukasz Golab, Flip Korn, and Divesh Srivastava. Explanation tables. *IEEE Data Eng. Bull.*, 41(3):43–51, 2018. URL `http://sites.computer.org/debull/A18sept/p43.pdf`.

[GLS+13] Salvador García, Julián Luengo, José Antonio Sáez, Victoria López, and Francisco Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Trans. Knowl. Data Eng.*, 25(4):734–750, 2013. doi:10.1109/TKDE.2012.35.

[GOV18] Weihao Gao, Sewoong Oh, and Pramod Viswanath. Demystifying fixed k-nearest neighbor information estimators. volume 64, pages 5629–5661, 2018. doi:10.1109/TIT.2018.2807481.

[GR09] Henrik Grosskreutz and Stefan Rüping. On subgroup discovery in numerical domains. *Data Min. Knowl. Discov.*, 19(2):210–226, 2009. doi:10.1007/s10618-009-0136-3.

[Gra18] Artur Gramacki. *Nonparametric kernel density estimation and its computational aspects.* Springer, 2018.

[GRD+11] Ling Guo, Daniel Rivero, Julian Dorado, Cristian R. Munteanu, and Alejandro Pazos. Automatic feature extraction using genetic programming: An application to epileptic EEG classification. *Expert Syst. Appl.*, 38(8):10425–10436, 2011. doi:10.1016/j.eswa.2011.02.118.

[HAMS97] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range queries in OLAP data cubes. pages 73–88, 1997. doi:10.1145/253260.253274.

[Has11] H. M. Hashemian. State-of-the-art predictive maintenance techniques. *IEEE Trans. Instrumentation and Measurement*, 60(1):226–236, 2011. doi:10.1109/TIM.2010.2047662.

[HCGdJ11] Francisco Herrera, Cristóbal J. Carmona, Pedro González, and María José del Jesús. An overview on subgroup discovery: foundations and applications. *Knowl. Inf. Syst.*, 29(3):495–525, 2011. doi:10.1007/s10115-010-0356-2.

[Hel16] Sumyea Helal. Subgroup discovery algorithms: A survey and empirical evaluation. *J. Comput. Sci. Technol.*, 31(3):561–576, 2016. doi:10.1007/s11390-016-1647-1.

[HMF+16] Ramon Huerta, Thiago Mosqueiro, Jordi Fonollosa, Nikolai F Rulkov, and Irene Rodriguez-Lujan. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems*, 157:169 – 176, 2016. doi:https://doi.org/10.1016/j.chemolab.2016.07.004.

[HNO08] Nicholas J. A. Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy via approximation theory. In *IEEE Symposium on Foundations of Computer Science, FOCS'08*, pages 489–498, 2008. doi:10.1109/FOCS.2008.76.

[HPS15] Nikolai Helwig, Eliseo Pignanelli, and Andreas Schütze. Condition monitoring of a complex hydraulic system using multivariate statistics. In *IEEE International Instrumentation and Measurement Technology Conference (I2MTC'15)*, pages 210–215, 2015. doi:10.1109/I2MTC.2015.7151267.

[HSPVB07] Katerina Hlaváčková-Schindler, Milan Paluš, Martin Vejmelka, and Joydeep Bhattacharya. Causality detection based on information-theoretic approaches in time series analysis. *Physics Reports*, 441(1):1 – 46, 2007. doi:https://doi.org/10.1016/j.physrep.2006.12.004.

[HZ01] Eric A. Hansen and Shlomo Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artif. Intell.*, 126(1-2):139–157, 2001. doi:10.1016/S0004-3702(00)00068-0.

[IKM07] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Database cracking. In *Conference on Innovative Data Systems Research CIDR'07*, pages 68–78, 2007. URL `http://cidrdb.org/cidr2007/papers/cidr07p07.pdf`.

[Jay57] E. T. Jaynes. Information theory and statistical mechanics. *Phys. Rev.*, 106:620–630, May 1957. doi:10.1103/PhysRev.106.620.

[JLJL16] Jian Jin, Ying Liu, Ping Ji, and Hongguang Liu. Understanding big consumer opinion data for market-driven product design. *International Journal of Production Research*, 54(10):3019–3041, 2016. doi:10.1080/00207543.2016.1154208.

[KA14] Justin B. Kinney and Gurinder S. Atwal. Equitability, mutual information, and the maximal information coefficient. *Proceedings of the National Academy of Sciences*, 111(9):3354–3359, 2014. doi:10.1073/pnas.1309933111.

[KBG$^+$07] Shiraj Khan, Sharba Bandyopadhyay, Auroop R. Ganguly, Sunil Saigal, David J. Erickson, Vladimir Protopopescu, and George Ostrouchov. Relative performance of mutual information estimation methods for quantifying the dependence among short and noisy data. *Phys. Rev. E*, 76:026209, Aug 2007. doi:10.1103/PhysRevE.76.026209.

[KBHJ08] Yuliya Kopylova, Duncan A. Buell, Chin-Tser Huang, and Jeff Janies. Mutual information applied to anomaly detection. *Journal of Communications and Networks*, 10(1):89–97, 2008. doi:10.1109/JCN.2008.6388332.

[Kel15] Fabian Keller. *Attribute Relationship Analysis in Outlier Mining and Stream Processing.* PhD thesis, Karlsruhe Institute of Technology, 2015. URL `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000048790`.

[KL87] LF Kozachenko and Nikolai N Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987.

[KM19] Kiran Karra and Lamine Mili. Copula index for detecting dependence and monotonicity between stochastic signals. *Inf. Sci.*, 485:18–41, 2019. doi:10.1016/j.ins.2019.02.007.

[KMB12] Fabian Keller, Emmanuel Müller, and Klemens Böhm. Hics: High contrast subspaces for density-based outlier ranking. In *IEEE International Conference on Data Engineering (ICDE'12*, pages 1037–1048, 2012. doi:10.1109/ICDE.2012.88.

[KMB15] Fabian Keller, Emmanuel Müller, and Klemens Böhm. Estimating mutual information on data streams. In *International Conference on Scientific and Statistical Database Management SSDBM'15*, pages 3:1–3:12, 2015. doi:10.1145/2791347.2791348.

[KS96] Sanjiv Kapoor and Michiel H. M. Smid. New techniques for exact and approximate dynamic closest-point problems. *SIAM J. Comput.*, 25(4):775–796, 1996. doi:10.1137/S0097539793259458.

[KS09] Philipp Kranen and Thomas Seidl. Harnessing the strengths of anytime algorithms for constant data streams. *Data Min. Knowl. Discov.*, 19(2):245–260, 2009. doi:10.1007/s10618-009-0139-0.

[KSG04] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Phys. Rev. E*, 69:066138, Jun 2004. doi:10.1103/PhysRevE.69.066138.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Conference on Neural Information Processing Systems NIPS'12*, pages 1106–1114, 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks`.

[L$^+$08] Guillaume Leduc et al. Road traffic data: Collection methods and applications. *Working Papers on Energy, Transport and Climate Change*, 1(55), 2008.

[Laz10] Nicole Lazar. Ockham's razor. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(2):243–246, 2010. doi:10.1002/wics.75.

[LBL16] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1675–1684, 2016. doi:10.1145/2939672.2939874.

[LHS13] David López-Paz, Philipp Hennig, and Bernhard Schölkopf. The randomized dependence coefficient. In *Conference on Neural Information Processing Systems NIPS'13*, pages 1–9, 2013. URL `http://papers.nips.cc/paper/5138-the-randomized-dependence-coefficient`.

[LQ65] Don O Loftsgaarden and Charles P Quesenberry. A nonparametric estimate of a multivariate density function. *The Annals of Mathematical Statistics*, 36(3):1049–1051, 1965. doi:10.1214/aoms/1177700079.

[LR91] Anna Lubiw and András Rácz. A lower bound for the integer element distinctness problem. *Inf. Comput.*, 94(1):83–92, 1991. doi:10.1016/0890-5401(91)90034-Y.

[MHF$^+$15] Son T. Mai, Xiao He, Jing Feng, Claudia Plant, and Christian Böhm. Anytime density-based clustering of complex data. *Knowl. Inf. Syst.*, 45(2):319–355, 2015. doi:10.1007/s10115-014-0797-0.

[MMM14] Ben Murrell, Daniel Murrell, and Hugh Murrell. R2-equitability is satisfiable. *Proceedings of the National Academy of Sciences*, 111(21):E2160–E2160, 2014. doi:10.1073/pnas.1403623111.

[MN90] Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(2):215–241, 1990. doi:10.1007/BF01840386.

[MR05] Oded Maimon and Lior Rokach, editors. *The Data Mining and Knowledge Discovery Handbook*. Springer, 2005.

[MTV11] Michael Mampaey, Nikolaj Tatti, and Jilles Vreeken. Tell me what i need to know: succinctly summarizing data with itemsets. In *ACM International Conference on Knowledge Discovery and Data Mining SIGKDD'11*, pages 573–581, 2011. doi:10.1145/2020408.2020499.

[NLM99] Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pages 61–67, 1999.

[PAS06] Steven J. Phillips, Robert P. Anderson, and Robert E. Schapire. Maximum entropy modeling of species geographic distributions. *Ecological Modelling*, 190(3):231 – 259, 2006. doi:https://doi.org/10.1016/j.ecolmodel.2005.03.026.

[PGS12] Barnabás Póczos, Zoubin Ghahramani, and Jeff G. Schneider. Copula-based kernel dependency measures. In *International Conference on Machine Learning ICML'12*, 2012. URL `http://icml.cc/2012/papers/417.pdf`.

[PK09] Angeliki Papana and Dimitris Kugiumtzis. Evaluation of mutual information estimators for time series. *International Journal of Bifurcation and Chaos*, 19(12):4197–4215, 2009. doi:10.1142/S0218127409025298.

[PLV02] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Conference on Empirical Methods in Natural Language Processing EMNLP'02*, 2002. URL `https://www.aclweb.org/anthology/W02-1011/`.

[PPL97] Stephen Della Pietra, Vincent J. Della Pietra, and John D. Lafferty. Inducing features of random fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(4):380–393, 1997. doi:10.1109/34.588021.

[PQB05] Ernesto Pereda, Rodrigo Quian Quiroga, and Joydeep Bhattacharya. Nonlinear multivariate analysis of neurophysiological signals. *Progress in Neurobiology*, 77(1):1 – 37, 2005. doi:https://doi.org/10.1016/j.pneurobio.2005.10.003.

[QGP10] Peng Qiu, Andrew J. Gentles, and Sylvia K. Plevritis. Reducing the computational complexity of information theoretic approaches for reconstructing gene regulatory networks. *Journal of Computational Biology*, 17(2):169–176, 2010. doi:10.1089/cmb.2009.0052.

[Qui87] J. Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987. doi:10.1016/S0020-7373(87)80053-6.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Rén59] Alfréd Rényi. On measures of dependence. *Acta mathematica hungarica*, 10(3-4):441–451, 1959. doi:10.1007/BF02024507.

[Ric06] John Rice. *Mathematical statistics and data analysis*, chapter Survey Sampling, pages 199–220. Nelson Education, 2006.

[Rok09] Lior Rokach. *Pattern Classification Using Ensemble Methods*, volume 75 of *Series in Machine Perception and Artificial Intelligence.* WorldScientific, 2009. doi:10.1142/7238.

[RRF$^+$11] David N. Reshef, Yakir A. Reshef, Hilary K. Finucane, Sharon R. Grossman, Gilean McVean, Peter J. Turnbaugh, Eric S. Lander, Michael Mitzenmacher, and Pardis C. Sabeti. Detecting novel associations in large data sets. *Science*, 334(6062):1518–1524, 2011. doi:10.1126/science.1205438.

[Rug04] Salvatore Ruggieri. Yadt: Yet another decision tree builder. In *IEEE International Conference on Tools with Artificial Intelligence ICTAI'04*, pages 260–265, 2004. doi:10.1109/ICTAI.2004.123.

[Sar01] Sunita Sarawagi. User-cognizant multidimensional analysis. *VLDB J.*, 10(2-3):224–239, 2001. doi:10.1007/s007780100046.

[Sch95] Thomas Schreiber. Efficient neighbor searching in nonlinear time series analysis. *International Journal of Bifurcation and Chaos*, 05(02):349–358, 1995. doi:10.1142/S0218127495000296.

[SG05] Mirco Speretta and Susan Gauch. Personalized search based on user search histories. In *International Conference on Web Intelligence WI'05*, pages 622–628, 2005. doi:10.1109/WI.2005.114.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. doi:10.1002/j.1538-7305.1948.tb01338.x.

[SHR$^+$07] Antti Sorjamaa, Jin Hao, Nima Reyhani, Yongnan Ji, and Amaury Lendasse. Methodology for long-term prediction of time series. *Neurocomputing*, 70(16-18):2861–2869, 2007. doi:10.1016/j.neucom.2006.06.015.

[Šid67] Zbyněk Šidák. Rectangular confidence regions for the means of multivariate normal distributions. *Journal of the American Statistical Association*, 62(318):626–633, 1967. doi:10.1080/01621459.1967.10482935.

[Sil86] Bernard W. Silverman. *Density Estimation for Statistics and Data Analysis*. Springer, 1986. doi:10.1007/978-1-4899-3324-9.

[SJD13] Felix Martin Schuhknecht, Alekh Jindal, and Jens Dittrich. The uncracked pieces in database cracking. *PVLDB*, 7(2):97–108, 2013. doi:10.14778/2732228.2732229.

[Skl59] M Sklar. Fonctions de repartition an dimensions et leurs marges. *Publ. inst. statist. univ. Paris*, 8:229–231, 1959.

[SMH$^+$03] Harshinder Singh, Neeraj Misra, Vladimir Hnizdo, Adam Fedorowicz, and Eugene Demchuk. Nearest neighbor estimates of entropy. *American journal of mathematical and management sciences*, 23(3-4):301–321, 2003. doi:10.1080/01966324.2003.10737616.

[SRB$^+$07] Gábor J Székely, Maria L Rizzo, Nail K Bakirov, et al. Measuring and testing dependence by correlation of distances. *The annals of statistics*, 35(6):2769–2794, 2007.

[VB19] Michael Vollmer and Klemens Böhm. Iterative estimation of mutual information with error bounds. In *International Conference on Extending Database Technology EDBT'19*, pages 73–84, 2019. doi:10.5441/002/edbt.2019.08.

[VET$^+$19] Michael Vollmer, Adrian Englhardt, Holger Trittenbach, Pawel Bielski, Shahab Karrari, and Klemens Böhm. Energy time-series features for emerging applications on the basis of human-readable machine descriptions. In *International Conference on Future Energy Systems e-Energy'19*, pages 474–481, 2019. doi:10.1145/3307772.3331022.

[VGBS19] Michael Vollmer, Lukasz Golab, Klemens Böhm, and Divesh Srivastava. Informative summarization of numeric data. In *International Conference on Scientific and Statistical Database Management SSDBM'19*, pages 97–108, 2019. doi:10.1145/3335783.3335797.

[VH07] Martin Vejmelka and Katerina Hlavácková-Schindler. 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*, pages 790–797, 2007. doi:10.1007/978-3-540-71618-1_88.

[VL18] Sebastián Ventura and José María Luna. Subgroup discovery. In *Supervised Descriptive Pattern Mining*, pages 71–98. Springer, 2018.

[vLK12] Matthijs van Leeuwen and Arno J. Knobbe. Diverse subgroup set discovery. *Data Min. Knowl. Discov.*, 25(2):208–242, 2012. doi:10.1007/s10618-012-0273-y.

[VRB18] Michael Vollmer, Ignaz Rutter, and Klemens Böhm. On complexity and efficiency of mutual information estimation on static and dynamic data. In *International Conference on Extending Database Technology EDBT'18*, pages 49–60, 2018. doi:10.5441/002/edbt.2018.06.

[Wel62] BP Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962. doi:10.1080/00401706.1962.10490022.

[Wil85] Dan E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 14(1):232–253, 1985. doi:10.1137/0214019.

[WJ95] Matt P. Wand and M. Chris Jones. *Kernel Smoothing*. Springer, 1995. doi:10.1007/978-1-4899-4493-1.

[WL09] Janett Walters-Williams and Yan Li. Estimation of mutual information: A survey. In *Rough Sets and Knowledge Technology RSKT'09*, pages 389–396, 2009. doi:10.1007/978-3-642-02962-2_49.

[WW39] Abraham Wald and Jacob Wolfowitz. Confidence limits for continuous distribution functions. *The Annals of Mathematical Statistics*, 10(2):105–118, 1939. doi:10.1214/aoms/1177732209.

[YWKT07] Ying Yang, Geoffrey I. Webb, Kevin B. Korb, and Kai Ming Ting. Classifying under computational resource constraints: anytime classification using probabilistic estimators. *Machine Learning*, 69(1):35–53, 2007. doi:10.1007/s10994-007-5020-z.

[ZCWQ03] Aoying Zhou, Zhiyuan Cai, Li Wei, and Weining Qian. M-kernel merging: Towards density estimation over data streams. In *International Conference on Database Systems for Advanced Applications DASFAA'03*, pages 285–292, 2003. doi:10.1109/DASFAA.2003.1192393.

[Zha00] Guoqiang Peter Zhang. Neural networks for classification: a survey. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 30(4):451–462, 2000. doi:10.1109/5326.897072.

[Zil96] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996. URL `http://www.aaai.org/ojs/index.php/aimagazine/article/view/1232`.