Privacy-preserving AI using declarative constraints

Moitree Basu

Doctoral school of Mathematics and Digital Sciences (MADIS)

Doctoral Dissertation

Privacy-preserving AI using declarative constraints

Specialization : Computer Science and Applications

prepared and publicly defended by

Moitree Basu

to obtain the degree of
Doctorate from Lille University

at Villeneuve d'Ascq, France
on April 4th, 2024

Theses defended before the jury composed of

Patrick Baillot	Research Director, CNRS Lille	President
Jean-Francois Couchot	Professor, FEMTO-ST Besançon	Reviewer
Oana Goga	Researcher, CNRS Paris	Examiner
Zied Bouraoui	Associate Professor HDR, Université d'Artois	Reviewer
Jan Ramon	Research Director, INRIA Lille	Supervisor

École Doctorale Mathematiques, Sciences du Numérique et de leurs Interactions (MADIS)

Thèse de Doctorat

IA préservant la vie privée à l'aide de contraintes déclaratives

---

Spécialisation : Informatique et applications

préparée et défendue publiquement par

Moitree Basu

pour obtenir le grade de
Docteur de l'Université de Lille

à Villeneuve d'Ascq, France
le 4 Avril 2024

Thèse soutenue devant le jury composé de

Patrick Baillot	Directeur de recherches, CNRS Lille	Président
Jean-Francois Couchot	Professeur, FEMTO-ST Besançon	Rapporteur
Oana Goga	Chargée de recherches, CNRS Paris	Examinatrice
Zied Bouraoui	Maître de conférences HDR, Université d'Artois	Rapporteur
Jan Ramon	Directeur de Recherches, INRIA Lille	Directeur

In loving memory of
my father.

Declarations

Funding. I'm partially funded by the Région Hauts de France and ANR PAMELA (ANR-16-CE23-0016-01).

Acknowledgements

I would like to thank everyone who, directly or indirectly, helped me in the journey of walking towards this PhD.

First, I would like to thank my supervisor Dr. Jan Ramon for his guidance and supervision. He has been not only a mentor to me, but also a wise critic of my progress during this whole time. He is often very specific about doing some things, and never accepts anything less than that, which often pushed me to explore more. I think, his meticulousness brought the best out of me, every time.

I thank my reviewers, juries, and examiners, Prof. Patrick Baillot, Prof. Jean-Francois Couchot, Dr. Oana Goga, and Dr. Zied Bouraoui, for spending their precious time to read my dissertation, giving feedback to improve the content, and attending the defense. Their opinion and questions are a strong guiding force for me towards making the dissertation better.

I thank my supervisors of my Masters thesis, Dr. Ingmar Steiner and Prof. Dietrich Klakow, for building the foundation of research and also for their recommendation letter for the PhD position.

I thank Marc Tommasi for being an amazing leader to the Magnet (MAchine learninG in information NETworks) team, of which I was a part. I spent the majority of my time in the labs of Inria, and I want to thank all my former and current officemates for making the office space a positive and reflective atmosphere. I thank them all, with whom I shared my workspace every day and ideas often. I was also associated with the Cristal Lab of the University of Lille, and I would like to thank them.

The members with whom I worked more closely are Dr. Carlos Cotrini (Post Doctoral Fellow), Pradipta Deb (Research Engineer), and Arijus Pleska (PhD Student). I would like to extend my gratitude to them for having my back whenever needed and being more than a colleague to me. Carlos helped me in understanding academic writing far more than I could have expected in the short time we worked together. He brought a fresh perspective to my project as he joined further down the line, and that helped me grow in new dimensions. Pradipta attended so many meetings with me, and a lot of ideas were conceived when we brainstormed together. He taught me to question everything and defend my own ideas unapologetically. Arijus and I jointly put effort into a project which apart from directly affecting my thesis, helped me absorb and realize a lot of concepts in more depth. We were so different in our approach, that I learned to address differences of opinion in my work and be a team-player. I thank Marc and Mikaela for their constructive feedback to make my defense slides more professional, concise, and attractive.

I thank the Euraxess center, and Philippe Ducamp for finding the first apartment in France and for helping me settle with assistance to a lot of administrative responsibilities.

I would also like to thank the HR-team of Inria Lille for their enormous support throughout the process. I thank them for helping with all documentation and for answering all my questions about the many PhD formalities. I would

also like to thank the 'Agence nationale de la recherche' and region of Hauts de France for supporting me financially. I want to thank all the other members of the group and administrations for the company and assistance during academic trips and events.

I convey my eternal gratitude to my parents, Shyamali Basu and Sumitava Basu for being the two pillars of my life since day one and providing me with everything they could, and sometimes even more. Then I am forever grateful to my life partner Pradipta Deb for supporting me unconditionally, suffering through every hardship with me, being the cheerleader on the hardest days, and the torchbearer on the darkest paths.

I thank my friends in different countries and continents who often brought the light breeze of laughter and fun. Everyone who believed in me at some points, or was doubtful in my capabilities, has helped me in some way or the other.

I thank everyone who supported me during the Covid-19 pandemic, mental ups-and-downs, and assisted in relaxation and sustenance for years. It would not have been possible to reach here without every one of your big or small contribution.

Thanks to all for everything.

Best wishes,
Moitree Basu

Abstract

Machine learning and Deep learning-based technologies have gained widespread adoption, quickly displacing traditional artificially intelligent (AI) systems. Contemporary computers are remarkable in processing enormous amounts of personal data through these machine learning (ML) algorithms. However, this technological advancement brings along significant privacy implications, and this problem can only be expected to escalate in the foreseeable future.

Studies have shown that it is possible to deduce sensitive information from statistical models computed on datasets, even without direct access to the underlying training dataset. Apart from the privacy-related concerns regarding statistical models, the complex systems learning and employing such models are increasingly difficult for users to understand, and so are the ramifications of consenting to the submission and use of their private information within such frameworks. Consequently, transparency and interpretability emerged as pressing concerns.

In this dissertation, we study the problem of specifying privacy requirements for machine learning based systems, in a manner that combines interpretability with operational feasibility. Explaining privacy-improving technology is a challenging problem, especially when the objective is to construct a system that at the same time is interpretable and has a high utility. In order to address this challenge, we propose to specify privacy requirements as constraints, thereby allowing for both interpretability and automated optimization of the utility.

Keywords: **Privacy-preservation, Differential privacy, Logical Bayesian Networks, Specification language, Verifiability, Interpretability, Probabilistic predicates, Logical predicates**

Résumé

Les technologies basées sur l'apprentissage automatique et l'apprentissage profond ont été largement adoptées, supplantant rapidement les systèmes traditionnels d'intelligence artificielle (IA). Les ordinateurs modernes sont remarquables pour traiter d'énormes quantités de données personnelles grâce à ces algorithmes d'apprentissage automatique. Toutefois, cette avancée technologique a des répercussions importantes sur la vie privée, et on ne peut que s'attendre à ce que ce problème s'aggrave dans un avenir proche.

Des études ont montré qu'il est possible de déduire des informations sensibles à partir de modèles statistiques calculés sur des ensembles de données, même sans accès direct à l'ensemble de données d'apprentissage sous-jacent. Outre les préoccupations liées à la protection de la vie privée concernant les modèles statistiques, les systèmes complexes qui apprennent et utilisent ces modèles sont de plus en plus difficiles à comprendre pour les utilisateurs, tout comme les ramifications du consentement à la soumission et à l'utilisation de leurs informations privées dans de tels cadres. Par conséquent, la transparence et l'interprétabilité sont devenues des préoccupations majeures.

Dans cette thèse, nous étudions le problème de la spécification des exigences en matière de protection de la vie privée pour les systèmes basés sur l'apprentissage automatique, d'une manière qui combine l'interprétabilité et la faisabilité opérationnelle. Expliquer une technologie améliorant la protection de la vie privée est un problème difficile, en particulier lorsque l'objectif est de construire un système qui soit à la fois interprétable et d'une grande utilité. Afin de relever ce défi, nous proposons de spécifier les exigences en matière de protection de la vie privée sous forme de contraintes, ce qui permet à la fois l'interprétabilité et l'optimisation automatisée de l'utilité.

Mots-clés : **Préservation de la vie privée, confidentialité différentielle, réseaux bayésiens logiques, langage de représentation, vérifiabilité, interprétabilité, prédicats probabilistes, prédicats logiques**

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last decade, the cost of storage space has dropped drastically while computational power has surged, resulting in an explosion of data. Concurrently, machine learning and deep learning-based technologies have seen extraordinary usage, rapidly supplanting traditional rule-based AI systems. These modern computers demonstrate extraordinary proficiency in processing massive amounts of personal data, albeit with considerable privacy concerns.

In today's world, a better understanding of these technologies and their potential misuse is critical. This issue's trajectory is predicted to worsen in the near future. Studies have revealed the potential of deducing sensitive information from statistical outputs provided by machine learning models, so simply storing data securely is no longer sufficient. Similar deductions can be made even when one does not have direct access to the underlying training dataset, as demonstrated in prior research. Moreover, the growing complexity of these systems renders it increasingly challenging for users to comprehend the implications of consenting to the submission and utilization of their private information within such frameworks.

Chapter outline: *Next, in Section 1.1, we will state our main objectives. Then, in Section 1.2, we will elaborate on our desired properties, followed by a summary of contributions in Section 1.3. Finally, we will outline in Section 1.4 how the body of our work and the findings of our studies are organized in the next chapters.*

1.1 Objectives

The rising complexity in machine learning algorithms, and intelligent computing systems deploying them to train sensitive personal data, provides an exposure to fraudulent actors to deduce private information from individuals, often with negative consequences. A body of literature has discussed techniques for re-identifying individuals by combining data from various sources, such as voter registration records, de-identified hospital records [10], or by determining an

individual's inclusion in a machine learning training dataset solely through interactions with the trained model [115], [96].

Many inference tasks possess the potential for producing adverse results, even without access to the training dataset. In such cases, it is critical to protect the privacy of the participants as well as any other entities that may be vulnerable to such risks. So we aim to design a framework that preserves the privacy of individuals and protects their sensitive information from being revealed. We emphasize that our principal concern of implementing more stringent privacy-guarantee risks yielding lower utility of results.

We would also like to aim for a more automated design of such a framework, by eliminating the requirement of making lower-level engineering choices, and focusing on the privacy requirements while optimizing for utility.

Finally, the ability to articulate the privacy assurances granted by a system is extremely useful in bridging the often problematic communication gap between computer scientists who develop solutions and legal experts seeking to grasp these guarantees without being bogged down by intricate technical details. Our framework's goal is to create a system that allows us to describe our privacy requirements and learn a model while meeting those specifications.

Objective 1: Our objective will be to *design an unambiguous and interpretable declarative language* to specify the privacy requirements.

Objective 2: Another objective is to *design an automated utility maximizing privacy preservation framework* from the specifications of privacy requirements.

1.2 Desired properties

This section emphasizes that the framework we intend to develop must have certain properties like: *privacy*, *interpretability*, *utility*, *transparency*, *verifiability*, and *automation*. We discuss the need for these properties and argue how the state of the art still falls short in achieving some or all of them.

1.2.1 Privacy

To begin, let us consider the **first property**.

Preserving the privacy of persons or organizations within a dataset is a key concern and a prominent topic of research. This includes protecting individuals from inadvertently exposing sensitive information and examining the nature of privacy assurances provided by a system.

Before delving deeper into the privacy debate, it is critical to understand that our premise assumes the presence of a semi-honest or *honest-but-curious (HBC)* adversary with access to the results of the inference process applied to the datasets' contents, as defined by [102]. An HBC adversary is defined in this context as a genuine participant in a communication protocol who follows the prescribed protocol without deviation, while actively seeking to harvest all possible information from legitimately received messages. Furthermore, we contend that an honest but curious adversary refrains from colluding, thereby

avoiding the exchange of information with other parties. Given this scenario, it is crucial to put in place adequate measures that preserve the dataset's privacy and prevent the inadvertent disclosure of sensitive information to such individuals.

Privacy-preserving procedures usually use encryption techniques [32] or insert noise into the data [43]. When using the latter approach, determining the appropriate noise level required to establish privacy guarantees could prove exceedingly difficult.

Furthermore, a substantial subset of these technologies is capable of exclusively verifying privacy attributes, necessitating the end-user's responsibility to design privacy-preserving techniques. However, this requires expertise in both cryptography and statistics, making the creation of privacy-preserving systems a complicated and potentially error-prone undertaking. Additionally, many solutions have limits, especially addressing certain aspects of privacy, such as information confidentiality or individual anonymity. As a result, there is an urgent need for solutions that provide more comprehensive and holistic privacy assurance.

1.2.2 Utility

Next, let us consider the **second property**.

Data privacy protection through privacy-preserving methods usually imposes a trade-off with utility [135]. The adoption of privacy safeguards either frequently results in increased computing costs or a reduction in the precision of computation results. For example, the combination of many techniques, as mentioned in [124], is known to result in a perceptible deterioration in the quality of output in the field of differential privacy.

As a result, solutions that strike a suitable balance between privacy and utility are in high demand. Ultimately, acquiring relevant insights from inference work is crucial. At the end of a study, if the output of a machine learning model lacks significant value and fails to provide insights into the underlying database on which it was trained, all other desirable qualities become ineffective.

1.2.3 Interpretability

Now, let's address the **third property**.

As system complexity grows, users face an increasingly difficult challenge: understanding the consequences of disclosing information to these intricate systems. In response to the intricate web of privacy issues woven by such advanced systems, interpretability-driven solutions have emerged. A subset of these solutions makes use of formal frameworks to assist users in gaining a complete grasp of the privacy issues connected with communications with a certain system [14], [131], [101].

Regrettably, these existing methods still have limitations in expressiveness, particularly in handling the complexities of modern systems that combine diverse approaches such as machine learning, differential privacy, and secure multiparty computation. A key restriction of present systems is their ability to pro-

vide mere ‘verification’ - that is, the capability of establishing a system’s privacy. Declarative specifications, on the other hand, can provide a greater range of capabilities, such as analysis, automated solution construction, parameter optimization, privacy strategy selection, and more.

The domain of interpretability encompasses several dimensions, including a nuanced understanding of the terms and conditions outlined in privacy policies, a thorough grasp of contractual obligations before consent, a contextual appreciation of the concept of privacy, and verification of the algorithm’s efficacy in upholding privacy guarantees. It also refers to the mechanisms by which the algorithm implements these assurances.

The need for openness comes in light of legislative developments, such as the EU General Data Protection Regulation (GDPR) [2, 65]. This rule specifies that users have the right to know how their data is used and what measures are in place to protect their personal information. In addition to that, explaining privacy protection measures is critical since it encourages confidence among system users.

1.2.4 Transparency

In addition to the three primary properties previously discussed, we also attain several desirable properties indirectly. One of them, transparency, is the **fourth property**.

Transparency implies that privacy conditions are unambiguous and easily understood by all parties involved, including data collection agents, those with limited access to the data, individuals capable of running queries and accessing the outcomes, and participants in the study. It mandates that all stakeholders have access to privacy terms and associated protocols, which may include unrestricted access to all source code.

In the traditional centralized model, a central data user combines all data and performs computations based on their specific requirements. As the number of participants in a communication increases, so does the complexity of the process. In contrast, in a decentralized system, all players must participate in the complete spectrum of calculations, depending on the individual protocol used (e.g., due to the encryption requirements). As a result, unless a "data-sharing" protocol is used to transfer the data, either in its original or more obfuscated form, the data user cannot prevent the fact that data subjects become aware of the nature of the computations. Transparency, particularly in such decentralized circumstances, reduces the possibility of privacy conflicts. Numerous multi-party protocols improve transparency by facilitating the disclosure of computations to all involved parties.

1.2.5 Verifiability

The **fifth property** is verifiability.

The utilization of a language for representing system components should also help with the property of verifiability. When the information process is

represented using such a language, the various stakeholders involved in the process must be able to authenticate the privacy assurances offered by their counterparts. This feature enables companies to ensure unequivocal data privacy protection throughout the entire process pipeline. In this context, verification implies being able to validate that all privacy criteria, as expressed in specifications, are met.

The development and application of such a language in analogous environments has the potential to automate the modeling of information process pipelines, as well as the verification of their privacy-preserving properties and the detection of any violations thereof. The use of such automation has the potential to significantly improve the efficacy and effectiveness of the review and verification processes, resulting in significant time, effort, cost, and resource savings.

1.2.6 Automation

The last one, automation, is the **sixth property**.

Automation has the potential to streamline and reduce the effort required to make lower-level design decisions. As a result, engineers may concentrate their efforts entirely on the definition of system requirements, without having to consider implementation alternatives.

The optimal choice can be determined automatically by solving the inference issue described in Chapter 3 and 4. Once the requirements are effortlessly integrated into the language templates and tuned within the inference mechanism, the optimization process will eliminate the need to address additional, trivial design considerations.

1.3 Contribution

The fundamental goal of this thesis is to pursue the development of privacy-preserving systems while adhering to the aforementioned properties, as stated in the objectives in Section 1.1.

Based on the preceding discussion, it is evident that the current solutions exhibit shortcomings in meeting the desired properties. They either provide insufficient privacy measures, or result in significant utility losses, or fail to accurately characterize modern complex information systems. Furthermore, many frequently used solutions are largely focused on verifying privacy features. In response to the need to improve user privacy while also improving interpretability for complex systems, we propose the implementation of a unique approach that includes the following important components.

Our scientific contributions are divided into five distinct areas:

1. **Designing a privacy specification language**
   Our first contribution is the development of a novel, robust, and com-

prehensive specification language designed for specifying information processing pipelines and the privacy requirements that accompany them. The fundamental goal for this language's design is to be consistent, with an emphasis on the interpretability, verifiability, and transparency of its declarations. Furthermore, we intend to evaluate this language across a variety of real-world circumstances to enable the formalization of natural language texts into a probabilistic framework. This transformation aims to substantially reduce the uncertainty that is sometimes inherent in the use of natural language within the privacy policies of websites, corporate documents, legislative texts, and other legal documents.

2. **Development of an analysis framework**
   Our second contribution focuses on the introduction of an analysis framework based on the aforementioned language that is capable of checking and certifying compliance with privacy constraints. The primary goal is to assess the level of privacy preservation provided by complex information systems using this privacy specification language.

3. **Automated synthesizer for privacy-preserving mechanisms**
   Our third contribution concentrates on the development of an automated synthesizer capable of developing privacy-preserving mechanisms based on a set of specified strategies. This program tries to improve algorithmic performance while adhering to privacy requirements. In this case, we aim to use the constraint programming or constraint optimization problem (CP) approach. It is imperative that the language aligns with this contribution while maintaining interpretability without sacrificing the algorithm's performance or utility. Our approach involves systematic problem formulation and treating privacy requirements as constraints within an optimization problem, with the objective function representing utility or, conversely, loss. The loss function accounts for a variety of cost factors, including predicted output inaccuracy, computational costs, and storage requirements.

4. **Composition strategy for multiple modules**
   Our fourth contribution proposes a composition strategy for multiple modules that utilizes the synthesizer's skills in constructing privacy-preserving mechanisms based on a specified set of constraints. In this case, the user only needs to define the privacy requirements and a selection of strategies (such as secure multi-party computing techniques or differential-privacy-based noise mechanisms), and the synthesizer decides on the best approach for them. It also computes the optimal parameters associated with the selected approach.

5. **Tailored noise mechanism**
   The tailored noise mechanism (TNM), our fifth contribution, explores the complexities of dealing with highly non-linear attribute transformations within a given dataset. First, we develop a noise mechanism for

privatizing features by solving a constraint problem in such a way that only relevant intervals of feature transformations are selected while minimizing variance and bias in the privatization noise. Second, we look at two approaches to creating constraint programs: (a) transforming attributes into features before privatization, and (b) privatizing attributes before transforming them into features. Following that, we use the approach described in our previous contribution to select the superior constraint optimization problem, thereby providing the best alternative.

To demonstrate the power of our solution, we illustrate how it can model a scenario based on a distributed setting of medical centers. We formalize and analyze privacy requirements for patients, hospital staff, and medical centers. Afterward, we show how to automatically construct mechanisms that compute statistics about the patients, while following these privacy requirements.

1.4 Thesis outline

Chapter 1 commences with a concise **introduction** to the thesis.

Chapter 2 describes the **background** notations, definitions, basic concepts, and state-of-the-art for this dissertation.

Chapter 3, is based on our work on "**Interpretable privacy with optimizable utility**" [104].

Chapter 4, presents the different key components of the **specification language** in detail, with examples on the purview of a real-world problem. We explore how **inference** tasks can optimize utility while preserving privacy when equipped with comprehensive problem descriptions and process specifications.

Chapter 5 discusses **tailored noise mechanism**, explaining how to design the noise distribution tailored for special situations where traditional noise mechanisms fall short.

Chapter 6 addresses potential directions for **future research** within the various areas examined in this thesis.

Chapter 7 provides a comprehensive **thesis conclusion** by summarizing the content, followed by an exhaustive **bibliography** comprising all the citations used in this document.

Chapter 2

Background

This work sits at the crossroads of multiple fields of study. Each of them is vast and complex in its own way. Understanding our work requires some knowledge of all these fields, some very basic and some in-depth. So, before we can jump onto our projects, we must look thoroughly at the basics of the three main pillars:

- Machine learning
- Data privacy
- Logic theory

Before delving into all of the principles of the aforementioned three domains, we remark that we will be using concepts of **statistics**, **probability**, **linear algebra**, and **CVXOPT**.

Linear algebra We will commonly use the underlying ideas provided by Linear algebra to elucidate the formulation of the constraint optimization problem. We investigate fundamental concepts such as scalar, vector, matrix, determinant of a matrix, square matrix, trace of a matrix, transpose of a matrix, inverse of a matrix, identity matrix, eigenvalue, eigenvector, eigenvalue decomposition, singular value decomposition, and linear, quadratic, cubic, or higher order systems in this context. It is worth noting that, while these notions are used in our derivations, as we are not conceptually going to use Linear Algebra, we are omitting the detailed discussion of them in this summary.

Following that, our discussion gets into a technical review of key topics from statistics and probability theory. A comprehensive understanding of these concepts is imperative for grasping the foundational principles upon which our ideas are built.

Most of the contents in the following Sections 2.1, and 2.2 have been directly inspired by some websites [1].

[1] probabilitycourse.com, wikipedia.org, britannica.com, encyclopediaofmath.org

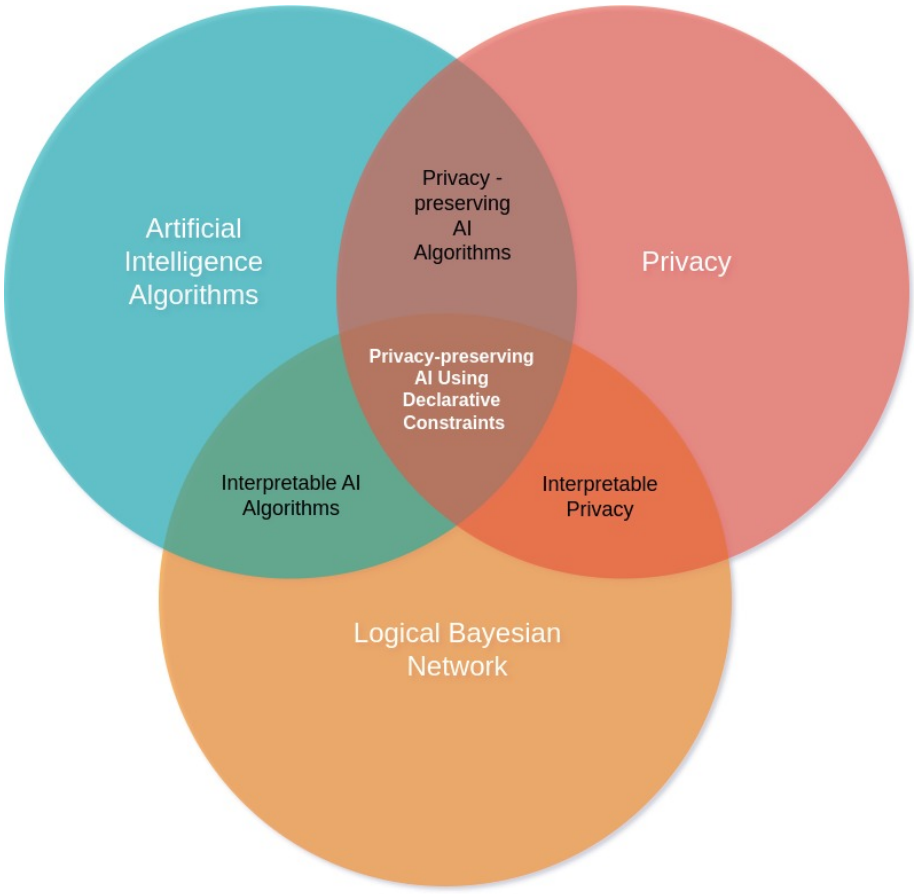


Figure 2.1: The niche of the dissertation topic at the intersection of modules.

2.1 Statistics

In this dissertation, a variety of statistical languages will be used in the explanation of the features covering machine learning and privacy, demanding a clear exposition. To help with this, we drew on relevant literature sources [38, 69, 88, 120, 125, 127] to create a concise explanation of the principles discussed in this section.

2.1.1 Random variable

A random variable (RV) X, is a measurable function $X : \Omega \to E$, mapping from a sample space Ω (comprising a set of potential outcomes) to a measurable space E.

The probability that the random variable X assumes a value within a measurable set $S \subseteq E$ is formally defined as follows:

$$\Pr(X \in S) = \Pr(\omega \in \Omega \mid X(\omega) \in S) \tag{2.1}$$

Random variables can have many variations: independent, dependent, correlated, etc. They can also be classified as discrete, continuous, or mixed types, depending on the probability distribution of the random variable.

2.1.2 Mean

In statistics, given a dataset, **mean** is used to summarize the dataset for a better and more holistic understanding. Mean can be of different types.

- **Arithmatic mean** often referred to simply as the mean, is computed as the sum of all values within a given list or set of numbers, divided by the total number of items contained in that specific set or list. Consequently, when considering a list of n numbers denoted as $x_1, x_2, \ldots, x_n$, the arithmetic mean is formally defined as follows:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + ... + x_n}{n} \tag{2.2}$$

- **Geometric mean** is a mathematical measure computed as the product of all values within a given list or set of numbers, raised to the power of the reciprocal of the total number of items contained in that specific set or list. Therefore, in the context of a list consisting of n numbers, denoted as $x_1, x_2, \ldots, x_n$, the geometric mean is formally defined as,

$$\bar{x} = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = (x_1 x_2 x_3 \cdots x_n)^{\frac{1}{n}} \tag{2.3}$$

- **Harmonic mean** is calculated as the reciprocal of the sum of all the reciprocals of the values within a given list or set of numbers. This result is then multiplied by the total number of items contained in that specific set or list. Thus, when considering a list composed of n numbers, denoted as $x_1, x_2, \ldots, x_n$, the formal definition of the harmonic mean is articulated as follows:

$$\bar{x} = n\sum_{i=1}^{n} \left(\frac{1}{x_i}\right)^{-1} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} \tag{2.4}$$

Besides these common ones, many definitions of mean exist and they are also computed in different contexts, like the mean of a function, weighted mean, mean of a probability density function, etc.

2.1.3 Expectation

The weighted average of a large yet finite number of independently selected observations driven by the outcome of a random variable is generally called the expectation of that random variable. For discrete random variables, we can compute the weighted arithmetic mean, whereas for continuous random variables, we take an integral over the random variable.

If the random variable is named X, then the expectation will be denoted as $\mathbb{E}(X)$ in our text. If the independently selected finite list of possible outcomes is $x_1, x_2, \cdots, x_l$, and the list of their corresponding probability of occurrence is $p_1, p_2, \cdots, p_l$, then the expectation of X is defined as,

$$\mu = \mathbb{E}(X) = \sum_i x_i pi = x_1 p_1 + x_2 p_2 + \cdots + x_l p_l, \tag{2.5}$$

where, $p_1 + p_2 + \cdots + p_l = 1$. Some basic rules around expectation are:

- Non-negativity of expectation: If $X > 0$, then $\mathbb{E}[X] \geq 0$.
- Linearity of expectation:
  - $\mathbb{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbb{E}[X_i]$, where $X_i (1 \leq i \leq n)$ are random variables.
  - $\mathbb{E}[\sum_{i=1}^{n} a_i X_i] = \sum_{i=1}^{n} a_i \mathbb{E}[X]$, where $a_i (1 \leq i \leq n)$ are constants and $X_i (1 \leq i \leq n)$ are random variables.
- Monotonicity of expectation: If $X \leq Y$ then $\mathbb{E}[X] \leq \mathbb{E}[Y]$, given $\mathbb{E}[X]$ and $\mathbb{E}[Y]$ exist.
- Non-degeneracy of expectation: If $\mathbb{E}[|X|] = 0$, then $X = 0$.
- Non-multiplicativity of expectation: $\mathbb{E}[XY] \neq \mathbb{E}[X] \cdot \mathbb{E}[Y]$, if X and Y are dependent, but $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$, if X and Y are independent.

2.1.4 Variance

Variance is the expectation of the squared deviation of a random variable, a probability distribution, or a dataset from the mean or expected value of the sample (or population). It stands as a pivotal indicator of the degree of dispersion within a set of values, with its center of reference being the mean. In essence, a low variance signifies that the data points cluster closely around the mean or expectation, whereas a high variance signifies a broader range of values with significantly more dispersed data points.

$$\sigma^2 = Var(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2. \tag{2.6}$$

2.1.5 Standard deviation

The standard deviation (SD) of a random variable, a probability distribution, or a dataset is the square root of its variance. It is another measure of dispersion, centered around the mean. Similar to variance, a low SD shows less dispersion around the mean or the expectation, and a high SD shows wide variation.

$$\sigma = SD(X) = \sqrt{\mathbb{E}[X^2] - (\mathbb{E}[X])^2}. \tag{2.7}$$

2.1.6 Statistical sampling

In statistics, sampling is a method of selecting a subset of individual people or items from a *population*. This subset is called the *sample*, which is ideally representative of the population. This sample is studied to understand some characteristics of the population from which the sample is drawn. Collecting a sample can be efficient in terms of cost and time and often supports feasibility, as studying the entire population is not only expensive and time-consuming, but it can often be an impossible task. Every item or individual point of the population, hence in the sample, is an *observation*. Depending on the study of the sample and the characteristics of the population, we can classify sampling as stratified sampling, systematic sampling, simple random sampling, cluster sampling, etc. Depending on the method of choosing samples, the sampling error and corresponding bias change.

2.1.7 U-statistics

In theoretical statistics, a U-statistic is a class of statistics that can produce an unbiased, minimum-variance estimation of estimable statistical parameters for various classes of probability distributions. Hence, they are frequently used in estimation theory. The "U" in the name of U-statistics comes from the unbiasedness of its estimation capability.

Definition. [75] Let $X_1, \cdots, X_n$ be n independent random vectors, $X_\nu = (X_\nu^{(1)}, \cdots, X_\nu^{(r)})$, and $\Phi(x_1, \cdots, x_m)$ a function of $m(\leq n)$ vectors $x_\nu = (X_\nu^{(1)}, \cdots, X_\nu^{(r)})$. A statistic of the form $U = \sum{}''\Phi(X_{\alpha_1}, \cdots, X_{\alpha_m})/n(n-1)\cdots(n-m+1)$, where the sum $\sum{}''$ is extended over all permutations $(\alpha_1, \cdots, \alpha_m)$ of m different integers, $1 \leq \alpha_i \leq n$, is called a U-statistic.

If $X_1, \cdots, X_n$ have the same (cumulative) distribution function $F(x)$, U is an unbiased estimate of the population characteristic.

$$\theta(F) = \int \cdots \int \Phi(x_1, \cdots, x_m) dF(x_1) \cdots dF(x_m). \tag{2.8}$$

$\theta(F)$ is called a regular function of the d.f. $F(x)$.

The variance of a U-statistic is a function of the sample size n and of certain population characteristics. Also, if $X_1, \cdots, X_n$ have the same distribution and

$\phi(x_1, \cdots, x_m)$ is independent of n, the d.f. of $\sqrt{n}(U - \theta)$ tends to a normal d.f. as $n \to \infty$ under the sole condition of the existence of $E(\Phi^2(X_1, \cdots, X_m))$.

Following our discussion of the fundamental ideas of statistics, to which we will frequently refer in our major body of work, we will look into probability theory.

2.2 Probability theory

Statistics and probability theory go hand-in-hand, and they are both essential for understanding the basics of machine learning and privacy. Next, we will discuss some of the preliminaries of probability. We have consulted some books [17, 37, 38, 58, 69, 80, 88, 94, 120] to summarize the concepts discussed in this section.

2.2.1 Law of large numbers

In probability theory, the theorem that describes that the average of all the results of performing the same experiment a large number of times should be close to the expected value is called the law of large numbers (LLN) [103],[35]. As the number of experiments or trials increases, the average result gets closer to the expectation. In mathematical terms, we can state it as,

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{X_i}{n} = \overline{X} = \mathbb{E}[X]. \tag{2.9}$$

The difference between, the expression above and the expected value of X, converges toward zero as n increases. For example, it shows that in many flips of an unbiased coin, the average probability of heads and tails is equal. We use these concepts in various situations.

The same intuition is used in another very similar concept, called the **central limit theorem**. It says that the mean or expected value of a sample will approach a normal distribution as the sample size increases and tends to converge to the mean of the population. So, with a large enough sample size, the sample mean converges to the population mean.

2.2.2 Addition law of probability

The probability of event X or Y occurring is shown by the addition of the probabilities of them individually,

$$\Pr(X \cup Y) = \Pr(X) + \Pr(Y) - \Pr(X \cap Y). \tag{2.10}$$

Now, for independent events X and Y, the overlap $(X \cap Y) = \phi$, hence this law boils down to,

$$\Pr(X \cup Y) = \Pr(X) + \Pr(Y), \tag{2.11}$$

where $\Pr(X \cap Y) = 0$.

2.2.3 Probability functions and distributions

In the sequel, we will look into the standard probability distribution functions.

2.2.3.1 Cumulative distribution function (CDF)

The cumulative distribution function of a random variable X can be defined as,

$$\mathbb{F}_X(x) = \Pr(X \leq x), \tag{2.12}$$

where X can take real values. The CDF takes the value of the probability that X takes on a value less than or equal to x.

The CDF of a continuous random variable X can be defined as,

$$\mathbb{F}_X(x) = \int_{-\infty}^{x} f_X(t)\, dt, \tag{2.13}$$

where f_X is the probability density function of the RV X. So, if f_X is continuous at x, then,

$$f_X(x) = \frac{d}{dx}\mathbb{F}_X(x). \tag{2.14}$$

If the distribution of the random variable X has a discrete component at a value $\mathbf{a}$, then the equation above gets the form,

$$\Pr(X = a) = F_X(a) - \lim_{x \to a^-} F_X(x), \tag{2.15}$$

which takes on value 0, if this discrete component does not exist and $\mathbb{F}_X$ is continuous at $\mathbf{a}$.

2.2.3.2 Probability density function (PDF)

In probability theory, PDF is used to express the probability of an observation around a target value when the observation is of a random variable with continuous univariate distribution. We can define it as,

$$\Pr[a \leq X \leq b] = \int_a^b f_X(x)dx, \tag{2.16}$$

where, X has density f_X.

From the definition of the cumulative distribution function of RV X in equations 2.13 and 2.14, we can say, $f_X(x)dx$ is the probability of X falling within the infinitesimal interval $[x, x + dx]$.

2.2.3.3 Probability mass function (PMF)

The probability of a discrete random variable taking a specific value in the distribution is expressed by the PMF. We can define it as,

$$f(x) = \begin{cases} \Pr(X = x), & \text{if } x \in S \\ 0, & \text{if } x \notin S \end{cases} \tag{2.17}$$

where, $\sum_{x \in S} f(x) = 1$.

2.2.3.4 Joint probability distribution

If two random variables are defined on the same probability space, then the joint probability distribution is the corresponding probability distribution on any possible pairs of outputs.

We can formulate the joint probability distribution of random variable X and Y as, $f_{X,Y}(x, y)$.

2.2.3.5 Prior probability distribution

A prior probability distribution of a quantity is the probability distribution that expresses the prior beliefs about this quantity before some evidence about the quantity is noticed.

A prior probability distribution can be expressed as $\Pr(A)$ is the probability of observing event A without any given conditions.

2.2.3.6 Conditional probability distribution

Conditional probability is the probability of one event occurring given that another event is true. The conditional probability distribution is stated as $\Pr(A \mid B)$, which represents the probability of occurring of event A, given event B is true. We call this the **posterior probability** of A, given that B is true.

2.2.3.7 Marginal probability distribution

If more than one random variable is defined in a random experiment, other than the joint probability distribution and conditional probability distribution, the individual probability distribution of each variable is referred to as its marginal probability distribution.

The marginal probability distribution of individual random variables X and Y in a joint probability distribution can be expressed as,

$$\Pr_X(x_i) = \sum_j \Pr(x_i, y_j), \text{and} \tag{2.18}$$

$$\Pr_Y(y_j) = \sum_i \Pr(x_i, y_j). \tag{2.19}$$

2.2.3.8 Likelihood

Likelihood is exactly the opposite of conditional probability distribution. Likelihood can be stated as, $L(A \mid B)$, which represents the probability of occurring of event B, given event A is true. So, we must notice, that $L(A \mid B) = \Pr(B \mid A)$

2.2.4 Bayes' theorem

One of the most popular theorems used in almost all disciplines of science is the following,

$$\Pr(A \mid B) = \frac{\Pr(B \mid A)\Pr(A)}{\Pr(B)}, \tag{2.20}$$

where $\Pr(B) \neq 0$.

Bayes' theorem [13] provides a framework for updating the prior probability distribution when new data becomes available, allowing us to calculate the posterior probability distribution. This is achieved by utilizing the conditional probability distribution of the uncertain quantity concerning the new data.

2.2.5 Common probability distributions

There are many probability distributions like Laplace, Gaussian, binomial, Poisson, uniform, Bernoulli, etc., which are commonly used by all scientific communities. Every probability distribution has its pros and cons and is suitable for different problem scenarios. We will use a few of them later in our work as well, so a quick refresher through their definitions will be useful here.

2.2.5.1 Definition: Laplace distribution

The Laplace distribution [35] (centered at 0) with scale b is the distribution with probability density function:

$$Lap(x|b) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right). \tag{2.21}$$

2.2.5.2 Definition: Gaussian distribution

The Gaussian distribution [60] represents the probability density function of a normally distributed random variable $X \sim \mathcal{N}(\mu, \sigma^2)$, with mean $\mu = 0$ and variance σ^2 is defined as,

$$g(X) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right). \tag{2.22}$$

2.2.5.3 Definition: Binomial distribution

Bernoulli's trial is an experiment with only two possible outcomes (boolean). The binomial distribution [36] is a discrete probability distribution with parameter $n \in \mathbb{N}$ and $p \in [0, 1]$, representing the probability of getting exactly k-number of successes in n independent Bernoulli's trial with probability of success p and failure $1 - p$, defined as,

$$\Pr(k, n, p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \tag{2.23}$$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, is the binomial co-efficient.

2.2.5.4 Definition: Uniform distribution

A uniform distribution is a symmetric probability distribution where the outcome lies between two bounds, lower bound a and upper bound b. The probability density function is defined as,

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{if } a \leq x \leq b \\ 0, & \text{otherwise} \end{cases} \tag{2.24}$$

Similarly, there are many other frequently used probability density functions like Poisson, Bernoulli, Student's t, geometric, etc.

Next, we delve into the concepts of disciplines like machine learning, optimization, data privacy, logic theory, etc.

2.3 Basics of machine learning

Machine learning is an old science, and it has been applied to different real-world problems like clustering and classification. Ahead of delving into practical applications, it is essential to discuss how this tool works.

To gain a working knowledge of the key concepts and reasoning behind them, a brief explanation of these machine intelligence algorithms is provided in the next few sub-sections. We have consulted some books [16, 37, 66, 94, 125, 127] to summarize the concepts discussed in this section.

Learning

Learning is the process through which computer systems improve their performance on certain tasks by studying and responding to data in machine learning. It entails the creation of algorithms and models that can recognize patterns, forecast outcomes, and take actions based on previous experiences. Machine learning is divided into three types: supervised, unsupervised, and reinforcement learning, each with its own unique approach to extracting knowledge from

data. Machines can essentially acquire and apply information from data autonomously, allowing them to solve complicated issues, make decisions, and improve their performance over time.

Inference

The primary goal of a learning algorithm is to find the internal pattern of a dataset by setting values for some parameters and using these parameters to predict outputs for new data points. This process is called inference. Inference can be of two kinds: **deductive inference** and **inductive inference**.

In deductive inference, specific assertions are derived from general principles, while in inductive inference, general principles are learned from observations. Inductive inference typically involves the following steps:

- Collecting data.
- Constructing a model.
- Making predictions.

Now, let's discuss learning algorithms and classify them based on their functional forms.

2.3.1 Parametric vs non-parametric learning algorithms

Some learning algorithms have fixed parametric structures and functional forms, while others do not. Algorithms with a specific functional form are known as parametric learning algorithms. These algorithms aim to estimate parameters to plug into the functional form for predicting values for new data points. An example of a parametric method is linear regression, which assumes a fixed linear form, and the slope and bias parameters need to be estimated.

Non-parametric learning algorithms [70] do not make assumptions about the functional form of the model and are flexible regarding parameters and their values. For example, in the k-nearest Neighbors (KNN) Algorithm, the functional form of the underlying model is not assumed, and it varies as the number k varies. The complexity and degrees of freedom can vary in non-parametric learning.

2.3.1.1 Predictability versus interpretability

To better understand learning models, two additional terms should be introduced: predictability and interpretability.

Predictability refers to the ease with which a system can predict the relationship between predictor and response variables and interactions among different predictors. Simpler systems are easier to interpret, but complex functions are

difficult to interpret. However, complex models often perform better in prediction tasks. For example, linear regression is interpretable but has low predictability, while neural networks are highly predictable but less interpretable. Choosing the model's complexity depends on the problem's specific requirements.

2.3.2 Different types of learning algorithms

Learning algorithms can be classified into five main types:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Structured output learning
- Reinforcement learning

Supervised learning

Supervised learning algorithms [16] are driven or supervised by labels corresponding to each data point in the training dataset. Given n pairs of training data $(X_i, Y_i)_{i=1\cdots n}$, where each data point consists of the predictor variable X_i and the response variable Y_i as the label, the goal of supervised learning is to learn a mapping function $f : X \to Y$ from a set of possible functions in the hypothesis space. In classification tasks, the output space Y is discrete, while in regression settings, Y is continuous.

Unsupervised learning

Unsupervised learning algorithms [16] are not supervised by labels. Training data points consist only of predictor variables. Unsupervised learning includes clustering, density estimation, and dimensionality reduction.

Semi-supervised learning

Semi-supervised learning algorithms [136] are supervised by labels for some data points in the training dataset, while others resemble unsupervised learning. The goal is to construct a function for the entire dataset, $A : L \times U \to F$, where F represents the set of possible functions.

Reinforcement learning

Reinforcement learning [119] is a key paradigm in machine learning and artificial intelligence that is concerned with teaching agents to make successive decisions in dynamic situations. Reinforcement learning agents learn through interaction

with their surroundings, as opposed to supervised learning, where algorithms learn from labeled instances. Based on their activities, these agents receive feedback in the form of incentives or penalties, allowing them to develop optimal methods through trial and error. Reinforcement learning has found applications in a wide range of fields, including robotics and computer games, as well as autonomous vehicles and recommendation systems. It provides a promising strategy for handling complicated issues where judgments must be made over time, making it an important field of AI research and development.

2.3.3 Model selection and assessment

In these learning algorithms, we train models, evaluate them, and often select one model over another.

Model selection

Model selection involves choosing the function space within which a statistical method learns from data. The selected model's proximity to the true underlying model, which represents the true data population distribution p_{data}, can determine the model's success or failure. Model selection is essential when multiple models within a function class are evaluated based on their performance on sample datasets.

Model assessment

Model assessment involves evaluating the performance of different functions within a function space. Different models within the same function class may have varying assessments in terms of how well they fit the data and generalize the distribution. The error introduced by the chosen model compared to the best model in the function space is called empirical error.

2.3.4 Input, action, outcome, and hypothesis spaces

Describing models and machine learning algorithms requires the introduction of some relevant terminologies.

- **Input space** is the domain of values that the input variable or predictor can take.
- **Action space** is the set of actions allowed to the agent or algorithm.
- **Outcome space** is the range of values that the response or dependent variable can take.
- **Hypothesis space** is the space of all possible hypotheses to be considered. The best hypothesis fits the data well and is optimal.

2.3.5 Basics of optimization

Now, that we have discussed about key concepts of basic machine learning, we dig deeper into the concepts of optimization, and critical points, as those are some of the principal concerns in this dissertation.

2.3.5.1 Critical points

The first-order derivative of the function at any point is the **gradient** of the function at that point. The function is only derivable if it is continuous and differentiable at that point. Now the second-order derivative is called the rate of change of the gradient of the function at a given point. The points at which the function gradient reaches a value of zero are called the **critical points** of the function. See Fig. 2.2, 2.3 for the illustrations.

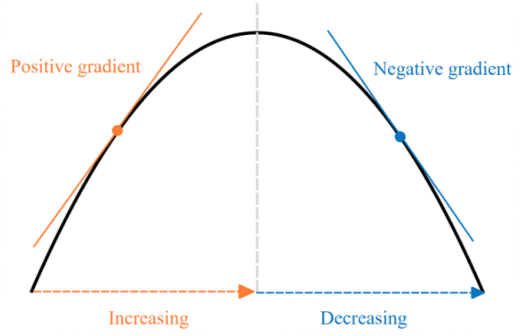


Figure 2.2: Gradient: The positive and negative tangent lines, shown in orange and blue respectively, touch the curve of the function, shown in black, at the bold orange and blue points. [Source]

There are three different types of critical points:

- Minima
- Maxima
- Saddle point

A point on the function is called a **minima** if the neighboring points on that function are higher than the point. Similarly, a point on the function is called a **maxima** if the neighboring points on that function are lower than the point. These maxima and minima can be global, local, or both in the case of a purely convex function. But in very high dimensional problems reaching a local extreme point is as good as reaching a global one [34]. The last critical point is the **saddle point**, which has both higher and lower points as neighbors on the function. See Fig. 2.3 for illustration.

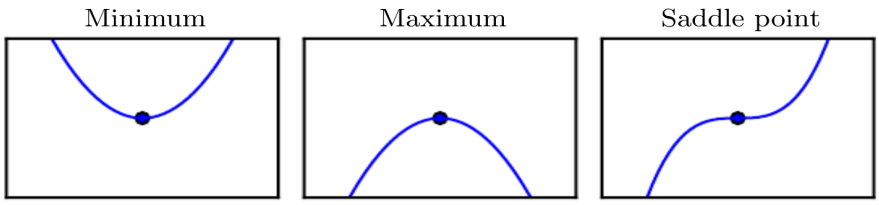


Figure 2.3: Critical points: The bold blue point on the left-most figure is the minima of the function. The bold blue point on the middle figure is the maxima of the function. On the right-most figure, the saddle-like structure has a critical point, shown in bold blue. [Adapted from 66]

Most simple functions are either convex or concave functions where either minima or maxima exist. But complex problems dealing with complex functions can consist of **multiple extrema**, both local and global optima. In such problems, finding the optimum solution is not as easy as simple convex (or concave) functions and requires a more advanced and sophisticated optimization algorithm to avoid getting stuck at **local optimum** points. See Fig. 2.4 for illustration.

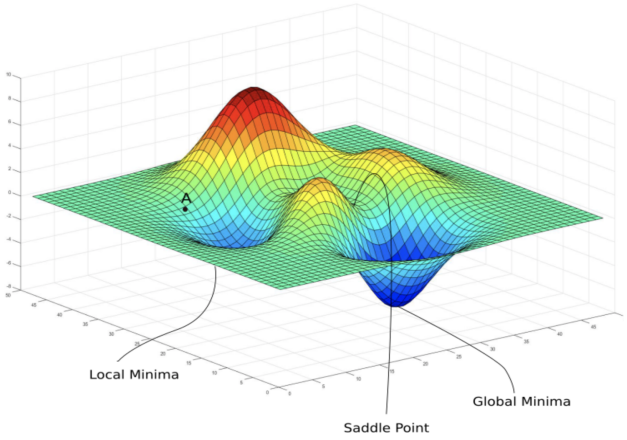


Figure 2.4: Multiple extrema: Multiple critical points and extremum points exist on the function, including local minima, global minima, and saddle points. [Source]

2.3.5.2 Optimization

Optimization is a method of nudging the solution in the right way so that eventually the optimal solution is reached. There are many optimization methods available like first-order Gradient descent, second-order Newton method, etc.

gradient descent (GD) [108] is the most popular and simple optimization method which randomly initializes at any point on the function and then takes a step in the opposite direction of the gradient (for minimization problems). This step size is decided by a factor called **learning rate** (η) and it can vary. Too big a step size can throw the algorithm off track, and the algorithm may never reach the optima, whereas too small a learning rate can delay the optimization procedure.

Batch gradient descent updates the parameters considering all the data points in the dataset. **Mini-batch gradient descent** is a variation of gradient descent where each update of parameters does not depend on the whole training dataset; rather, a mini-batch of some randomly selected data points taken from the original dataset affects every update. The bigger the batch size is, the less zigzag the trail of optimization, but each parameter update is more time-consuming. The decision on the mini-batch size and its trade-offs is to be made as a hyper-parameter and is adjusted using the validation set.

The last but very common variant of gradient descent is **stochastic gradient descent (SGD)** [108]. Here, each parameter update depends on only one training data point. It converges near the solution much faster than batch or mini-batch gradient descent. Especially when the training dataset is huge, it requires an enormous amount of computing power to train on them. So, for the inner product $x_i'w = x_{i,1}w_1 + x_{i,2}w_2 + \ldots + x_{i,p}w_p$, we can express it as,

$$w^{\text{new}} = w^{\text{old}} + \frac{\eta}{1 + \eta||x_i||^2}(y_i - x_i'w^{\text{old}})x_i, \tag{2.25}$$

where features are $x_1, \ldots, x_n \in \mathbb{R}^p$ and observations are $y_1, \ldots, y_n \in \mathbb{R}$. Here, the learning rate is normalized, hence numerically stable for all η.

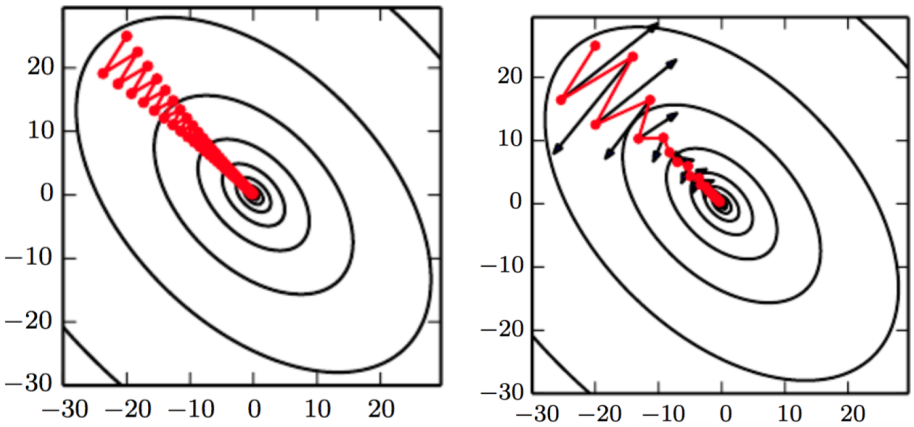
But it takes a back-and-forth path towards the optima and can jitter around the optima rather than reaching the exact point, as depicted in Fig. 2.5a, 2.5b. This problem can be solved by introducing little tweaks in the updates like introducing simple or Nesterov momentum terms. This takes inspiration from the concept of momentum in Physics where a moving particle incurs acceleration as it moves through space. Similarly, the learning rate or the step size gains momentum over different optimization steps. An exponential decay factor is introduced here to create the momentum depending on the previous gradients and their influence on the current one while updating the weights w that minimizes the $Q(w)$ below,

$$w := w - \eta\nabla Q_i(w) + \alpha\Delta w, \tag{2.26}$$

where η is the learning rate, $(0 \leq \alpha \leq 1)$ is the exponential decay factor.

The problem above can also be solved using adaptive learning rates in the optimization algorithm. This hyper-parameter can be set at the fixed value using the trial and error method or can be adjusted automatically through the course of learning, e.g., in algorithms like AdaGrad [41], RMSProp [121], Adam [81], etc. The Robbins–Monro algorithm [107] theoretically achieves calculating the optimal convergence rate, concerning the objective function values at $O(1/n)$, whereas [28] and [53] achieved $O(1/\sqrt{n})$.

The momentum accumulates the decaying average of gradients from past iterations to calculate the current update direction and size. The most common and best-performing first-order method for many problems is ADAM optimization, which uses adaptive momentum. It's an amalgamation of RMSProp and momentum.



(a) Naive SGD (b) SGD with Momentum

Figure 2.5: SGD without and with momentum: (a) Contour of Stochastic Gradient Descent cost function, where the jitter (shown in red) effect delays the optimization. [Adapted from 66]

Sometimes learning rate decay is done with a fixed decay rate, where after a fixed number of iterations, the learning rate decays by the rate factor. It can also decay after seeing no noticeable change in performance for a fixed number of iterations.

Now, we look further into a specialized alley of optimization, constraint optimization, and we will thoroughly use this concept in our proposed framework, as mentioned in Section 1.3.

2.3.5.3 Constraint optimization

In the mathematical world of optimization, the process of optimization in a constrained environment is called constrained or constraint optimization [15]. Here, an objective function is optimized with respect to some variables in the presence of some constraints on those variables. Constraint optimization has two principal components: the objective function and the set of constraints.

Now, the objective function is generally a loss or cost function in a minimization problem setting, where we try to find the minimal point after optimization. In the case of utility, performance, or reward maximization, we try to find the maximal point as a result of the optimization. The constraints can also be of various natures:

- **Hard constraints** are the ones that absolutely need satisfying while finding a solution to the optimization problem. These constraints are non-negotiable and can often slow down the optimizer, making it harder to find a feasible solution for the objective function while satisfying the hard constraints.
- **Soft constraints** are less strict in satisfaction criteria and negotiable at times. The optimizer should try to find an optimal solution for the objective function while trying its best to satisfy the soft constraints.

Besides, the constraints can also be classified as equality constraints or inequality constraints. The standard nature of a constraint optimization problem is as follows:

$$\begin{aligned} \min \quad & f(\mathbf{x}), \\ \text{subject to} \quad & g_i(\mathbf{x}) = c_i \ \text{ for } i = 1, \ldots, n \\ \text{and} \quad & h_j(\mathbf{x}) \geq d_j \ \text{ for } j = 1, \ldots, m \end{aligned} \tag{2.27}$$

where $g_i(\mathbf{x}) = c_i$ for $i = 1, \ldots, n$, and $h_j(\mathbf{x}) \geq d_j$ for $j = 1, \ldots, m$ are hard constraints that are required to be satisfied while optimizing the objective function $f(\mathbf{x})$. The $g_i(\mathbf{x}) = c_i$ for $i = 1, \ldots, n$ are the equality constraints, whereas, $h_j(\mathbf{x}) \geq d_j$ for $j = 1, \ldots, m$ are inequality constraints.

Having discussed optimization, we now look at how we deal with a dataset for training, testing, and validation purposes.

2.3.6 Dividing the dataset

An empirical risk minimizer has a probability of not reaching the optimal solution and getting diverted to (or stuck at) a sub-optimal solution for unseen data. However, the objective of the minimizer is to find the best prediction result (or inference) on previously unobserved data points. To find a solution to this, generally, the model is not trained on the whole dataset available; rather, they are divided into the following three subsets:

- Training dataset
- Test dataset
- Validation dataset

The training algorithm is run on the largest part of the dataset, called the **training dataset**. For large datasets, more than eighty percent of the total dataset is used as the training set. Generally, if more data points are available for training, the performance of the model improves. The remaining data points are divided into the **validation set** and the **test set**. The test dataset is kept aside and is never used before the model is finalized. However, every time

the model is trained on the training set, the validation set is used to get an intermediate idea of the model's performance on unobserved data and to tune the hyperparameters.

When the dataset is not large enough to divide it into a training set, a test set, and if required, a validation set, statistical methods like **cross-validation (CV)**, **leave-one-out cross-validation (LOOCV)**, and **k-fold cross-validation** come to the rescue.

2.3.6.1 Cross-validation and re-sampling algorithms

When the dataset is not large enough to divide it into a training set, a test set, and if required, a validation set, then statistical methods like **cross-validation (CV)** [82] and other resampling methods come to the aid. In a naive CV, the dataset is divided into a random split (generally 50/50) to be used as a training and test dataset. This way, the trained model is tested on previously unseen data points. Unfortunately, it reduces the training data size significantly, hence overestimating the test error. See Fig. 2.6.

Figure 2.6: Cross-validation: The naive cross-validation splits the dataset into a 50/50 division as the training data and the test data. [Adapted from 77]

Another method called **leave-one-out CV (LOOCV)** [77] improves over the vanilla CV, where if there are n data points available, then all but one data point is used as the training data and the one data point (also called the held-out data point) acts as the test data. However, the test dataset is too small to reflect proper test error. So the whole training method is repeated n times, every time with a different held-out data point as the test data. Then the final test error is the mean of all the n test errors from n different trainings. It solves the problem of very few training data points, but as the training sets from different runs are almost the same, it has a very high variance. It is also computationally expensive as it needs to train the model n times on the $(n-1)$ data points, every time. See Fig. 2.7.

The third resampling method called **k-fold cross-validation** [72], is somewhere between the above two methods. It tries to address the problems both these methods have. Here, the dataset of n data points is divided into k different non-overlapping folds of almost the same size ($\cong \frac{n}{k}$). Now the function is trained with $(k-1)$ folds of data, i.e., $\frac{(k-1)*n}{k}$ many data points as the training set and the rest as the test set. The whole process is repeated k times, every time with a different fold as the held-out test set and the rest of the $(k-1)$ folds

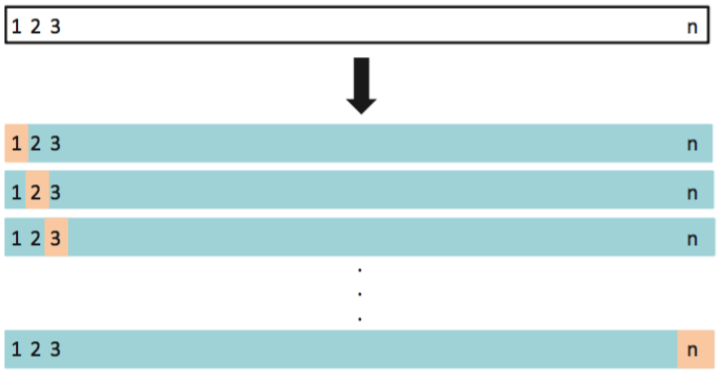


Figure 2.7: Leave-one-out cross-validation: In this cross-validation, at every run, all data points but one are used as the training dataset and the held-out data point acts as the test data. [Adapted from 77]

as the training set. In the end, an average is taken over all the k-test errors to get the final result. This method has more data than CV as well as less overlap of the training dataset between different runs. The computational expense is also somewhere in between CV and LOOCV. Though finding the right number of folds, appropriate for the problem in hand is a challenge, and the test error estimation depends a lot on the value of k. See Fig. 2.8.

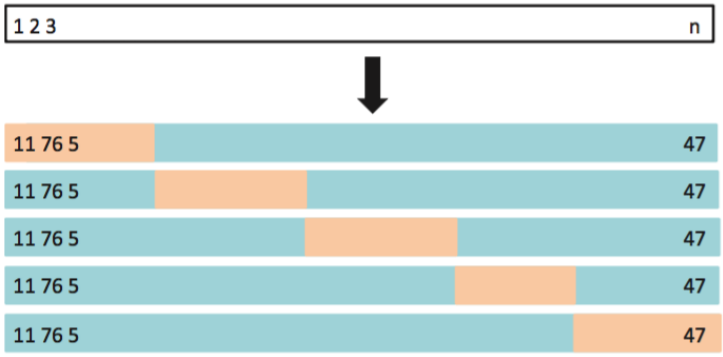


Figure 2.8: K-fold cross-validation: In this cross-validation method, n data points are divided into nearly equal-sized k folds. At each run, $(k-1)$ folds are used as the training set, and the rest of one fold is the test set. [Adapted from 72]

K-fold CV with $k < n$ is computationally more efficient than LOOCV, as k-fold CV needs to fit the model k times, whereas LOOCV needs to fit the model n times. Moreover, k-fold CV estimates the test error rate more accurately than

LOOCV as a result of the bias-variance trade-off.

2.3.7 Training and fitting the model

The estimated model can approximate the true function using the training dataset to different extents. Intuitively, if the estimated model fits the training dataset exactly, then there would be zero loss, which might sound like a perfect solution to any problem. However, the original goal of any learning algorithm is to reach a point when the estimated model performs well on previously unseen data points.

This may seem unintuitive, but in reality, nobody cares much about the model's performance on the training data, especially in prediction problems in supervised learning, and how well it learns to memorize the training dataset. Instead, a model that has comparatively poor performance on the training data but performs significantly well on test data is considered better than a perfect fit on the training dataset.

How well the estimated model fits the observed dataset and performs on the test dataset also depends on the fitting of the model on the training dataset. If the estimated function is very flexible and goes through all data points in the training dataset, then it is called **overfitting** the dataset. It will perform well on the training dataset but will not generalize well to perform on the test data. The system merely memorizes the observed points, learns all the noise and idiosyncrasies of the training dataset, and does not capture the nature of the true function. Similarly, an **underfit** model is so simple that it lacks the flexibility to capture the variations in the dataset. It fails to represent both the training and the test dataset and, consequently, the ground truth function, leading to very low performance. A model that neither underfits nor overfits the dataset, having the right capacity (similar to the ground truth or the original distribution unknown to all), is ideal. See Fig. 2.9 for illustration.

A model with flexibility somewhere in between, one that approximates the training dataset well and has a significantly low test error rate, is called a correct or **perfect fit**. A perfect fit should be the model that comes closest to the **Bayes decision function** and has an error with the least possible difference from the **Bayes error**. The Bayes error is the lowest possible achievable error by any learning algorithm when the real distribution of data is unknown to the system. This variation in fitting is subjected to the bias-variance trade-off, which we will explore next.

2.3.7.1 Bias variance trade-off

As the complexity of the model increases, the training error of the model steadily decreases. However, this may not hold for the test error. Initially, as the training error decreases with increasing complexity, the test error also decreases, but only up to a certain point. After reaching that certain point, despite the training error continuing to decrease, the test error starts to increase as the complexity of the model increases.

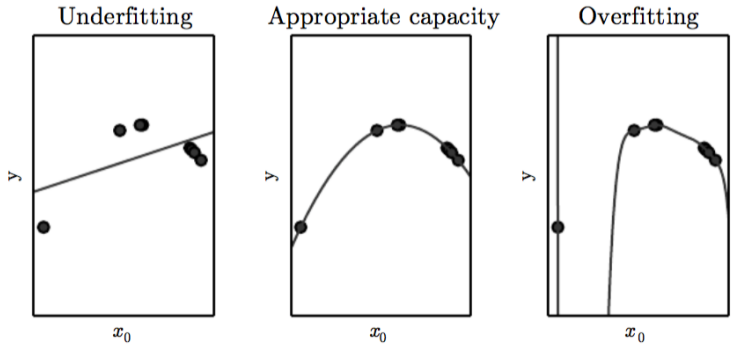


Figure 2.9: Different types of capacity: The left image shows a linear model underfitting the dataset and failing to follow the pattern. The middle image has the appropriate capacity where the model can perfectly follow the true dataset. The right image is overfitting as it blindly follows the data points and cannot capture the pattern present in the original dataset. [Adapted from 66]

To explain this phenomenon, we need to understand the bias-variance trade-off [72]. But before that, let's discuss two fundamental concepts and their interactions:

- **Variance** of a fitted model measures how much the model parameters will change if we use a different sample training set from the original dataset.
- **Bias**, on the other hand, indicates how closely a model follows the underlying trend in the dataset.

If a model closely follows every point in the training dataset, it has a very low bias but may fail to generalize to the underlying trend in the original dataset (the unknown distribution), resulting in a very high variance. Conversely, a highly generalized model has low variance but may exhibit high bias, as it could deviate significantly from the underlying pattern in the data. Bias and variance are always in opposition, with one increasing as the other decreases and vice versa.

The **residual error** of a fitted model is proportional to the sum of the variance term and the squared bias term. Our primary goal is to reduce the overall error of the trained model, so we must set the model parameters in such a way that both bias and variance settle at their optimal values. Finding the sweet spot that balances these contributing factors is a significant challenge for most machine learning algorithms. A model with high bias is underfitting the dataset, while a model with high variance is overfitting the dataset. See Fig. 2.10 for illustration.

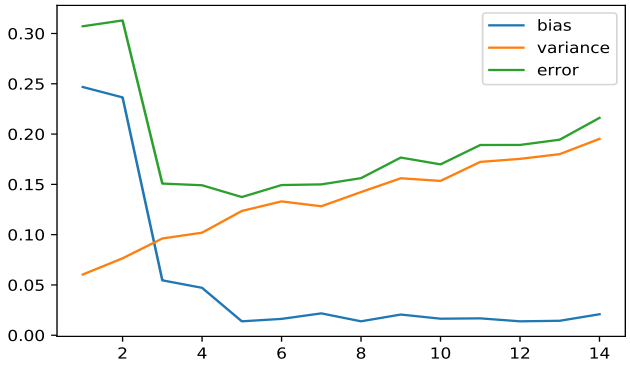


Figure 2.10: Bias-variance trade-off: With increasing complexity of the model, the bias curve in blue monotonically decreases, while the variance curve in orange monotonically increases. Consequently, a function of both terms in green, with the growing complexity of the model, initially decreases the test error but then starts to rise. [Adapted from 72]

2.3.7.2 Regularization

Training less can lead to underfitting, while excessive training can result in overfitting and an increase in generalization error. Regularization is a popular technique to combat overfitting. The challenge of determining the appropriate model capacity during training can be addressed through regularization, which can be achieved in two ways: constraint and penalty methods.

Constraining the dimensionality of the function space simplifies the function and prevents it from becoming excessively complex, which helps avoid overfitting. For example, Ivanov regularization imposes such constraints. In contrast, penalizing the function does not constrain the original function space but reduces its complexity to smooth out sharp changes. This regularization method reduces parameter values and encourages them to approach zero. It is also known as weight decay. The most well-known regularization techniques through penalty are Ivanov and Tikhonov regularization methods, [99], which are essentially equivalent ways to achieve a similar level of regularization on a function.

The two most popular regularization methods are:

- Lasso or L1 regularization
- Ridge or L2 regularization

Lasso or L1 regularization [77] aims to minimize the error function while adding an L1 norm term (first-order). This results in reduced parameter values, effectively bringing the estimated weight vector $\hat{\beta}$ closer to the blue diamond in

Fig. 2.11. It ultimately settles at the point where the error function contour touches the diamond:

$$\hat{\beta} \equiv \arg\min_{\beta} \big(||y - X\beta||^2 + \lambda ||\beta||_1 \big). \tag{2.28}$$

Similarly, **ridge or L2 regularization** [77] aims to minimize the error function while adding an L2 norm term (second-order). This also leads to reduced parameter values, bringing $\hat{\beta}$ closer to the blue circle in Fig. 2.11. It ultimately settles at the point where the error function contour touches the circle:

$$\hat{\beta} \equiv \arg\min_{\beta} \big(||y - X\beta||^2 + \lambda ||\beta||_2^2 \big). \tag{2.29}$$

Both regularization methods have their advantages. L1 regularization introduces sparsity to the parameters, allowing it to set some parameter values to zero, reducing the problem's dimensionality and simplifying the optimization process. However, L1 regularization is non-differentiable and piecewise, making gradient updates of parameters challenging through backpropagation. On the other hand, L2 regularization is continuous and differentiable but cannot induce sparsity in parameters. As a result, L1 regularization is often used for feature selection, particularly in high-dimensional problem spaces.

Next, we will provide a brief overview of various types of losses. It is important to understand that our main goal, optimizing a desired result (in our case, utility or privacy parameters), requires us to minimize loss effectively.

2.3.8 Different types of loss

Loss functions are problem-specific, and their properties vary depending on their application in regression or classification problems. Generally, classifications focus on the fraction of test data points that are correctly classified, while regression problems are concerned with the residual error.

The classification loss function is called **0/1 loss**, which assigns a "0" for every correctly classified data point and a "1" for every wrongly classified data point to count the number of wrongly classified data points, normalized by the total number of data points in the test set. The 0/1 loss is continuous, convex but not differentiable, and is defined as:

$$\ell(\hat{y}, y) = 1(\hat{y} \neq y). \tag{2.30}$$

Simple **residual loss** is the difference between the prediction and the ground truth, expressed as:

$$\ell(\hat{y}, y) = \hat{y} - y. \tag{2.31}$$

Sometimes, **absolute loss** or **Laplace loss** is used, but it is not differentiable and is defined as:

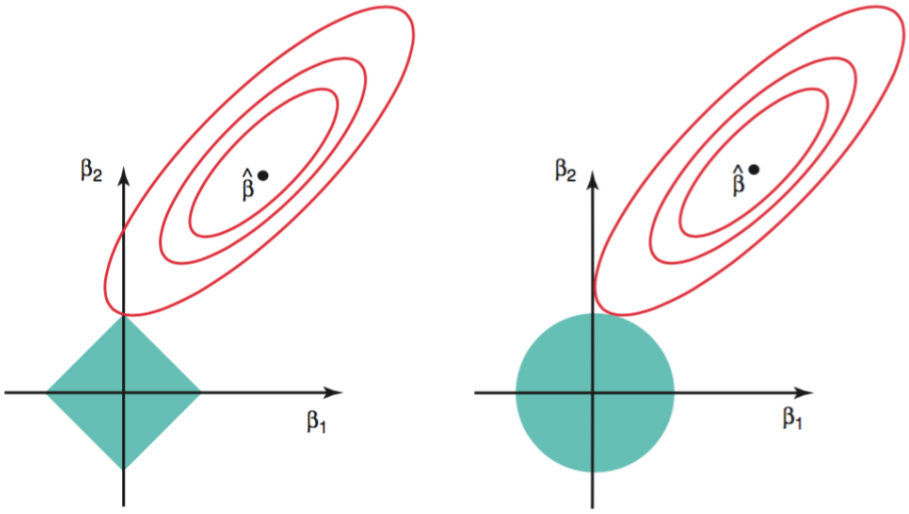


Figure 2.11: L1 and L2 regularization with the residual sum of squares (RSS) contour function: The left figure illustrates L1 regularization, where the green diamond represents the L1 function, and the error contour is shown in red. Similarly, in the right image, the green circle is the L2 function, and the contour plot is in red. In both L1 and L2 regularization, total cost minimization aims to move the error contour's minimum point toward the regularizer function. The final value of $\beta^\star$ is where the regularizer function plot touches the error contour. In L1 regularization, the point where the error contour touches the diamond's corner sets the weight component along the β_2-axis to zero. [Adapted from 77]

$$\ell(\hat{y}, y) = |\hat{y} - y|. \tag{2.32}$$

Outliers are data points that can be considered noise and do not follow the pattern of the true underlying distribution, such as the price of Bill Gates's house in a house price prediction problem.

Square loss is used to prevent positive and negative residuals from nullifying each other and is differentiable. However, the square loss is more affected by outliers and is not robust. It is defined as:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2. \tag{2.33}$$

Hinge loss is a maximum margin classification error, commonly used in support vector machine (SVM) [31], and is sometimes referred to as **SVM loss**. In margin-based losses, a positive margin signifies correct classification, while a negative margin indicates incorrect classification. Here, a larger margin is considered a classification with more confidence. Hinge loss can be defined as:

$$\ell(\hat{y}, y) = \max(0, 1 - \hat{y} \cdot y). \tag{2.34}$$

Logistic loss or **binary loss** is a normalized loss function that provides values within the range of 0 and 1. Logistic loss is mainly used in classification problems. Prediction accuracy increases as the log loss decreases, but it does not solely consider the boolean nature of a 0/1 loss.

Cross-entropy loss or **log loss** is another commonly used loss function in classification problems. Its output has a range $\in [0, \infty)$ and is defined as:

$$\ell(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}). \tag{2.35}$$

There are many other loss functions like huber loss, ramp loss, or perceptron loss, but they are not widely used. In all cases, minimizing the error function is equivalent to maximizing likelihood/log-likelihood, minimizing negative log-likelihood, or minimizing cross-entropy [21].

2.4 Common machine learning algorithms

Machine learning algorithms are diverse in nature and application. Some of them have proven to be more popular in terms of ease of use, generalizability in different applications, adaptability, and other factors. To provide a varied perspective, we will briefly discuss some of the most versatile ML algorithms that have stood the test of time and remain useful to this day. We have consulted several books [16, 37, 66, 94, 125, 127] to summarize the concepts discussed in this section.

2.4.1 Prediction

Every machine learning algorithm serves a specific purpose, and one of the most popular purposes is prediction. Prediction algorithms can predict the value of a target variable for specific values of feature variables that influence the outcome or the target variable. Whether it's predicting the weather, rainfall, or apartment prices in a specific location, prediction algorithms play a crucial role in understanding the behavior of these target variables. One of the most common and simplest prediction algorithms is linear regression.

Linear regression Linear regression is an approach for modeling the linear relationship between one or more explanatory variables and a response variable. Simple linear regression deals with only one explanatory variable, while multiple linear regression deals with multiple explanatory variables.

For a dataset of n data points with a response variable y_i and explanatory variables x_{ij}, where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, d$, i.e., the explanatory variables x_i are d-dimensional vectors, the linear regression model is defined as:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id} + \varepsilon_i \quad \text{for } i = 1, \ldots, n. \tag{2.36}$$

In vector notation, we can write this as $y = X^T \beta + \epsilon$, where

$$
\begin{aligned}
y &= \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \\
X &= \begin{bmatrix} \mathbf{x}_1^\mathsf{T} \\ \mathbf{x}_2^\mathsf{T} \\ \vdots \\ \mathbf{x}_n^\mathsf{T} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ 1 & x_{21} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nd} \end{bmatrix}, \\
\beta &= \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}, \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}.
\end{aligned} \tag{2.37}
$$

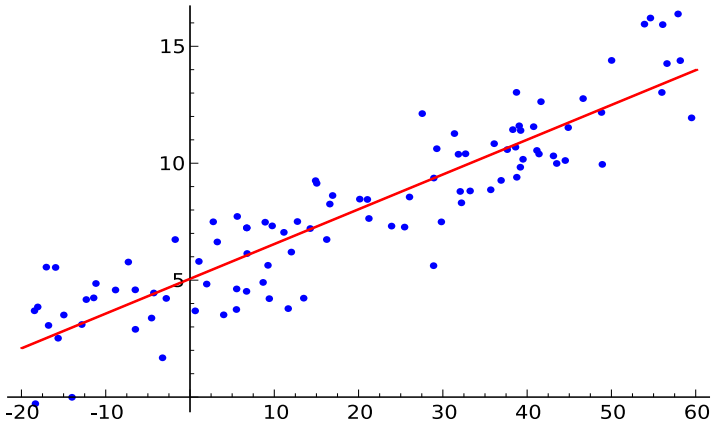


Figure 2.12: Linear regression: An example plot of simple linear regression line on fitted data with one explanatory variable and one dependent variable. [Source]

To illustrate this concept with a simple plot, refer to Fig. 2.12. This is a two-dimensional plot where one axis (horizontal) represents the explanatory variable, and the other axis (vertical) represents the dependent variable. The red line is the approximation of the line fitted by the linear regression model to the blue data points. After training, during testing, given a new data point with a value of the explanatory variable, the corresponding value of the dependent variable is predicted by the value on the red line.

2.4.2 Classification

Just as we've seen how linear regression is used for prediction, there are many types of classification algorithms as well. Some of the popular ones like k-nearest neighbor (KNN) and naive Bayes are straightforward and effective on simple datasets. However, for more complex datasets where a fine and sophisticated solution is required, there are advanced classification methods.

Naive Bayes classifier Based on Bayes' theorem, this supervised learning algorithm makes the "naive" assumption of conditional independence between every pair of features to find the outcome with the highest probability. The algorithm calculates the posterior probability using the prior probability, likelihood, and evidence:

$$\text{posterior probability} = \frac{\text{prior probability} \times \text{likelihood}}{\text{evidence}} \,. \tag{2.38}$$

The Naive Bayes algorithm performs well on tasks like text categorization, email (spam or ham) classification, and disease diagnosis.

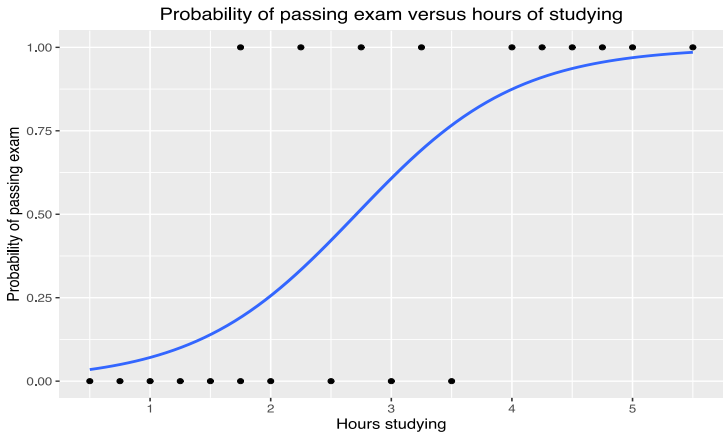


Figure 2.13: Logistic regression: An example plot of a simple logistic regression curve on fitted data with one explanatory variable and one dependent variable. The horizontal axis shows the explanatory variable "hours studying", and the vertical axis shows the "probability of passing the exam." [Source]

Logistic regression Logistic regression is one of the most common and practical classification algorithms, as illustrated in Fig. 2.13. It models the probability of a binary outcome and is widely used in various applications.

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{2.39}$$

K-nearest neighbor (KNN) KNN [16] is a non-parametric supervised algorithm that assigns every point in the dataset to the class where the majority of its $k \geq 1$ nearest data points in the training set are assigned. In specific applications, KNN can also be used for regression purposes.

Support vector machine (SVM) SVM [123] is a more sophisticated supervised algorithm used for binary classification. It categorizes data points in a training dataset into one of two classes while maximizing the width of the boundary between the classes. New test data points are assigned to a class depending on their position around the boundary.

To achieve this, SVM places a set of hyperplanes in an ideally high-dimensional space to separate the classes and tries to find the widest gap between them. The pair of hyperplanes with the widest gap between them is chosen. The region between these two parallel hyperplanes is called the margin (or hard margin). The plane parallel to these hyperplanes, cutting the margin in half, is chosen as the class separator. Hence, it's also known as the maximum-margin hyperplane algorithm. The data points on the two margins are called the support vectors.

If the classes are not linearly separable, the kernel trick is used with a non-linear kernel to find the hyperplanes. Additionally, when the dataset is not linearly separable, SVM often finds a soft margin instead of a hard margin. A soft margin allows the model to make some mistakes, meaning a few training samples may fall on the wrong side of the class separator. This approach leads to a more generalized final model at the cost of a higher training cost.

2.4.3 Dimensionality reduction

Another important field of machine learning application is dimensionality reduction. High-dimensional spaces can lead to sparse and intractable observations, known as the "curse of dimensionality". Domains dealing with a high number of features, such as bioinformatics or speech recognition, often require mechanisms to reduce the dimension of their data for easier analysis. Dimensionality reduction aids in data visualization, pattern recognition, data interpretation, and noise reduction.

To address this, we often try to map data into a lower-dimensional space while retaining the insights from the original high-dimensional representation. Feature selection aims to find a subset of all initial features that provide a meaningful representation of a lower-dimensional dataset. Feature extraction or projection, on the other hand, transforms the dataset into a smaller set of derived features with a meaningful and non-redundant representation of the original dataset.

Principal component analysis (PCA) PCA [78]is the most widely used linear dimensionality reduction technique based on feature projection. It linearly maps high-dimensional data into a lower-dimensional representation while retaining the maximum amount of variance in the data, preserving meaningful information from the original dataset.

This simple yet effective method computes the eigenvectors of the covariance matrix of the data and then sorts them based on their eigenvalues. To find the p principal components, we select the first p eigenvectors corresponding to the largest p eigenvalues in the covariance matrix, as they capture the highest variance in the high-dimensional space. Kernel or graph-based principal component analysis deals with a non-linear way of feature projection.

2.4.4 Ensemble learning

Ensemble learning methods rely on a group of individually weak base models to optimize the final, stronger prediction results through their combined wisdom. They aggregate a group of base models with high variance to ultimately achieve a model with lower variance. Ensemble methods often use models prone to overfitting, such as Decision trees and Random forests, and a more generalized solution can be obtained by aggregating an ensemble of different models.

Two types of ensemble learning methods are **bagging** and **boosting** [66]. In both methods, multiple learning models are trained with their own set of weights, and by sharing these weights, they tend to reach parameters that lead to better performance than their individual performances. Bagging methods are generally used when individual base models show high variance and low bias while boosting methods are preferred when individual base models exhibit low variance and high bias.

In bagging, a random sample of training data points is selected with replacement to create many datasets with overlap, and they are trained in parallel. **Random forest** uses many **decision trees**, and aggregating many such high-variance models yields better performance than individual models.

On the other hand, individual base models are trained sequentially in boosting methods. **AdaBoost** [56], **XGBoost** [27], or **GradientBoost** [57] are some variants of Boosting algorithms. Iterative learning of multiple weak-learner models eventually results in a strong learner with improved model performance.

2.4.5 Clustering

Clustering, or cluster analysis, is the unsupervised study of discovering natural groupings or associations in data. Clusters are dense groups of observations in feature space that are closer to some clusters and distant from others. The proximity of observations within a cluster is often measured using a distance or similarity metric.

Clusters may have centroids and boundaries separating them. Finding clusters can be useful in knowledge discovery or pattern finding among observations. Beyond using quantitative measurements or distance metrics, domain experts often subjectively analyze the data to achieve better clustering performance.

K-means One of the most widely used clustering algorithms is known as k-means clustering [52]. It utilizes variance as the distance metric and aims to minimize the variance among clusters. The algorithm partitions the population

into k sets with optimized inter-cluster variance based on a sample set of size n. Observations are assigned to the cluster with the nearest centroid.

Given a sample set of observations $\{x_1, x_2, \ldots, x_n\}$, where each observation x_i is a d-dimensional real vector, k-means clustering divides the sample into a set of clusters $S = \{S_1, S_2, \ldots, S_k\}$ by minimizing the within-cluster variance in terms of the sum of squares:

$$\arg\min_S \sum_{i=1}^{k} \sum_{x \in S_i} \|\mathbf{x} - \mu_i\|_2^2 = \arg\min_S |S_i| \mathrm{Var}(S_i), \tag{2.40}$$

where $|S_i|$ is the size of cluster S_i, and μ_i is the centroid of cluster S_i, i.e., $\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x$.

2.4.6 Traditional deep neural networks

The recent surge in applying neural networks across various fields has surpassed traditional implementations and their performances. Neural networks had been relatively dormant until the last decade when they experienced unprecedented advancements. Starting from 2009, the widespread adoption of neural networks can be attributed to significant increases in **computational power**, including graphics processing units (GPUs) and more recently TPUs, making efficient data processing accessible. These networks excel in handling tensors and other linear algebraic elements efficiently, specializing in matrix operations and parallel processing. Moreover, open access to vast **datasets** has greatly contributed to the progress of neural networks as a tool for dealing with such datasets and uncovering hidden patterns within them.

Numerous research organizations and universities host datasets on open-source platforms, such as MNIST [39], IMAGENET [109], CIFAR-10, and CIFAR-100 [84].

2.4.6.1 Feedforward neural networks

A neural network with a single layer is called a single-layer perceptron (SLP), and a network with multiple hidden layers is referred to as a multi-layer perceptron (MLP) [66]. An SLP is essentially a linear regression model (optionally with non-linearity applied at the end), while complex networks with multiple layers are required to solve more challenging problems with intricate functional forms. An SLP can be expressed using the following functional form:

$$f(x; w, b) = w^T x + b, \tag{2.41}$$

whereas the same architecture with one hidden layer between the input and output layers can be expressed by the following function:

$$f(x; w_1, b_1, w_2, b_2) = w_2^T \max(0, w_1^T x + b_1) + b_2, \tag{2.42}$$

with a ReLU non-linearity applied to the first layer. Similarly, additional layers can be stacked to increase the complexity of the functional form.

Forward propagation involves network depth or the number of layers l, weight matrices $W^{(i)}$, bias vectors $b^{(i)}$, where $i \in \{1, \ldots, l\}$, the input features X, and the output or ground truth values y. The feedforward network takes the input vector X at the input layer, multiplies it with the weights of the first hidden layer, and adds the bias. After calculating the pre-activation, a non-linear activation function is applied to it. Similar operations occur in other hidden layers, propagating activations through the network to the output layer where the outcome is generated.

Backpropagation Depending on the outcome at the output layer and the ground truth, the cost is computed. The backward propagation of information from the cost calculated at the output layer through the network to compute the gradient concerning the parameters is known as the **backpropagation** algorithm [26]. The calculated gradient in backpropagation is utilized by gradient-based methods, such as gradient descent, to perform learning. Backpropagation follows the chain rule of calculus:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}, \tag{2.43}$$

where $y = f_1(x)$ and $z = f_2(y) = f_2(f_1(x))$, and f_1 and f_2 are functions.

Many of the details of computing backpropagation are trivial these days, as advanced tools are used for implementation, which often have built-in libraries that compute backpropagation implicitly.

2.4.6.2 Convolutional neural network (CNN)

Convolutional neural networks (CNNs) [86] are specialized networks designed for conducting convolution operations on grid-structured data, such as images (2-D or higher dimensional grids) or time-series data (1-D grids). Convolution can be either discrete or continuous, depending on the properties of the signal, with continuous convolution being used for audio signals and discrete convolution for digital images. In both cases, the convolution operation can be understood as a weighted multiplication with a kernel function, which emphasizes local properties over global ones.

The kernel is placed on the grid, and multiplication and addition operations are performed to calculate the values in the convoluted matrix. The kernel is then moved to the next position, eventually covering the entire grid function. Convolution can be used in multiple directions, but the principles remain the same as in one-dimensional convolution. Three types of convolution techniques are available: full convolution, same convolution, and valid convolution, depending on factors like grid size, edge management, kernel depth, and the number of convolution layers.

2.4.6.3 Recurrent neural network (RNN)

Recurrent neural networks (RNN) [126] represent a family of neural networks specialized in handling sequential data. The name 'recurrent' comes from its

ability to repeatedly apply the same processing units to each time frame of the input data. RNNs can work with variable input sizes of time-series data. In an RNN, each unit's output is a function of the input up to that point and the previous output unit.

There are many variations of RNN, including teacher-forcing RNN, fully connected RNN, context-sensitive RNN, conditional RNN, bidirectional RNN, encoder-decoder sequence-to-sequence RNN, and RNN with attention mechanism. Due to their suitability for handling time-related data, RNNs find extensive applications in fields like speech recognition, natural language processing (NLP), and time-series analysis. Real-world applications of RNNs include machine translation, speech synthesis, virtual question-answering systems, and customer care chatbots.

2.4.6.4 Long-short term memory (LSTM)

Long-short term memory (LSTM) [74] is a type of neural network designed to address a limitation of traditional RNNs related to handling long-term dependencies caused by the vanishing gradient problem. In situations where there are long gaps between relevant pieces of information in a sequence, traditional RNNs struggle to establish connections. For instance, in natural language text, if a character talks about going to a country (e.g., Finland) and later mentions learning the language (e.g., Finnish) after several sentences, traditional RNNs may fail to associate these long-term dependencies and may not identify the name of the language.

LSTM addresses this problem using a gated recurrent neural network structure. It introduces a self-loop connection that allows gradients to flow without vanishing or exploding, enabling the network to create a weighted combination of previous memory content from earlier cells and more recent memory. This mechanism significantly improves the network's ability to capture long-term dependencies in sequential data.

Having discussed the basic concepts and popular algorithms of machine learning, we will next take a brief look at their numerous applications.

2.5 Applications of machine learning

The rise of machine learning algorithms has had a profound impact on various fields of application. Here, we will briefly discuss some of these applications.

2.5.1 Natural language processing (NLP)

Natural language processing (NLP) is a field that focuses on enabling computers to understand and generate human language. It encompasses tasks such as text analysis, sentiment analysis, language translation, and chatbot development. machine translation, a subfield of NLP, involves translating text or speech from one natural language to another. NLP systems often require domain expertise

to understand the nuances and context of language, as literal translation can be inadequate for capturing the intended meaning.

2.5.2 Speech

The study of speech involves understanding and processing spoken language. This field includes automatic speech recognition (ASR), text-to-speech (TTS) synthesis, privacy-preserving speaker recognition, speaker anonymization, and more. ASR systems convert spoken language into text, while TTS systems generate natural-sounding speech from written text. Speech-related applications are widespread, from voice assistants to intelligent medical systems.

2.5.3 Virtual agents and robotics

Robotics, a long-standing field, has witnessed a surge in popularity and applications in recent years. Virtual agents, including chatbots and customer support systems, have become integral in various industries. These agents are capable of understanding and responding to natural language, enabling efficient communication with users. Robots, both physical and virtual, are used in industries ranging from manufacturing to healthcare, performing tasks efficiently and autonomously.

2.5.4 Computer vision

Computer vision is a field that leverages AI and machine learning to interpret and analyze visual data, such as images and videos. Applications include image recognition, semantic image segmentation, style transfer, and identity verification. Computer vision technology is used in self-driving cars, medical imaging, social media content analysis, and more.

2.5.5 Recommendation systems

Recommendation systems analyze user behavior to suggest relevant content or products. This technology is widely used on platforms like Netflix, Amazon, and e-commerce websites to recommend movies, products, and services. Effective recommendation strategies can lead to increased user engagement and sales.

2.5.6 Fraud detection

Machine learning algorithms are employed in financial organizations to detect fraudulent activities. These algorithms learn from patterns in financial data to identify anomalies and potential fraud. Fraud detection is crucial for safeguarding financial systems and protecting consumers.

2.5.7 Task automation

Machine learning plays a significant role in automating tasks across various domains. From high-frequency stock trading and web crawling to automated content generation and language detection, these technologies streamline processes and improve efficiency.

2.6 Adverse effects and malicious uses of ML

Despite its numerous benefits, machine learning also raises concerns regarding adverse effects and malicious applications. Next, we will explore some of these concerns.

2.6.1 Discrimination and bias

Machine learning algorithms often face criticism for being biased and discriminatory. Bias can emerge from historical, and societal prejudices present in the data used for training. AI-powered systems, including those used in hiring processes and predictive analytics, have been found to exhibit gender, racial, or physical feature biases. Addressing these biases and achieving fairness in algorithms remains a challenge.

2.6.2 Automation and job security

The fear of automation replacing human jobs has persisted since the advent of computers. Recent AI advancements, such as natural language models like ChatGPT and Dall-e, have amplified concerns about job security. While these concerns are sometimes unfounded, it's essential to consider the shifting demand for new roles created by technological advances and the need for reskilling and reintegration of the workforce.

2.6.3 Unaccountability and personal ethics

The accountability of AI and machine learning systems is a critical issue. Legislation exists in some regions, but enforcement and oversight are still evolving. Researchers, governments, and companies must take responsibility for ethical AI practices. Personal ethics play a vital role, as some AI-related consequences may not be formally provable but can still have significant societal impacts.

2.6.4 Privacy attacks

Privacy attacks are becoming increasingly common, targeting individuals and organizations. Bots and cybercriminals spread misinformation, engage in political propaganda, and impersonate humans on social media platforms. Hacking, cybercrimes, data breaches, and unwanted surveillance are ongoing concerns.

Legislation like GDPR aims to protect privacy. We will delve deeper into this concern in the next section.

2.7 Data privacy

Data privacy refers to the right of individuals or communities to control the usage of their data. It empowers individuals to choose whether to disclose their data to specific parties or keep it hidden from others. Individuals have the authority to decide who can access their personal information and to what extent. For instance, someone may agree to share their aggregated health data but decline to disclose their medical records. Data privacy also allows individuals to assign different levels of sensitivity to various types of personal data. They may be more cautious about revealing their names and addresses compared to their age.

Data is essential for various types of studies, including model creation, statistical analysis, prediction generation, data analysis, result inference, interpolation of missing data, and more. Balancing the use of data with the preservation of data privacy is both a philosophy and a subject of cutting-edge research. The goal is to strike a balance between privacy preservation and extracting valuable information from data.

To better understand the concepts related to data privacy, let's review some key definitions, including sensitivity and differential privacy.

2.7.1 Sensitivity

For any two neighboring datasets, denoted as D_1 and D_2, which differ in at most one element, the sensitivity of a function $f : \mathbb{D} \to \mathbb{R}^k$ is defined as follows:

$$\Delta f = \max ||f(D_1) - f(D_2)||_1 \tag{2.44}$$

Here, Δf represents the maximum difference in the values of f when applied to two datasets that differ in at most one data point. Smaller values of Δf indicate that less noise is required to achieve differential privacy [48]. Sensitivity depends on the specific query function and is independent of the underlying database. Two commonly used sensitivity measures are l_1-sensitivity and l_2-sensitivity.

2.7.2 Differential privacy

Differential privacy [48] is a state-of-the-art privacy mechanism that quantifies privacy guarantees numerically. It defines neighboring datasets as those that differ in exactly one element ($\# ((D_1 - D_2) \cup (D_2 - D_1)) = 1$). A randomized function A taking datasets as input is said to provide (ϵ, δ)-differential privacy if, for any subset $S \subseteq \mathit{Range}(K)$ and any two neighboring datasets D_1 and D_2, the following inequality holds:

$$\Pr[A(D_1) \in S] \leq e^{\epsilon} \Pr[A(D_2) \in S] + \delta. \tag{2.45}$$

Typically, ϵ and δ are very small, and A provides ϵ-differential privacy when $\delta = 0$.

In simple terms, differential privacy ensures that the output of a function $A(D)$ does not change significantly when one data point is added to or removed from the dataset D. The values of ϵ and δ determine the tolerance to changes in $A(D)$.

A formal verification method of differential privacy in interactive systems can be found in [122].

Now that we have a basic understanding of the privacy metric, let's explore methods for achieving privacy. Privacy can be preserved using mechanisms such as encryption and the addition of noise.

2.7.3 Encryption for privacy

Data encryption is a mechanism used to protect sensitive information from unauthorized access or disclosure. Encryption serves three primary purposes: authentication (verifying the origin of the data), integrity (detecting any unauthorized data manipulation), and non-repudiation (creating a binding assurance that parties involved cannot later deny participation in data exchange).

Encryption is essential for both data at rest (e.g., stored in the cloud) and data in transit (e.g., during transmission). In encryption, the sensitive data is referred to as the 'plaintext', while the encrypted data is called the 'ciphertext'. Encryption mechanisms typically fall into the following two categories.

1. **Symmetric encryption (Private-key cryptography):** This method uses a single secret private key for both encryption and decryption. The sender uses the key to encrypt the plaintext, and the recipient uses the same key to decrypt the ciphertext. The key must be securely shared between the parties. Examples of symmetric key cryptography methods include data encryption standard (DES), triple DES, and advanced encryption standard (AES) [20].

2. **Asymmetric encryption (Public-key cryptography):** This method employs a pair of keys, consisting of a public key and a private key [20]. The public key is used for encryption, while the private key is used for decryption. Asymmetric encryption allows secure communication even over unsecured channels, as only the intended recipient with the private key can decrypt the message. However, managing key pairs requires a more complex key distribution system. Examples of asymmetric key cryptography methods include RSA [106] and Paillier cryptosystems [100].

2.7.4 Adding noise for privacy

The central concept here is to avoid sharing the exact result and instead reveal a noisy version of it. Adding noise is an effective and cost-efficient method for achieving privacy, but it can be challenging in various aspects. For instance, in

an interactive system, a repeated querying attack can compromise privacy even if only noisy answers are provided consistently.

To illustrate the concept of privacy and how the addition of noise can protect sensitive information, consider the famous example of Terry Gross's height [50]. This example has been widely used in related literature:

> Suppose one's exact height was considered a highly sensitive piece of information, and that revealing the exact height of an individual was a privacy breach. Assume that the database yields the average heights of women of different nationalities. An adversary who has access to the statistical database and the auxiliary information ''Terry Gross is two inches shorter than the average Lithuanian woman'' learns Terry Gross' height, while anyone learning only the auxiliary information, without access to the average height, learns relatively little.

Determining the appropriate noise budget and noise type can be challenging. Moreover, noise can be added at various stages in an information pipeline, raising questions about where and how much noise should be introduced. These decisions can depend on the specific situation, problem setting, the function to be applied to the data, or the level of trust between the parties involved. Additionally, increasing the amount of noise generally provides stronger privacy guarantees but can reduce the utility or performance of the algorithm.

The two most commonly used noise mechanisms are Laplacian and Gaussian noise.

2.7.4.1 Laplacian noise mechanism

In the Laplacian noise mechanism, noise is added by drawing from a Laplace distribution, as defined by Eq. (2.21), to perturb the original data and preserve privacy.

Definition: Laplace mechanism. Given any function $f : \mathbb{N}^{|\chi|} \to \mathbb{R}^k$, the Laplace mechanism is defined as follows:

$$M_L\left(x, f(.), \epsilon\right) = f(x) + (Y_1, ..., Y_k). \tag{2.46}$$

Here, the Y_i are independent Laplace random variables drawn from $\text{Lap}(\Delta f/\epsilon)$, and Δf is the l_1-sensitivity of function f as defined in Eq. (2.21).

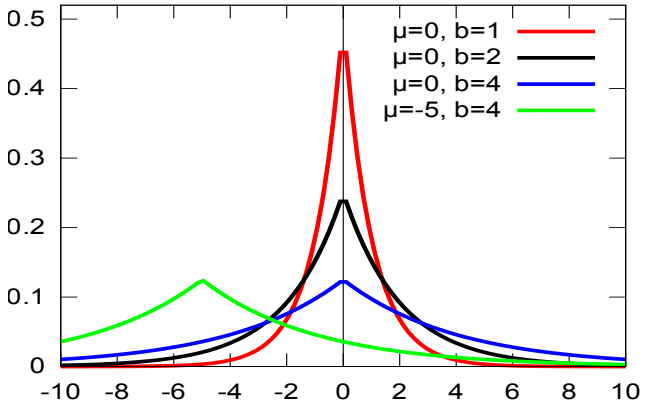


Figure 2.14: Laplace probability density function (for various μ and b). [Source]

The Laplace mechanism provides a privacy guarantee known as ϵ-differential privacy. The noise is scaled by the sensitivity of the query divided by ϵ, inversely proportional to ϵ. This means that more sensitive query results require a stronger privacy guarantee, leading to the addition of more noise. The Laplace mechanism is sometimes referred to as $(\epsilon, 0)$-differential privacy, as δ is always set to 0. Laplace noise provides better accuracy than the Gaussian mechanism discussed next but is less flexible. We illustrate the Laplace probability density function in Fig. 2.14.

2.7.4.2 Gaussian noise mechanism

An alternative to the Laplacian mechanism is the Gaussian mechanism, where noise is added by drawing from a Gaussian distribution, as defined in Eq. (2.22).

Definition: Gaussian mechanism. For $\epsilon \in (0, 1)$ and $\delta \in (0, 1)$, given any function f, the Gaussian mechanism is defined as follows:

$$M_G\left(x, f(.), \epsilon, \delta\right) = f(x) + \mathcal{N}(\mu, \sigma^2), \tag{2.47}$$

where, $\mu = 0$, and $\sigma^2 = \frac{2ln(1.25/\delta).(\Delta f)^2}{\epsilon^2}$.

The Gaussian mechanism offers a more relaxed privacy guarantee compared to the Laplace mechanism. Instead of ϵ-differential privacy, it provides (ϵ, δ)-differential privacy. The variance of the Gaussian distribution depends on the privacy parameters, i.e., ϵ, δ, and sensitivity. The Gaussian mechanism is often more effective than the Laplace mechanism when the l_2-sensitivity is lower than the l_1-sensitivity, resulting in the addition of less noise. The Laplace mechanism is limited to scenarios where only l_1-sensitivity is applicable. We illustrate the Gaussian probability density function in Fig. 2.15.

Now that we have discussed these fundamental methods for achieving privacy, let's delve deeper into the practical aspects of how differential privacy

works. We will explore the various ways in which these concepts can be applied in the framework of a privacy-preserving system.

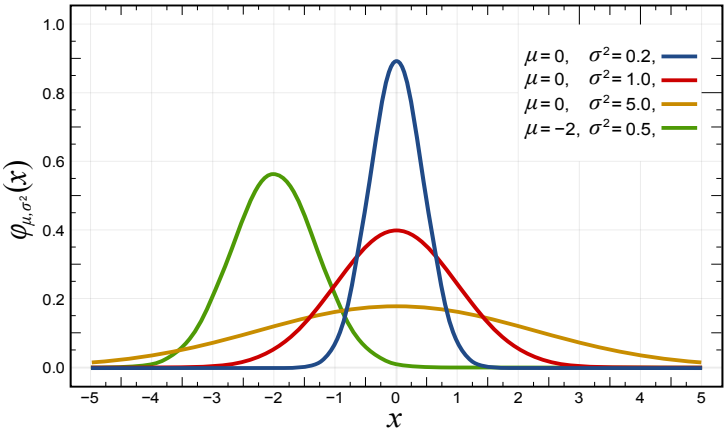


Figure 2.15: Gaussian probability density function (for various μ and σ^2). [Source]

2.7.5 Classical differential privacy concepts

Differential privacy says, in a study, the participation of a single individual mustn't significantly affect the outcome. This implies, that if a random algorithm is applied to a non-correlated dataset containing the data entry of one individual, such as Alice, replacing that data entry with another, say, Bob, should not make the new outcome more or less likely (within the ϵ factor) compared to the previous outcome.

Therefore, if an adversary possesses some prior knowledge (auxiliary information) and applies a random algorithm to the dataset to conclude the data entry, their results should not be significantly better than a factor of e^{ϵ} compared to when everything else is the same except for the data entry replacement. This represents a strong privacy guarantee, but not an absolute one. It does not rely on computational power or any auxiliary information, yet it may still bear some minimal risk to privacy. However, the benefits of the results obtained from algorithm runs on the database outweigh this minimal additional privacy risk. This concept can also be extended to group privacy but with a linear increase in the ϵ factor.

The choice of where to add noise can be crucial in terms of the resulting privacy and utility. This decision can vary widely depending on the problem setting. Factors such as the data source, collection method, query types, how the database is treated, and the frequency of such queries can all influence the choice of privacy mechanisms. There are two different approaches to classical differential privacy methods: global differential privacy and local differential privacy.

2.7.5.1 Central differential privacy

The classic central differential privacy mechanism [43] involves adding noise to the output variable y_i to obtain a noisy version $\hat{y}_i$ before revealing the response. This mechanism protects the privacy of the published output.

This mechanism is useful when the data source and the collection method are fully trusted, and individual data points in the database are trusted. The sensitivity of the function does not significantly affect the outcome, as the non-noisy versions are fed directly into the algorithm. This mechanism performs particularly well with large datasets and when only a few queries are run over them, as noise is only added to the output. However, if multiple queries are made, the realized loss may still be high. Additionally, if the number of outputs is greater than the number of inputs, the loss may be even higher.

2.7.5.2 Local differential privacy

Local differential privacy (LDP) [42] is a state-of-the-art privacy mechanism where sufficient noise is added to every input variable x_i to obtain a fully private version $\hat{x}_i$. Any query can then be answered using these noisy versions, and the output can be computed as $\mathcal{O} = f(\hat{x})$. A major drawback of this approach is that it requires a significant amount of noise, leading to a larger privacy budget and, consequently, higher expected loss and sub-optimal utility.

Local DP is more effective when the data collection and processing organization is not trusted, and individual inputs need to be differentially private. It also works well when multiple queries are frequently made over the same set of data points. However, if the queries are run on non-overlapping data points, LDP may result in lower utility. Additionally, the algorithm needs to be robust to general perturbations of individual inputs, as locally added noise may significantly affect algorithms with high sensitivity.

2.7.5.3 Other variants of differential privacy concepts

Research exists on other variants of differential privacy, their analysis, application, and comparison with the traditional CDP or LDP mechanisms. In this paper [6], they discuss three of such variants of differential privacy, namely Approximate differential privacy [45], Hypothesis test differential privacy [98], and Renyi differential privacy [92].

2.7.5.3.1 Approximate differential privacy. This is the lenient version of differential privacy where $\delta > 0$, i.e., the privacy notion allows with an ideally tiny probability δ, the differential privacy guarantee may not hold. When this δ probability is set to 0, we get the pure differential privacy, parameterized by ϵ only. Research [118] exists which tries to find better performance on privacy preservation in both pure and approximate settings.

2.7.5.3.2 Hypothesis test differential privacy. If we have a database access mechanism M, which returns random output Y, and we consider the hypothesis testing experiment, where a null hypothesis $H0$ and alternative hypothesis $H1$ is chosen such that:

$$H0 : Y \text{ came from a database } D0 \ ,$$
$$H1 : Y \text{ came from a database } D1 \ .$$

Now, for a choice of a rejection region S, the probability of type I error, i.e., when the null hypothesis is true but rejected, is defined as $P(M(D_0) \in S) \equiv P_{FA}(D_0, D_1, M, S)$ and the probability of type II error, i.e., when the null hypothesis is false but retained, is defined as $P(M(D_1) \in (\bar{S}) \equiv P_{MD}(D_0, D_1, M, S)$, and where $\bar{S}$ is the complement of S.

According to this algorithm [79], for any $\epsilon \geq 0$ and $\delta \in [0, 1]$, a database mechanism M is (ϵ, δ)-differentially private if and only if the following conditions are satisfied for all pairs of neighboring databases D_0 and D_1, and all rejection region $S \subseteq \chi$:

$$P_{FA}(D_0, D_1, M, S) + e^{\epsilon} P_{MD}(D_0, D_1, M, S) \geq 1 - \delta, \tag{2.48}$$
$$e^{\epsilon} P_{FA}(D_0, D_1, M, S) + P_{MD}(D_0, D_1, M, S) \geq 1 - \delta. \tag{2.49}$$

Here, the FA stands for false alarm and MD stands for missed detection. We can infer from this, that it is impossible to achieve small values for both probabilities of false alarm (type I error) and probabilities of missed detection (type II error) from data obtained through a differentially private mechanism, and vice-versa.

We'll discuss an analysis on the composition of differentially private mechanisms using Renyi differential privacy mechanism in Section 2.7.6.2.

Works like [3], [114] studies differential privacy mechanism in deep neural networks. In this paper [25], an alternative approach of perturbing the objective function to gain better privacy-utility trade-off over traditional output perturbation methods is studied.

Differential privacy has been studied in the context of different problems like gradient descent [128], boosting [49], empirical risk minimization [12], datamining [47], [18], distributed aggregation [44] etc. Many other works concentrate on analysis of privacy-preservation under different scenarios and tries to provide better bounds on the performance [40], [87], [7], [89], [71], [23] in terms of different differential privacy concepts.

2.7.6 Differential privacy composition rules

Differential privacy composition rules provide upper bounds on privacy parameters when multiple differentially private components in a pipeline are combined.

2.7.6.1 Classical differential privacy composition

It states that if there is one (ϵ_1, δ_1)-differentially private step and a (ϵ_2, δ_2)-differentially private step, then the combination of these two steps will provide

at worst a $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-differentially private guarantee [46] .

The composition rule similarly applies to ϵ-differential mechanisms. According to [46], the composition of an ϵ_1-differentially private mechanism and an ϵ_2-differentially private mechanism is at worst $(\epsilon_1 + \epsilon_2)$-differentially private.

2.7.6.2 Rényi differential privacy composition rule

Rényi differential privacy [92] provides a tighter bound on the composition of multiple heterogeneous components with individual differential privacy specifications. This relaxation of differential privacy is based on the parameterized Rényi Divergence.

Definition: Rényi divergence. For two probability distributions P and Q defined over $\mathcal{R}$, the Rényi divergence [105] of order $\alpha > 1$ is defined as follows:

$$D_\alpha(P \parallel Q) \triangleq \frac{1}{\alpha - 1} \log \mathbb{E}_{x \sim Q} \left(\frac{P(x)}{Q(x)} \right)^\alpha. \tag{2.50}$$

The interval $(1, \infty)$ of Rényi divergence is defined by continuity, where for $\alpha = 1$, $D_1(P \parallel Q)$ is equal to the Kullback-Leibler divergence. For $\alpha = \infty$, $D_\infty(P \parallel Q)$ or for any randomized mechanism f, it is ϵ-differentially private if and only if the distribution of f over any two adjacent inputs D and D' satisfies the Rényi divergence constraint, which should be less than or equal to the factor ϵ. This allows us to define Rényi-differential privacy.

Definition: (α, ϵ)-RDP. A randomized mechanism $f : D \rightarrowtail \mathcal{R}$ is said to have (α, ϵ)-Rényi differential privacy, or (α, ϵ)-RDP for short, if for any adjacent $D, D' \in \mathcal{D}$, the following condition holds:

$$D_\alpha(f(D) \parallel f(D')) \leq \epsilon. \tag{2.51}$$

Rényi divergence can be defined for fractional α, i.e., $\alpha \leq 0$, and even negative values. Rényi-DP can be used as a relaxed privacy measure of classic differential privacy in other situations, such as basic sequential composition or group privacy.

Studies of advanced composition algorithm of DP can be found in [49] and [98]. The paper [8] studies Rényi-DP-based interpretation of hypothesis testing.

2.8 Logic

In our work, we utilize elements from logic programming and other constructs inspired by logic. In this section, we will discuss the preliminary concepts of logic, followed by a detailed discussion of the logical Bayesian network [54], [55]. These foundational concepts are crucial for building our specification language.

2.8.1 Logic preliminaries

Now, let's delve into the definitions of some essential concepts that we will directly or indirectly use to design our logic-based privacy specification language. These definitions have mainly been taken from the book [112] and from this source.

Logic. Logic is the study of the validity of different kinds of inference. This term is often used synonymously with deductive logic, the branch of logic concerned with inferences whose premises cannot be true without the conclusion also being true. The other major branch of logic, inductive logic, deals with inferences whose premises can be true even if the conclusion is false.

Premise. A premise is a statement meant to support the conclusion of an argument.

Term. Traditionally, the subject or predicate in a categorical proposition is a term.

Atom. An atom is a formula that contains no logical connectives or equivalently a formula that has no strict subformulas.

Literal. A literal is a sentence that is either an atomic sentence or the negation of an atomic sentence.

Constant. A constant is a symbol that, under the principal interpretation, is a name for something definite, be it an individual, a property, a relation, etc.

Variable. A variable is an expression of first-order logic that is like individual constants in that it may be the argument of a predicate, but unlike constants, it can be bound by quantifiers.

Quantifier. A quantifier is an operator of which it is true that both the constant or form it is used with and the constant or form produced are propositions or propositional forms.

Logical consequence. A logical predicate is a sentence S, which is a logical consequence of a set of premises if the premises all can't be true, while the conclusion S is false.

Logical contradiction. A logical contradiction is a sentence that comes out false in every possible circumstance.

Logical truth. A logical truth is a sentence that is a logical consequence of any set of premises. That is, no matter what the premises may be, the conclusion can't be false. A logical truth thus comes out true in every possible circumstance.

Logically equivalent sentences. Two sentences are logically equivalent if they have the same truth values in all possible circumstances.

Logical implication. The relation that holds between two propositions when one is deducible from the other is called logical implication.

Negation. A negation is a first-order logic sentence that begins with a negation sign ($\neg$). The negation of a true sentence is false; the negation of a false sentence is true.

Negation elimination ($\neg$ Elim). A negation elimination is a rule of systems F and FT that permits us to infer a sentence from the negation of its

negation (e.g., to infer S from $\neg\neg S$).

Negation introduction ($\neg$ Intro). A negation introduction is a rule of systems F and FT that permits us to prove S by showing that $\neg S$ leads to a contradiction.

Logical predicate. A logical predicate is a symbol in first-order logic that is used to express a property of objects or a relation between objects.

Arithmetical predicate. A predicate that can be explicitly expressed in terms of the truth-functional connectives of propositional calculus, the universal and existential quantifiers, constant and variable natural numbers, and the addition and multiplication functions.

Relation. A relation is defined as a set of ordered pairs.

Reflexive relation. A relation R is reflexive if "aRa" holds for all a that are members of the field of R, irreflexive if "aRa" holds for no members of the field of R, and nonreflexive if "aRa" holds for some but not all members of the field of R.

Symmetric relation. A relation R is symmetric if for all a and b that are members of the field of R, "aRb" if and only if "bRa", asymmetric if for all a and b that are members of the field of R, "aRb" if and only if not-"bRa", and nonsymmetric when "aRb" and "bRa" hold for some but not all a and b that are members of the field of R.

Transitive relation. A relation R is transitive when for all a, b, and c that are members of the field of R, if "aRb" and "bRc", then "aRc", intransitive when for all a, b, and c that are members of the field of R, if "aRb" and "bRc", then not-"aRc", and nontransitive when if "aRb" and "bRc", then "aRc" holds for some but not all of the a, b, and c that are members of the field of R.

Conjunction. Conjunction is a binary propositional connective, usually read as "and ($\wedge$)", whose truth table is such that "$A \wedge B$" is false when A or B or both are false and is true when both are true.

Disjunction. Disjunction is a binary propositional connective, usually read as "or ($\vee$)", whose truth table is such that "$A \vee B$" is true when either or both of A, B is true and is false when both are false.

Clause. A clause is a disjunction of literals; that is, an expression of the form $\ell_1 \vee \ldots \vee \ell_n$, where $\vee$ is the disjunction operation and $\ell_1, \ldots, \ell_n$ are literals.

Prefix vs. infix notation. In prefix notation, the predicate or relation symbol precedes its arguments, e.g., $Larger(a, b)$. In infix notation, the relation symbol appears between its two arguments, e.g., $a = b$.

Logically possible. A sentence is logically possible if there is no logical reason it cannot be true, i.e., if there is a possible circumstance in which it is true.

Material conditional. A material conditional is a truth-functional version of the conditional "if P then Q". The material conditional $P \rightarrow Q$ is false if P is true and Q is false but otherwise true.

Well-formed formula (WFF). Well-formed formulas are the "grammatical" expressions of first-order logic. A WFF is either atomic (an n-ary predicate followed by n individual symbols), or a complex WFF is constructed using

connectives, quantifiers, and other WFFs. Atoms are the simplest well-formed formulas of logic.

Satisfaction. An object named a satisfies an atomic well-formed formula $S(x)$ if and only if $S(a)$ is true, where $S(a)$ is the result of replacing all free occurrences of x in $S(x)$ with the name a.

Necessary and sufficient conditions. A necessary condition for a statement S is a condition that must be held for S to obtain. $S \rightarrow P$ says that P is a necessary condition for S. A sufficient condition for a statement S is a condition that guarantees that S will be obtained. $P \rightarrow S$ says that P is a sufficient condition for S.

Presupposition. The presuppositions of a sentence S are those conditions that must be fulfilled for S to have a truth value, i.e., for S to make any claim at all.

Proof by cases. A proof strategy that consists of proving some statement S from a disjunction by proving S from each disjunct.

Proof by contradiction (indirect proof). To prove $\neg S$ by contradiction, we assume S and prove a contradiction. In other words, we assume the negation of what we wish to prove and show that this assumption leads to a contradiction.

Proofs without premises. A proof without premises, as the name implies, contains no premises. Such proof typically begins with a subproof assumption and ends when all subproofs have been closed. The conclusion of a proof without premises is called a theorem of the system of proof. In a sound system, every theorem is a logical consequence of the empty set of premises, i.e., a logical truth.

Conditional proof. A proof that begins by making certain assumptions, $A_1, A_2, \cdots, A_n$, deducing B from them, and then asserting based on this the truth of the hypothetical proposition 'if A_1, then if A_2, then if $\cdots$, then if A_n, then B'. The rule of conditionalization is the rule that allows one to make this last step based on the preceding ones.

After a quick review of the basic definitions of preliminary concepts in logic, we will proceed to provide a thorough description of the Logical Bayesian network.

2.8.2 Logical Bayesian network

The logical Bayesian network (LBN) model is based on the principles of knowledge-based model construction (KBMC), which combines the concepts of logical programming and probabilistic models. It leverages the advantages of both fields: the ability to represent relational data through logical programs and the ability to model noisy data through probabilistic models. LBN is often used in conjunction with machine learning.

We chose to use the concepts of LBN among the many alternatives because we are also utilizing the intersection between the two fields: logical modeling and probabilistic modeling. Despite not being the most popular framework, LBNs have the advantage of being quite interpretable, aligning more with our end goal. Since relational databases can be easily converted into logical programs [33], also

known as inductive logic programming [93], logical programs are well-suited for modeling the data in relational databases.

Next, we will delve into the definitions of the different components of LBN. We will use these components to formally define the LBN [55].

2.8.2.1 Definition: Random variable declaration

A random variable declaration is a range-restricted clause of the form

$$\text{random}(pAtom) \leftarrow lit_1, \ldots, lit_n,$$

where $n \geq 0$, $pAtom$ is a probabilistic atom, and $lit_1, \ldots, lit_n$ are logical literals.

2.8.2.2 Definition: Conditional dependency clause

A conditional dependency clause is a clause of the form

$$pAtom \mid pAtom_1, \ldots, pAtom_n \leftarrow lit_1, \ldots, lit_m,$$

where $n, m \geq 0$, $pAtom, pAtom_1, \ldots, pAtom_n$ are probabilistic atoms, and lit_1, $\ldots, lit_m$ are logical literals.

2.8.2.3 Definition: Logical CPD

A logical CPD for a probabilistic predicate p is a function mapping a set of ground probabilistic atoms to a conditional probability distribution (CPD) in the range of p.

Now, let's look at the formal definition of LBN.

2.8.2.4 Definition: Logical Bayesian network

A Logical Bayesian network [55] is a tuple $(\mathcal{R}, \mathcal{D}, \mathcal{C})$ with $\mathcal{R}$ as a set of random variable declarations, $\mathcal{D}$ as a set of conditional dependency clauses, and $\mathcal{C}$ as a set of logical CPDs, one for each probabilistic predicate.

In addition to the constructs we have encountered so far, we need one more: the dependency statement, to understand the semantics of the distribution given by the logical Bayesian network.

2.8.2.5 Definition: Dependency statement

A dependency statement is of the form

$$a \mid a_1, \ldots a_n \leftarrow l_1, \ldots, l_m,$$

where $n \geq 1$, $m \geq 0$, $a, a_1, \ldots, a_n$ are probabilistic atoms, and $l_1, \ldots, l_m$ are logical literals.

Here, a is called the head, $a_1, \ldots, a_n$ is called the body, and $l_1, \ldots, l_m$ is called the context, which is also optional. They have also defined the **parent**

term, which can be derived from the above definition. Any variable in the body is the parent of the variable in the head for a true context.

Bayesian networks are often used for the representation of joint probability distribution and for computing probabilistic inference. LBNs formally extend the directed acyclic Bayesian networks to the case of relational data to represent directed probabilistic logical models. LBN uses the context-specific independence property of Bayesian networks, which exploits the superiority of logical probability trees over logical probability tables with rules of combination.

2.8.2.6 Definition: Semantics of an LBN

The joint probability distribution defined by a Bayesian network for a set of random variables $X = \{X_1, \ldots, X_n\}$ is given by

$$\Pr(X_1, \ldots, X_n) = \prod_{i=1}^{n} \Pr(X_i | Pa(X_i)),$$

where $Pa(X_i)$ denotes the set of parents of X_i for all i.

Next, we will briefly define how LBN can be used to create a predicate dependency graph, a concept that will be used in our work to illustrate an example scenario.

2.8.2.7 Definition: Predicate dependency graph of an LBN

The predicate dependency graph of an LBN is the graph that contains a node for each probabilistic predicate and a directed edge from a node p_1 to a node p_2 if the LBN contains a dependency statement with predicate p_2 in the head and p_1 in the body. We will illustrate this with an example taken from [54], in the next section.

2.8.2.8 Running example of LBN

We now introduce a small example to illustrate the components of the logical Bayesian network, which we have taken verbatim from [54], based on the "university"-example [63].

```
There are students and courses. We know which students take
which courses. Each student has an IQ and a final ranking, and
each course has a difficulty level. A student taking a certain
course gets a grade for that course. The grade of a student for
a course depends on the IQ of the student and the difficulty of
the course. The final ranking of a student depends on their
grades for all the courses they're taking.
```

Now we specify the clauses they have used in the original paper (and thesis) [54] to explain their example. First, we list the logical predicates:

`student/1, course/1, and takes/2.`

Then, the probabilistic predicates are:
`iq/1, diff/1, ranking/1, grade/2.`
The associated ranges for the above predicates are {low, high}, {low, middle, high}, {A, B, C}, and {A, B, C}, respectively.

Next, the random variable declarations are as follows:
`random(iq(S)) ← student(S).`
`random(ranking(S)) ← student(S).`
`random(diff(C)) ← course(C).`
`random(grade(S,C)) ← takes(S,C).`

Then we list the conditional dependency statements:
`ranking(S) | grade(S,C) ← takes(S,C).`
`grade(S,C) | iq(S), diff(C).`

The mapping of the problem into the Bayesian network in terms of a normal logic program or as known as Bayesian ground facts:
`student(john).`, `course(ai).`, `takes(john,ai).`, `student(pete).`,
`course(db).`, `takes(john,db).`, `takes(pete,ai).`

The LBN contains CPD of iq(john):
`p(iq(john)), low: 0.4, high: 0.6.`

LBN gives the user the freedom to design their predicates to be either deterministic or probabilistic, as they choose for individual applications. Deterministic predicates become logical predicates, and CPD is defined for the probabilistic ones. We can see the predicate dependency graph of the running example in Fig. 2.16.

Also, LBN allows negated atoms. To express if "a student has a grade for a course if he was taking that course unless he was absent on the exam," we write:
`random(grade(S,C)) ← takes(S,C), not(absent(S,C)).`

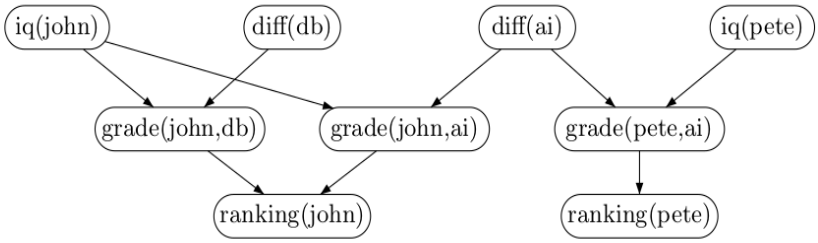


Figure 2.16: Structure of the Bayesian network induced for the running example. [Adapted from 54]

2.9 Constraint program solvers and CVXOPT

We have discussed constraint optimization in detail in Section 2.3.5.3. We will now discuss some of the solver tools for such optimization problems in many iterations. It implements a stopping condition that is met when a certain value of satisfaction criteria goes lower than a pre-specified threshold. So, we cut some slack to the constraint satisfaction criteria and declare that we have reached the optimal solution once the stopping condition is satisfied.

The constraint optimization problem requires empirical evaluation. Awareness of an accessible and well-maintained constraint optimization solver saves time when preparing experiments. There are many Python-based solvers with their advantages and disadvantages. We looked into multiple of them like **Gurobi**, **PuLP**, and **Scipy** methods (scipy.optimize.minimize or scipy.optimize.lin-prog).

After our evaluation, we decided to use CVXOPT [5], a Python programming language-based software used for convex optimization. It uses extensive Python libraries to provide a simpler way to solve convex optimization problems. Next, we discuss the basic elements of the problem definition that are accepted by the CVXOPT solver.

Nonlinear convex optimization CVXOPT solvers consider the constraint optimization problem in the following form:

$$\begin{aligned} \min \quad & f_0(x), && && (2.52)\\ \text{subject to} \quad & f_k(x) \leq 0, && k = 1, \ldots, m && (2.53)\\ & Gx \preceq h, && && (2.54)\\ & Ax = b. && && (2.55) \end{aligned}$$

where $x \in \mathbb{R}^n$ are constraint variables, and n is the number of constraint variables.

As we can see in the notations above, the CVXOPT solvers accept some pre-specified data structures representing the following four structures of the constraint programs:

- **The objective function**: Here, f_0 is a twice-differentiable convex objective function and is subject to the constraints. The objective function $f_0 : \mathbb{R}^n \to \mathbb{R}^+$ is mapped to the positive real number space. The form of the generalized nonlinear objective function stated above changes into $f_0(x) = c^T x$ when we have a linear objective function, where c is a real single-column dense matrix.
- **The linear equality constraints**: The matrix $G \in \mathbb{R}^{l \times n}$ is a real dense or sparse matrix that contains the coefficients of the constraint variables

in the linear equality constraints. The argument h is a real single-column dense matrix that contains the constant terms in the linear equality constraints.

- **The linear inequality constraints**: The matrix $A \in \mathbb{R}^{k \times n}$ is a real dense or sparse matrix that contains the coefficients of the constraint variables in the linear inequality constraints. The argument b is a real single-column dense matrix that contains the constant terms in the linear inequality constraints. If there are no equality constraints, then the default values for matrices A and b are sparse matrices with zero rows.
- **The non-linear convex constraints**: Each of the constraint functions $f_1(x), \ldots, f_m(x) : \mathbb{R}^n \to \mathbb{R}$ are also twice-differentiable convex functions, mapped to the real number space ($\mathbb{R}$). Here, $k = 1, \ldots, m \in \mathbb{N}$ are the number of non-linear convex constraints.

In all three next Chapters 3, 4, 5 of this dissertation, which represents our main body of research work, the constraint optimization problem is implemented using this tool to find the optimal solution.

Chapter 3

Interpretable privacy with optimizable utility

abstract

In this chapter, we discuss the problem of specifying privacy requirements for machine learning-based systems in an interpretable yet operational way. Explaining privacy-improving technology is a challenging problem, especially when the goal is to construct a system that is interpretable and has high performance. To address this challenge, we propose to specify privacy requirements as constraints, leaving several options for the concrete implementation of the system open, followed by a constraint optimization approach to achieve an efficient implementation, next to the interpretable privacy guarantees.

3.1 Introduction

Over recent years, one has seen an increasing interest in privacy as awareness of the privacy risks of data processing systems has increased. Legislation was introduced to protect data, and sufficient data and insights became available to create technology capable of realizing several tasks while preserving the privacy of participants. One of the most popular notions of privacy, which we will also adopt in this chapter, is differential privacy [43] and its extensions, e.g., [116], [51].

An important aspect of this evolution concerns informing users of the privacy guarantees a system offers. Some legislation, such as Europe's GDPR [2], requires transparency, i.e., users have the right to know how their data are used and how their sensitive data are protected. Explaining privacy protection strategies is also important to increase trust among the users of a system. Finally, being able to explain what privacy guarantees a system offers is also helpful in the sometimes challenging communication between computer scientists who

develop solutions and legal experts who are interested in understanding the guarantees without the burden of having to investigate many technical details.

While a large number of papers in the machine learning community study a single machine learning problem and strategies to perform that machine learning task in a privacy-preserving way, real-world systems are often complex, consisting of several machine learning, preprocessing, prediction (inference) steps, user interactions, and data transfers. The privacy requirements of interest to a user are requirements on the system as a whole, combining the behavior of its many components, including their privacy guarantees. While some researchers have focused on analyzing the privacy guarantees of complete systems, the literature on that topic is still rather limited.

Such large systems combine heterogeneous components, each having its own characteristics that concern their effects on the privacy of the data. There is an increasing need for systems that allow one to specify and explain the privacy guarantees for a complete system. However, next to interpretability, performance, e.g., in terms of precision of computation, communication, and storage cost, is also required. In this chapter, we study strategies to achieve both interpretability and good performance.

In particular, we argue that composition rules for differential privacy, which start from the building blocks and combine them bottom-up, may not offer sufficient flexibility. We suggest an alternative approach where privacy requirements are specified top-down and implementation choices, such as the allocation of "privacy budget" to several components or the choice between more costly multi-party computing and less accurate noisy data sharing, are optimized afterward.

We start in Section 3.2 with a brief review of relevant literature and a discussion of the advantages and drawbacks of several strategies. We then sketch our ideas in Section 3.3 and provide a number of examples to illustrate them.

3.2 Existing approaches

An important notion in the context of privacy is differential privacy [43] and we have discussed them in Section 2.7.2.

Several variants and generalizations of differential privacy have been proposed, including proposals focusing on the adversarial model [51] and proposals allowing for more refined secret definitions [116]. In this chapter, we occasionally adopt the term 'secret' as introduced in Pufferfish privacy [116] to describe variables that are private but not tied to the individual level within a database of individuals, as typically seen in classic differential privacy. There are approaches like metric-based DP [4] where it improves utility of algorithms for the same level of privacy in terms of ϵ-DP guarantee provided by the traditional (local) differential privacy mechanism.

A wide variety of languages have been proposed to describe the privacy properties of systems. Some are aimed at compilers or circuit evaluators [83], and others are not necessarily aimed at privacy-preserving technology but rather

at trust or consent [101]. In the sequel, we will focus our discussion on languages aimed at specifying the privacy properties of systems using privacy-preserving technology.

A classical approach to studying the privacy of a compound system is to take the different components as input and analyze the behavior of the compound system. One basic strategy is to apply composition rules for differential privacy. The basic rule states that if data is queried twice, once with an (ϵ_1, δ_1)-differentially private algorithm and once with a (ϵ_2, δ_2)-differentially private algorithm, then the combination is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-differentially private. Even though this always holds, this is usually not a tight bound on the privacy of the combination. A number of improved bounds have been proposed, e.g., [79], but even those are often not immediately practical. One issue is that the order of steps to be performed in a system may not be known a priori, e.g., a system could branch due to an if-then-else decision, or parts could be repeated.

To address this problem and at the same time have a more uniform way to represent privacy properties, many authors have proposed languages to specify privacy properties, together with associated techniques to verify whether these properties are satisfied when several rules can be applied [129], [102], [97]. The advantage is that next to a language, there is a system that can attempt to verify whether a given system satisfies the described privacy properties. However, several problems remain. First, theorem-proving style techniques usually only work for a limited set of rules or reasoning primitives, and they don't scale very well with increasing problem size. Second, while verifying that a property holds, is interesting, optimizing the performance would be even better. In such theorem-proving settings, it remains the task of the human expert to design the characteristics of the individual components of the system and combine them such that they collaborate efficiently.

3.3 Privacy constraint optimization

Similar to earlier work discussed in the previous section, our approach starts from a language to describe privacy constraints. However, rather than aiming at verification, we aim at optimization. We propose to first formulate the problem and its privacy requirements in a systematic way as depicted in Fig. 3.1, and then to treat these privacy requirements as the constraints of an optimization problem where the objective function is the utility, or conversely the loss. The loss function can incorporate various types of costs, such as the expected error on the output, the computational cost of the resulting system, or its storage cost.

Below, we first sketch at a high level, how problems can be specified. Next, we provide examples of this idea applied to different types of problems. In this chapter, our goal is not to improve some quantifiable performance measures or to solve more difficult problems than before, but rather to illustrate that the idea of constraint programming with privacy requirements has several potentially interesting applications.

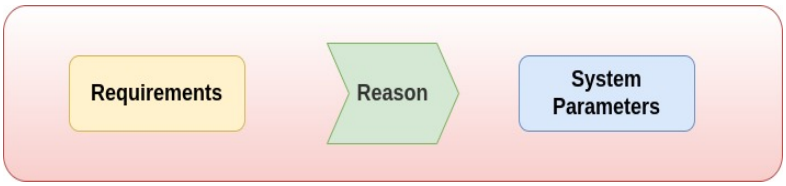


Figure 3.1: The high-level illustration of our approach.

3.3.1 Problem specification

For this chapter, our main aim is not to develop a complete language allowing for representing as many as possible privacy properties (languages to describe privacy properties (as in Fig. 3.2) have been proposed in the literature to some extent), our main objective is to open the discussion on how to optimize the performance of a system given fixed global privacy requirements.

Therefore, we will use only a basic language sufficient for our examples. In particular, we distinguish the following components in a specification:

- **Declaring relevant variables.** We will treat both data and models as random variables: data may be public or private and may be drawn jointly with other data variables from some distribution. According to the definition of differential privacy, a differentially private learning algorithm must be a randomized algorithm that outputs a model that follows some probability distribution conditioned on the training data.
- **Specifying relations between variables (background knowledge).** After specifying the relevant random variables, we can specify the conditional dependencies between these variables using a probabilistic model, e.g., a Bayesian network or a Markov random field.
- **Privacy requirements.** Then, we can specify the required privacy properties. This typically involves specifying that the several possible values of a secret can't be distinguished with significant probability by parties not authorized to know the secret.

Below, we present a number of examples of scenarios where we can apply the proposed technique of compiling privacy requirements to constraint programs.

3.3.2 Optimizing differential privacy noise as a function of the desired output

Assume we have n sensitive input variables and we want to answer m queries over these sensitive variables. For the simplicity of our presentation, we will assume that the answer to each query is a linear combination of the sensitive variables. We can specify our problem as follows:

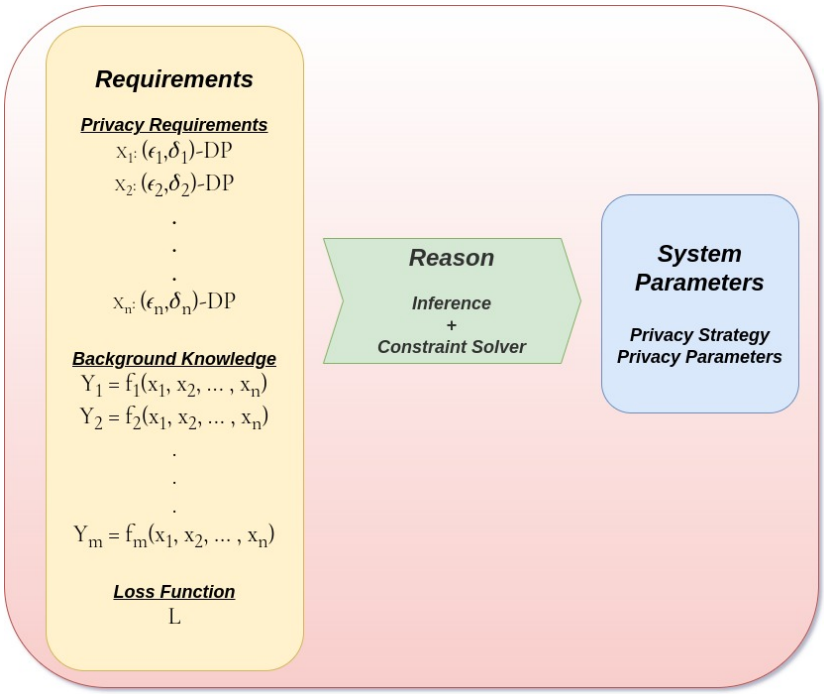


Figure 3.2: The detailed illustration of the components in our process, starting from the requirement specification, through inference and solving the constraint optimization problem, to achieve optimal system parameters like noise variances, and design choices on privacy strategies.

Variables:

- $x \in [0,1]^n$: input
- $y \in \mathbb{R}^m$: intermediate variable
- $A \in \mathbb{R}^{m \times n}$: constant
- $b \in \mathbb{R}^m$: constant
- $\mathcal{O}$: output

Background knowledge:

- $y = Ax + b$
- Loss function: $L = \|\mathcal{O} - y\|_2^2$

Privacy requirements:

- $\mathcal{O}$ is (ϵ, δ)-DP w.r.t. x.

Here, we organize our specification as outlined in Section 3.3.1. It is clear that revealing the exact answers to the queries y_i is unacceptable, resulting in some approximation of the original answers to the queries. So, we specify a loss function representing the cost of the approximation errors.

There are two classic approaches. First, one can use Local Differential Privacy (LDP) [42]. This means that to every input variable x_i sufficient noise is added to obtain a fully private version $\hat{x}_i$. Next, any query can be answered starting from these noisy versions, so, the output can be computed as $\mathcal{O} = A\hat{x} + b$. A major drawback of this approach is that this requires a lot of noise and hence the expected loss will be high. For the simplicity of our analysis, we use the Gaussian mechanism throughout this example. We get:

$$\mathbb{E}\left[L_{LDP}\right] = tr(A^\top A)\left(\frac{2\log(1.25/\delta)}{\epsilon^2}\right). \tag{3.1}$$

Second, one can use classic differential privacy for each query y_i separately, adding noise to every y_i to obtain a noisy version $\hat{y}_i$. If multiple queries are obtained, the realized loss may be still high, and if $m > n$, even higher than in the LDP case above:

$$L_{DP} = \left(\sum_{j=1}^{m}\left(\sum_{i=1}^{n}|A_{i,j}|\right)^2\right)\left(\frac{2\log(1.25/\delta)}{\epsilon^2}\right). \tag{3.2}$$

In contrast, given the specifications above, we propose to (semi-automatically) generate options to address the privacy requirements, not committing to adding noise to input (as in LDP) or output (as in classic DP), but to address the possibility to add noise at meaningful points in the computation. This could result in the following constraint optimization program:

Minimize

$$\mathbb{E}_{\eta,\xi}\left[L\left(\sigma_{(\eta)},\sigma_{(\xi)}\right)\right] = \mathbb{E}_{\eta,\xi}\left[\|\mathcal{O} - (Ax+b)\|_2^2\right]$$

Subject to

- $\hat{x} = x + \eta$
- $\mathcal{O} = \hat{y} = A\hat{x} + b + \xi$
- $\eta_i \sim \mathcal{N}\left(0, \sigma^2_{(\eta),i}\right)$
- $\xi_i \sim \mathcal{N}\left(0, \sigma^2_{(\xi),i}\right)$
- $\mathcal{O}$ is (ϵ, δ)-DP w.r.t. x.

For one query ($m = 1$), the optimal solution to this problem will correspond with classic differential privacy. In the case the number of queries m is large, the solution of the optimization problem will converge to the local differential privacy case. Between the two extremes, we expect a loss that is lower than either of the classic strategies.

The constraint program we consider is easy to solve numerically, and if some approximations are made which are commonly used for the Gaussian mechanism, we get a relaxed constraint optimization problem only involving quadratic functions.

We hence distinguish four steps to address problems with privacy requirements:

1. Specifying the problem and the privacy requirements
2. Adding options to realize the privacy requirements and casting it into a constraint optimization problem
3. Solving the constraint optimization problem
4. Executing the algorithm with the obtained solutions and parameters

We show next, how these can be realized mathematically. It establishes an argument that we can translate privacy requirements into a constraint problem, which we can then optimize using a classic constraint programming solver.

3.3.2.1 A simple case with normal random variables

Suppose that we require a linear model to be locally differentially private. More concretely, suppose we run m queries on n sensitive variables, where $n \geq m > 0$ are integers, and let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ be constant, $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, and

$$y = Ax + b. \tag{3.3}$$

Revealing the query-outputs y may reveal information about sensitive x. To control this leakage, we can instead add noise at different points of the calculation of y, once before the computation and once after, as follows:

$$\hat{y} = A(x + \eta) + b + \xi, \tag{3.4}$$

where η and ξ are normally-distributed random vectors such that,

$$\eta_i \sim \mathcal{N}(0, \sigma_{\eta,i}), \quad \text{for } i \leq n, \tag{3.5}$$

$$\xi_j \sim \mathcal{N}(0, \sigma_{\xi,j}), \quad \text{for } j \leq m. \tag{3.6}$$

Similarly, for an adjacent dataset (with difference in exactly one data-point), we can assume the output will be,

$$y' = Ax' + b. \tag{3.7}$$

Our goal is to make the observed noisy outputs y', (ϵ, δ)-differentially private such that the adjacent datasets where y and y' come from can't be differentiated significantly. So they reveal no significant information about the presence of either x or x' in the dataset. In particular, for any $k \in [n]$ and for any $x, x' \in [0,1]^n$ such that $x - x' = 1_k$, we require for some $\epsilon_k, \delta_k > 0$:

$$\Pr(x \mid \hat{y}) \leq e^{\epsilon_k} \Pr(x' \mid \hat{y}) + \delta_k. \tag{3.8}$$

To achieve this, it is sufficient to prove that the following Eq. (3.9) holds at least with probability $1 - \delta_k$ (over the noise),

$$\left| \frac{\Pr(x \mid \hat{y})}{\Pr(x' \mid \hat{y})} \right| \leq e^{\epsilon_k}. \tag{3.9}$$

So, from the Eq. (3.9), we can say that the following holds:

$$\begin{aligned}
&\left| \log \left(\frac{\Pr(x \mid \hat{y})}{\Pr(x' \mid \hat{y})} \right) \right| \\
&= \left| \log \left(\frac{\Pr(x) \Pr(\hat{y} \mid x) / \Pr(\hat{y})}{\Pr(x') \Pr(\hat{y} \mid x') / \Pr(\hat{y})} \right) \right| \quad \text{(using Bayes' theorem)} \\
&= \left| \log \left(\frac{\Pr(\hat{y} \mid x)}{\Pr(\hat{y} \mid x')} \right) \right| \\
&= \left| \log \left(\frac{(2\pi m)^{-1/2} \exp\left(-(\hat{y} - y)^\top \Sigma^{-1} (\hat{y} - y)/2\right)}{(2\pi m)^{-1/2} \exp\left(-(\hat{y} - y')^\top \Sigma^{-1} (\hat{y} - y')/2\right)} \right) \right| \\
&= \left| \log \left(\frac{\exp\left(-(\hat{y} - y)^\top \Sigma^{-1} (\hat{y} - y)/2\right)}{\exp\left(-(\hat{y} - y')^\top \Sigma^{-1} (\hat{y} - y')/2\right)} \right) \right| \\
&= \left| \log \left(\exp \left(\frac{-(\hat{y} - y)^\top \Sigma^{-1} (\hat{y} - y) + (\hat{y} - y')^\top \Sigma^{-1} (\hat{y} - y')}{2} \right) \right) \right| \\
&= \left| \frac{-(\hat{y} - y)^\top \Sigma^{-1} (\hat{y} - y) + (\hat{y} - y')^\top \Sigma^{-1} (\hat{y} - y')}{2} \right| \\
&= \frac{1}{2} \left| -\hat{y}^\top \Sigma^{-1} \hat{y} + 2\hat{y} \Sigma^{-1} y - y^\top \Sigma^{-1} y + \hat{y}^\top \Sigma^{-1} \hat{y} - 2\hat{y} \Sigma^{-1} y' + (y')^\top \Sigma^{-1} y' \right| \\
&= \frac{1}{2} \left| 2\hat{y} \Sigma^{-1} (y - y') + (y' + y)^\top \Sigma^{-1} (y' - y) \right| \\
&= \frac{1}{2} \left| (y' + y - 2\hat{y})^\top \Sigma^{-1} (y' - y) \right| \\
&= \frac{1}{2} \left| (Ax' + b + (Ax + b) - 2(A(x + \eta) + b + \xi))^\top \Sigma^{-1} (y' - y) \right| \\
&\quad \text{(substituting } y \text{ and } y', \text{ by } Ax + b \text{ and } Ax' + b \text{ above, respectively)} \\
&= \frac{1}{2} \left| (Ax' + b - (Ax + b) - 2(A\eta + \xi))^\top \Sigma^{-1} (y' - y) \right| \\
&= \frac{1}{2} \left| (y' - y)^\top \Sigma^{-1} (y' - y) - 2 (A\eta + \xi)^\top \Sigma^{-1} (y' - y) \right| \qquad (3.10)
\end{aligned}$$

So, as we stated in Eq. (3.9), we require this expression on the righthand-side to be smaller than ϵ_k with probability at least $1-\delta_k$. Due to the absolute value, we require a probability of at least $1-\delta_k/2$ on both sides. This yields us the following:

$$\Pr\left(\frac{1}{2}\left(-2\left(A\eta+\xi\right)^\top\Sigma^{-1}(y'-y)+\left(y'-y\right)^\top\Sigma^{-1}\left(y'-y\right)\right)\geq\epsilon_k\right)\leq\frac{\delta_k}{2}. \tag{3.11}$$

Equivalently we can write,

$$\Pr\left(2\left(A\eta+\xi\right)^\top\Sigma^{-1}(y-y')\geq 2\epsilon_k-\left(y-y'\right)^\top\Sigma^{-1}\left(y-y'\right)\right)\leq\frac{\delta_k}{2}. \tag{3.12}$$

The variance of the term appearing on the left side of Eq. (3.12) is,

$$\begin{aligned}&\mathrm{var}\left(2\left(A\eta+\xi\right)^\top\Sigma^{-1}(y'-y)\right)\\ &= 4(y'-y)^\top\Sigma^{-1}\mathrm{var}(A\eta+\xi)\Sigma^{-1}(y'-y)\\ &= 4(y'-y)^\top\Sigma^{-1}(y'-y).\end{aligned} \tag{3.13}$$

For any centered Gaussian random variable z with variance σ_z^2, we have the following tail bound:

$$\Pr(z\geq\lambda)\leq\frac{\sigma_z}{\lambda\sqrt{2\pi}}\exp\left(-\lambda^2/2\sigma_z^2\right). \tag{3.14}$$

We will apply this to Eq. (3.12), setting the following assignments,

$$z = 2\left(A\eta+\xi\right)^\top\Sigma^{-1}(y-y'), \tag{3.15}$$

$$\lambda = 2\epsilon_k-\left(y-y'\right)^\top\Sigma^{-1}\left(y-y'\right), \tag{3.16}$$

$$\sigma_z^2 = 4(y'-y)^\top\Sigma^{-1}(y'-y). \qquad \text{(refer to Eq. (3.13))} \tag{3.17}$$

The term appearing on the right-hand side of Eq. (3.14) is smaller than $\delta_k/2$ if the following holds,

$$\log\left(\frac{\lambda}{\sigma_z}\right)+\frac{1}{2}\left(\frac{\lambda}{\sigma_z}\right)^2\geq\log\left(\frac{2}{\delta_k\sqrt{2\pi}}\right). \tag{3.18}$$

We will denote with $\sigma^2_{GM}(\epsilon,\delta)$, the minimal variance of additive Gaussian noise needed to make a variable in the range $[0,1]$, (ϵ,δ)-differentially private. E.g., [48] shows that if $\sigma\epsilon\geq 3/2$ and $(\sigma\epsilon)^2\geq 2\log(1.25/\delta)$ then $\sigma\geq\sigma^2_{GM}(\epsilon,\delta)$.

We briefly repeat here an adapted version of their derivation, where we set $\sigma=2/\sigma_z$. To make this inequality hold, we require that both the following Eq. (3.19) and (3.20) hold,

$$\log\left(\frac{\lambda}{\sigma_z}\right)\geq 0\equiv\lambda\geq\sigma_z, \tag{3.19}$$

and

$$\frac{1}{2}\left(\frac{\lambda}{\sigma_z}\right)^2 \geq \log\left(\frac{2}{\delta_k\sqrt{2\pi}}\right). \tag{3.20}$$

From Eq (3.16) and (3.17), we see that,

$$\lambda = 2\epsilon_k - \sigma_z^2/4. \tag{3.21}$$

We observe that, if $2\epsilon_k/\sigma_z \geq 3/2$, then the following satisfies Eq. (3.19),

$$\begin{aligned}
\frac{\lambda}{\sigma_z} &= \frac{2\epsilon_k}{\sigma_z} - \frac{\sigma_z}{4} \\
&= \frac{2\epsilon_k}{\sigma_z} - \frac{\sigma_z}{2\epsilon_k}\frac{\epsilon_k}{2} \\
&\geq \frac{2\epsilon_k}{\sigma_z} - \frac{\sigma_z}{2\epsilon_k}\frac{1}{2} \\
&\geq \frac{3}{2} - \frac{2}{3}\frac{1}{2} > 1.
\end{aligned}$$

Moreover, if also $2\epsilon_k/\sigma_z \geq \sqrt{2\log(1.25/\delta_k)}$, there holds,

$$\begin{aligned}
\frac{1}{2}\left(\frac{\lambda}{\sigma}\right)^2 &= \frac{1}{2}\left(\frac{2\epsilon_k}{\sigma_z} - \frac{\sigma_z}{2\epsilon_k}\frac{\epsilon_k}{2}\right)^2 \\
&\geq \frac{1}{2}\left(\frac{2\epsilon_k}{\sigma_z} - \frac{\sigma_z}{2\epsilon_k}\frac{1}{2}\right)^2 \\
&= \frac{1}{2}\left(\left(\frac{2\epsilon}{\sigma_z}\right)^2 - 1 + \left(\frac{\sigma_z}{2\epsilon_k}\frac{1}{2}\right)^2\right) \\
&\geq \frac{1}{2}\left(\left(\frac{2\epsilon_k}{\sigma_z}\right)^2 - 1 + \left(\frac{2}{3}\frac{1}{2}\right)^2\right) \\
&= \frac{1}{2}\left(\left(\frac{2\epsilon_k}{\sigma_z}\right)^2 - \frac{8}{9}\right) \\
&\geq \frac{1}{2}\left(2\log(1.25/\delta_k) - \frac{8}{9}\right) \\
&\geq \log\left(\sqrt{\frac{2}{\pi}}\frac{1}{\delta_k}\right),
\end{aligned}$$

and which satisfies Eq. (3.20).

Next to Dwork's conditions [48], other upper bounds for the function $\sigma^2_{GM}(\epsilon,\delta)$ have been proposed, e.g., [9]. In general, in the sequel, we will assume that

$$(2/\sigma_z)^2 \geq \sigma^2_{GM}(\epsilon,\delta). \tag{3.22}$$

Substituting σ_z^2 from Eq (3.17), we see this is equivalent to

$$4\left(4(y'-y)^\top\Sigma^{-1}(y'-y)\right)^{-1} \geq \sigma^2_{GM}(\epsilon_k,\delta_k).$$

In this equation we can also substitute $y - y' = Ax + b - Ax' - b = A(x - x')$, to obtain,

$$\left((x' - x)^\top A^\top \Sigma^{-1} A(x' - x)\right)^{-1} \geq \sigma^2_{GM}(\epsilon_k, \delta_k). \tag{3.23}$$

We also know that $x - x' = 1_k$ as per our assumption, which simplifies the inequality to,

$$\left(A^\top_{:,k} \Sigma^{-1} A_{:,k}\right)^{-1} \geq \sigma^2_{GM}(\epsilon_k, \delta_k) \tag{3.24}$$

$$\equiv A^\top_{:,k} \Sigma^{-1} A_{:,k} \leq \sigma^{-2}_{GM}(\epsilon_k, \delta_k) \tag{3.25}$$

$$\equiv A^\top_{:,k} \Sigma^{-1} A_{:,k} \leq \frac{2\log(1.25/\delta_k)}{\epsilon^2_k}. \tag{3.26}$$

For each k, this is a convex constraint on the variables σ^2_η and σ^2_ξ.

Before we design the final constraint optimization problem, we will need to calculate the Σ as below,

$$\begin{aligned}
\Sigma &= var_{\eta,\xi}(\hat{y}) && (3.27)\\
&= var_{\eta,\xi}(A(x+\eta) + b + \xi) && (3.28)\\
&= \mathbb{E}_{\eta,\xi}\left[(\hat{y} - \mathbb{E}[\hat{y}])(\hat{y} - \mathbb{E}[\hat{y}])^\top\right] \quad (\text{as } \mathbb{E}[\hat{y}] = \mathbb{E}[A(x+\eta) + b + \xi] = Ax + b)\\
&= \mathbb{E}_{\eta,\xi}\left[(A(x+\eta) + b + \xi - (Ax + b))(A(x+\eta) + b + \xi - (Ax + b))^\top\right]\\
&= \mathbb{E}_{\eta,\xi}\left[(A\eta + \xi)(A\eta + \xi)^\top\right]\\
&= \mathbb{E}_{\eta,\xi}\left[A\eta\eta^\top A^\top + 2A\eta\xi^\top + \xi\xi^\top\right]\\
&= A\mathbb{E}[\eta\eta^\top]A^\top + \mathbb{E}[\xi\xi^\top]\\
&= A\,\mathrm{diag}(\sigma_\eta)\,A^\top + \mathrm{diag}(\sigma_\xi) && (3.29)\\
&= [A\ I]\mathrm{diag}(\sigma_\eta, \sigma_\xi)^2 \begin{bmatrix} A^\top \\ I \end{bmatrix} && (3.30)\\
&= [A\ I]\mathrm{diag}(\sigma_\eta, \sigma_\xi)^2 [A\ I]^T. && (3.31)
\end{aligned}$$

Forming the final constraint optimization problem

We know, that choosing too large values for each $\sigma_{\eta,i}$ and each $\sigma_{\xi,j}$ aids in protecting the privacy of x but sacrifices the utility of $\hat{y}$. In contrast choosing too small values for $\sigma_{\eta,i}$ and $\sigma_{\xi,j}$, increases the utility of $\hat{y}$ but sacrifices the privacy of x. We strike a balance by constructing a constraint program that maximizes the utility subject to a constraint on the desired privacy. So, we formalize our structure using two components: **utility maximization** and **privacy constraint**.

1. ***Maximizing utility*** The objective of this program is to minimize a function that measures the loss in utility between y and $\hat{y}$. A natural function for this is

$$\mathbb{E}_{\sigma_\eta, \sigma_\xi}\left[\|y - \hat{y}\|^2\right]. \tag{3.32}$$

2. ***Privacy constraint*** Obviously, the optimal of this function is when the variances are all zero, so we constraint σ_η and σ_ξ so that $\hat{y}$ is (ϵ, δ)-locally-differentially private, for a fixed $\epsilon > 0$ and $\delta > 0$. We've shown that a sufficient condition is

$$A^\top \Sigma^{-1} A \leq \sigma_{GM}^{-2}(\epsilon, \delta). \tag{3.33}$$

We explain the terms appearing in this constraint.

- $\Sigma = [A\ I]\text{diag}(\sigma_\eta, \sigma_\xi)^2[A\ I]^T$. is the covariance matrix of $\hat{y}$.
- $A \in \mathbb{R}^{m \times n}$ is the weight matrix.
- $\sigma_{GM}(\epsilon, \delta)$, for $\epsilon > 0$ and $\delta > 0$ is the minimum additive Gaussian noise required to make a random variable with range $[0, 1]$, (ϵ, δ)-differentially private.

Combining the aforementioned objective function to maximize utility and the constraint to specify privacy requirements, we get a constraint optimization problem, which is:

$$\begin{aligned} &\text{minimize} \\ &\quad \textstyle\sum_{i=1}^{n} \alpha_{\eta,i}\sigma_{\eta,i}^2 + \sum_{j=1}^{m} \alpha_{\xi,j}\sigma_{\xi,j}^2 \\ &\text{subject to} \\ &\quad \forall k \in \{1,2 \ldots \text{n}\}\text{: } A_{:,k}^\top \Sigma^{-1} A_{:,k} \leq \sigma_{GM}^{-2}(\epsilon_k, \delta_k). \end{aligned} \tag{3.34}$$

Here, α_η and α_ξ are vectors representing the costs induced by the two noise variables, σ_η and σ_ξ respectively, in turn, help minimize the loss expression for optimum utility.

Solving the derived constraint problem, i.e., optimizing the linear objective function while satisfying the convex constraints will provide the optimal solution for the variables σ_η^2 and σ_ξ^2, which are the noise variances. This, in turn, optimizes the utility under the convex constraints or privacy requirements. Because of the convex nature of the constraints, the optimization problem can be solved efficiently. This template is the generalized version of both local and global DP and hence achieves better utility than the two classic DP approaches.

3.3.3 Shaping differential privacy noise

In several situations, the classic additive noise mechanisms don't provide an adequate solution. Consider for example the following problem. Suppose that we have a finite domain $\mathcal{X}$ of positive numbers. Consider n parties numbered from 1 to n. Each party i has a sensitive value $x_i \in \mathcal{X}$ which it doesn't want to be revealed. At the same time, the parties collectively would like to compute

$k \geq 2$ means of their private values, including the arithmetic mean m_1 and harmonic mean m_{-1} where

$$m_p = \left(\frac{1}{n} \sum_{i=1}^{n} x_i^p \right)^{1/p}. \tag{3.35}$$

This gives us the following problem:

Variables:
- $x \in [0,1]^n$: input
- $p \in \mathbb{R}^k$: constant
- $\mathcal{O}$: output

Background knowledge:
- Loss function: $L = \sum_{i=1}^{k} \left(\mathcal{O}_i - m_{p_i}(x)\right)^2$

Privacy requirements:
- $\mathcal{O}$ is (ϵ, δ)-DP w.r.t. x.

The several parties don't trust a common curator and therefore decide they will all share noisy versions $\hat{x}_i$ of their private values x_i and perform the computation on these noisy values. This is also the setting considered by local differential privacy.

Classic additive noise mechanisms such as the Laplace mechanism and the Gaussian mechanism have the drawback that the noise distribution has an infinite domain. So, for every value x_i, especially if x_i is one of the smallest elements of $\mathcal{X}$, there is a probability that $\hat{x}_i$ is close to 0 or negative, which would make the estimation of m_{-1} very difficult.

Several solutions are conceivable. First, for every i we could approximate the p_i-mean $m_{p_i}(x)$ using a separate noisy version of x^{p_i}. Averaging over values of x^{p_i} with additive zero-mean (Laplacian or Gaussian) error would give an unbiased estimate. Still, this would imply that the k means to be computed should share the available privacy budget. We could follow an approach similar to the one in Section 3.3.2 to optimally spread the privacy budget.

Another option is to use the full privacy budget for a single noisy version $\hat{x}_i$ of x_i for every i. As we can't use classical additive noise mechanisms, we consider an arbitrary parameterized distribution and aim at estimating optimal parameters for it subject to a number of desirable properties. We should take into account that if the smallest (respectively largest) possible noisy versions of the x_i can't be much smaller (resp. larger) than the smallest (resp. largest) possible value of $\mathcal{X}$ (in our case to avoid zero or negative noisy versions which may harm the approximation of $m_{-1}(x)$), then we can't use zero-mean additive noise. A common solution is to choose for $\hat{x}_i$ with probability α an unbiased estimator of x and with probability $1-\alpha$ some background distribution B.

In particular, we consider a distribution over a domain $\mathcal{Y} \supseteq \mathcal{X}$. For $(x, y) \in \mathcal{X} \times \mathcal{Y}$, let $f_{x,y} = P\left(\hat{x}_i = y \mid x\right)$, i.e., $f_{x,y}$ is the probability, given that a private

value is x, that the noisy version is y. This naturally leads to the following quadratic program:

Minimize

$$\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} f_{x,y}(x-y)^2$$

Subject to

- $\forall x, \sum_{y \in \mathcal{Y}} f_{x,y} y = \alpha x + (1-\alpha)\mathbb{E}[B]$
- $\forall x, \sum_{y \in \mathcal{Y}} f_{x,y} y^{-1} = \alpha x^{-1} + (1-\alpha)\mathbb{E}\left[B^{-1}\right]$
- $\forall x, \sum_{y \in \mathcal{Y}} f_{x,y} = 1$
- $\mathcal{O}$ is (ϵ, δ)-DP w.r.t. x, i.e., $\forall x_1, x_2, y : f_{x_1,y} \leq e^{\epsilon} f_{x_2,y} + \frac{\delta}{|\mathcal{X}|}$

3.3.4 Combining building blocks

The examples in Sections 3.3.2 and 3.3.3 focused on isolated problems. Practical systems are often large and consist of many steps. Even if for each of these steps a privacy-preserving solution is available, these still need to be combined into a global solution.

Classic approaches to differential privacy often use combination rules, e.g., the combination of an (ϵ_1, δ_1)- differentially private step and an (ϵ_2, δ_2)- differentially private step is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$- differentially private. The disadvantage is that the privacy budget is not recycled and therefore is exhausted quickly.

The same approach can be taken using constraint programs. However, additionally, we can attempt to obtain globally better solutions. First, we can combine two constraint programs, which share variables and/or constraints. An optimal solution for the combined program may be globally more optimal than the combination of the solutions of the individual programs. Second, it becomes easier to program design choices. Often, several possible solution strategies exist, especially when considering in distributed settings the trade-off between encryption (which is more expensive in terms of computational cost) or adding noise (which decreases the utility of the output). In such situations, we can introduce both solution strategies as separate sub-programs of the larger constraint program, and introduce an additional variable π which is 0 when the first solution is used and 1 if the other solution is used. While constraint optimization typically works with real-valued variables, if the constraint programs corresponding to the two solutions don't share parameters then the design choice variable π will be either 0 or 1 in the optimum of the constraint program. In this way, in several cases, the user can focus on specifying requirements and possible solution strategies, while the optimization algorithm computes the value of the alternatives and selects the best solution.

3.4 Example scenario: distributed medical centers

In this section, we present the modeling, analysis, and synthesis of privacy requirements in a running example based on a distributed setup of multiple medical centers where each of them is computing statistics on their dataset. We will be able to see the concepts that we have built till now being applied in practice here. We will also demonstrate how to use individual constraint programs representing individual components as a building block into the framework of a combination of constraint programs, as discussed in Section 3.3.4.

3.4.1 Problem statement

Let's assume, that in our setup of multiple medical centers, the patient features, diagnosis, treatment, and prognosis are sensitive to the patient and the center. There are different privacy requirements on the patient data, as well as statistics computed on the data of individual centers (we call this, local aggregation), and statistics computed on the data of multiple centers (we call this, global aggregation). We will explain the requirements in Section 3.4.2.

Let's assume we have $r > 1$ medical centers, each with their own patients. We assume, no person is a patient in more than one hospital. Then we can give the U_i patients of center i the unique numbers, i.e., $U_{i-1} + 1, \cdots, U_i$. Total number of patients is $N = \sum_{i=1}^{r} |U_i|$.

Every patient j has q features $x_{j,k}$ with $k = 1...q$, where we set $x_{j,0} = 1$ for mathematical convenience, as the co-efficient of the bias term. So, the patient feature vector with q distinct features, represented by unique patient ID, is $(q+1)$-dimensional.

In an ideal world, with a completely trustworthy central aggregator, the patient data from all the medical centers could be centralized and accessible for computation, then the query computed on the patient data could be defined as:

$$Q = \sum_{1 \leq j \leq N} x_{j,l} x_{j,k}, \quad \forall k, l \in 0...q. \tag{3.36}$$

The first feature, $x_{j,1}$ is a special feature (which we could call for now "Kalium", the Latin name for the chemical element Potassium), which indicates the level of Potassium in a human body.

So, for the first feature, $x_{j,1}$ ("Kalium" value), our query becomes,

$$Q = \sum_{1 \leq j \leq N} x_{j,1} x_{j,k}, \quad \forall k \in 0...q. \tag{3.37}$$

We remark here that, this can be used to find indication of a disease *hyperkalemia*, which is the medical term that describes a potassium level in human blood higher than the normal range.

In real world, a distributed multi-centric setup does not facilitate us to compute statistics on a global dataset. So, individual centers will compute their

local aggregation. Once the local aggregations of the centers are computed and published, we can compute the global aggregation.

Publishing the results of a query can have severe repercussions on the various stakeholders (patients or centers), which is why privacy requirements play a major role in alleviating such negative consequences. For instance, in our running example, the result of a query revealing an aberrant average "Kalium" value, indicating *hyperkalemia* in particular, can cause a bad reputation for a medical center, as it can be caused either by features of the patient population, or it can indicate incompetence or inadequate care from the medical center.

We summarize the general notations introduced so far in Section 3.4.1 and used frequently in this example in Table 3.1.

Symbol	Meaning
r	number of medical centers, where $r \geq 1$
U_i	the set of patients in center i, where $1 \leq i \leq r$
N	total number of patients
q	number of patient features, where $q \geq 0$
Q	computed query on the patient data
j	index for patients
k	index for patient features

Table 3.1: Mathematical notations for the terms involved in our example scenario.

3.4.2 Privacy requirements

Now, to shield privacy from such adverse repercussions of revealing useful aggregation, we will specify all the privacy requirements in our scenario. Before that, we summarize the notations for different privacy models in Table 3.2

Symbol	Meaning
$\epsilon^{(pp)}, \delta^{(pp)}$	inter-center differential privacy budget
$\epsilon^{(pl)}, \delta^{(pl)}$	intra-center differential privacy budget
$\epsilon^{(u)}, \delta^{(u)}$	feature specific (here "Kalium") differential privacy budget

Table 3.2: Notations for different differential privacy-specifications in terms of (ϵ, δ) budgets, according to the privacy requirements.

- Individual patients need privacy. We assume the following properties for the privacy budget.
  - For all patients $(x_{j,:})$, with $k = 1 \dots q$ patient features, the privacy budget should be $(\epsilon^{(pp)}, \delta^{(pp)})$-DP in any center-wise published partial statistics.

  - For all patients $(x_{j,:})$, with $k = 1 \dots q$ patient features, the privacy budget should be $(\epsilon^{(pl)}, \delta^{(pl)})$-DP in any locally published (within the medical center) partial statistics seen by researchers in the hospital.
  - Therefore, $(\epsilon^{(pp)}, \delta^{(pp)}) \ll (\epsilon^{(pl)}, \delta^{(pl)})$ as hospital trusts its own researchers better.

- Center-wise partial aggregation also needs privacy. Here, we assume properties for another privacy budget.
  - For any center i, with U_i patients, and $k = 1 \dots q$ patient features, $\sum_{j \in U_i} x_{j,1} x_{j,k}$ should be $(\epsilon^{(u)}, \delta^{(u)})$-DP for any aggregation.

- For global aggregations, each
  - center calculates partial aggregations, for our generalized query defined in Eq. 3.36,

    $$Q_{i,k,l} = \sum_{j \in U_i} x_{j,k} x_{j,l}, \text{ where } k, l \in 1 \dots q,$$

    and choose between
    * costly encryption for fully private computation of global sum $Q_{k,l}$
    * adding some more noise to these partial sums and publishing them for public computation of global sum $Q_{k,l}$

3.4.3 Inferring on our example

Now, that we have described the setting of our problem and broken down the goals, what exactly we would like to achieve through the method we proposed in Section 3.3 is described in the following and illustrated in the Fig. 3.3, 3.4, 3.5, 3.6. We systematically introduce the actions taken to achieve the privacy requirements.

- Preprocess and normalize all data (scale if needed): $x_{j,k} \in [0, 1]$.
- We consider two scenarios: $v \in \{\mathrm{E}, \mathrm{C}\}$, as depicted in Fig. 3.4, 3.5,
  - $v = E$ is the **Encryption**-based design choice providing higher accuracy,
  - $v = C$ is the **Cheaper** design choice where more noise is added.
- Add noise $\eta_{v,j,k} \sim \mathcal{N}\left(0, \sigma^2_{v,i}\right)$ to patient data $x_{j,k}$ with $j \in U_i$, as depicted in Fig. 3.3.
- Compute local statistics: $Q_{v,i,k,l} = \sum_{j \in U_i} \left(x_{j,k} + \eta_{v,j,k}\right) \left(x_{j,l} + \eta_{v,j,l}\right)$, as depicted in Fig. 3.3.
- The global aggregation is computed by choosing the optimal design choice, as depicted in Fig. 3.6. Either (1) publish the $Q_{\mathrm{C},i,k,l}$ and then aggregate publicly (Fig. 3.4), or (2) compute secure private aggregation algorithm "GOPA"-sum [110] $Q_{\mathrm{E},i,k,l}$ (Fig. 3.5).

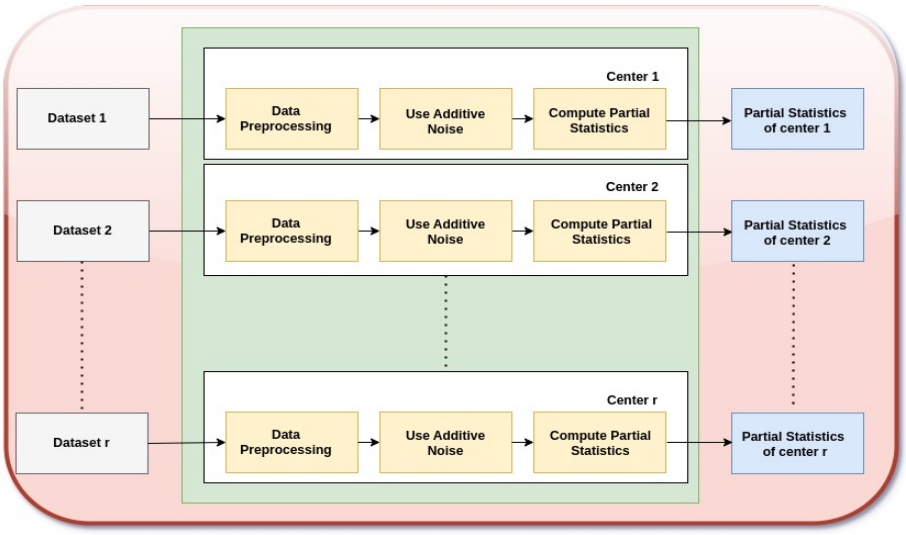


Figure 3.3: Every medical center performs data pre-processing on its noisy dataset and computes local aggregation specific to its center. This result is available to their local researchers, but they may add more noise when they intend to publish the local result to the foreign researchers of other medical centers.

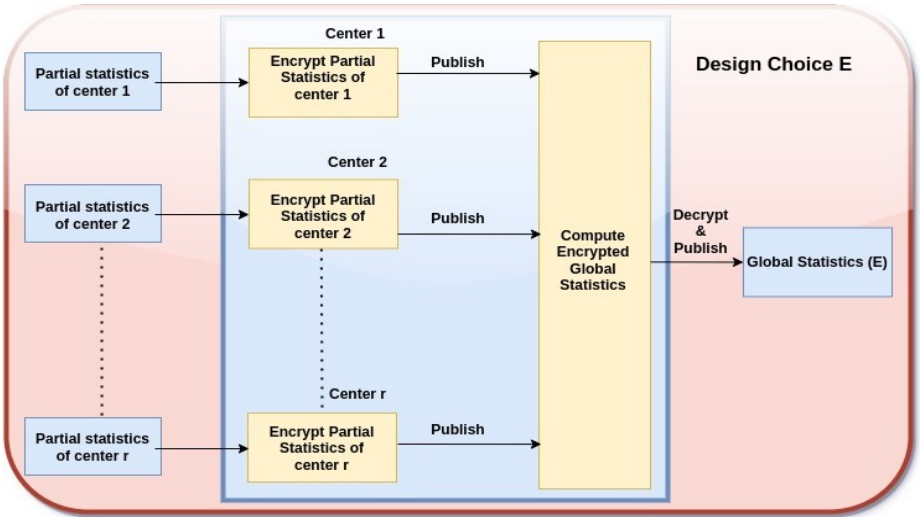


Figure 3.4: Design choice E, representing the cryptographic method of computationally expensive encryption of individual medical center's noisy partial statistics, computing global aggregation on the encrypted data, then decrypting and publishing the final result.

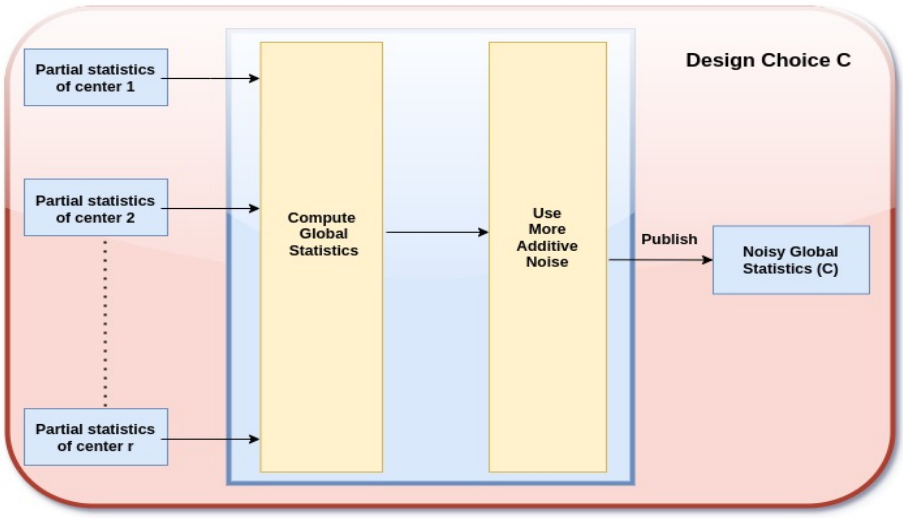


Figure 3.5: Design choice C, representing the computationally inexpensive method of computing global aggregation on the medical centers' noisy partial statistics, then adding more noise before publishing the final result.

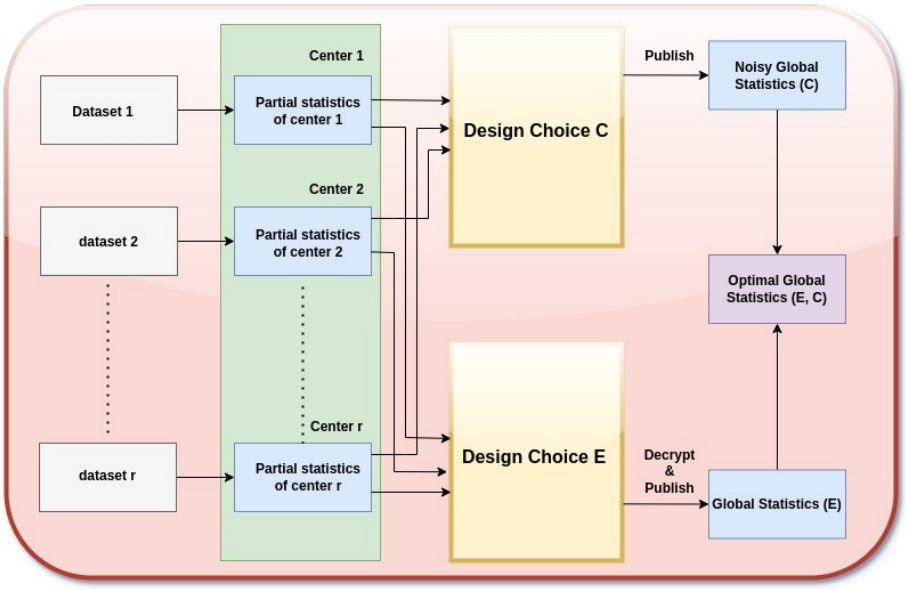


Figure 3.6: The Optimal solution will be the one that finds the approach between the two design choices that provides the minimal loss in terms of utility, privacy, and computational cost.

For illustration, we assume the two design choices, as depicted in Fig. 3.4, 3.5, but we emphasize that the reader can also make other design decisions depending on the privacy requirements and the problem settings.

Our goal would be to compute the optimal variances of these noise variables, added before and after the computation. In the process of achieving that, we would know the positions of these noise variables to be added and how the building blocks are combined while choosing the most optimal privacy-preservation strategy. We illustrated that in Fig. 3.6.

We can take the differential-privacy specifications to reason about them, derive the constraint optimization program, and find the optimal parameters using a constraint solver. The constraint optimization problem will take the following form.

Final constraint optimization problem We can infer the following constraint optimization program:

Objective function

- $\min \lambda Obj_{\mathrm{E}} + (1-\lambda) Obj_{\mathrm{C}}$
- $Obj_{\mathrm{E}} = L_{enc} + \sum_{i=1}^{r} \|U_i\| \sigma^2_{\mathrm{E},i}$
- $Obj_{\mathrm{C}} = \sum_{i=1}^{r} \|U_i\| \sigma^2_{\mathrm{C},i}$
- $0 \leq \lambda \leq 1$

Privacy Constraints

- Patient privacy:
  - In scenario E, only multi-centric aggregate statistics are published:
    $A^\top \left([A^{(1)} \dots A^{(r)} \quad I] \mathrm{diag}(\sigma_{\mathrm{E}})^2 [A^{(1)} \dots A^{(r)} \quad I]^\top \right)^{-1} A \quad \leq \sigma^{-2}_{GM}(\epsilon^{(pp)}, \delta^{(pp)})$.
  - In scenario C, statistics of every unit are published:
    for $i : 1 \dots r$: $A^\top \left([A^{(i)}\ I] \sigma^2_{\mathrm{C},i} [A^{(i)}\ I]^\top \right)^{-1} A \leq \sigma^{-2}_{GM}(\epsilon^{(pp)}, \delta^{(pp)})$.
  - In scenario $v \in \{\mathrm{E}, \mathrm{C}\}$, statistics of every unit are seen by the local researchers:
    for $i : 1 \dots r, v \in \{\mathrm{E}, \mathrm{C}\}$: $A^\top \left([A^{(i)}\ I] \sigma^2_{v,i} [A^{(i)}\ I]^\top \right)^{-1} A \leq \sigma^{-2}_{GM}(\epsilon^{(pl)}, \delta^{(pl)})$.

- Hyperkalemia confidentiality per unit:
  - In scenario C, the local hyperkalemia statistics are published: for $i : 1 \dots r$: $(A')^\top \left([A^{(i)\prime} \quad I] \sigma^2_{\mathrm{C},i} [A^{(i)\prime} \quad I]^\top \right)^{-1} A' \leq \sigma^{-2}_{GM}(\epsilon^{(u)}, \delta^{(u)})$.
  - In scenario E, the local hyperkalemia statistics are part of the eventually published overall aggregates: $(A')^\top \left([A^{(1)\prime} \dots A^{(r)\prime} \quad I] \mathrm{diag}(\sigma_{\mathrm{E}})^2 [A^{(1)\prime} \dots A^{(r)\prime} \quad I]^\top \right)^{-1} A' \leq \sigma^{-2}_{GM}(\epsilon^{(u)}, \delta^{(u)})$.

Here,

- A: feature weight matrix,
- $\sigma^{-2}_{GM}(\epsilon, \delta)$: minimal variance of additive Gaussian noise needed to achieve (ϵ, δ)-differential privacy ,
- σ^2_E, σ^2_C: variances of noise terms in design choice E & C respectively.

3.5 Discussion and conclusions

In this chapter, we argue that the explainability of privacy-preserving systems can be helped by clearly specifying the privacy guarantees satisfied by the systems, and we propose to see these privacy requirements as constraints in an optimization problem.

First, during the design and development phase, this methodology helps the developer to focus on the requirements rather than on implementation choices. In fact, the constraint optimization problem represents the space of all possible high-level implementations, and the solver accordingly finds the most interesting implementation strategy.

Second, in the deployment phase, such an explicit representation of the privacy guarantees facilitates answering user queries about exactly to what extent sensitive data is protected.

We presented a few examples showing that in several cases the translation of privacy requirements to constraint optimization problems is reasonably easy, and often yields constraint optimization problems that can be solved efficiently. Of course, this doesn't constitute a proof that such a methodology will deliver good results in all cases. An interesting line of future work is to explore more different situations and analyze whether the obtained constraint optimization problems remain tractable and scale well with the problem complexity.

Another idea for future work may be to explore whether this methodology also allows us to translate interpretable fairness requirements to efficiently

solvable constraint optimization problems. However, a number of additional challenges may arise there, e.g., there is no widespread consensus on a single good notion of fairness (as is the case with differential privacy in the privacy domain). Second, while in the current chapter on privacy, we rely on the relation between uncertainty (e.g., variance) and privacy strength, which often leads to efficiently solvable constraints, it is not immediately clear whether we could rely on a similar relation in the fairness domain.

Chapter 4

AI using declarative privacy constraints

Abstract

The increasing complexity of systems has resulted in the need for interpretable systems that protect the users' privacy, without sacrificing utility. To address this concern, we propose a novel approach that can model complex systems and specify privacy requirements for them. Furthermore, we propose a synthesizer that automatically produces solutions meeting those requirements. This synthesizer automatically chooses among privacy-preserving techniques and optimizes the parameters associated with the chosen technique. We demonstrate the application of our specification language in the context of a medical problem where useful statistics are computed on sensitive information in a cost-effective, interpretable, and privacy-preserving way.

4.1 Introduction

The advent of systems based on machine learning that can handle huge amount of personal data has led to privacy concerns. Indeed, researchers have shown that it is possible to infer private information from machine-learning models, even when one does not have access to the training dataset [10, 96, 115]. Furthermore, such systems are becoming increasingly complex, so users cannot understand the consequences of giving private information to these systems. This highlights the need for systems that achieve three goals: *privacy*, *interpretability*, and *utility*. We discuss these goals and argue why the state of the art still falls short of achieving them.

Privacy

The increasing complexity of systems can be exploited by malicious users to infer private information from individuals, sometimes with harmful consequences. Several works have demonstrated how to re-identify individuals by combining information from different systems like voter-registration records and anonymized hospital records [10] or how to infer if a person was part of a training dataset for a machine learning system by only interacting with the trained model [115].

To prevent such leakages, researchers have proposed solutions based on encryption [32] and differential privacy [43]. Unfortunately, these solutions can only deal with individual components of a data pipeline and cannot provide security guarantees for the whole pipeline. In addition, many of them can only verify privacy properties, leaving the burden of designing privacy-preserving mechanisms to the end user.

Utility

The protection brought by privacy-preserving solutions also comes at the cost of utility [135]. Adding privacy usually brings extra computation costs or less precision in the computation outcomes. For example, in differential privacy, composing different mechanisms brings a noticeable loss of outcome quality [124]. Solutions that strike the right balance between privacy and utility are needed.

Interpretability

As systems become more complex, users have a harder time understanding the consequences of giving information to these systems. To mitigate and understand the privacy risks of complex systems, solutions based on *interpretability* have been proposed. Some of them use formal systems to help all stakeholders better understand the privacy implications of interacting with a system [14, 101, 131]. Unfortunately, these solutions are still not expressive enough to capture the complex systems that today combine different solutions based on machine learning, differential privacy, and secure multi-party computation.

Contribution

From this recount, we argue that current solutions fall short in terms of privacy, utility, or interpretability; they offer limited privacy protection, sacrifice too much utility, or cannot describe current complex systems. Furthermore, many popular solutions can only verify privacy properties. To address the need for providing more privacy to users and more interpretability to complex systems, we propose a novel mechanism that introduces the following components and depicts them in Fig. 4.1.

- A novel, powerful, and holistic *specification language* for describing information processing pipelines and their privacy requirements. It can for-

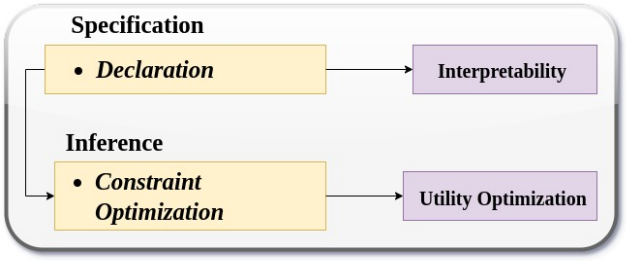


Figure 4.1: Block diagram for desired properties through specification and inference.

malize legal documents written in natural language like privacy policies, terms and conditions, and privacy legislation.

- We present an analysis framework, based on this language, that can verify and certify that the privacy constraints are being respected.
- We present a *synthesizer* that automatically produces privacy-preserving mechanisms, given a set of strategies. One only needs to specify the privacy requirements and a set of strategies (e.g., secure multi-party computation strategies or differential-privacy noise mechanisms) and the synthesizer automatically chooses the optimal strategy among them. Furthermore, it computes the optimal parameters for the chosen strategy.

To demonstrate the power of our solution, we illustrate how it can model a scenario based on a medical study. We formalize and analyze privacy requirements for patients and medical centers. Afterward, we show how to automatically construct mechanisms that compute statistics about the patients and reveal them to the researchers of medical centers, while respecting the privacy requirements. Throughout this chapter, we will alternate between concepts and their illustrations through the running example for better understanding.

4.2 Related work

The increasing awareness of privacy has spawned efforts to make ML-based systems privacy-preserving. We contribute to these efforts with the first system that directly synthesizes differentially-private ML-based systems from privacy specifications. Previous works only propose languages that specify privacy requirements for existing systems or propose verification mechanisms to ensure that an already built system is differentially private. We give next an overview of them.

4.2.1 Specification works

Several works offer languages for specifying and reasoning about privacy requirements. For example, Jeeves [131] allows us to specify how sensitive values can be disclosed. Jeeves also comes with a model that checks if the system adheres to the specified privacy policies. S4P [14] is a language that can specify privacy policies and, in addition, reason about them. S4P can then give an understanding of the implications of a specified privacy policy. SecreC [19] is another language proposed for systems dedicated to secure multi-party computation. Other works, like CCPL [73], propose languages that are more accessible and simplify the specification of privacy policies. PILOT [101] empowers users to specify their privacy preferences and then use these specifications to control what type of data websites are allowed to collect from users. P2U [76] focuses on specifying privacy policies for secondary data sharing.

These works are intended to specify and reason about privacy policies, however, they cannot synthesize differentially private mechanisms from a given specification, as we do. Our main advantage is that we do not only offer a specification language, but we also offer a procedure that automatically computes the parameters that fulfill the requirements related to the privacy mechanisms.

4.2.2 Verification works

Some works go beyond specification and offer frameworks that can verify and analyze privacy requirements. Eddy [22] is a framework for specifying privacy requirements. It can then analyze such requirements to understand information flows and to identify potential conflicts in those requirements. ZKay [117] is a language that specifies private values in blockchain contracts and ensures that they are not unintendedly leaked.

Some works are focused on differential privacy. For example, Duet [97] is a framework for specifying and verifying differentially-private mechanisms. Duet can reason about systems that use differentially private mechanisms and verify that they correctly implement their specifications. CertiPriv [11] provides the machine-checked proof of correctness for differentially-private mechanisms. Xu's work [129] offers a process calculus for reasoning about systems that implement differentially private mechanisms. In particular, it can investigate the degree of privacy under the composition of different mechanisms. DFUzz [59] uses a functional programming language with linearly indexed types to decide if queries about sensitive information in a system are differentially private. PINQ [90] is a system for processing queries that guarantees differential privacy of the outcome. It works for systems with differentially-private mechanisms and it ensures that the query outcome satisfies differential privacy. Also, privacy-preserving inference in Bayesian networks has been considered before, e.g., [133] discusses how to add noise after an inference task to make the result differentially private.

Despite the expressive power of such systems and their ability to model and reason about the privacy properties of complex systems, these solutions

still require the user to specify and design differentially private mechanisms. This process requires mathematical expertise, as the design of such mechanisms is non-trivial. The advantage of our framework is that it only requires the user to specify the variables that they wish to make differentially private, their privacy budget, and the mechanisms that should be used for differential privacy. The system can be adjusted to incorporate additional provisions for privacy-preservation strategies as well. The process of computing the right parameters for the mechanism works automatically.

4.3 Preliminaries

We will thoroughly use the concepts of differential privacy [43] (we have discussed them in detail in Section 2.7.2) in this chapter.

4.3.1 Logic preliminaries

We begin by recalling some notation from logic programming. We assume given some sets $\mathcal{V}$, $\mathcal{C}$, $\mathcal{R}$, and $\mathcal{F}$, denoting *variables*, *constants*, *relation symbols*, and *function symbols*, all of them countable. We assume that each relation symbol $r \in \mathcal{R}$ and each function symbol $f \in \mathcal{F}$ comes with an *arity* $m \in \mathbb{N}$. Sometimes, we use f/m instead of f or r/m instead of r to make the arity explicit. A *term* is any variable, constant or expression of the form $f(t_1, \ldots, t_m)$, where f is a function symbol of arity m and $t_1, \ldots, t_m$ are terms. An *atom* is any expression of the form $r(t_1, \ldots, t_m)$, where r is a relation symbol of arity m and $t_1, \ldots, t_m$ are terms. A *literal* is any expression of the form $r(t_1, \ldots, t_m)$ or $\neg r(t_1, \ldots, t_m)$, where $\neg$ is the negation operator and $r(t_1, \ldots, t_m)$ is an atom. A *clause* is an expression of the form

$$H :- B_1, B_2, \ldots, B_n, \tag{4.1}$$

where (i) H is an atom, and (ii) $B_1, B_2, \ldots, B_n$ are literals. The expression on the left-hand side of ': $-$' is called the clause's *head* and the expression on its right-hand side is called the *body*. A *(logical) predicate* is any literal. A *Logic program* is a finite set of clauses.

4.3.2 logical Bayesian network

Now, to quickly review, logical Bayesian networks (LBN)[54] have the following components: a set of *random variable declarations*, a set of *conditional dependency clauses*, and a set of *logical CPDs*, one for each probabilistic predicate.

Among the components of LBN [55], as reviewed in Section 2.8.2, we will use probabilistic predicates and logical predicates as used in LBN. Logical predicates describe logical, deterministic background knowledge, whereas probabilistic predicates have an associated range and are used to represent random variables. We can use LBN for computing conditional probabilities of our random variables. Besides, the Bayesian network seems fit for our purpose, as there is no strong need for cyclic dependencies, as in Markov networks.

4.4 Language

Before we start discussing the specifications, we introduce the problem scenario to depict a real-world situation where those specifications can be applied.

Running example: problem statement

We will consider a running example involving cancer patients [1]*, as depicted in Fig. 4.2. In our example, we want to predict the prognosis (in terms of* ***patient's expected length of survival****, and* ***if the patient survives for more than six months****) of patients. For this, we want to take into account genomic information of the cancer. In particular, cancer causes gene alterations in patients, e.g., mutations or deletions. Various drugs can be used to treat these patients but cancers can get resistant to drugs by developing more gene alterations. In that case, a new drug should be used which is still effective, if such a drug is available.*

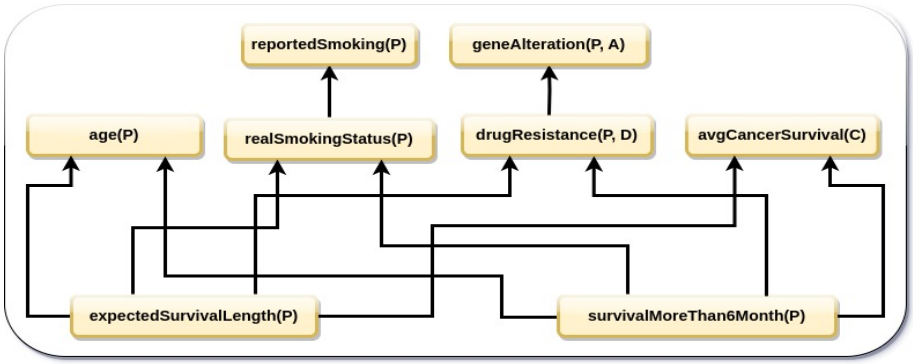


Figure 4.2: This is a block diagram illustrating the skeleton of our running example. The blocks, representing the random variables, may be directly or transitively dependent on other RVs. The direct dependency is indicated by solid arrows. The input RVs are **age(P)**, **reportedSmoking(P)**, **geneAlteration(P, A)**, **avgCancerSurvival(C)**. The **realSmokingStatus(P)**, **drugResistance(P, D)** are the intermediate RVs, which directly depend on some of the input RVs. The two output RVs are **expectedSurvivalLength(P)** and **survivalMoreThan6Month(P)**. The output RVs may directly or indirectly depend on some or all of the intermediate RVs and input RVs.

Declarations. We now present our declarative language, whose main components are represented in Fig. 4.3. We first present basic components which are

[1] This simplified example is based on a pilot study in the TRUMPET project `https://trumpetproject.eu/`

also relevant in ordinary logical Bayesian networks. Then, we will discuss the components needed to specify the privacy requirements to make the inference differentially private.

4.4.1 Random variable declaration

The `random`/1 predicate declares random variables (RV), members of the set $\mathcal{R}$ in the definition of LBN [54]. We will be using a similar syntax for declaring RVs in our scenario.

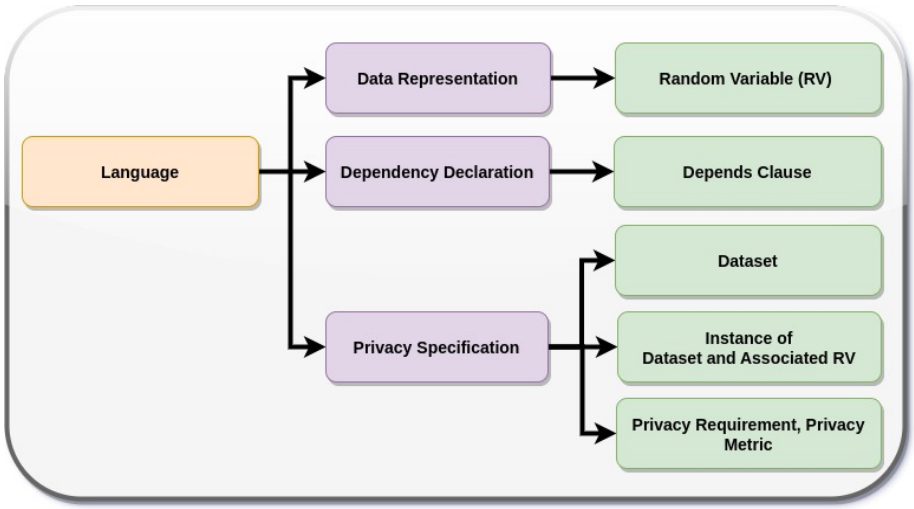


Figure 4.3: Components and sub-components of the language

Example: In our running example the random variable declarations include:

- `random(geneAlteration(P, A)) :- patient(P), alteration(A).`
  % `geneAlteration(P, A)` indicates (boolean) the existence of gene alteration `A` in patient `P`. We can read the declaration as, "if there exists patient `P`, and alteration `A`, then `geneAlteration(P, A)` is a random variable".
- `random(realSmokingStatus(P)) :- patient(P).`
  % `realSmokingStatus(P)` indicates (boolean) if patient `P` smokes.
- `random(reportedSmoking(P)) :- patient(P).`
  % `reportedSmoking(P)` indicates (boolean) what the patient `P` reported about their smoking status.
- `random(age(P)) :- patient(P).`
  % `age(P)` indicates (years) the age of patient `P`.

- `random(drugResistance(P, D)) :- patient(P), drug(D).`
  % `drugResistance(P, D)` indicates the resistance (numeric) of drug `D` that is being administered on patient `P`. We can read the declaration as, "if there exists patient `P`, and drug `D`, then `drugResistance(P, D)` is a random variable".

- `random(avgCancerSurvival(C)) :- cancer(C).`
  % `avgCancerSurvival(C)` indicates the average survival length (months) of patients with a certain type of cancer `C`. We can read the declaration as, "if `C` is type of cancer, then `avgCancerSurvival(C)` is a random variable".

- `random(expectedSurvivalLength(P)) :- patient(P).`
  % `expectedSurvivalLength(P)` indicates (months) the expected length of survival of patient `P`.

- `random(survivalMoreThan6Month(P)) :- patient(P).`
  % `survivalMoreThan6Month(P)` indicates (boolean) if patient `P` is likely to survive more than six months.

Relational database. This component is used to describe how the data is organized. We distinguish between *relations* and *random variable declarations*.

We take relational database systems for representing data as they have a suitable layout of related tables or relations. A logical Bayesian network favors relational databases as relational databases can easily be modeled into logical programs.

Data organized in relational databases can be described using relation predicates. Now, an individual cell, which is an intersection point of an attribute and a row in a relation, can be considered as a random variable.

A dataset is a set of identifiers, and then to every identifier we associate many random variables, and the random variables may be associated with some relations. We will discuss about this in detail in Section 4.4.3.

Example: In our running example, we declare a dataset `patientDataset`, and some relations like `patientData`, `geneAlterationHistory`, etc. to represent the dataset. Now, the random variables associated with the dataset may come from any of these relations.

For instance, for a patient `P`, for the random variables `age(P)`, `reportedSmoking(P)`, the functors `age`, and `reportedSmoking` are originated from relation `patientData`, whereas, for random variable `geneAlteration(P, A)`, the functor `geneAlteration` has originated from relation `geneAlterationHistory`.

Having learned the association between relations, datasets, and random variables, from now on, we will primarily use the term random variables and reason about their characteristics for the rest of the chapter.

4.4.2 Dependency declaration

In the context of logical Bayesian networks, if a random variable (RV) u depends directly on another RV v, we call u the *child random variable* and v the *parent random variable*. Moreover, when a random variable has no parent in a network and is known to the user even before an algorithm (or query) is run, it is called an *input random variable*. An *output random variable* is a RV that will be observed by a certain observer.

In a logical Bayesian network, there could be dependencies among multiple parent and child random variables, which, when stacked on top of each other, transitively create indirect dependencies between the output and input variables in the network. To understand the privacy implications of revealing any of these output variables, we must identify all random variables it depends on (directly or transitively).

In practice, to have a complete declaration of dependency, we combine the two components of LBN, *conditional dependency clauses* and *logical conditional probability distributions*, members of the set $\mathcal{D}$, and $\mathcal{C}$, respectively, in the definition of LBN [54]. In our language, the dependencies among random variables are declared using the predicate '`depends`'. We show the new declaration below.

```
depends(ChildRV, ParentRVs, CPF) :-  cond_1, ..., cond_n,
CPF = Expression.
% We can read this as "ChildRV is dependent on ParentRVs as defined by the
CPF expression if cond_1, ... , cond_n are all true".
```

In this chapter, the `depends` predicate indicates that the `ChildRV` depends on the list of random variables `ParentRVs`. The dependency only holds under the optional conditions `cond_1, ..., cond_n,` and is defined by the conditional probability function CPF. The CPF is a function, defined by the given '`Expression`', which binds the child with the parents. It can have any of the following nature.

- **Probabilistic CPF**: As the name suggests, these functions are distinguished by the incorporation of randomness. They represent the idea that, when given an identical set of input conditions, the outcome is not fixed but rather contains a range of possible outcomes, each associated with a specific probability. This probabilistic paradigm recognizes that outcomes in many real-world circumstances are not established with absolute certainty. Instead, they manifest as stochastic processes with several possible outcomes, each with its probability.
- **Deterministic CPF**: On the other hand, this type of function displays a characteristic of absolute predictability. A deterministic function produces an outcome with absolute certainty when given a specific set of input conditions or prior knowledge. In essence, the event's probability is either 0 or 1, suggesting that the conclusion is certain and unmistakable. This type of deterministic framework is frequently used in settings when

occurrences are completely predictable and lack randomness, leaving no opportunity for probabilistic uncertainty.

The clause will provide the list of random variables, `ParentRVs`, on which the `ChildRV` is dependent, based on the CPF provided.

Example: We assume a patient named `uma` with cancer type `c`. Drug `{d1, d2, d3, d4}` are being administered to her. Gene alterations `{a1, a2, a3, a4, a5, a6}` have occurred to `uma` and she developed resistance to drugs `{d1, d2}`. We assume, gene alterations `{a1, a2, a3}` caused resistance to drug `d1` and gene alterations `{a3, a4, a5, a6}` caused resistance to drug `d2` respectively. In our running example, the dependency declarations include:

- ```
  depends(realSmokingStatus(P), ParentRVs, CPF) :-
      ParentRVs = [reportedSmoking(P)],  % appends all the
            % parent RVs in a list and assign them to ParentRVs
      CPF = bernoulli(0.2 + 0.8 * reportedSmoking(P)).
   % bernoulli(L) is a bernoulli distribution with parameter L:
                % 1 with probability L, 0 with probability 1-L
  ```

  Explanation: Now, this `depends` predicate, when applied for patient `uma`, indicates the dependency of RV `realSmokingStatus(uma)` on the `ParentRVs`, where the `CPF` provides the probabilistic distribution parameterized by the expression given by the `ParentRVs`, in this case, `reported-Smoking(uma)`.

  We assign the probability of `realSmokingStatus(uma)` to 1 if the patient reports smoking. If the patient denies smoking, then the value of `realSmokingStatus(uma)` will be 1, with 0.2 probability, which means we consider the fact that the patient might be withholding actual information 20% of the time.

- ```
  weightGA_sum([V], W*V) :-
      V=geneAlteration(P, A), weightGA(A, D, W).
  weightGA_sum([V1|RV], S1+RS) :-
      weightGA_sum([V1], S1), weightGA_sum(RV, RS).
                           % weightGA(A, D, W) : W is the weight
                              % for gene alteration A and drug D

  depends(drugResistance(P, D), ParentRVs, CPF) :-
      findall(geneAlteration(P, A), random(geneAlteration(P, A)),
                                        ParentRVs),
                % puts all the parent RVs in the list ParentRVs
      weightGA_sum(ParentRVs, TotalWeight),
      CPF = exp(-TotalWeight).
                        % exp(-X) is an exponential distribution
  ```

  Explanation: Now, this `depends` predicate, when applied for patient `uma`, indicates the dependency of RV `drugResistance(uma, d1)` on the

ParentRVs, where the CPF provides the probabilistic distribution parameterized by the expression given by the ParentRVs, in this case, geneAlteration(uma, a1), geneAlteration(uma, a2), geneAlteration(uma, a3). Similarly, it can indicate the dependency of RV drugResistance(uma, d2), by the combinations of RVs in ParentRVs, in that case, geneAlteration(uma, a3), geneAlteration(uma, a4), geneAlteration(uma, a5), geneAlteration(uma, a6).

- ```
  depends(expectedSurvivalLength(P), ParentRVs, CPF) :-
      patientHasCancer(P, C),
                          % every patient only has one cancer
      findall(drugResistance(P,D), random(drugResistance(P,D)),
                                      DRList),
      append([avgCancerSurvival(C), realSmokingStatus(P),
               age(P)], DRList, ParentRVs),
                            % appends all the parent RVs in a
                              % list and assigns to ParentRVs
      CPF = poisson(1/(realSmokingStatus(P) * 0.5+
          max(DRList) + avgCancerSurvival(C) * 0.02 +
          age(uma) * 0.01)).          % poisson(L) is a poisson
                            % distribution with λ parameter L
  ```

  Explanation: Now, this depends predicate, when applied for patient uma, indicates the dependency of RV expectedSurvivalLength(uma) on the ParentRVs, where the CPF provides the probabilistic distribution parameterized by the expression given by the ParentRVs, in this case, [avgCancerSurvival(c), drugResistance(uma, d1), drugResistance(uma, d2), realSmokingStatus(uma), age(uma)].

  So, if the λ value for patient uma turns out to be ≈ 8.1, then the probability for expectedSurvivalLength(uma)≥ 12 can be computed by:
  poisson(expectedSurvivalLength(uma)≥ 12, $\lambda = 8.1$) ≈ 0.11928.

Of course, these depends predicates are only examples to show how the dependencies between RVs can be represented in our language. In reality, for example, the survival of a patient depends on a lot of other (important) factors like type and amount of comorbidities of the patient, various kinds and counts of gene alterations, gender, medical history, drug side effects, etc. We leave the decision of determining the actual dependency relationship between RVs, to the domain experts dealing with the actual use case.

4.4.3 Privacy specification

We allow for defining multiple *privacy requirements*. This enables us to construct scenarios in which it is beneficial to reveal information to numerous viewers at times, and certain observers are permitted to obtain more detailed information than others. For example, a medical center could give doctors full access to all medical patient data, while only privacy-preserving aggregated information
  ```

is shared with the public. Now, all privacy specifications connected to a single privacy requirement should be satisfied with a single privacy budget. Hence, the privacy budget must be divided into multiple portions if more random variables are involved in a single privacy requirement.

1. `dataset(DatasetName, Domain).` declares the dataset `DatasetName` and specifies that its domain is `Domain`, i.e., if we consider `D` to be an instance of `DatasetName`, then `D` $\subseteq$ `Domain`.

   Example: `dataset(patientDataset, domain(P, patient(P))).`
   Explanation: In our running example, the `patientDataset` represents the `dataset` of all patients and the corresponding `Domain` is the set containing a complete representation of all possible patients. The `Domain` is represented by a term `domain(V, Goal)` where it can be tested whether an element belongs to the `domain` by unifying `V` with the element and calling the `Goal`, which is the condition, that will succeed depending on whether `V` is in the domain. For this example, `P` and `patient(P)` correspond to `V` and `Goal` respectively.

2. `dataset_instance(DatasetName, Instance).` declares that `Instance` is an instance of the dataset `DatasetName`.

   Example: `dataset_instance(patientDataset,`
   `patient_instance(P)) :- patient(P).`
   Explanation: `patient_instance(P)` is an instance of dataset `patientDataset` for patient `P`.

3. `instance_rv(DatasetName, Instance, RV).` declares that `RV` is a random variable belonging to the instance `Instance` of the dataset `DatasetName`.

   Example: `instance_rv(patientDataset, patient_instance(P),`
   `age(P))`
   Explanation: In dataset `patientDataset`, `age(P)` is a random variable belonging to instance `patient_instance(P)`, for patient `P`.

4. `privacy_requirement(PrivReqName, Dataset, PrivMetric).` is a declaration specifying that `PrivReqName` is the name of a privacy requirement and `PrivMetric` is a term specifying the used statistical privacy metric and budget. Examples of such terms include $dp(\epsilon)$ for ϵ-differential privacy, $dp(\epsilon, \delta)$ for (ϵ, δ)-differential privacy or $renyi\_dp(\alpha, \epsilon)$ for (α, ϵ)-Rényi differential privacy [92].

   The declarations of `PrivReqName`, can be generalized over all observers. In case we have different access specified for different observers, we can use `Observer` as a parameter to the `PrivReqName`.

Example: `privacy_requirement(patientPrivacy, patientDataset, dp(0.1, 0.01)).`
Explanation: A privacy requirement named `patientPrivacy`, which specifies the privacy to observe dataset `patientDataset` with privacy metric (ϵ, δ)-dp, where, $\epsilon = 0.1$ and $\delta = 0.01$. We can specify different privacy requirements for different observers,

```
privacy_requirement(patientPrivacy(doctor),
                    patientDataset, dp(0.25, 0.03)).
privacy_requirement(patientPrivacy(researcher),
                    patientDataset, dp(0.2, 0.02)).
privacy_requirement(patientPrivacy(public),
                    patientDataset, dp(0.1, 0.01)).
```

We specify privacy requirement for three different observers, `doctor`, `researcher`, and `public`, such that `doctor` observes the `patientDataset` with highest precision and the `public` observes it with least precision.

5. `dp_rv(PrivReqName, RV).` declares the relevant observations which should be privatized: all variables `RV` such that `dp_rv(PrivReqName, RV)` is true, should together satisfy the privacy requirement.

   Example: `dp_rv(patientPrivacy(doctor), age(P)) :- patient(P).`
   Explanation: The `dp_rv` predicate declares the random variable `age(P)` associated with privacy requirement `patientPrivacy(doctor)`. According to our definition of predicates `privacy_requirement` and `dp_rv`, the random variable `age(P)` will share the `PrivMetric` defined by `dp(0.25, 0.03)` with other random variables associated with the same privacy requirement `patientPrivacy(doctor)`.

Additionally, some random variables are *personal* in the sense that they are related to an identifiable natural person. Many privacy regulations, like the GDPR [2], have special requirements for such personal information, e.g., a person can request the deletion of all their personal information. If some `RV` is personal to a party, then implicitly the party has access to observe the `RV` with the highest precision. For instance, in our running example, all the random variables associated with a patient instance, identified by `patient(P)`, in the dataset `patientDataset`, are personal to patient `P`.

Example: `personal_rv(P, age(P)) :- patient(P).`
Explanation: Random variable `age(P)` is personal to patient `P`.

4.4.4 Summarization of declarations

In this section, we present a summary of the various components presented in our specification language and how they might be organized in the context of a logical Bayesian network. Here, we will briefly summarize the components of the declarative language introduced so far.

1. ***Random variable declaration*** We declare all the random variables (RV) for a query, using the definitions described in Section 4.4.1.

2. ***Dependency declaration*** In Section 4.4.2, we specify which input RVs (direct or indirect child RVs) are used to compute the value of the observed output RVs, using one or more `depends` predicate. The binding between all the ancestor RVs and observed RV is decided based on the conditional probability function `CPF`(s).

3. ***Privacy specification*** As described in Section 4.4.3, we specify privacy requirements using the five principal privacy specification constructs.

These specifications will be used in the inference in Section 4.5.

4.5 Inference

The goal of this section is to discuss our strategy to infer the appropriate privacy mechanisms and derive the optimal privacy parameters, which in turn will be used to solve inference tasks. Here, we address which sensitive information is revealed in the form of random variables if a statistic computed on them, is released. Moreover, it also inspects the precise spots in the LBN where the privacy techniques (in this case adding noise) should be positioned to achieve the best results.

We organize this section as follows. Section 4.5.1, outlines the technique we employ to construct a constraint problem to achieve the optimal solution and discuss how to select amongst various noise mechanisms, given a set of privacy specifications. In Section 4.5.2, 4.5.3, 4.5.4, we delve deeper into the design of the constraint problem in a generalized scenario. In Section 4.5.5, we demonstrate how to apply the inference concepts discussed so far, in the context of our running Example 4.4.

4.5.1 Design overview of the constraint problem

Now, we give an overview of our design process for the constrained optimization problem, given the specification of the problem and their privacy requirements, defined in Section 4.4.

Constraint problem structure

We primarily attempt to build an objective function and a set of constraints. In our approach, the objective function is a loss function that must be optimized, and the constraints are generated from the privacy requirements. Section 4.5.2 contains more information about this. Once such a constraint problem is generated, following that, a classical constraint optimization solver can compute the optimal solution for the constraint problem.

Forward computation

Throughout this chapter, for simplicity of explanation, we will assume forward computation, i.e., we want to infer a random variable given priors and evidence about some of its ancestors. For example, we see forward inference as inferring one by one the random variables, starting from the known input nodes until the queried nodes, where we can add noise to guarantee privacy but can also maximize utility without being too fixated on either of the traditional differential privacy approaches [48].

Trade-offs between differential privacy approaches

From [42] and [48], we get the basic definitions of two classical Differential Privacy approaches. Local differential privacy (LDP), adds noise to the input, and then computations happen over the noisy data. This often requires a larger privacy budget, resulting in high loss and low utility. On the other hand, central differential privacy (CDP) first performs computations on original inputs and then adds noise to get noisy results. This works well when a few queries are run on a large number of sensitive variables (refer to Fig. 4.4). So, LDP is more expensive, than CDP, in terms of noise for the same privacy budget. For a fixed privacy level, and fewer outputs, CDP has better utility, but if the number of outputs increases, the utility of LDP will outperform CDP. Hence, we see that the Local and Central DP approaches have complementary benefits. Combining the advantages of both can lead to a more optimal noise budget, thus maximizing the utility of the algorithm.

Combining forward computation with generalized DP approach

We can generalize this idea by looking at forward inference (for simplicity of deriving the constraint problem, but not restricted only to it) in a probabilistic model, as the computation of one or more outputs from a set of inputs through a circuit of optional internal nodes. In our case, we chose to use a logical Bayesian network, and depending on the graph topology and the conditional probability functions, adding noise at inputs, outputs, and intermediate nodes or a combination of these may be optimal. So our solution approaches the middle road between CDP and LDP and adds noise carefully at places that perform best in optimizing privacy and utility. We will take this approach in generating the constraint problem from the specifications in the following section.

4.5.2 Transformation: specifications to constraints

In this section, we sketch on a more granular level, how we can transform the privacy specifications into a constraint problem.

Let's assume a logical Bayesian network, where $\mathcal{RV} = \{x_i : i \in [k]\}$ is the set of all random variables in the network, and every partial ordering (or node) in the network is defined by a random variable x_i. Each RV x_i is dependent on its direct ancestor RVs, known as its parent RVs, and indexed by the set

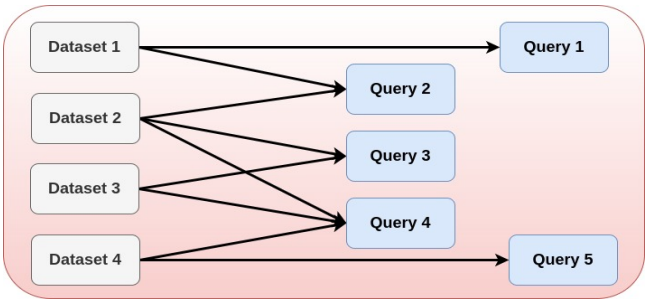


Figure 4.4: Queries and datasets can have one-to-many, many-to-one, or many-to-many relationship. So, in a study, a query can run on one or more datasets, and one dataset can also participate in multiple studies.

$Pa(i) = \{j : (x_j | x_i)\}$. We call x_i an input variable if $Pa(i) = \emptyset$. We call x_i an output variable if we intend to reveal its value to an observer.

Each x_i is defined by a function f_i on the combination of its parent random variable, x_j. So, for input RVs, $f_i \in \mathbb{R}$ is a constant. The nature of f_i can be of any operations that are permitted on the x_i. We summarize all of the assumptions and notations frequently used in this section in Table 4.1.

Symbol	Meaning
$[k]$	$\{1, 2, \cdots, k\}$
$\mid$	Symbol indicating dependency relation, and induces a partial order on the random variables
$\mathbb{1}_i$	Unit vector of shape (1 x k) with a 1 on position i and 0 elsewhere
x_i	Random variables (RV) for $1 \leq i \leq k$, where $x_i \in \mathbb{R}$
f_i	Functions applied on the combinations of the parent RVs of x_i
$\mathcal{RV} = \{x_i : i \in [k]\}$	Set of all random variables
$Pa(i)$	Set of indices of all the parents of a node x_i
$\eta \in \mathbb{R}^k$	Noise, which is a normally distributed random vector
D	Dataset, if a particular patient $P \in Dataset$
D'	Dataset, if a particular patient $P \notin Dataset$
O	Observation

Table 4.1: General mathematical notations used for the transformation from the privacy specification to a generalized constraint problem.

Our approach incorporates privacy criteria that outlines what can be inferred about any sensitive variables that are either directly or indirectly connected to the output variable, from observing the output. The output variable is not necessarily sensitive; it could be computed from variables, some or all of which may be sensitive. If an output variable is sensitive in nature, we will also need

a privacy requirement for it.

4.5.2.1 Privatization of observed variables

Based on our previous assumptions about inference, we can write,

$$x_i = f_i(x_{Pa(i)}), \quad \text{for } i \in [k], \tag{4.2}$$

as $Pa(i) = \{j : (x_j|x_i)\}$ is the set of all indices of direct parents of x_i.

To avoid revealing the sensitive value of any x_i, one must not publish their original values or the original values of output variables that are computed (directly or indirectly) from one or more such x_i. Therefore, noise is added individually to each x_i, so that, when we publish the output variables to the observer, it accumulates the noise added to all the noisy variables that are used to compute the output. Hence, publishing the output variables is privacy-preserving.

In our framework, the noise is a random vector $\eta \in \mathbb{R}^k$, and it's defined as

$$\eta_i \sim \mathcal{N}(0, \sigma_{\eta,i}), \quad \text{for } i \in [k]. \tag{4.3}$$

To protect the privacy of the x_i values, we add noise locally to all of them, including the input RVs and intermediate RVs (partial orderings [54]). These noisy $\hat{x}_i$ enables us to attain differential privacy, which we can define as,

$$\hat{x}_i = f_i(\hat{x}_{Pa(i)}) + \eta_i, \quad \text{for } i \in [k]. \tag{4.4}$$

If the noise components η_i ($i \in [k]$) are independent of each other, then we can consider the variance of a variable $\hat{x}_i$ w.r.t η_i as

$$\begin{aligned} \text{var}_{\eta_i}(\hat{x}_i) &= \text{var}_{\eta_i}(f_i(\hat{x}_{Pa(i)})) + \text{var}_{\eta_i}(\eta_i), & (4.5)\\ &= \sigma^2_{\eta,i}\ . \qquad (\text{since, } \text{var}_{\eta_i}(f_i(\hat{x}_{Pa(i)})) = 0) & (4.6) \end{aligned}$$

If we consider Eq. (4.5), we assume that $\Sigma_\eta(\hat{x}_j) = \text{var}_\eta(\hat{x}_j)$ is known for all $j \in Pa(i)$. Then, assuming f_i is linear in all its arguments (locally but in a sufficiently large sphere around $x_{Pa(i)}$) will give us,

$$f_i(\hat{x}_{Pa(i)}) \approx f(x_{Pa(i)}) + (\hat{x}_{Pa(i)} - x_{Pa(i)})^\top . \nabla_{x_{Pa(i)}} f_i(x_{Pa(i)}). \tag{4.7}$$

Now, we can substitute the value of $\hat{x}_i$ from Eq. (4.4),

$$\Sigma_\eta(\hat{x}_i) = \Sigma_\eta(f_i(\hat{x}_{Pa(i)})) + \Sigma_\eta(\eta_i).$$

Using, first order Taylor series approximation of f_i around $x_{Pa(i)}$, we can re-write as,

$$\begin{aligned} \Sigma_\eta(\hat{x}_i) &= \Sigma_\eta(f_i(x_{Pa(i)}) + (\hat{x}_{Pa(i)} - x_{Pa(i)})^\top \nabla_{x_{Pa(i)}} f_i(x_{Pa(i)})) + \Sigma_\eta(\eta_i),\\ &= \Sigma_\eta(f_i(x_{Pa(i)})) + \Sigma_\eta((\hat{x}_{Pa(i)} - x_{Pa(i)})^\top \nabla_{x_{Pa(i)}} f_i(x_{Pa(i)}))\\ &\quad + \Sigma_\eta(\eta_i). \end{aligned} \tag{4.8}$$

We know that, $\Sigma_\eta(f_i(x_{Pa(i)})) = 0$. Therefore, we can write,

$$\begin{aligned}\Sigma_\eta(\hat{x}_i) &= \sum_{j \in Pa(i)} \left(\frac{\partial f}{\partial x_j} x_{Pa(i)}\right)^2 \Sigma_\eta(f_i(\hat{x}_j)) + \Sigma_\eta(\eta_i) \\ &= \sum_{j \in Pa(i)} \left(\frac{\partial f}{\partial x_j} x_{Pa(i)}\right)^2 \Sigma_\eta(f_i(\hat{x}_j)) + \mathbb{1}_i \mathbb{1}_i^\top \sigma_{\eta,i}^2. \end{aligned} \tag{4.9}$$

As we have already assumed that, $\Sigma_\eta(\hat{x}_j) = \text{var}_\eta(\hat{x}_j)$ is known for all $j \in Pa(i)$, then from Eq. (4.9), we get an iterative formula of the form,

$$\Sigma_\eta(\hat{x}_i) = \sum_{j \in Pa(i)} \beta_{i,j} + \mathbb{1}_i \mathbb{1}_i^\top \sigma_{\eta,i}^2 \ . \tag{4.10}$$

For input variables, f_i being a constant, $\beta_{i,j}$ would be 0, and we would simply have $\Sigma_\eta(\hat{x}_i) = \mathbb{1}_i \mathbb{1}_i^\top \sigma_{\eta,i}^2$.

In Eq. (4.9), hypothetically, if f_i are purely linear and we know the $\Sigma_\eta(x_j)$, where j are parents of i, then we can iteratively compute the noise contributed by Eq. (4.9) directly. The noise for the current node x_i, plus the noise added at the parents, which is the direct covariance matrix on the x_j, multiplied by the gradients of that function, summed over all parents.

This way, we can start computing with the input variables, where the first term is zero (as the gradient is zero), and then step by step, go through all the dependencies in the graph and compute for all variables the variances subjected to the noise.

Now, of course, in practice, f_i is not always linear. Since the derivation up to this point, is based on our first approximation (i.e. f_i is linear in all its arguments (locally but in a sufficiently large sphere around $x_{Pa(i)}$)), it is good if only a small amount of noise is needed. Otherwise, if we consider a very small epsilon value which means a lot of noise needs to be added, then for most of the non-linear dependencies f_i, variance approximation for all the variables, will be way off.

4.5.2.2 Privacy constraints from DP guarantees

Let's consider, the variable, $\hat{x}_i$, that's revealed to some observer. The main criterion is that $\hat{x}_i$ must not expose too much about the sensitive variable x_i itself and other sensitive, ancestor random variables in the network.

Using the definition of classical differential privacy [43], we can write, for any observation O, patient P, adjacent datasets D and D', the following condition needs to be true to achieve (ϵ, δ)-differential privacy.

$$\Pr(O \mid P \in D) \leq e^\epsilon \Pr(O \mid P \notin D) + \delta. \tag{4.11}$$

If we consider D', an adjacent dataset to dataset D, where there is one patient P for which $P \notin D \equiv P \in D'$, then we can simplify this inequality as,

$$\Pr(O \mid D) \leq e^{\epsilon} \Pr(O \mid D') + \delta. \tag{4.12}$$

In this scenario, O is a noisy random variable that is observed by some observer. Each of the noisy $\hat{x}_i$ is differentially private with corresponding $\epsilon_i > 0, \delta_i > 0$.

We consider that we have a conditional probability function that yields a probability distribution for the observed random variable (RV) dependent on its respective parent RVs. The joint probability decomposition property of basic Bayesian networks allows us to describe the joint probability of the parents,

$$\Pr(x_{Pa(i)}) = \Pr(\cap_{j \in Pa(i)} x_j), \tag{4.13}$$

as the product of conditional probabilities of the individual parent RVs of the observed RV,

$$\Pr(x_{Pa(i)}) = \prod_{j \in Pa(i)} \Pr(x_j \mid x_{Pa(j)}), \tag{4.14}$$

where $x_{Pa(i)} = \{x_j, \forall j \in Pa(i)\}$ and $x_{Pa(j)} = \{x_l, \forall l \in Pa(j)\}$.

Therefore, the following inequality holds,

$$\Pr(x_{Pa(i)}, D \mid \hat{x}_i) \leq e^{\epsilon} \Pr(x_{Pa(i)}, D' \mid \hat{x}_i) + \delta. \tag{4.15}$$

4.5.3 Defining the constraint problem

To begin, we represent our random variables and the relationships among them in an LBN-like structure, where each RV is a partial ordering (or node of the LBN graph). We start with the sensitive RVs, progress through optional intermediate RVs, and reach the observed RV(s), as shown in Fig. 4.5. We also define notations for each of the network components in terms of (a) input or sensitive variables (S), (b) intermediate variables (Z), and (c) output or observed variables (O) in Table 4.2. The intermediate variables are all (strict or otherwise) ancestors of observed variables, and they are also (strict or otherwise) descendants of sensitive variables.

Now, we recall that in Section 4.5.2.2, we assumed that the sensitive variables S, as defined in Table 4.2, are functions of the dataset (in our case, D or D'). We denote by $S(D)$ the vector of values of the sensitive variables S given dataset D. Conditioning a probability on D implies conditioning the probability (also) on $S(D)$. Considering the symbols defined in Table 4.2, we can write,

$$\Pr(\hat{x}_{ZO} \mid D) = \prod_{i \in V_{ZO}} \Pr(\hat{x}_i \mid x_{Pa(i)}, D). \tag{4.16}$$

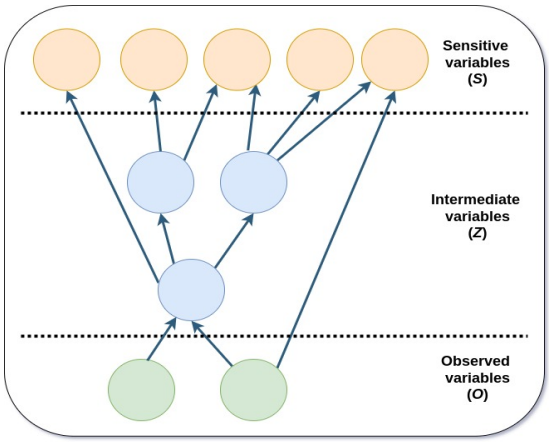


Figure 4.5: The different zones in a logical Bayesian network, the set of sensitive variables (S), the set of intermediate variables (Z), and the set of observed variables (O).

Symbol	Meaning
V_S	the set of indices of sensitive variables
V_O	the set of indices of observed variables
V_Z	the set of indices of the intermediate variable
V_A	$V_S \cup V_Z \cup V_O$
V_{ZO}	$V_Z \cup V_O$
x_O	the vector of all variables represented by the indices in V_O, i.e., $x_O = (x_i)_{i \in V_O}$
x_Z	the vector of all variables represented by the indices in V_Z, i.e., $x_Z = (x_i)_{i \in V_Z}$
x_{ZO}	the vector of all variables represented by the indices in V_{ZO}, i.e., $x_{ZO} = (x_i)_{i \in V_{ZO}}$.

Table 4.2: The notations used to explain different components of the logical Bayesian network.

We proceed to marginalize the probabilities:

$$\begin{aligned} \Pr(\hat{x}_O = \psi_O \mid D) &= \sum_{\psi_Z \in \mathrm{dom}(\hat{x}_Z)} \Pr(\hat{x}_O = \psi_O, \hat{x}_Z = \psi_Z \mid D) \\ &= \sum_{\psi_Z \in \mathrm{dom}(\hat{x}_Z)} \prod_{i \in V_{ZO}} \Pr(\hat{x}_i = \psi_i \mid \hat{x}_{Pa(i)} = \psi_{Pa(i)}, D) \end{aligned} \tag{4.17}$$

Here:

- $\Pr(\hat{x}_O = \psi_O \mid D)$: This is the conditional probability of the random

variable $\hat{x}_O$ taking on a specific value ψ_O given the dataset D.

- $\sum_{\psi_Z \in \text{dom}(\hat{x}_Z)}$: This represents the summation over all possible values ψ_Z in the domain of the random variables in set $\hat{x}_Z$.
- $\Pr(\hat{x}_O = \psi_O, \hat{x}_Z = \psi_Z \mid D)$: This is the joint probability of both $\hat{x}_O$ taking on the value ψ_O and $\hat{x}_Z$ taking on the value ψ_Z given the dataset D. It's the product of their conditional probabilities.

Expanding the $\Pr(\hat{x}_O = \psi_O, \hat{x}_Z = \psi_Z \mid D)$, we get

- $\prod_{i \in V_{ZO}}$: This signifies a product over all random variables $\hat{x}_i$ in the set V_{ZO}.
- $\Pr(\hat{x}_i = \psi_i \mid \hat{x}_{Pa(i)} = \psi_{Pa(i)}, D)$: This is the conditional probability of the random variable $\hat{x}_i$ taking on the value ψ_i given its parents (denoted by $\hat{x}_{Pa(i)}$) taking on the values $\psi_{Pa(i)}$ and the dataset D.

So, the equation is essentially summing over all possible combinations of values for $\hat{x}_Z$ and calculating the joint probability of $\hat{x}_O$ and $\hat{x}_Z$ by multiplying the conditional probabilities of each variable in V_{ZO}.

Classical Differential privacy condition [43] mandates that,

$$\forall \psi_O \in \text{dom}\,(\hat{x}_O) : \Pr(\hat{x}_O = \psi_O \mid D) \leq e^{\epsilon} \Pr(\hat{x}_O = \psi_O \mid D') + \delta. \tag{4.18}$$

We can rewrite this as,

$$\begin{aligned} &\forall \psi_O \in \text{dom}\,(\hat{x}_O) : \\ &\sum_{\psi_Z \in \text{dom}(\hat{x}_Z)} \prod_{i \in V_{ZO}} \Pr(\hat{x}_i = \psi_i \mid \hat{x}_{Pa(i)} = \psi_{Pa(i)}, D) \\ \leq \quad & e^{\epsilon} \sum_{\psi_Z \in \text{dom}(\hat{x}_Z)} \prod_{i \in V_{ZO}} \Pr(\hat{x}_i = \psi_i \mid \hat{x}_{Pa(i)} = \psi_{Pa(i)}, D') + \delta \end{aligned} \tag{4.19}$$

To prove $\sum_x \prod_y p_{x,y} \leq \sum_x \prod_y p'_{x,y}$ is true, it is sufficient to prove that the following holds,

$$\forall x, \prod_y p_{x,y} \leq \prod_y p'_{x,y}. \tag{4.20}$$

Therefore, from Eq. (4.19), it is adequate to demonstrate, $\forall \psi_z \in \text{dom}(\hat{x}_Z)$,

$$\frac{\prod_{i \in V_{ZO}} \Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)}{\prod_{i \in V_{ZO}} \Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D')} \leq e^{\epsilon}, \tag{4.21}$$

or,

$$\left| \log \left(\frac{\prod_{i \in V_{ZO}} \Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)}{\prod_{i \in V_{ZO}} \Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D')} \right) \right| \leq \epsilon \tag{4.22}$$

holds with probability at least $(1 - \delta)$ (over the noise).

From the expression on the left side of Eq. (4.22), we can write:

$$\begin{aligned}
&\left|\log\left(\frac{\prod_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)}{\prod_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D')}\right)\right| \\
&= \left|\log(\Pi_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)) - \log(\Pi_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D'))\right| \\
&= \left|\Sigma_{i\in V_{ZO}}\log\left[\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)\right] - \Sigma_{i\in V_{ZO}}\log\left[\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D')\right]\right| \\
&= \left|\Sigma_{i\in V_{ZO}}\left(\log\left(\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)\right) - \log\left(\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D')\right)\right)\right| \\
&= \left|\Sigma_{i\in V_{ZO}}\log\left(\frac{\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)}{\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D')}\right)\right| \\
&= \Sigma_{i\in V_{ZO}}\left|\log\left(\frac{\Pr(\hat{x_i} \mid \hat{x}_{Pa(i)}, D)}{\Pr(\hat{x_i} \mid \hat{x}_{Pa(i)}, D')}\right)\right| .
\end{aligned}$$

Considering, the values of $\hat{x}_i$ follow a Gaussian distribution, we write,

$$\left|\log\left(\frac{\prod_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)}{\prod_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D')}\right)\right| \tag{4.23}$$

$$=\Sigma_{i\in V_{ZO}}\left|\log\left(\frac{\frac{1}{\sigma_{\eta,i}(\hat{x}_i).\sqrt{2\pi}}\exp(-\frac{1}{2}\left(\frac{\hat{x_i}-x_i}{\sigma_{\eta,i}(\hat{x}_i)}\right)^2)}{\frac{1}{\sigma_{\eta,i}(\hat{x}_i).\sqrt{2\pi}}\exp(-\frac{1}{2}\left(\frac{\hat{x_i}-x'_i}{\sigma_{\eta,i}(\hat{x}_i)}\right)^2)}\right)\right| \tag{4.24}$$

$$=\Sigma_{i\in V_{ZO}}\left|\log\left(\frac{\exp(-\frac{1}{2}\left(\frac{\hat{x_i}-x_i}{\sigma_{\eta,i}(\hat{x}_i)}\right)^2)}{\exp(-\frac{1}{2}\left(\frac{\hat{x_i}-x'_i}{\sigma_{\eta,i}(\hat{x}_i)}\right)^2)}\right)\right| \tag{4.25}$$

$$=\Sigma_{i\in V_{ZO}}\left|\log\left(\exp\left(-\frac{1}{2}\left(\frac{\hat{x_i}-x_i}{\sigma_{\eta,i}(\hat{x}_i)}\right)^2+\frac{1}{2}\left(\frac{\hat{x_i}-x'_i}{\sigma_{\eta,i}(\hat{x}_i)}\right)^2\right)\right)\right| \tag{4.26}$$

$$=\Sigma_{i\in V_{ZO}}\left|\frac{1}{2}\left(\frac{\hat{x_i}-x'_i}{\sigma_{\eta,i}(\hat{x}_i)}\right)^2-\frac{1}{2}\left(\frac{\hat{x_i}-x_i}{\sigma_{\eta,i}(\hat{x}_i)}\right)^2\right| \tag{4.27}$$

$$=\Sigma_{i\in V_{ZO}}\left|\frac{1}{2}\left(\frac{{\hat{x_i}}^2+x_i'^2-2\hat{x_i}x'_i-{\hat{x_i}}^2-x_i^2+2\hat{x_i}x_i}{\sigma^2_{\eta,i}(\hat{x}_i)}\right)\right| . \tag{4.28}$$

From Eq. (4.5), we write:

$$\left|\log\left(\frac{\prod_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)}{\prod_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D')}\right)\right| = \Sigma_{i\in V_{ZO}}\left|\frac{1}{2}\left(\frac{\left(x_i'^2-x_i^2\right)+2\hat{x_i}(x_i-x'_i)}{\mathrm{var}_{\eta_i}(\hat{x}_i)}\right)\right| . \tag{4.29}$$

Assuming the $\hat{x}_i$ has come from the noisy parents of x_i random variable after

applying transformation function f_i and adding more noise η_i, we can write,

$$\left|\log\left(\frac{\prod_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)}{\prod_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D')}\right)\right| \tag{4.30}$$

$$= \sum_{i\in V_{ZO}}\left|\frac{1}{2}\left(\frac{(x_i'^2 - x_i^2) + 2\hat{x}_i(x_i - x_i')}{\mathrm{var}_{\eta_i}(\hat{x}_i)}\right)\right| \tag{4.31}$$

$$= \sum_{i\in V_{ZO}}\frac{1}{2}\left|\frac{(x_i' + x_i)(x_i' - x_i) + 2\hat{x}_i(x_i - x_i')}{\mathrm{var}_{\eta_i}(\hat{x}_i)}\right| \tag{4.32}$$

$$= \sum_{i\in V_{ZO}}\frac{1}{2}\left|(x_i' + x_i - 2\hat{x}_i)\left(\mathrm{var}_{\eta_i}(\hat{x}_i)\right)^{-1}(x_i' - x_i)\right| \tag{4.33}$$

$$= \sum_{i\in V_{ZO}}\frac{1}{2}\left|(x_i' - x_i + 2(x_i - \hat{x}_i))\left(\mathrm{var}_{\eta_i}(\hat{x}_i)\right)^{-1}(x_i' - x_i)\right| \tag{4.34}$$

$$= \sum_{i\in V_{ZO}}\frac{1}{2}\left|(x_i' - x_i)^2\left(\mathrm{var}_{\eta_i}(\hat{x}_i)\right)^{-1} + 2(x_i - \hat{x}_i)(\mathrm{var}_{\eta_i}(\hat{x}_i))^{-1}(x_i' - x_i)\right| . \tag{4.35}$$

So, we can write this as,

$$\left|\log\left(\frac{\prod_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D)}{\prod_{i\in V_{ZO}}\Pr(\hat{x}_i \mid \hat{x}_{Pa(i)}, D')}\right)\right|$$
$$= \frac{1}{2}\Big|(x'_{ZO} - x_{ZO})^\top \Sigma^{-1}_{\eta_{ZO}}(\hat{x}_{ZO})\,(x'_{ZO} - x_{ZO}) + 2(x_{ZO} - \hat{x}_{ZO})^\top\Sigma^{-1}_{\eta_{ZO}}(\hat{x}_{ZO})(x'_{ZO} - x_{ZO})\Big| . \tag{4.36}$$

So as we stated in Eq. (4.22), we require the expression on the righthand-side of the equality sign in Eq. (4.36), to be smaller than ϵ with probability at least $1-\delta$. Due to the absolute value, we require a probability of at least $1-\delta/2$ on both sides:

$$\Pr\Big(\frac{1}{2}\big((x'_{ZO} - x_{ZO})^\top \Sigma^{-1}_{\eta_{ZO}}(\hat{x}_{ZO})\,(x'_{ZO} - x_{ZO}) + 2(x_{ZO} - \hat{x}_{ZO})^\top\Sigma^{-1}_{\eta_{ZO}}(\hat{x}_{ZO})(x'_{ZO} - x_{ZO})\big) \geq \epsilon\Big) \leq \frac{\delta}{2}. \tag{4.37}$$

Equivalently, we can write,

$$\Pr\Big(2(x_{ZO} - \hat{x}_{ZO})^\top\Sigma^{-1}_{\eta_{ZO}}(\hat{x}_{ZO})(x'_{ZO} - x_{ZO}) \geq 2\epsilon - (x'_{ZO} - x_{ZO})^\top \Sigma^{-1}_{\eta_{ZO}}(\hat{x}_{ZO})\,(x'_{ZO} - x_{ZO})\Big) \leq \frac{\delta}{2}. \tag{4.38}$$

The variance of the term w.r.t η appearing on the left side of Eq. (4.38) inside the probability is,

$$\begin{aligned}
&\text{var}\left(2(x_{ZO}-\hat{x}_{ZO})^\top\Sigma_{\eta_{ZO}}^{-1}(\hat{x}_{ZO})(x'_{ZO}-x_{ZO})\right)\\
&= 4(x'_{ZO}-x_{ZO})^\top\Sigma_{\eta_{ZO}}^{-1}(\hat{x}_{ZO})\text{var}((x_{ZO}-\hat{x}_{ZO})^\top)\Sigma_{\eta_{ZO}}^{-1}(\hat{x}_{ZO})(x'_{ZO}-x_{ZO})\\
&= 4(x'_{ZO}-x_{ZO})^\top\Sigma_{\eta_{ZO}}^{-1}(\hat{x}_{ZO})\Sigma_{\eta_{ZO}}(\hat{x}_{ZO})\Sigma_{\eta_{ZO}}^{-1}(\hat{x}_{ZO})(x'_{ZO}-x_{ZO})\\
&= 4(x'_{ZO}-x_{ZO})^\top\Sigma_{\eta_{ZO}}^{-1}(\hat{x}_{ZO})(x'_{ZO}-x_{ZO}) \qquad (4.39)
\end{aligned}$$

For any centered Gaussian random variable g with variance σ_g^2, we have the following tail bound[43]:

$$P(g\geq\lambda)\leq\frac{\sigma_g}{\lambda\sqrt{2\pi}}\exp\left(-\lambda^2/2\sigma_g^2\right). \qquad (4.40)$$

We will apply this to Eq (4.38), setting the following assignments,

$$\begin{aligned}
g &= 2(x_{ZO}-\hat{x}_{ZO})^\top\Sigma_{\eta_{ZO}}^{-1}(\hat{x}_{ZO})(x'_{ZO}-x_{ZO}), \qquad (4.41)\\
\lambda &= 2\epsilon-(x'_{ZO}-x_{ZO})^\top\Sigma_{\eta_{ZO}}^{-1}(\hat{x}_{ZO})(x'_{ZO}-x_{ZO}). \qquad (4.42)
\end{aligned}$$

We can see from Eq. (4.39),

$$\sigma_g^2 = 4(x'_{ZO}-x_{ZO})^\top\Sigma_{\eta_{ZO}}^{-1}(\hat{x}_{ZO})(x'_{ZO}-x_{ZO}). \qquad (4.43)$$

Considering the righthand side is of Eq. (4.40) is smaller than $\delta/2$, we can write,

$$\begin{aligned}
&\log(\frac{\sigma_g}{\lambda\sqrt{2\pi}}\exp\left(-\lambda^2/2\sigma_g^2\right))\leq\log(\delta/2),\\
&\log(\frac{\sigma_g}{\lambda\sqrt{2\pi}})+\log(\exp\left(-\lambda^2/2\sigma_g^2\right))\leq\log(\delta/2),\\
&\log\left(\frac{\lambda}{\sigma_g}\right)+\frac{1}{2}\left(\frac{\lambda}{\sigma_g}\right)^2\geq\log\left(\frac{2}{\delta\sqrt{2\pi}}\right). \qquad (4.44)
\end{aligned}$$

We will denote with $\sigma_{GM}^2(\epsilon,\delta)$, the minimal variance of additive Gaussian noise needed to make a variable in the range $[0,1]$, (ϵ,δ)-differentially private. E.g., Authors of [48] showed that if $\sigma\epsilon\geq 3/2$ and $(\sigma\epsilon)^2\geq 2\log(1.25/\delta)$ then $\sigma^2\geq\sigma_{GM}^2(\epsilon,\delta)$ holds.

We briefly repeat here an adapted version of their derivation, where we set $\sigma=\frac{2}{\sigma_g}$. To make this inequality hold, we require that both the following Eq. (4.45) and (4.46) hold,

$$\log\left(\frac{\lambda}{\sigma_g}\right)\geq 0\equiv\lambda\geq\sigma_g, \qquad (4.45)$$

and

$$\frac{1}{2}\left(\frac{\lambda}{\sigma_g}\right)^2\geq\log\left(\frac{2}{\delta\sqrt{2\pi}}\right). \qquad (4.46)$$

From Eq (4.42) and (4.43), we see that,

$$\frac{\lambda}{\sigma_g} = \frac{2\epsilon}{\sigma_g} - \frac{\sigma_g}{4} = \frac{2\epsilon}{\sigma_g} - \frac{\sigma_g}{2\epsilon} \times \frac{\epsilon}{2}. \tag{4.47}$$

We observe that if $\frac{2\epsilon}{\sigma_g} \geq 3/2$ and $\epsilon \leq 1$ then, the following satisfies Eq (4.45),

$$\frac{\lambda}{\sigma_g} \geq \frac{3}{2} - \frac{2}{3}.\frac{1}{2} > 1. \tag{4.48}$$

Moreover, if also $\frac{2\epsilon}{\sigma_g} \geq \sqrt{2\log(1.25/\delta)}$, there holds

$$\begin{aligned}
\frac{1}{2}\left(\frac{\lambda}{\sigma}\right)^2 &= \frac{1}{2}\left(\frac{2\epsilon}{\sigma_g} - \frac{\sigma_g}{2\epsilon}.\frac{\epsilon}{2}\right)^2 \\
&\geq \frac{1}{2}\left(\frac{2\epsilon}{\sigma_g} - \frac{\sigma_g}{2\epsilon}.\frac{1}{2}\right)^2, (\text{because } \epsilon \leq 1) \\
&= \frac{1}{2}\left(\left(\frac{2\epsilon}{\sigma_g}\right)^2 - 1 + \left(\frac{\sigma_g}{2\epsilon}.\frac{1}{2}\right)^2\right) \\
&= \frac{1}{2}\left(\left(\frac{2\epsilon}{\sigma_g}\right)^2 - 1 + \left(\frac{2}{3}.\frac{1}{2}\right)^2\right) \\
&= \frac{1}{2}\left(\left(\frac{2\epsilon}{\sigma_g}\right)^2 - \frac{8}{9}\right) \\
&\geq \frac{1}{2}\left(2\log(1.25/\delta) - \frac{8}{9}\right) \\
&\geq \log\left(\sqrt{\frac{2}{\pi}}\frac{1}{\delta}\right),
\end{aligned} \tag{4.49}$$

and which satisfies Eq. (4.46).

Next to Dwork's conditions [48], other upper bounds for $\sigma_{GM}(\epsilon, \delta)$ have also been proposed, e.g., [9]. In general, in the sequel we will assume that,

$$\left(\frac{2}{\sigma_g}\right)^2 \geq \sigma^2_{GM}(\epsilon, \delta). \tag{4.50}$$

Substituting from Eq. (4.43), we see this is equivalent to

$$\begin{aligned}
&\frac{4}{4(x'_{ZO} - x_{ZO})^\top \Sigma^{-1}_{\eta_{ZO}}(\hat{x}_{ZO})(x'_{ZO} - x_{ZO})} \geq \sigma^2_{GM}(\epsilon, \delta), \\
&\equiv (x'_{ZO} - x_{ZO})^\top \Sigma^{-1}_{\eta_{ZO}}(\hat{x}_{ZO})(x'_{ZO} - x_{ZO}) \leq \sigma^{-2}_{GM}(\epsilon, \delta).
\end{aligned} \tag{4.51}$$

We re-write this as follows,

$$\sum_{i \in V_{ZO}} (x'_i - x_i)^2 \Sigma^{-2}_{\eta_i}(\hat{x}_i) \leq \sigma^{-2}_{GM}(\epsilon, \delta),$$

$$\equiv \sum_{i \in V_{ZO}} \left(\frac{(f_i(x'_{pa(i)}) - f_i(x_{pa(i)}))^2}{(\Sigma_{\eta_i}(\hat{x}_i))^2} \right) \leq \sigma_{GM}^{-2}(\epsilon, \delta). \tag{4.52}$$

Substituting the value of $(\Sigma_{\eta_i}(\hat{x}_i))^2$ from Eq. (4.9) here, we form our final constraint program as defined in Eq. (4.54).

4.5.4 Forming the final constraint optimization problem

From the tradeoffs between utility and privacy, we know that a large value for each $\sigma_{\eta,i}$ protects the privacy of sensitive x_i but compromises the utility of any observed $\hat{x}_i$. On the other hand, a small value for $\sigma_{\eta,i}$ maximizes the utility of observable $\hat{x}_i$ but loses in terms of the privacy of sensitive x_i. We try to strike a balance by developing a constraint problem that maximizes utility subject to privacy restrictions. So, we formalize our structure using two components: **utility maximization** and **privacy constraint**.

1. ***Utility maximization*** The objective for this part is to minimize a function that measures the loss in utility between a certain RV x_i and the corresponding $\hat{x}_i$. A natural function for this is

$$\mathbb{E}_{\sigma_\eta} \left[\|x_i - \hat{x}_i\|^2 \right]. \tag{4.53}$$

   We define the cost function to be minimized as the weighted sum of the variance of the noise component, that is added to the x_i. We provide a concrete expression for this in Eq. (4.54).

2. ***Privacy constraint*** Obviously, the optima of the objective function in Eq. (4.53) is reached, when the noise variances are all zero, so we put a constraint on σ_η, such that $\hat{x}_i$ is (ϵ, δ)-differentially private, for a fixed $\epsilon > 0$ and $\delta > 0$. The expression of the constraint is also written in Eq. (4.54).

Final constraint optimization problem Combining the aforementioned objective function to maximize utility and the constraint to specify privacy requirements, we get a constraint optimization problem, which is:

$$\begin{aligned} &\text{minimize} \\ &\quad \textstyle\sum_{i=1}^{k} \alpha_{\eta,i} \sigma_{\eta,i}^2 \\ &\text{subject to} \\ &\quad \textstyle\sum_{i \in V_{ZO}} \left(\frac{(f_i(x'_{Pa(i)}) - f_i(x_{Pa(i)}))^2}{\left(\sum_{j \in Pa(i)} \left(\frac{\partial f}{\partial x_j} x_{Pa(i)} \right)^2 \Sigma_\eta(f_i(\hat{x}_j)) + \mathbb{1}_i \mathbb{1}_i^\top \sigma_{\eta,i}^2 \right)^2} \right) \leq \sigma_{GM}^{-2}(\epsilon, \delta), \end{aligned} \tag{4.54}$$

where α_η is a vector representing the cost induced by the noise η. Next, we explain the components of our constraint as presented in Eq. (4.54).

- In the **numerator of the left side** of Eq. 4.54, the expression $(f_i(x'_{Pa(i)}) - f_i(x_{Pa(i)}))^2$, is the squared difference between two corresponding intermediate or observed (as $i \in V_{ZO}$) random variables (or partial orderings of LBN) belonging to two neighboring dataset D and D'.
- The **denominator of the left side** of Eq. (4.54) represents the variance of noisy x_i variable w.r.t. noise variable η.
  - Expression $\left(\frac{\partial f}{\partial x_j} x_{Pa(i)}\right)^2$ is the squared partial derivative of the parent RVs (indexed by j) of a particular RV x_j.
  - Expression $\Sigma_\eta(f_i(\hat{x}_j))$ is the variance of the output of each intermediate or observed RVs w.r.t noise variable η.
  - Expression $\mathbb{1}_i \mathbb{1}_i^\top \sigma^2_{\eta,i}$ represents the noise variance of the i-th RV.
- On the **right side** of Eq. 4.54, the expression $\sigma^2_{GM}(\epsilon, \delta)$ is the variance of the minimal additive Gaussian noise required to make a random variable with range $[0, 1]$, (ϵ, δ)-differentially private (e.g., [48] shows that if $\sigma\epsilon \geq 3/2$ and $(\sigma\epsilon)^2 \geq 2\log(1.25/\delta)$, then $\sigma^2 \geq \sigma^2_{GM}(\epsilon, \delta)$).

Solving this Constraint Optimization Problem (CP) optimizes the objective function while satisfying the constraints. It shows that if we add noise to the variables in a Bayesian network and translate that into a CP, solving the CP gives a differentially private solution, with optimized utility and the desired privacy guarantee, which in turn is achieved by computing the optimized noise variance $\sigma_{\eta,i}$ for x_i variables.

4.5.5 Inference on our running example

In this section, we present the synthesis of the constraint problem (CP), based on Eq. (4.54), for the scenario in our running Example 4.4. We will infer privacy constraints from the specifications 4.4.3 of a query on some observed random variable. Doing so, we will be able to see the concepts that we have built till now being applied in practice here.

We assumed a patient named `uma` with cancer of type `c`. Drug `{d1, d2, d3, d4}` are being administered to her. A doctor observes the RV `expectedSurvivalLength(uma)`, with $(0.25, 0.03)$-differentially privacy. We summarize some assumptions for the LBN presented in Fig.4.6 and the assumed notations according to Table 4.2.

- For the ease of expressing our example mathematically, we assign identifiers to the set of RVs such that the set $V_A =$ `{age, rsm, rst, ga1, ga2, ga3, ga4, ga5, ga6, dr1, dr2, acs, esl}` stores the identifiers, as given in Table 4.3, and depicted in Fig. 4.6.

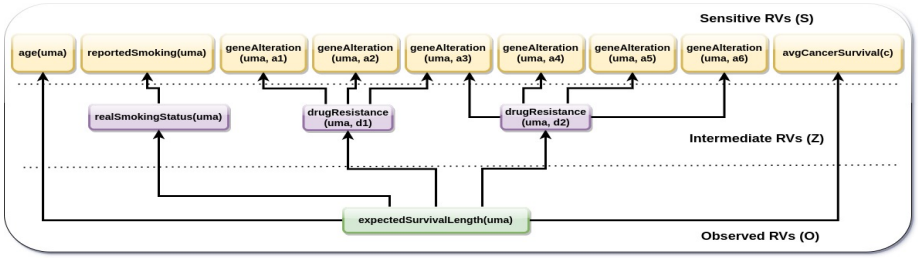


Figure 4.6: The different zones of LBN belonging to the sensitive RVs, the intermediate RVs, and the observed RVs in the logical Bayesian network in our running example, when we observe the RV `expectedSurvivalLength(uma)`.

Identifiers	Random Variable
age	`age(uma)`
rsm	`reportedSmoking(uma)`
rst	`realSmokingStatus(uma)`
ga1	`geneAlteration(uma, a1)`
ga2	`geneAlteration(uma, a2)`
ga3	`geneAlteration(uma, a3)`
ga4	`geneAlteration(uma, a4)`
ga5	`geneAlteration(uma, a5)`
ga6	`geneAlteration(uma, a6)`
dr1	`drugResistance(uma, d1)`
dr2	`drugResistance(uma, d2)`
acs	`avgCancerSurvival(c)`
esl	`expectedSurvivalLength(uma)`

Table 4.3: The identifiers for random variables associated with observed RV `expectedSurvivalLength(uma)`, in our running example.

- We consider the observed RV is `expectedSurvivalLength(uma)`. So, the set V_O holds the identifier {`esl`} of the RV `expectedSurvivalLength(uma)`.
- The observed RV depends on intermediate RVs `realSmokingStatus(uma)`, `drugResistance(uma, d1)`, `drugResistance(uma, d2)`. So, set V_Z holds the identifiers {`rst, dr1, dr2`}.
- The set V_{ZO} holds the identifiers {`esl, rst, dr1, dr2`} representing the identifiers of intermediate and output RVs.
- These intermediate and observed RVs depend on some sensitive, input RVs. So, the set V_S holds the identifiers {`age, rsm, ga1, ga2, ga3, ga4, ga5, ga6, acs`} representing the RVs in the set, {`age(uma), reportedSmoking(uma), geneAlteration(uma, a1), geneAlteration(uma, a2), geneAlteration(uma, a3), geneAlteration(uma, a4), geneAlteration(uma, a5), geneAlteration(uma, a6), avgCancerSurvival(c)`}.

Furthermore, from this, we can infer a constraint problem, optimizing which will help us achieve our objective of publishing `expectedSurvivalLength(uma)` to some observer in a privacy-preserving way. This is achieved by satisfying the (ϵ, δ)-differential privacy requirement, specified in Section 4.4.3. We can use the constraint problem structure we've derived in Eq. 4.54, to model this.

We, minimize
$\sum_{i \in V_A} \alpha_{\eta,i} \sigma^2_{\eta,i}$
subject to

$$\sum_{i \in V_{ZO}} \left(\frac{(f_i(x'_{Pa(i)}) - f_i(x_{Pa(i)}))^2}{\left(\sum_{j \in Pa(i)} \left(\frac{\partial f}{\partial x_j} x_{Pa(i)} \right)^2 \Sigma_\eta (f_i(\hat{x}_j)) + \mathbb{1}_i \mathbb{1}_i^\top \sigma^2_{\eta,i} \right)^2} \right) \leq \sigma^{-2}_{GM}(\epsilon, \delta). \tag{4.55}$$

In this case, an observer `doctor` observes the RV `expectedSurvivalLength(uma)` for patient `uma`, with (0.25, 0.03)-differential privacy. From this CP, we can iteratively calculate the ground-level constraints to reach all the sensitive, input RVs through the intermediate RVs, starting from our observed RV.

Here, we show a small intermediate step of the computation of the constraint program for the RV `realSmokingStatus(uma)`, represented by identifier `rst` in the set V_{ZO}. For this RV, the component of the constraint (4.55) is,

$$\left(\frac{(f_{rst}(x'_{rsm}) - f_{rst}(x_{rsm}))^2}{\left(\left(\frac{\partial}{\partial x_{rsm}} f_{rst}(x_{rsm}) \right)^2 \Sigma_\eta (f_{rst}(\hat{x}_{rsm})) + \mathbb{1}_{rst} \mathbb{1}_{rst}^\top \sigma^2_{\eta,rst} \right)^2} \right) \tag{4.56}$$

Assuming, the **tangent** function is applied to the parent random variable of `rst`, we can rewrite the above as

$$\left(\frac{(tan(x'_{rsm}) - tan(x_{rsm}))^2}{\left(\left(\frac{\partial}{\partial x_{rsm}} tan(x_{rsm}) \right)^2 \Sigma_\eta (tan(\hat{x}_{rsm})) + \mathbb{1}_{rst} \mathbb{1}_{rst}^\top \sigma^2_{\eta,rst} \right)^2} \right) \tag{4.57}$$

Similarly, for every random variable in the set V_{ZO}, we can compute the corresponding component of the constraint (4.55). Solving the constraint problem in Eq. (4.55), optimizes the objective function to minimize the loss incurred by the introduction of noise to the random variables in the LBN 4.6, hence maximizing the utility, while satisfying the constraints representing the privacy requirements 4.4.3. It computes the optimized noise variance for all the RVs associated with observed RV `expectedSurvivalLength(uma)`. Solving the CP provides a $(0.25, 0.03)$-differentially private solution. Similarly, we can synthesize the constraint problem for our other observed RV `survivalMoreThan6Month(uma)`.

4.6 Discussion and conclusion

In this chapter, we introduce strategies for the development of efficient and interpretable privacy-preserving systems.

First, we adopt an innovative approach to define privacy requirements in a data pipeline, which culminates in the design of a constraint optimization problem that yields privacy-preserving solutions. From the approaches presented in

this study, it is evident that privacy specifications can be effectively translated into constraint problems that can be efficiently resolved.

Second, our proposed declarative language for specifying privacy requirements is built on the base of LBN (logical Bayesian network) and presents a lucid framework. This approach makes it easier to explain the privacy constraints imposed on a system. Furthermore, this enables us to answer questions regarding the privacy-preservation guarantees of the system and encourages us to reason with them.

Third, our constraint optimization approach liberates developers from the responsibility of making complex implementation choices, allowing them to solely focus on the specification of the requirements. The optimal solution is then determined automatically by solving the constraint optimization problem.

Fourth, the strategy we present is generalizable and easily adaptable to many scenarios, depending on the particular problem and intended solution. The outcome of the constraint problem could be used for computing the optimal noise variance, optimal noise placement within the data pipeline, or other relevant parameters.

This research opens up new possibilities for investigation, presenting questions which we will discuss in detail in Chapter 6.

Chapter 5

Tailored noise mechanism

Abstract

In this chapter, we study the distributed privacy-preserving averaging (alternatively, aggregation) of sensitive attributes (alternatively, features) locally privatized by participating parties. Every participant intends to collaboratively compute the averages over those privatized features, but only after optimizing a constraint problem to obtain privatization functions for privatizing the sensitive features. A central curator can run a solver to solve a constraint optimization problem to find the optimal value of the privacy parameters or the privacy functions. The goal of such a mechanism is to compute the privatization functions yielding maximum utility with the desired privacy preservation guarantee. Once computed, the optimum parameters are shared with distributed parties, who then locally privatize their sensitive features.

Now, computing these features can be performed by transforming sensitive attributes, and such transformations may contain singularities or high-magnitude gradients, leading to the risk of obtaining an outlier feature. This concern can be alleviated by designing a tailored noise mechanism for privatizing sensitive features using privatization functions. These functions are obtained by solving a *convex constraint optimization problem*, which performs *bias-variance minimization* of noise and intends to *select informative intervals of transformation* only.

Declaration

This work is jointly conducted by my fellow doctoral student Arijus Pleska and me. It can be divided into the following collaborative axes on which we worked together:

1. Expression of models whose parameters are obtained from U-statistics. We mainly focused on this work from different aspects concerning our theses.

- Moitree: Generalized expression of models and their combination.
- Arijus: Comparison of the model where privatization occurs after and before transformations (with high-magnitude gradients) of sensitive features.

2. Discretization of domains of features.
3. Expression of the constraints and the objective function of the convex program for obtaining privatization functions.
4. Experimental setup.

We collaboratively worked on the last three axes, where we both contributed in improving the final results.

5.1 Introduction

We consider a set of parties that want to collaboratively run some aggregation on sensitive features, but only after local privatization functions are applied to them. In this chapter, we study the privacy-preserving aggregation of sensitive attributes once they are converted into features by applying useful non-linear transformation functions to them.

So far, we have discussed solutions that involve only additive noise, such as Laplacian or Gaussian noise distributions. However, we have always agreed that we may need specially tailored noise distributions for specific problem scenarios. This takes a step further than using simple classical noise distributions in a local differentially private setup. In our earlier chapters, we discussed approaches involving specific noise distributions and finding the right places to apply them. In contrast, this chapter takes a more generalized approach that encompasses the earlier models and explores ways to shape unusual noise distributions carefully curated for individual problems and their privacy guarantee needs. This approach also aims to achieve unbiased estimates of the parameters.

In Chapters 3 and 4, we discussed combining building blocks by either choosing the more optimal building block in a parallel situation or using an aggregation of multiple such building blocks.

In our tailored noise mechanism (TNM) approach, we create two different regression models. These regression models can be fit on instances whose linear attributes are transformed into features by a highly non-linear (possibly discontinuous) transformation function, either before or after applying noise.

This helps us avoid the problem of calculating highly non-linear functions like the inverse of sensitive variables. For example, if we take the inverse of sensitive variables x after adding simple Gaussian noise (with negligible or close-to-zero value), then after applying transformation functions we get $(x + noise)^{-1}$, and the additive noises can create overshooting of the largest (or undershooting of the smallest) possible values in the distribution. A similar concern applies to other non-linear transformations like $log(x + noise)$ or $tan(x + noise)$. We need a carefully tailored distribution of noise to address this problem.

Moreover, if the transformation is discontinuous, e.g., the logarithm is discontinuous at 0+ (on the real number line, the value immediately to the right of 0), then the application of a classic noise mechanism is likely to cause a higher loss of information than usual or lead to a suboptimal solution. So, if a classic noise mechanism is applied before the transformation, it is likely to result in an extreme loss of utility because noisy points might fall close to the discontinuities of the transformation function. Otherwise, if the classic noise mechanism is applied after the transformation, it is known that it adds more noise than is needed for the fixed privacy budget.

So, unlike the previous chapters, instead of deciding the amount and position of required noise, here we try to achieve privacy preservation and utility maximization by tailoring the shape of the noise distribution itself. We use constraint optimization to find the privatization function that is optimal for providing privacy to our sensitive features. Once the privatization function is obtained by running a constraint optimization solver, the different parties can locally privatize their sensitive features and collaboratively compute the averaging on privatized features. We show that TNM outperforms traditional privacy solutions on both synthetic and actual datasets, resulting in a reduced loss value between true and predicted values.

Outline: Next in Section 5.2, we describe the preliminaries of concepts related to the tailored noise mechanism. In Section 5.3, we briefly review the literature available in the related field. In Section 5.4, we go into further detail in modeling the constraint optimization problem through different steps, like domain discretization, defining the objective function, and the constraints. Then in Section 5.5, we describe the datasets, experimental setup, and evaluation criteria. In Section 5.6, we present the result with our interpretation, followed by a result summary. Finally, in Section 5.7, we conclude with a discussion of our limitations and possible future directions.

5.2 Preliminaries

In this segment, we will provide a concise problem statement in Section 5.2.1 and explain our approach to resolving the aforementioned problem in Section 5.2.2. In Section 5.2.3, we will provide some mathematical notations and their technical explanations that will be useful in understanding the content of the following sections.

5.2.1 Problem statement

Our problem is based on a distributed multi-party computation setting. These distributed parties intend to collaboratively compute the aggregation of some sensitive information. For that purpose, they privatize their sensitive features locally before publishing them to the central curator. Once they receive such privatized features, they compute a statistical model fitting those data. For privatization, we need to add noise, and unfortunately, classic additive noise

mechanisms, e.g., the Laplace mechanism 2.7.4.1 or the Gaussian mechanism 2.7.4.2, can sometimes have undesirable properties, such as providing suboptimal utility by using unnecessarily high noise [9].

We can explain with a similar example introduced in Section 5.1. We know that the reciprocal function x^{-1} has a singularity at 0. Now, if the i-th party has x_i sensitive information, where $0 \leq i \leq n$, then after the publication of $\tilde{x}_i = (x_i + noise)$ by adding Gaussian *noise*, the parties want to compute together the aggregate $\sum_{i=0}^{n}(\tilde{x}_i)^{-1}$, $\forall i$. Then there is the risk that for some i, $\tilde{x}_i$ gets close to 0, and hence $\tilde{x}_i^{-1}$ becomes arbitrarily large, as does $\sum_i(\tilde{x}_i)^{-1}$.

Now, designing appropriate features for fitting an accurate statistical model is crucial. So, once the privatized versions of corresponding sensitive attributes are published, features can be computed from them. New features can also be computed from other existing features. The transformation functions (or feature functions) applied to attributes or features to compute new features can contain singularities and gradients of high magnitude. We refer to *sensitive features* and *sensitive attributes* interchangeably. Some examples of such feature functions or transformations are the identity function, reciprocals, logarithms, tangents, etc.

5.2.2 Proposed solution

In this work, to avoid the fundamental disadvantages of classical additive noise mechanisms, we decided to choose our tailored noise distribution. This is achieved by a few steps as described below:

- In the tailored noise mechanism (TNM), to alleviate the problem of obtaining an outlier feature due to the transformation of a privatized feature in an interval where the transformation has gradients of high magnitudes or singularities, we apply discretization strategies. The target is to select informative intervals of transformation only.

- TNM produces discrete probability distributions for privatization of features, inspired by the discrete Gaussian mechanism [24]. The discretization is applied to both the domains of sensitive features and the domains of privatized features alike. Besides, we work with bounded domains for the privatized features, in a similar way to the bounded domains of the corresponding sensitive features.

- We design a constraint optimization problem (CP), solving which provides a privatization function with utility-maximizing noise parameters. Ideally, a central curator can run a solver to find the conditional probability mass functions or the optimized privatization functions and then publish them to all the distributed parties participating in the study. We denote the spaces of the sensitive features and the privatized features by χ and $\tilde{\chi}$, respectively, such that $\chi \subseteq \tilde{\chi}$.

  The goals of the constraint problem can be broken down into the following:

- The objective function should minimize the variance of the tailored noise.
- The objective function should minimize the bias of the tailored noise.
- The constraints should abide by the differential privacy requirements.
- The constraints should minimize the absolute difference between the privatized features and corresponding sensitive features.

5.2.3 Notations

In our distributed setting, the n participating parties publish their data-tuple, consisting of a sensitive feature vector of m features in the space χ and the corresponding scalar target value. Our statistical model maps the feature vector to the scalar target value.

We want to approximate the true parameters of the statistical model h^*, parameterized by θ^*. We consider l models, indexed by $k \in [l]$ (where $l \geq 1$ is an integer).

For some model k, we will have a function $g_k : X \to \hat{\theta}$, which will provide us with the closest approximation of θ^*, and we call those estimated parameters $\hat{\theta}$. We define that these functions can be written in the form,

$$g_k \left(f_{k,1}(X), f_{k,2}(X), \ldots, f_{k,q_k}(X) \right), \tag{5.1}$$

where the $f_{k,s}$ are functions computing U-statistic s in model k from matrix X, q_k is the number of U-statistics in model k, and g_k are functions computing parameters from U-statistics in model k. The g_k varies depending on the fitting method of the model or privatization method. In particular, let $\mathcal{L}$ be an objective function. Then, we want to find a model that minimizes $\mathcal{L}(\theta^*, \hat{\theta})$, such that it minimizes the sum of the squared differences between the elements of the true parameters θ^* and the corresponding elements of the estimated parameters $\hat{\theta}$.

To summarize, the central curator solves a convex program for computing the privatization functions, represented by conditional probabilities $\Pr_{k,j}(\tilde{x} \mid x)$, $\forall k \in [l]$, $j \in [m_k]$, $x \in \chi_{k,j}$, $\tilde{x} \in \tilde{\chi}_{k,j}$, i.e., the probability that some party draws for its j-th feature a noisy value $\tilde{x}$ if the real value was x. These CPFs are published to the parties. The parties agree to locally privatize the m_k sensitive features used for fitting model k, using this conditional probability distribution.

Next, the parties share the privatized features with the central curator, who then computes the matrix X composed of features and target values over n distributed parties. Afterward, the central curator fits a statistical model $\hat{h}$, mapping a feature vector to a target value using functions g and f. All the notations are represented in a clear and concise way in Table 5.1.

5.3 Related work

A lot of the general body of work that has inspired and influenced us has already been discussed in earlier chapters. In this section, we will briefly summarize the most relevant literature we have studied and found inspiration from.

Symbol	Meaning
n	Number of parties
m'	Number of raw attributes
m	Number of features, such that $m \geq m'$
X	Data matrix containing both features and the target values
x, x_1, x_2	Values of a sensitive feature
$\tilde{x}$	Value of a privatized (noisy) feature
χ	Discretized feature domain of sensitive feature, $\chi \subseteq \mathbb{R}^m$
$\tilde{\chi}$	Discretized feature domain of privatized (noisy) feature
θ^*	True (optimal) parameters of a statistical model
$\hat{\theta}$	Estimated parameters of a statistical model
m_k	Number of sensitive features in k-th model, indexed by j
l	Number of models for parameter estimation, indexed by k
T	Set of privatized features, such that $T \subseteq \tilde{\chi}_{k,j}$
q_k	Number of U-statistics in model k, $q_k \in \mathbb{N}$
g_k	Function computing parameters from U-statistics in model k, $g_k : \mathbb{R}^{q_k} \rightarrow \mathbb{R}^{m+1}$
$f_{k,s}$	Function computing U-statistic $s \in [q_k]$ in model k from matrix X, $f_{k,s} : \mathbb{R}^{n \times (m+1)} \rightarrow \mathbb{R}$
r_j	Number of features computed from attribute j, indexed by t
$\mathcal{L}$	Objective function
$\mathcal{L}_k^{variance}$	Variance component of objective function in model k
$\mathcal{L}_k^{bias}$	Bias component of objective function in model k
ϵ_k	Privacy budget for every sensitive feature of model k
$\rho_k^{variance}$	Scaling factor for variance in model k
ρ_k^{bias}	Scaling factor for bias in model k
$\Pr_{k,j}^{\max}(\tilde{x})$	The highest value in $\{\Pr_{k,j}(\tilde{x} \mid x) : x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j}\}$
$b_{k,j,x}$	Bias term for model k, sensitive feature j, and $x \in \chi_{k,j}$

Table 5.1: Notations used in privatization of sensitive features using tailored noise mechanism.

The two major pillars of our work are based on the papers [64] and [29, 30]. Both of these works focus on privacy-preserving mechanisms driven by utility maximization. They solve convex programs to achieve the aforementioned goal. [29] applies additional constraints to [64] and is characterized by count data. Additionally, [68], deals with count data under a very similar family of problems. Similarly, [62] and [113] deal with similar problems but involve central differential privacy and local differential privacy, respectively.

Moreover, we have already discussed our inspiration from [24] in Section 5.2.2 and will delve into it further in the upcoming Section 5.4.1. This work discusses the discretized Gaussian mechanism applied within the differential privacy framework. In [91], the authors worked on ϵ-differential privacy using the Laplace mechanism. They discussed the issues they faced due to the least significant bits, where our discretization strategy can avoid dependence on the

least significant bits by discretizing the features into bins and rounding their representative values to a higher significant bit.

The staircase discretization mechanism [61] splits the range of continuous data into discrete intervals or bins with varying widths according to the desired level of privacy protection and the sensitivity of the data. Wider intervals are usually used in areas where the data is less sensitive or where adding more noise won't substantially degrade its utility. By providing more flexibility than equal width and equal frequency discretization, this choice of dynamically adjusted interval widths aims to balance the trade-off between privacy protection and data utility. This allows for more nuanced privacy-preserving transformations of the data by taking the sensitivity of the data into consideration to achieve optimal privacy-preserving transformations.

Unfortunately, despite all the advantages of the staircase discretization mechanism over traditional discretization methods, in our current local differential privacy setting, the mechanism may face challenges. The staircase discretization mechanism relies on central coordination to determine appropriate interval widths based on the sensitivity of the data. Besides, data distributions of different data contributors may have varying sensitivity. The risk of privacy breaches may increase and compromise the overall effectiveness of the privacy mechanism, if individual data contributors struggle to coordinate discretization parameters, and compare sensitivities. This can inadvertently lead to failure to ensure consistent and appropriate discretization across all contributors, and set appropriate noise levels to reach optimal privacy.

5.4 Modeling our approach

In this section, we aim to derive a Constrained Optimization Problem. The purpose of the proposed tailored noise mechanism is to compute the conditional probability for the noisy distribution of data, privatizing the sensitive values of the attributes.

The goal of this optimization problem will be:

1. The sampled data according to this probability distribution of the privatization function should provide an $(\epsilon, 0)$-differential privacy guarantee (see Section 5.4.3.1).

2. The variance of the privatization function should be minimized (see Section 5.4.2.1).

3. The biasedness of the privatization function should be minimized (see Sections 5.4.2.2 and 5.4.3.2).

4. The optimization problem must satisfy the basic, implicit mathematical conditions of standard probability theory (see Section 5.4.3.3).

We remark that the constraint optimization program should also guarantee that:

- The noisy points don't fall on the discontinuities of the feature and target variable transformations.
- The optimum noise is used as the privacy budget requires, to maximize the utility while preserving the privacy guarantee.

If we consider multiple regression models covering the cases of adding the differential privacy noise before and after the transformation, we can solve the constraints for these regression models so that the model with the best tradeoff between utility and data privacy can be chosen. The notations for different privatization models are defined in Table 5.2.

Symbol	Meaning
dp	Application of privatization function to achieve differential privacy
tns	Application of transformation function to compute features from attributes
$\circ$	Function composition, where $F_1 \circ F_2$ means applying F_2 first, and then F_1

Table 5.2: Notations used in representing the of application of modules.

- **Model** tns $\circ$ dp: The tailored mechanism where the differential privacy noise is applied to the attribute before they are transformed into features.
- **Model** dp$\circ$tns: The tailored mechanism where attributes are transformed into features before the differential privacy noise is applied to them.

In the following sections, we will discuss how we formalize the constraint optimization problem to achieve the goals discussed above through our tailored noise mechanism approach. First, in Section 5.4.1, we describe the discretization strategies for sensitive and noisy feature domains. Then, in Section 5.4.2, we define the objective function with the intention of minimization of loss and maximization of utility. Then, in Section 5.4.3, we define the different constraints that also satisfy the requirements we listed above.

5.4.1 Domain discretization

The noisy distribution is discrete because the local differential privacy requirement is defined as a constraint program. Also, the noisy distribution is multi-dimensional as we have several sensitive statistics for one data instance: the features and the target variable.

We will use a very commonly practiced data mining method to discretize our data domain, called **binning**. We will perform discretization of both the domains of sensitive features as well as the noisy, privatized features. At the core of discretization is a strategy to divide the continuous domain of a feature

variable into bins and pick a representative value from each of the bins. All the values of the feature that fall within a bin get mapped to the representative value of that corresponding bin. There are many binning methods in practice, and we will use two of the traditional strategies as mentioned in [134].

5.4.1.1 Equal width discretization

In equal width discretization (commonly called as equal-width binning method), given the number of **intervals** (or as commonly known as **bins**) as n, the total interval of the domain of the variable is divided into n bins of equal width. At the intersection of the bins, a **threshold** is placed. So, if the maximum and minimum value of the variable domain is y_{max} and y_{min}, then the interval $(y_{max} - y_{min})$ is divided into n bins of w width each, as follows,

$$w = \frac{(y_{max} - y_{min})}{n}. \tag{5.2}$$

The maximum and minimum values of the variable domain i.e., y_{max} and y_{min} are also referred to as the highest and the lowest threshold respectively.

This method is also called equal-distance discretization as the intervals are created at equal distances from each other. Deciding the number of bins, which computes the width of individual bins, is an important task. As the bins get narrower, the frequency of the data points in individual bins changes, and that may over-emphasize the noise present in the data. On the other hand, if we make the bins wider, a lot of details or patterns in the data may be overlooked. Now, the representative value of each bin is the middle value of every bin. As a result of creating equally-sized bins, the representative values are positioned at equal distances from each other as well.

We can also use equal-width binning for discretizing the transformed features. For that purpose, we need to first map the original feature to its transformation scale (say, by taking the inverse or natural logarithm), and then compute the bin intervals, and their representative values following the same process as we have already discussed. Once the transformed feature values are mapped to the representative values at the transformed scale, we can inversely transform them back (by taking the inverse or exponent) to the original feature scale. This results in more appropriately placed bins and their corresponding representative values, according to the magnitude of the gradients of the transformed features.

5.4.1.2 Equal frequency discretization

Another discretization strategy, named equal frequency discretization places the threshold after an equal frequency of data points, hence every bin holds an equal number of occurrences. Now, the representative value of bins can be chosen by various methods like choosing the mean or median or mode of every bin. For our purpose, we have placed the representative value at the middle, i.e., at an equal distance from both intervals on either side.

If random sampling is used on a dataset for experimenting, then depending on the sampled dataset, the bins and their thresholds may vary. This can also influence the representative value of the bins, hence affecting the final result of discretization.

The discrete domain of one category of the noisy distribution is called the discrete noisy domain. Discretized domains have the drawback that the noisy data is not as smooth as working with continuous domains, resulting in some loss of utility.

5.4.2 Defining the objective function

Our goal is to minimize the loss by mathematically minimizing the loss function $\mathcal{L}$. In our approach to solving the current problem in the form of a constraint problem, we will interchangeably use the term "objective function" instead. Optimizing the objective function will, in turn, achieve optimization (in our case, minimization) of the loss.

In our approach, the objective function consists of two components, and our goal will be to minimize the total objective function for model k, i.e., $\mathcal{L}_k$. So, our goal is to minimize the following,

$$\mathcal{L}_k = \mathcal{L}_k^{var} + \mathcal{L}_k^{bias}, \text{ for all } k \in [l], \tag{5.3}$$

where, $\mathcal{L}_k^{var}$ is the variance minimization component of the objective function, and $\mathcal{L}_k^{bias}$ is the bias minimization component of the objective function.

Now, as we have defined, each of these l models has its loss, and the final loss will be the aggregation of all of them as below,

$$\mathcal{L} = \sum_{k \in [l]} \mathcal{L}_k. \tag{5.4}$$

Next, we define the aforementioned two components of the objective function for variance and bias minimization, respectively.

5.4.2.1 Variance minimization

For each of the models indexed by k above, there is a variance minimization component. In model $tns \circ dp$, the objective function contains the variance of both the attributes and the transformed features. The variance minimization component can be written as,

$$\mathcal{L}_k^{var} = \sum_{j \in [m_k], t \in [r_j], x \in \chi_{k,j}, \tilde{x} \in \tilde{\chi}_{k,j}} \rho_{k,j,t}^{var} \Pr_{k,j}(\tilde{x} \mid x) \left(f'_{k,j,t}(\tilde{x}) - f'_{k,j,t}(x) \right)^2, \tag{5.5}$$

where the scaling term can be defined as,

$$\rho^{var}_{k,j,t} = 10^{-2} \min \left(1, \frac{1}{\max_{x \in \chi_{k,j}, \tilde{x} \in \tilde{\chi}_{k,j}} \left(f'_{k,j,t}(\tilde{x}) - f'_{k,j,t}(x) \right)^2} \right). \tag{5.6}$$

This calculates the variance minimization by summing up the scaled conditional probability for all the sensitive feature $j \in [m_k]$ in model k, for all features $t \in [r_j]$ derived from attribute j, multiplied by the individual squared difference of the function f' value when applied to the sensitive feature value and the privatized noisy feature value. We add 10^{-2} in the scaling term $\rho^{var}_{k,j,t}$ to keep the proposed values of the constraint variable $\Pr_{k,j}(\tilde{x} \mid x)$ within the interval length of 1 during the optimization process.

5.4.2.2 Bias minimization

The biasedness of the privatization function can be minimized by minimizing the absolute difference between the sensitive feature and the mean of the privatization function. For each of the models indexed by k above, there is a bias minimization component, represented by the constraint variable $b_{k,j,x}$. We can define it as follows,

$$\mathcal{L}^{bias}_{k} = \sum_{j \in [m_k], x \in \chi_{k,j}} \rho^{bias}_{k,j} b^2_{k,j,x}, \tag{5.7}$$

where the scaling term can be defined as,

$$\rho^{bias}_{k,j} = 10^{-2} \frac{1}{\max_{\tilde{x} \in \tilde{\chi}_{k,j}} \tilde{x}^2}. \tag{5.8}$$

Similarly, like variance minimization 5.4.2.1, we also add 10^{-2} to the scaling term here, so that bias minimization has a lower impact on the overall optimization as compared to variance minimization.

5.4.3 Defining the constraints

Now, as we have already defined the objective function in Section 5.4.2, here we define the constraints. We have three types of constraints: (1) constraints for the satisfaction of differential privacy requirements, (2) constraints for bias minimization, and (3) implicit constraints for compliance with probability theory fundamentals. We define each of them in further detail next.

5.4.3.1 Differential privacy constraints

From our assumption, we design k models, and if we have m_k sensitive features in the k-th model, then the total privacy budget ϵ will be evenly divided for each sensitive feature. So, the privacy budget for each of the m_k sensitive features is:

$$\epsilon_k = \frac{\epsilon}{m_k}. \tag{5.9}$$

We assume we have two adjacent values x_1 and x_2 for sensitive feature x, such that we add noise locally to the values under the notion of *local differential privacy* to prevent revealing sensitive information. From the definition of ϵ-differential privacy, such that $\epsilon \geq 0$, we can write,

$$\begin{aligned} \forall k \in [l],\ j \in [m_k],\ x_1, x_2 \in \chi_{k,j},\ T \subseteq \tilde{\chi}_{k,j} : \\ \sum_{\tilde{x} \in T} \Pr_{k,j}(\tilde{x} \mid x_1) \leq e^{\epsilon_k} \sum_{\tilde{x} \in T} \Pr_{k,j}(\tilde{x} \mid x_2). \end{aligned} \tag{5.10}$$

As the number of subsets $T \subseteq \hat{\chi}_{k,j}$ is exponential in $\mid \tilde{\chi}_{k,j} \mid$, we reformulate Eq. (5.10) as,

$$\begin{aligned} \forall k \in [l],\ j \in [m_k],\ x_1, x_2 \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j} : \\ \Pr_{k,j}(\tilde{x} \mid x_1) \leq e^{\epsilon_k} \Pr_{k,j}(\tilde{x} \mid x_2). \end{aligned} \tag{5.11}$$

We can further reformulate Eq. (5.11) to reduce the number of constraints even more,

$$\begin{aligned} \forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j} : \\ \Pr_{k,j}(\tilde{x} \mid x) \leq e^{\epsilon_k} \overset{\max}{\Pr_{k,j}}(\tilde{x}), \end{aligned} \tag{5.12}$$

$$\overset{\max}{\Pr_{k,j}}(\tilde{x}) \leq e^{\epsilon_k} \Pr_{k,j}(\tilde{x} \mid x), \tag{5.13}$$

where $\Pr^{\max}_{k,j}(\tilde{x})$ is the highest valued among the constraint variables in $\{\Pr_{k,j}(\tilde{x} \mid x) : x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j}\}$.

We will scale the constraints Eq. (5.12) and (5.13), to avoid the coefficients of the constraint variables from reaching very high numerical values, such that,

$$\begin{aligned} \forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j} : \\ \frac{1}{e^{\epsilon_k}} \Pr_{k,j}(\tilde{x} \mid x) \leq \overset{\max}{\Pr_{k,j}}(\tilde{x}), \end{aligned} \tag{5.14}$$

$$\frac{1}{e^{\epsilon_k}} \overset{\max}{\Pr_{k,j}}(\tilde{x}) \leq \Pr_{k,j}(\tilde{x} \mid x). \tag{5.15}$$

Finally, we bring the constraints Eq. (5.14) and (5.15) into normal form by moving the terms with constraint variables to the left of the inequality sign,

$$\forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j} :$$
$$\frac{1}{e^{\epsilon_k}} \Pr_{k,j}(\tilde{x} \mid x) - \max_{k,j} \Pr(\tilde{x}) \leq 0, \tag{5.16}$$
$$\frac{1}{e^{\epsilon_k}} \max_{k,j} \Pr(\tilde{x}) - \Pr_{k,j}(\tilde{x} \mid x) \leq 0. \tag{5.17}$$

Eq. (5.32) and (5.33) are the first two *linear inequality constraints* in their normal form, which preserves ϵ-differential privacy.

5.4.3.2 Bias minimization constraints

Classical approaches to differential privacy apply zero-mean additive noise. In our approach, this is infeasible. However, we can require a more relaxed property of bias minimization. We achieve this by using the aforementioned constraint variable representing bias $b_{k,j,x}$, and the ultimate goal will be to find an optimum solution for the whole problem that also minimizes the bias variable. We define this as follows,

$$\forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j} : \sum_{\tilde{x} \in \tilde{\chi}_{k,j}} \tilde{x} \Pr_{k,j}(\tilde{x} \mid x) \leq x + b_{k,j,x}. \tag{5.18}$$

We can scale the constraint Eq. (5.18) by the maximum value of $|\tilde{x}|$,

$$\forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j} : \frac{1}{\max_{\tilde{x} \in \tilde{\chi}_{k,j}} |\tilde{x}|} \left(\sum_{\tilde{x} \in \tilde{\chi}_{k,j}} \tilde{x} \Pr_{k,j}(\tilde{x} \mid x) \right) \leq \frac{x + b_{k,j,x}}{\max_{\tilde{x} \in \tilde{\chi}_{k,j}} |\tilde{x}|}. \tag{5.19}$$

Finally, we bring the constraints Eq. (5.19) into normal form by moving the terms with constraint variables to the left of the inequality sign,

$$\forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j} : \frac{1}{\max_{\tilde{x} \in \tilde{\chi}_{k,j}} |\tilde{x}|} \left(\sum_{\tilde{x} \in \tilde{\chi}_{k,j}} \tilde{x} \Pr_{k,j}(\tilde{x} \mid x) - b_{k,j,x} \right) \leq \frac{x}{\max_{\tilde{x} \in \tilde{\chi}_{k,j}} |\tilde{x}|}. \tag{5.20}$$

Eq. (5.20) is a *linear inequality constraint* in its normal form, aiming at minimizing bias.

5.4.3.3 Implicit constraints

We know that the probabilities of a probability mass function (PMF) must always sum up to 1. So, we define the implicit constraint for probabilities of a PMF as,

$$\forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j} : \sum_{\tilde{x} \in \tilde{\chi}_{k,j}} \Pr_{k,j}(\tilde{x} \mid x) = 1. \tag{5.21}$$

Eq. (5.21) is a *linear equality constraint* in its normal form.

Additionally, we also know that each of the probabilities of a probability mass function (PMF) is (1) greater or equal to 0, and (2) less or equal to 1. Now, the first fact always holds if Eq. (5.21) is true. So, we define the implicit constraint for individual probabilities of a PMF as,

$$\forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j} : \Pr_{k,j}(\tilde{x} \mid x) \geq 0. \tag{5.22}$$

We bring the constraint Eq. (5.22) into normal form by moving the terms with constraint variables to the left of the inequality (less or equal) sign,

$$\forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j} : -\Pr_{k,j}(\tilde{x} \mid x) \leq 0. \tag{5.23}$$

Eq. (5.23) is a *linear inequality constraint* in its normal form.

We have introduced a constraint variable $\Pr^{\max}_{k,j}(\tilde{x})$, and that, by definition, must be greater or equal to individual probabilities in $\{\Pr_{k,j}(\tilde{x} \mid x) : x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j}\}$. So, we define another implicit constraint for that,

$$\forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j} : \Pr_{k,j}(\tilde{x} \mid x) \leq \overset{\max}{\Pr_{k,j}}(\tilde{x}). \tag{5.24}$$

We bring the constraint Eq. (5.24) into normal form by moving the terms with constraint variables to the left of the inequality (less or equal) sign,

$$\forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j} : \Pr_{k,j}(\tilde{x} \mid x) - \overset{\max}{\Pr_{k,j}}(\tilde{x}) \leq 0. \tag{5.25}$$

Eq. (5.25) is a *linear inequality constraint* in its normal form.

According to probability theory, the constraint variable $\Pr^{\max}_{k,j}(\tilde{x})$ must always be less or equal to 1. So, we define another implicit constraint for that,

$$\forall k \in [l],\ j \in [m_k],\ \tilde{x} \in \tilde{\chi}_{k,j} : \overset{\max}{\Pr_{k,j}}(\tilde{x}) \leq 1. \tag{5.26}$$

Eq. (5.26) is a *linear inequality constraint* in its normal form.

So, we can summarize the final constraint program in 5.4.3.3.

$\forall k \in [l],\ j \in [m_k],\ x \in \chi_{k,j},\ \tilde{x} \in \tilde{\chi}_{k,j},\ t \in [r_j],:$

$$\textbf{minimize :}\ \ \mathcal{L} = \sum_k \mathcal{L}_k^{var} + \mathcal{L}_k^{bias}, \tag{5.27}$$

Variance minimization objective

$$\mathcal{L}_k^{var} = \sum_{j,t,x,\tilde{x}} \rho_{k,j,t}^{var} \Pr_{k,j}(\tilde{x} \mid x) \left(f'_{k,j,t}(\tilde{x}) - f'_{k,j,t}(x)\right)^2, \tag{5.28}$$

$$\rho_{k,j,t}^{var} = 10^{-2}\min(1, \frac{1}{\max_{x,\tilde{x}} \left(f'_{k,j,t}(\tilde{x}) - f'_{k,j,t}(x)\right)^2}), \tag{5.29}$$

Bias minimization objective

$$\mathcal{L}_k^{bias} = \sum_{j,x} \rho_{k,j}^{bias} b_{k,j,x}^2, \tag{5.30}$$

$$\rho_{k,j}^{bias} = 10^{-2} \frac{1}{\max_{\tilde{x}} \tilde{x}^2}, \tag{5.31}$$

subject to :

Differential privacy constraints

$$\frac{1}{e^{\epsilon_k}} \Pr_{k,j}(\tilde{x} \mid x) - \overset{\max}{\Pr_{k,j}}(\tilde{x}) \leq 0, \tag{5.32}$$

$$\frac{1}{e^{\epsilon_k}} \overset{\max}{\Pr_{k,j}}(\tilde{x}) - \Pr_{k,j}(\tilde{x} \mid x) \leq 0, \tag{5.33}$$

Bias minimization constraint

$$\frac{1}{\max_{\tilde{x}} |\tilde{x}|} \left(\sum_{\tilde{x}} \tilde{x} \Pr_{k,j}(\tilde{x} \mid x) - b_{k,j,x} \right) \leq \frac{x}{\max_{\tilde{x}} |\tilde{x}|}, \tag{5.34}$$

Implicit probability constraints

$$\sum_{\tilde{x}} \Pr_{k,j}(\tilde{x} \mid x) = 1, \tag{5.35}$$

$$- \Pr_{k,j}(\tilde{x} \mid x) \leq 0, \tag{5.36}$$

$$\Pr_{k,j}(\tilde{x} \mid x) - \overset{\max}{\Pr_{k,j}}(\tilde{x}) \leq 0, \tag{5.37}$$

$$\overset{\max}{\Pr_{k,j}}(\tilde{x}) \leq 1. \tag{5.38}$$

5.5 Implementation and experiments

In this section, we describe our implementation of the tailored noise mechanism and the corresponding constraint optimization problem outlined in Section 5.4.3.3. First, we describe the datasets on which we trained our model in Section 5.5.1, then we give brief implementation information in Section 5.5.2. Finally, we provide a detailed description of our experimental setup in Section 5.5.3.

5.5.1 Dataset description

Here, we describe the different synthetic datasets, ds_0, ds_{1a}, ds_{1b}, ds_{2a}, ds_{2b}, ds_{3a}, ds_{3b}, and an augmented version of the small real dataset called misra1d, for which we run experiments. We specifically selected datasets with non-linear and non-differentiable features, as well as datasets involving probabilities and log-likelihood.

- **ds0**: In this dataset, the target is defined by $y_i = \theta_0 + \theta_1 a_i + \eta_i^{reg}$ for every $i \in [n]$.
  - Here a_i is an attribute whose value is independently drawn from a uniform distribution $U(\epsilon^*, 1)$.
  - Here, ϵ^* is the shortest distance for the value of any attribute allowed to reach from a non-continuity or point of singularity at the time of transformation into features. Ideally, this value is set to $\epsilon^* = 10^{-2}$.
  - The regression noise η_i^{reg} is independently drawn from $\mathcal{N}(0, 1)$. We assume the true regression parameters, $\theta_0 = \theta_1 = 1$.

  Our goal is to compute the estimated regression parameters $\hat{\theta}_0$ and $\hat{\theta}_1$ which predict our target function $y_i^{pr} = \hat{\theta}_0 + \hat{\theta}_1 a_i$.

- **ds1a**: In this dataset, the target is defined by $y_i = \theta_0 + \theta_1 \log(a_i) + \eta_i^{reg}$ for every $i \in [n]$.
  - Here a_i is an attribute whose value is independently drawn from a uniform distribution $U(\epsilon^*, 1)$.
  - Here, ϵ^* is the shortest distance for the value of any attribute allowed to reach from a non-continuity or point of singularity at the time of transformation into features. Ideally, this value is set to $\epsilon^* = 10^{-2}$.
  - The regression noise η_i^{reg} is independently drawn from $\mathcal{N}(0, 1)$. We assume the true regression parameters, $\theta_0 = \theta_1 = 1$.

  Our goal is to compute the estimated regression parameters $\hat{\theta}_0$ and $\hat{\theta}_1$ which predict our target function $y_i^{pr} = \hat{\theta}_0 + \hat{\theta}_1 \log(a_i)$.

- **ds1b**: In this dataset, the target is defined by $y_i = \theta_0 + \theta_1 a_i + \theta_2 \log(a_i) + \eta_i^{reg}$ for every $i \in [n]$.

- Here a_i is an attribute whose value is independently drawn from a uniform distribution $U(\epsilon^*, 1)$.
- Here, ϵ^* is the shortest distance for the value of any attribute allowed to reach from a non-continuity or point of singularity at the time of transformation into features. Ideally, this value is set to $\epsilon^* = 10^{-2}$.
- The regression noise η_i^{reg} is independently drawn from $\mathcal{N}(0, 1)$. We assume the true regression parameters, $\theta_0 = \theta_1 = \theta_2 = 1$.

Our goal is to compute the estimated regression parameters $\hat{\theta}_0$, $\hat{\theta}_1$, and $\hat{\theta}_2$ which predict our target function $y_i^{pr} = \hat{\theta}_0 + \hat{\theta}_1 a_i + \hat{\theta}_2 \log(a_i)$.

- **ds2a**: In this dataset, the target is defined by $y_i = \theta_0 + \theta_1 \frac{1}{a_i} + \eta_i^{reg}$ for every $i \in [n]$.
  - Here a_i is an attribute whose value is independently drawn from a uniform distribution $U(\epsilon^*, 1)$.
  - Here, ϵ^* is the shortest distance for the value of any attribute allowed to reach from a non-continuity or point of singularity at the time of transformation into features. Ideally, this value is set to $\epsilon^* = 10^{-2}$.
  - The regression noise η_i^{reg} is independently drawn from $\mathcal{N}(0, 1)$. We assume the true regression parameters, $\theta_0 = \theta_1 = 1$.

  Our goal is to compute the estimated regression parameters $\hat{\theta}_0$ and $\hat{\theta}_1$ which predict our target function $y_i^{pr} = \hat{\theta}_0 + \hat{\theta}_1 \frac{1}{a_i}$.

- **ds2b**: In this dataset, the target is defined by $y_i = \theta_0 + \theta_1 a_i + \theta_2 \frac{1}{a_i} + \eta_i^{reg}$ for every $i \in [n]$.
  - Here a_i is an attribute whose value is independently drawn from a uniform distribution $U(\epsilon^*, 1)$.
  - Here, ϵ^* is the shortest distance for the value of any attribute allowed to reach from a non-continuity or point of singularity at the time of transformation into features. Ideally, this value is set to $\epsilon^* = 10^{-2}$.
  - The regression noise η_i^{reg} is independently drawn from $\mathcal{N}(0, 1)$. We assume the true regression parameters, $\theta_0 = \theta_1 = \theta_2 = 1$.

  Our goal is to compute the estimated regression parameters $\hat{\theta}_0$, $\hat{\theta}_1$, and $\hat{\theta}_2$ which predict our target function $y_i^{pr} = \hat{\theta}_0 + \hat{\theta}_1 a_i + \hat{\theta}_2 \frac{1}{a_i}$.

- **ds3a**: In this dataset, the target is defined by $y_i = \theta_0 + \theta_1 \tan(a_i) + \eta_i^{reg}$ for every $i \in [n]$.
  - Here a_i is an attribute whose value is independently drawn from a uniform distribution $U(\epsilon^*, 1)$.
  - Here, ϵ^* is the shortest distance for the value of any attribute allowed to reach from a non-continuity or point of singularity at the time of transformation into features. Ideally, this value is set to $\epsilon^* = 10^{-2}$.

– The regression noise η_i^{reg} is independently drawn from $\mathcal{N}(0,1)$. We assume the true regression parameters, $\theta_0 = \theta_1 = 1$.

Our goal is to compute the estimated regression parameters $\hat{\theta}_0$ and $\hat{\theta}_1$ which predict our target function $y_i^{pr} = \hat{\theta}_0 + \hat{\theta}_1 \tan(a_i)$.

- **ds3b**: In this dataset, the target is defined by $y_i = \theta_0 + \theta_1 a_i + \theta_2 \tan(a_i) + \eta_i^{reg}$ for every $i \in [n]$.
  - Here a_i is an attribute whose value is independently drawn from a uniform distribution $U(\epsilon^*, 1)$.
  - Here, ϵ^* is the shortest distance for the value of any attribute allowed to reach from a non-continuity or point of singularity at the time of transformation into features. Ideally, this value is set to $\epsilon^* = 10^{-2}$.
  - The regression noise η_i^{reg} is independently drawn from $\mathcal{N}(0,1)$. We assume the true regression parameters, $\theta_0 = \theta_1 = \theta_2 = 1$.

  Our goal is to compute the estimated regression parameters $\hat{\theta}_0$, $\hat{\theta}_1$, and $\hat{\theta}_2$ which predict our target function $y_i^{pr} = \hat{\theta}_0 + \hat{\theta}_1 a_i + \hat{\theta}_2 \tan(a_i)$.

- **misra1d**: In this dataset, the target is defined by

$$y_i = \frac{\theta_1^{\mu} \theta_2^{\mu} a_i}{1 + \theta_2^{\mu} a_i} + \eta_i^{reg}, \text{ for every } i \in [n]. \tag{5.39}$$

  - Here, attribute $a_i = 10^3 \times a_i'$, and a_i' is independently drawn from a uniform distribution $U(0,1)$.
  - Regression parameters are valued at $\hat{\theta}_1^{\mu} = 4.37 \times 10^2$ and $\hat{\theta}_2^{\mu} = 3.02 \times 10^{-4}$.

  To convert Eq. (5.39) into linear regression settings, we assume the following approximation,

$$\begin{aligned} y_i^{pr} = & \frac{\hat{\theta}_1^{\mu} \hat{\theta}_2^{\mu} a_i}{1 + \hat{\theta}_2^{\mu} a_i} \\ = & \hat{\theta}_1^{\mu} \left(1 - \frac{1}{1 + \hat{\theta}_2^{\mu} a_i} \right) \\ \approx & \hat{\theta}_1^{\mu} \left(1 - \sum_{j \in [2]} \frac{\alpha_j}{1 + c_j a_i} \right) \\ = & \hat{\theta}_0 + \frac{\hat{\theta}_1}{1 + c_1 a_i} + \frac{\hat{\theta}_2}{1 + c_2 a_i}. \end{aligned}$$

  Our goal is to compute the estimated regression parameters $\hat{\theta}_0$, $\hat{\theta}_1$, and $\hat{\theta}_2$ which predict our target function

$$y_i^{pr} = \hat{\theta}_0 + \frac{\hat{\theta}_1}{1 + c_1 a_i} + \frac{\hat{\theta}_2}{1 + c_2 a_i}. \tag{5.40}$$

- Here, $c_1 = \hat{\theta}_2^{\mu} + \hat{\theta}_2^{\sigma}$ and $c_2 = \hat{\theta}_2^{\mu} + 1.1 \times \hat{\theta}_2^{\sigma}$.
- The sample standard deviation of $\hat{\theta}_2^{\mu}$ is $\hat{\theta}_2^{\sigma} = 2.93 \times 10^{-6}$.
- The regression noise $\eta_i^{reg} = 6.85 \times 10^{-2} \eta_i^{reg'}$ and $\eta_i^{reg'}$ is independently drawn from $\mathcal{N}(0, 1)$. We assume the true regression parameters, $\theta_0 = \theta_1 = \theta_2 = 1$.
- The values of parameters $\hat{\theta}_1^{\mu}$, $\hat{\theta}_2^{\mu}$, $\hat{\theta}_2^{\sigma}$, and the coefficient of η_i^{reg} are taken from the official website.
- Also, the coefficient 10^3 in the expression of a_i is approximately the order of magnitude of the largest difference between the two target values in the original dataset.

5.5.2 Implementation in CVXOPT

We utilized the cvxopt package discussed in Section 2.9 as a tool to specify the constraint variables, construct the constraints, and the objective function. We also used CVXOPT's default constraint solver.

In Section 5.4.3.3, the constraint optimization program has a quadratic objective function and linear constraints, which are scaled to avoid numerical irregularities. CVXOPT's default constraint solver is based on *Cholesky decomposition* as it is efficient in our implementation. The CVXOPT solver based on the *LDL decomposition* is better suited for unscalable convex programs with convex constraints and/or a convex objective function.

5.5.3 Experimental setup

Next, we will briefly discuss the regression setup in the experiments 5.5.3.1, privacy guarantees associated with TNM 5.5.3.2, hyperparameter setup of the experiments 5.5.3.3, and experimental method variation setup accommodating all the datasets 5.5.3.4.

5.5.3.1 Regression setup

We created a multivariate linear regression model with $k+1$ regression parameters and a scalar target value for our experiments. We expressed the relationship between one target value and its features as follows:

For each i where $i \in [n]$:

$$y_i = \theta_0 + \theta_1 x_{i,1} + ... + \theta_k x_{i,k} + \xi_i. \tag{5.41}$$

Here,

- $x_{i,1}, x_{i,2}, \ldots, x_{i,k}$ are (sensitive) features,
- $\theta_0, \theta_1, \ldots, \theta_k \in \mathbb{R}$ are regression parameters,
- ξ_i is regression noise that is an independent observation of $\mathcal{N}(0, \sigma^2)$,
- σ is the standard deviation of the regression noise.

5.5.3.2 Privacy guarantees

- **Tailored noise mechanism (TNM).** For TNM, we assume that the lowest and greatest values of each sensitive feature are known. Equal distance discretization 5.4.1.1 allows us to keep each agent's sensitive features hidden. The central curator can execute equal distance discretization without knowing any of the sensitive features. A word of caution: for equal frequency discretization, the central curator would need to be aware of each sensitive feature. Furthermore, based on the differential privacy constraints of TNM (Eq. 5.32), local privatization of sensitive characteristics is ϵ-differentially private. Each agent, therefore, needs to initially discretize its sensitive feature using the same discretization approach as the central curator before privatizing it. We contend that the privacy assurances of TNM based on equal distance discretization and of Laplace are comparatively equivalent under our same assumptions. However, the discretization in TNM causes a loss of utility due to the amount of information loss.

- **Classical mechanism (Laplace).** For classical differential privacy, we consider the Laplace mechanism 2.7.4.1. Furthermore, we presume that the minimum and maximum values of each sensitive feature are known. This allows us to compute the l1 sensitivity of Laplace with a guarantee of ϵ-differential privacy.

- **General assumptions**:
  - We split the total privacy budget among sensitive features evenly, as advised in [43].
  - Although a target value y_i (Eq. 5.41) is a linear combination of sensitive features, the exact values of the regression parameters and the regression noise are unknown, therefore the sensitive feature values cannot be derived. Hence, they are non-sensitive [111].

Finally, based on the mechanism of supervised learning and least square model fitting [130], we compute the $\hat{\theta}$ parameter estimates as follows:

$$\hat{\theta} = \left(\frac{1}{n} X^T X + \lambda I_{m+1}\right)^{-1} \frac{1}{n} X^T y, \tag{5.42}$$

where λ is the regularization parameter.

5.5.3.3 Hyperparameter setup

For our experiment, there are a few hyperparameters to consider. Keeping reproducibility in mind, we have kept them constant throughout all of our experimental results.

- **Experiment repetition number and number of agents.** We fixed the experiment repetition number to 2^7, and for each repetition, we re-privatized our dataset. In all of our experiments, our number of agents was fixed at 10^4.
- **Tolerance level and max iteration of constraint solver.** The threshold of a tolerance level for primal infeasibility, dual infeasibility, and gap for our constraint solver was set to 10^{-7}. We set the max iteration to 128.
- **Domain discretization.** We set our discretization strategy for our primary experiments 5.6.1 to be of equal distance (width), which has 20 bins for both sensitive and noisy (privatized) feature values. For secondary experiments 5.6.2, we vary our discretization strategy between equal distance (width) and equal frequency. We also vary the number of bins (between 20 and 41) for domain discretization of sensitive and privatized features.
- **CI significance.** We set the significance level of our confidence interval to 0.05.
- **Privacy budget.** Finally, we iterate over a list of ϵ values $\{2^{-1}, 2^0, 2^{1/2}, 2^1, 2^{3/2}, 2^2, 2^{5/2}, 2^3, 2^{7/2}, 2^4, 2^5, 2^6\}$ for evaluating our privacy budget.

5.5.3.4 Experimental method variation setup

All datasets in Section 5.5.1 were defined by keeping in mind one of the following experiment setups.

- **Setup 1.** In this case, the dataset was constructed in a way that one feature is computed from one attribute. Datasets {**ds1a**, **ds2a**, **ds3a**, **misra1d**} are examples of such settings.
- **Setup 2.** In this case, the dataset was constructed in a way that two or more features are computed from one attribute. Datasets {**ds1b**, **ds2b**, **ds3b**} are examples of such settings.

In both experiments, we compare four models: **baseline** (non-privatized features), **classic DP** (Laplace-privatized features), **tns** $\circ$ **dp** (where attributes are privatized first, and then the privatized attributes are transformed into features), and **dp** $\circ$ **tns** (where attributes are transformed into features first, and then the features are privatized).

In the next section, we will see the results of our experiments, but before that, let us take a look at the evaluation criteria.

Evaluation criteria. We picked **mean squared error** (MSE) as our assessment criterion because we are doing a least square fit for all permutations of our model. We divide the MSE between actual and predicted target values by the standard deviation of true target values $\sigma(y)^2$, as described by [67]. We

use 10-fold cross-validation. The regularization parameter is set to 10^3.

Hardware settings. All of the experiments we are going to present were performed on a Macbook Pro (M2 Max) machine with a 38-core GPU and 16-core neural engine resulting in 64GB of unified memory. The maximum runtime for a particular dataset was approximately 24 minutes.

5.6 Results

In this section, we will present our findings from the primary and secondary experiments and their interpretations in Sections 5.6.1, 5.6.2, followed by a result summary in Section 5.6.3.

5.6.1 Primary result interpretation

In this section, the results are interpreted for experiments where for all datasets, discretization strategy was 'equal width', and the number of discretization bins, for both sensitive and privatized feature domains, were kept constant 5.5.3.3.

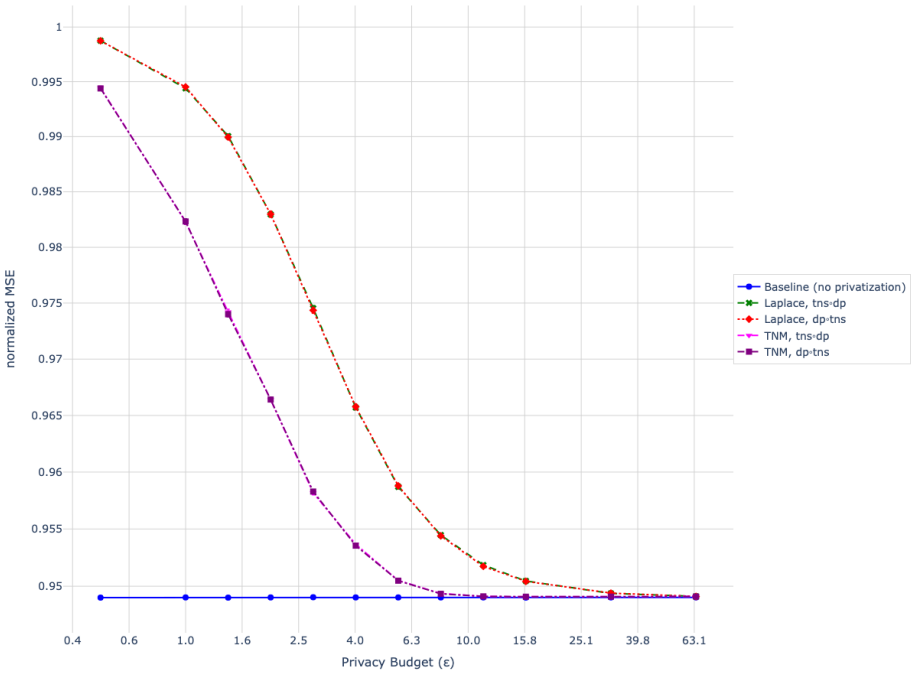


Figure 5.1: Experiment 1 on ds0

Interpretation:

Fig. 5.1 shows that the overall performance of TNM is much better than the classical DP (Laplace) mechanism. **TNM** reaches the same performance as the baseline model with a lower value of ϵ compared to **Laplace**. For both **TNM** and **Laplace**, the performance of **dp** $\circ$ **tns** and **tns** $\circ$ **dp** are very similar.

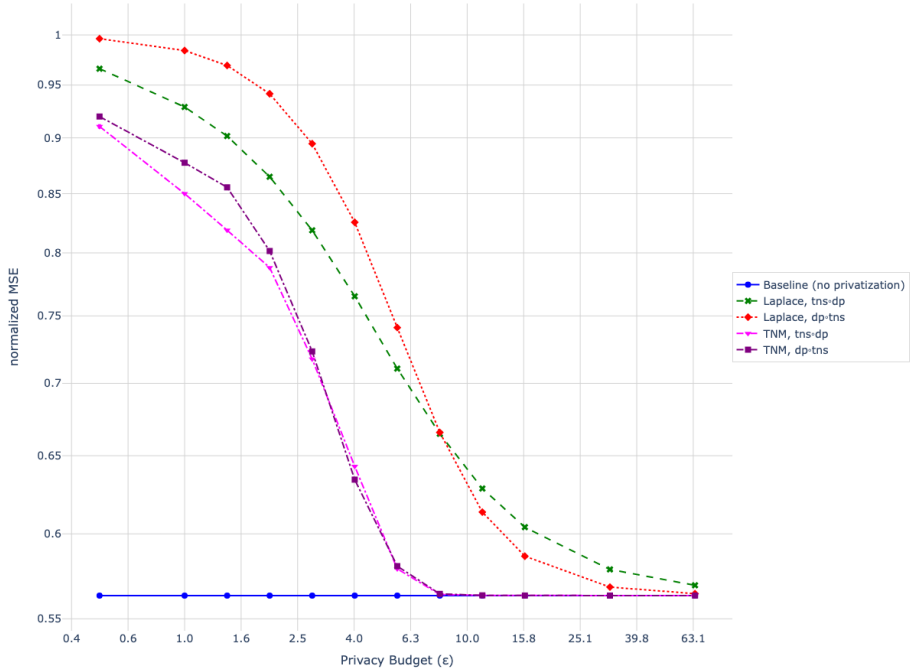


Figure 5.2: Experiment 1 on ds1a

Interpretation:

Fig. 5.2 shows that the overall performance of TNM is much better than the classical DP (Laplace) mechanism. **TNM** reaches the same performance as the baseline model with a lower value of ϵ compared to **Laplace**. The performance between two types of TNM models (**dp** $\circ$ **tns** and **tns** $\circ$ **dp**) is more or less similar, and the curves converge to each other for higher ϵ. For Laplace, **tns** $\circ$ **dp** performs better for lower ϵ values, whereas **dp** $\circ$ **tns** performs slightly better for higher ϵ values.

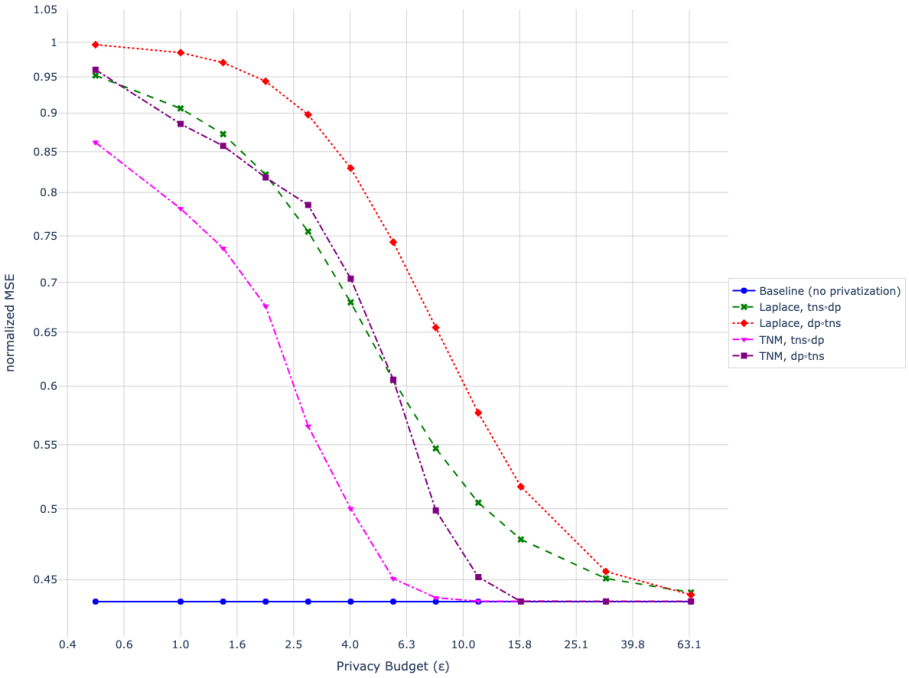


Figure 5.3: Experiment 2 on ds1b

Interpretation:

Fig. 5.3 again shows that the overall performance of TNM is much better than the classical DP (Laplace) mechanism for most of the values of ϵ. **TNM** reaches the same performance as the baseline model with a lower value of ϵ compared to **Laplace**. The performance of model **tns** $\circ$ **dp** is far better than model **dp** $\circ$ **tns**, for both Laplace and TNM, with **TNM** model **tns** $\circ$ **dp** outperforming the rest for lower values of ϵ.

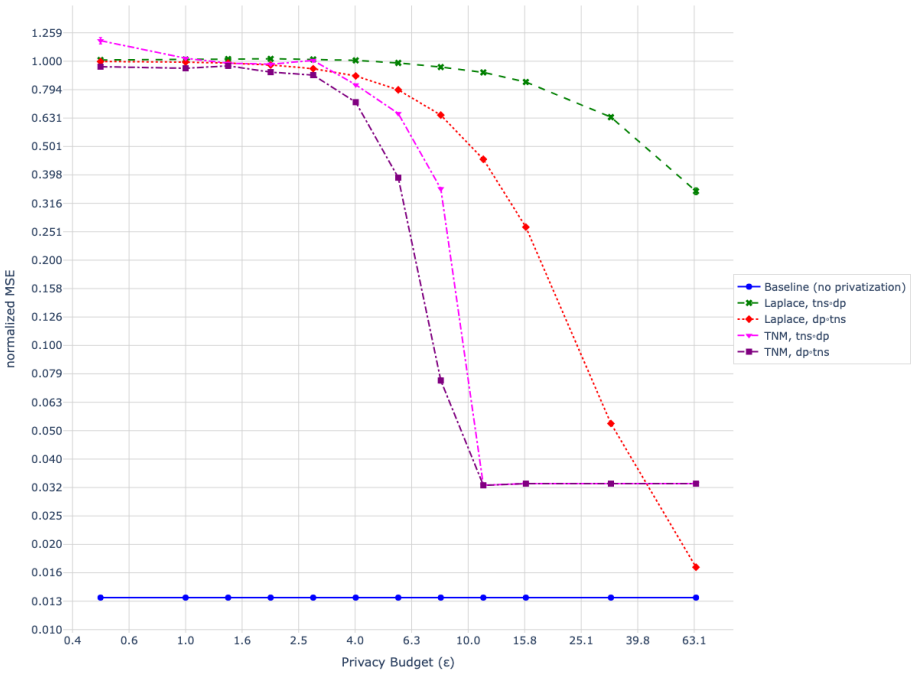


Figure 5.4: Experiment 1 on ds2a

Interpretation:

Fig. 5.4 shows that for **Laplace**, **dp** $\circ$ **tns** outperforms **tns** $\circ$ **dp** because of a higher probability of privatized characteristics ending up in an interval close to the singularity when the transformation is reciprocal. Similarly, **dp** $\circ$ **tns** performs better than **tns** $\circ$ **dp** in **TNM** for some lower values of ϵ.

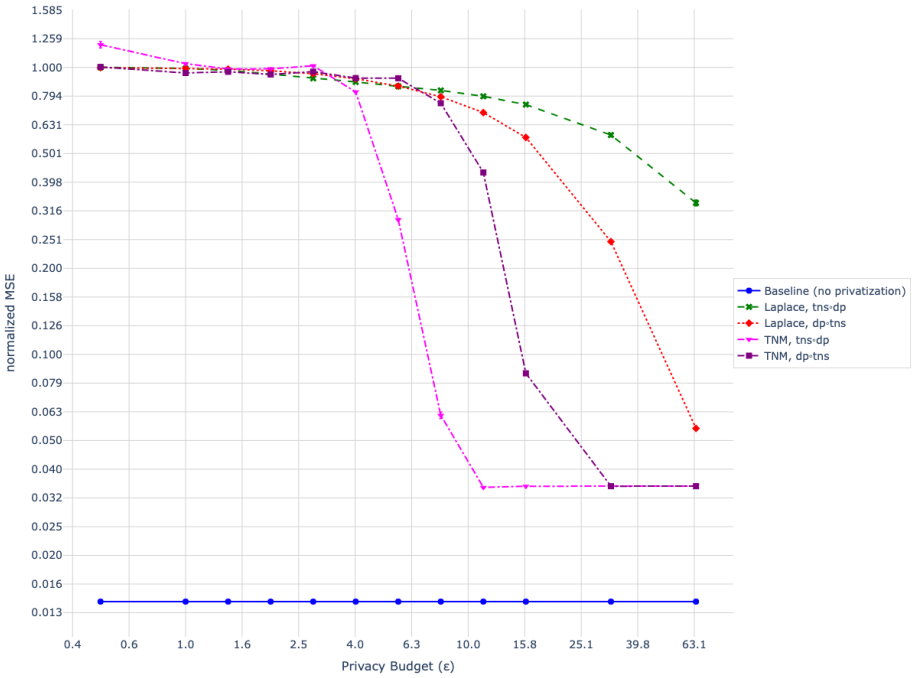


Figure 5.5: Experiment 2 on ds2b

Interpretation:

Fig. 5.5 shows very similar trends in curves as Fig. 5.4. It shows that for **Laplace**, **dp** $\circ$ **tns** outperforms **tns** $\circ$ **dp** because of a higher probability of privatized characteristics ending up in an interval close to the singularity when the transformation is reciprocal. However, similarly to Fig. 5.3, here also we see a big improvement in performance for **TNM** model **tns** $\circ$ **dp** for high ϵ values.

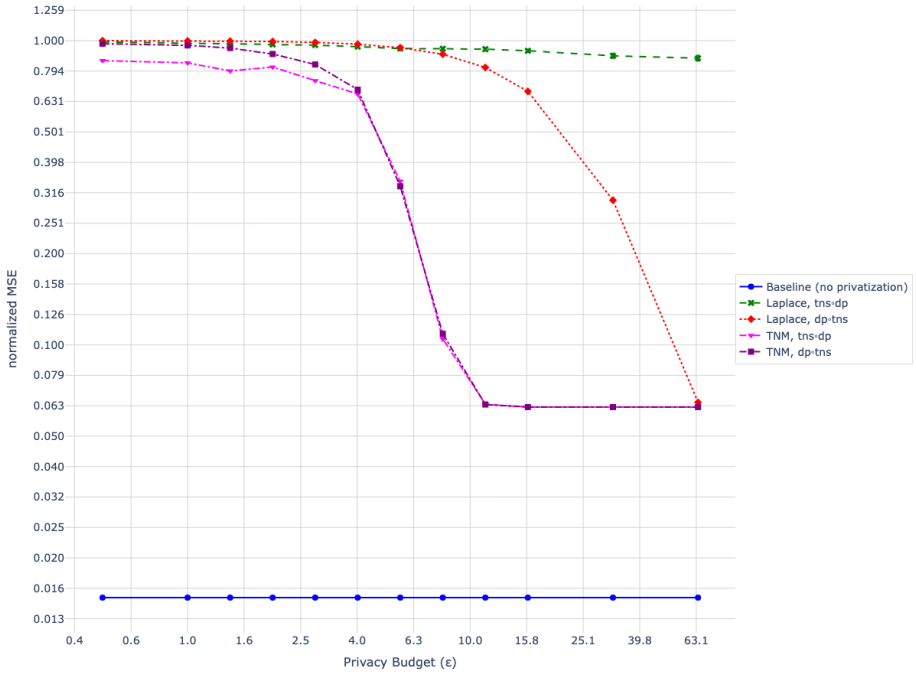


Figure 5.6: Experiment 1 on ds3a

Interpretation:

Fig. 5.6 shows that upon using a tangent transformation, it is highly likely that singularity will be reached quickly. For **Laplace**, **dp** $\circ$ **tns** outperforms **tns** $\circ$ **dp**. For **TNM**, performances are more or less similar, with **tns** $\circ$ **dp** outperforming **dp** $\circ$ **tns** by a small margin for lower values of ϵ.

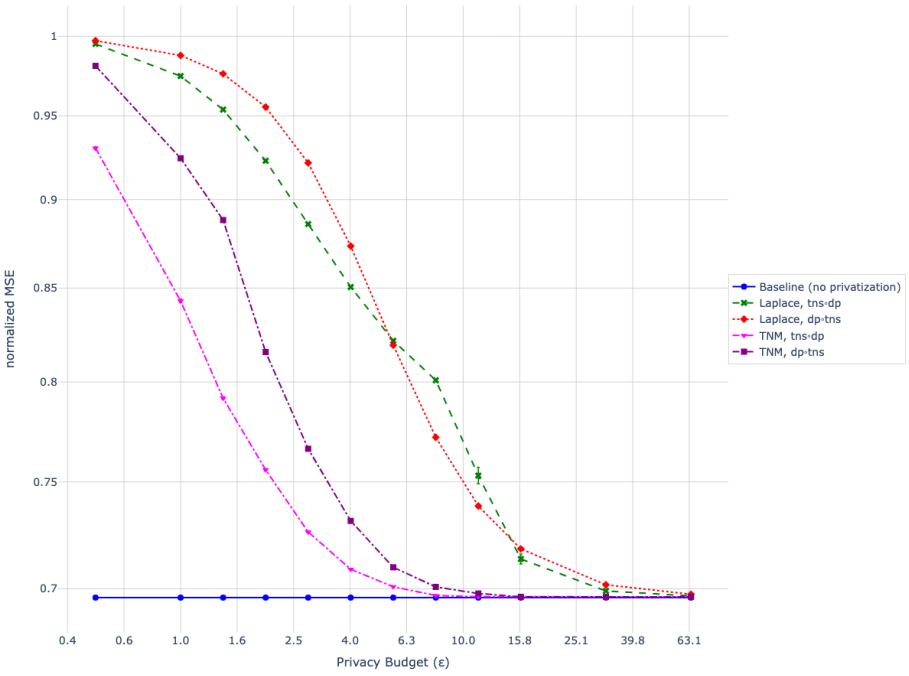


Figure 5.7: Experiment 2 on ds3b

Interpretation:

Continuing on the trends of results we have been getting for experiment setup 2, Fig. 5.7 shows that for **Laplace**, in most of the privacy budgets, **tns** $\circ$ **dp** performs better than **dp** $\circ$ **tns**. For **TNM**, **tns** $\circ$ **dp** is outperforming **dp** $\circ$ **tns** by a lot for lower values of ϵ.

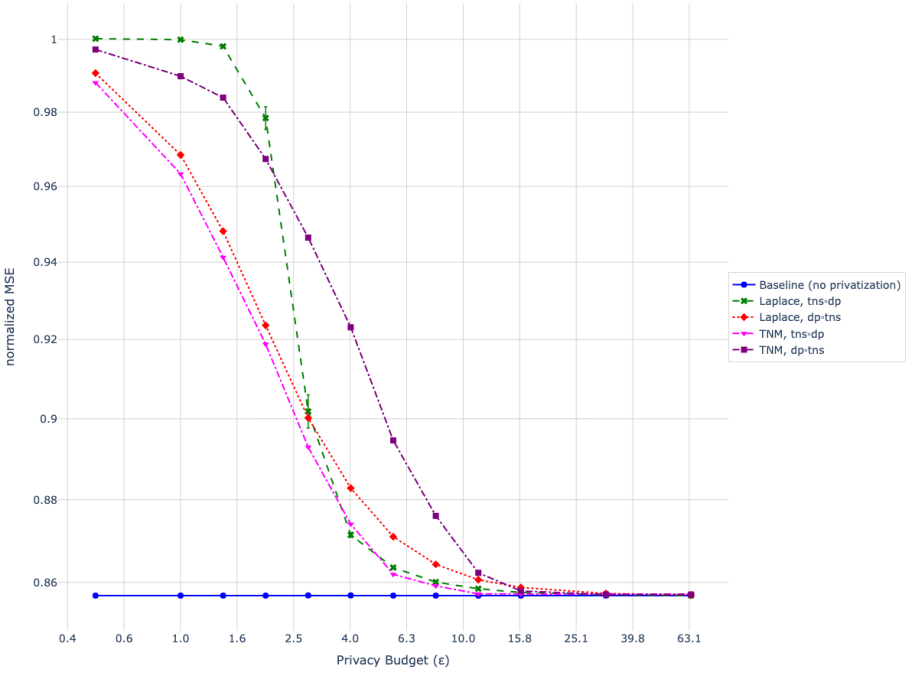


Figure 5.8: Experiment 1 on misra1d

Interpretation:

In Fig. 5.8, **TNM** outperforms **Laplace** solely for model **tns** $\circ$ **dp** due to the splitting of the budget for privacy in the other model **dp** $\circ$ **tns** and the changes in the target function in Eq. (5.40) that results in being reasonably distant from their singularities. In **Laplace**, model **tns** $\circ$ **dp** outperforms model **dp** $\circ$ **tns** for higher ϵ values but underperforms with smaller ϵ values. The model **tns** $\circ$ **dp** outperforms model **dp** $\circ$ **tns** in **TNM** because the privacy budget is not split in model **tns** $\circ$ **dp**.

5.6.2 Secondary result interpretation

In this section, we present the result interpretations for our secondary experiments where we vary the discretization strategy between 'equal width' and 'equal frequency'. The number of discretization bins, for both sensitive and privatized feature domains were also varied to represent fine and coarse discretization.

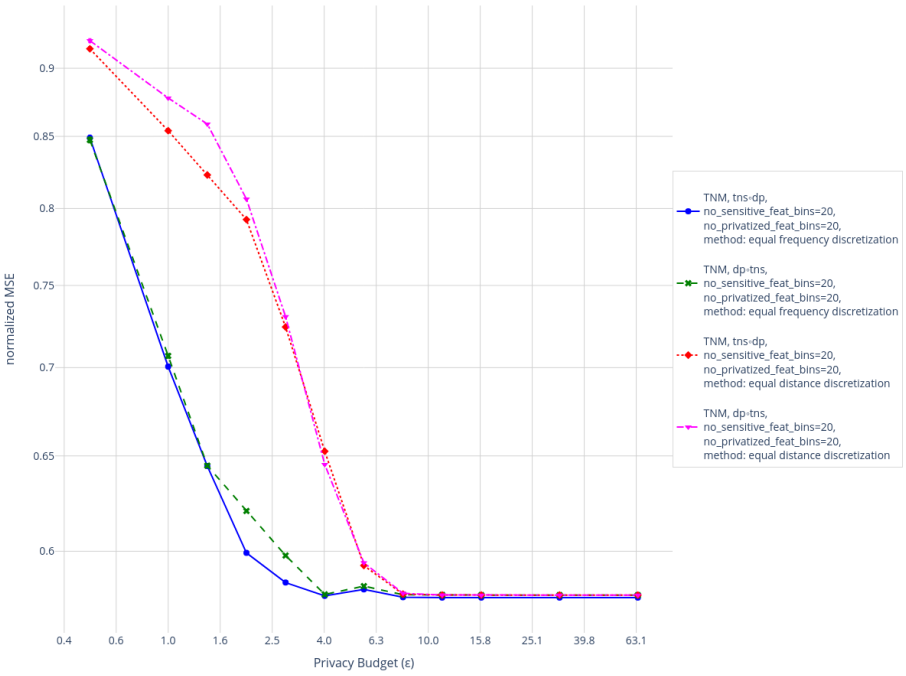


Figure 5.9: Experiment for comparison between '**equal distance**' and '**equal frequency**' discretization strategies on ds1a

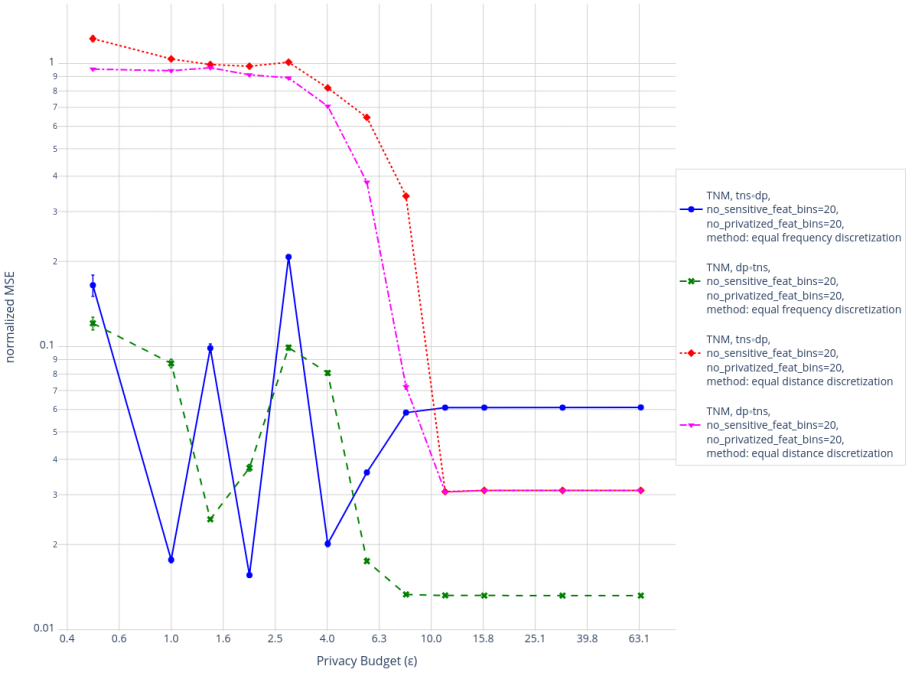


Figure 5.10: Experiment for comparison between '**equal distance**' and '**equal frequency**' discretization strategies on ds2a

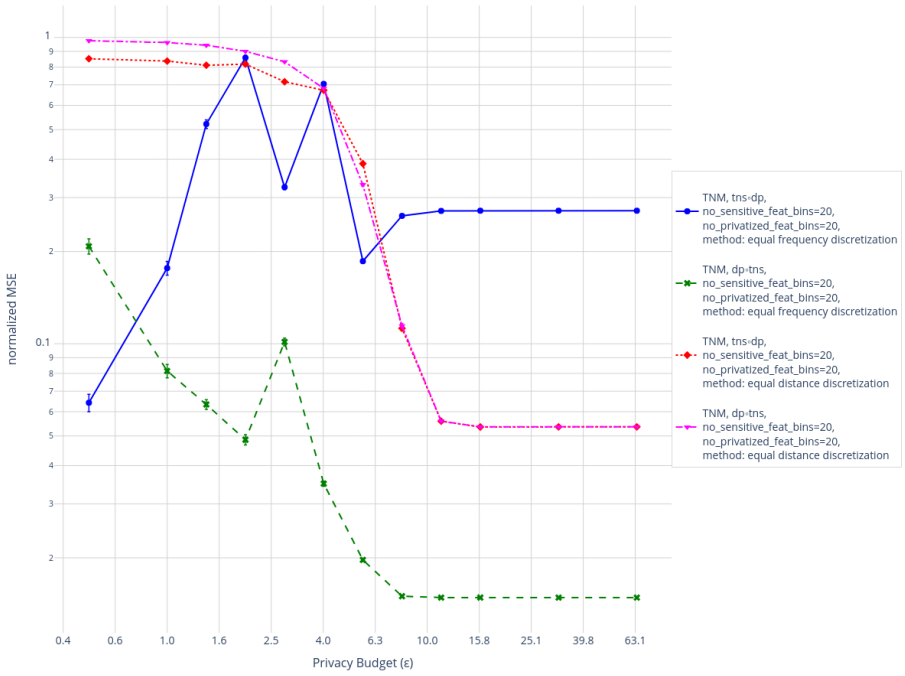


Figure 5.11: Experiment for comparison between '**equal distance**' and '**equal frequency**' discretization strategies on ds3a

Interpretation:

In Fig. 5.9, 5.10, 5.11, we see a general tendency, that a **equal frequency** discretization strategy mostly results in a better model, in terms of utility, over a **equal distance** strategy, for both **dp** $\circ$ **tns** and **tns** $\circ$ **dp** models. However, we see that in experiments for ds2a and ds3a (Fig. 5.10, 5.11), a gradual increase in privacy budget (higher values of ϵ), reflects towards loss in utility for a **tns** $\circ$ **dp** model. This is because, even with an equal frequency strategy, the discretization is not precise enough to handle high-magnitude gradients (since both reciprocal and tangential transformations reach singularity rather rapidly). We also observed similar, comparable results for datasets ds1b, ds2b and ds3b.

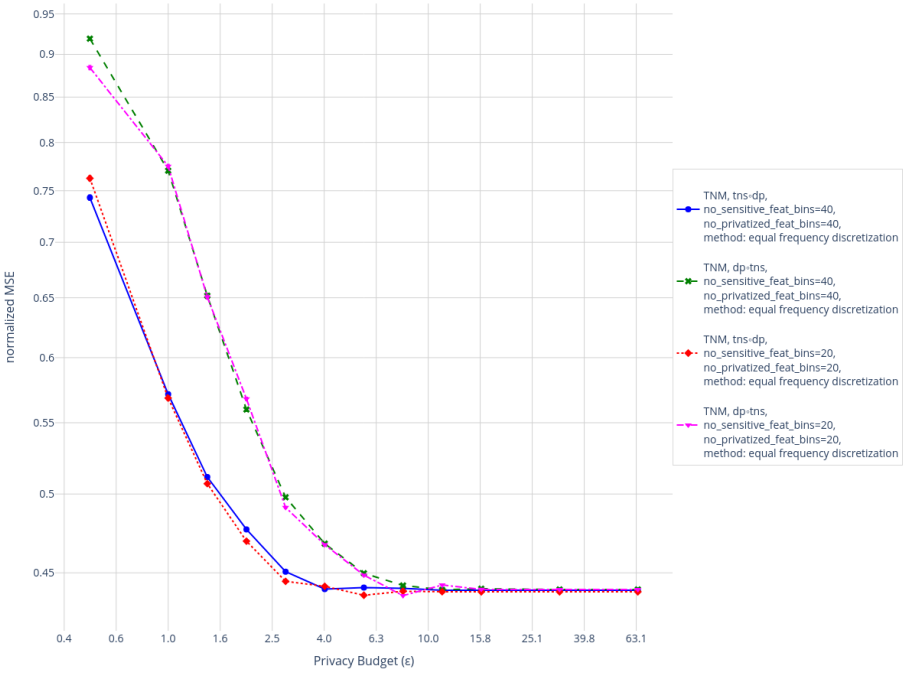


Figure 5.12: Experiment for comparison between '**fine**' and '**coarse**' discretization (**equal frequency**) strategies on the domains of sensitive features for ds1b

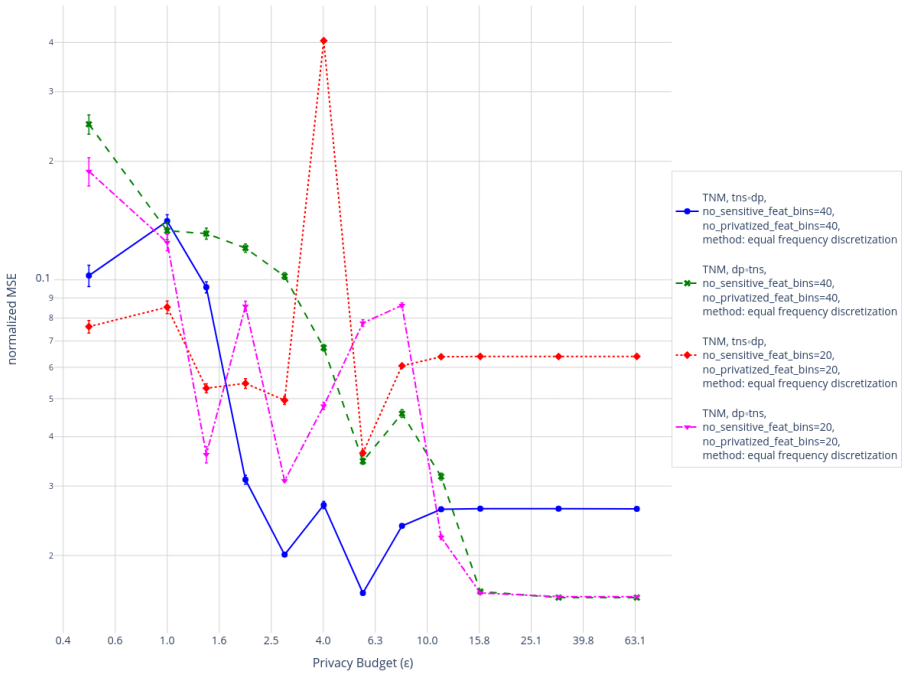


Figure 5.13: Experiment for comparison between '**fine**' and '**coarse**' discretization (**equal frequency**) strategies on the domains of sensitive features for ds2b

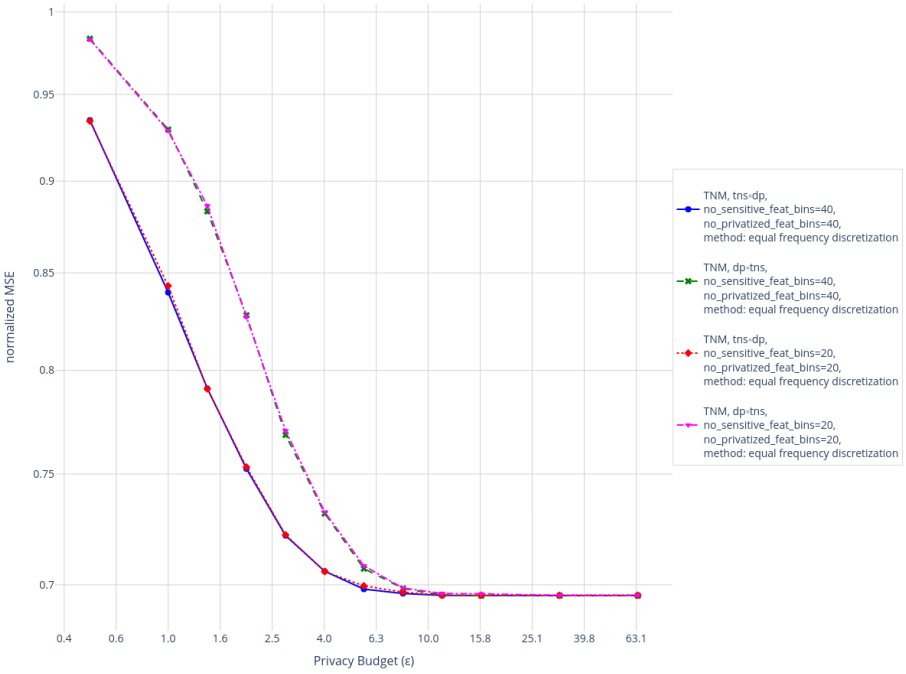


Figure 5.14: Experiment for comparison between '**fine**' and '**coarse**' discretization (**equal frequency**) strategies on the domains of sensitive features for ds3b

Interpretation:

In Fig. 5.12, 5.13, 5.14, we see a general tendency that a fine discretization (**equal frequency**) strategy mostly results in a better model in terms of utility, over a coarse discretization strategy, for the sensitive feature domains, for both **dp** $\circ$ **tns** and **tns** $\circ$ **dp** models. Given that, we would want to emphasize that 40 bins with an equal frequency discretization approach are insufficient to see a steady change (increase in utility) with a higher privacy budget (higher values of ϵ).

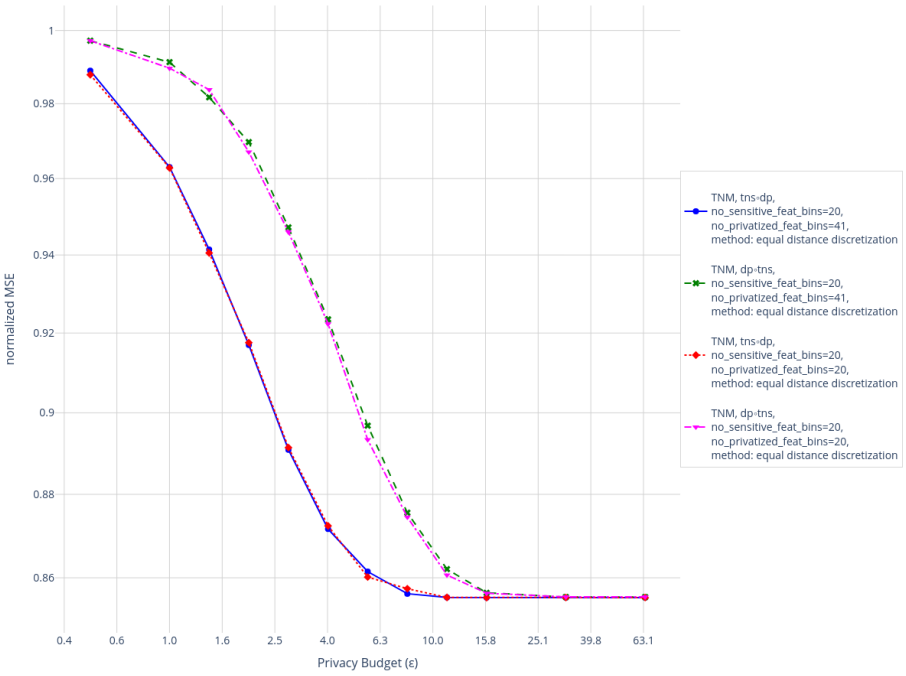


Figure 5.15: Experiment for comparison between '**fine**' and '**coarse**' discretization (**equal distance**) strategies on the domains of privatized features for misra1d

Interpretation:

In Fig. 5.15, we see that a fine discretization (**equal distance**) strategy does not result in any significant utility improvement, over a coarse discretization strategy, for the privatized feature domains, for any given privacy budget. We also ran experiments for **fine vs coarse** for equal frequency discretization strategy for sensitive feature domains instead of privatized features and observed similar results.

5.6.3 Result summary

Throughout our empirical studies, which included the examination of synthetic datasets as well as the augmentation of an authentic dataset with synthetic techniques, we continuously detected a notable tendency. Our investigations have demonstrated that our empirical findings provide insight into an important component of model performance. We see that using the tailored noise

mechanism (TNM) consistently results in a reduction in the mean squared error (MSE) between the true and predicted target values. This reduction is especially noticeable when compared to the performance of the Laplace mechanism.

This is true when the TNM is used to fit a linear regression model using the method of regularized least squares, as stated in Eq. (5.42), and when this model uses U-statistics as a foundational framework. This observation is significant since it holds across numerous feature functions, including the logarithmic, reciprocal, and tangent functions.

On the other hand, we also see that in some cases because of the nature of transformations applied, a risk of getting an outlier feature becomes higher. For example, in the case of the linear regression model applied to the **ds2a** dataset, we found that model **dp** $\circ$ **tns** (in which attributes are transformed into features before the introduction of differential privacy noise) outperformed model **tns** $\circ$ **dp** (in which differential privacy noise is first introduced to attributes before the transformation into features). This phenomenon can be linked to the relatively quick approach of the reciprocal transformation within **ds2a** dataset toward a singularity, which increases the likelihood of the presence of an outlier feature.

Finally, a general trend we have seen is that utility generally stagnates upon a higher privacy budget. One explanation for this behavior could be the fact that the regularization diagonal matrix in Eq. (5.42) becomes numerically inefficient for an inverse operation during the optimization process.

5.7 Future Scope and conclusion

The current scope of work has challenges and boundaries that led to the presence of some limitations in our work. These limitations also lead us to some concrete ideas for future directions.

- We explained that our technique is best suited for discrete privatization functions. To broaden the coverage, the inclusion of continuous privatization functions would be a place to start. This may lead us to explore the shape of the noise distribution to optimize the privatization functions. This involves investigating privatization functions parameterized by the shape of noise, not the estimation of conditional probabilities.

- The current formulation of the TNM is constrained to two regression models: a regression model where the differential privacy noise is added to the sensitive attributes before performing transformations and a regression model where the noise is added to the features after the transformation. In an ideal situation, users should be able to declare regression models themselves. It is possible, but designing the corresponding constraints and the objective function is non-trivial.

- In our current implementation, domain interpolation is not supported, but it might improve performance upon coarser discretization.

To conclude, we can say that in this article, we have presented the tailored noise mechanism (TNM) for privatizing features through the resolution of a constraint optimization problem. The TNM is designed to minimize the bias and variance of noise used for the privatization of the features. We also aim to develop a utility-maximization technique for the privatization of features emerging from the transformation of sensitive attributes, particularly when these transformations exhibit large gradients or singularities. This requirement is achieved using TNM by selecting only informative intervals for the transformation of sensitive attributes into features.

Chapter 6

Future work

We have articulated the research problem and conducted a comprehensive investigation within the scope of this dissertation. We explored a number of directions before focusing on a few that helped us solve the current challenge. We are aware that there are gaps or unresolved concerns in the current literature that can be addressed by future research. Further investigation is necessary, as it has the potential to have an impact on several layers of society, as well as academia and industry.

In this chapter, we outline some of the potential research directions for the future.

6.1 Improve inference

Improving inference is essential for developing more powerful and versatile privacy-preserving systems. So far in this dissertation, we have mostly focused on the language and explored the direction of inference in a few situations. We hope that in future work, we can have improved inference techniques that perform better in reasoning about these privacy requirements.

Once we have all the privacy requirements specified, we can explore how we can generate more complex, constraint problems automatically, and solve them. Also, we can research more advanced optimization techniques to find ways to satisfy all requirements, even if we don't get a convex constraint problem.

6.2 Language extension: Specifying dynamic behavior

In our privacy specification language, we have covered most of the static situations. In the future, we may introduce some auxiliary predicates for specifying a dynamic system for information exchange in a privacy-preserving manner.

In the case of a dynamic system, we can specify **actions** and the corresponding **effects** on the variables (and on the question of who has access to observe which information about the variables). In dynamic problems, for every time step, we can incrementally iterate over the two steps, as discussed in 3.3.2:

- Solving the constraint optimization problem
- Executing the algorithm with the obtained solutions and parameters

A deeper plunge can be taken into designing predicates for the portrayal of such dynamic behavior and extending the language. Next, we very briefly discuss a few of them:

Observe Parties can observe random variables during the processing of data. Whenever someone observes a random variable, either at the point when it gets a value or later when its value is disclosed, the observer comes to know the value. We formalize such observations with the predicate `observe(Obs, RV)`, representing that the person `Obs` observes the random variable `RV`. For instance, `observe(Obs, gender(P))` ← `treats(P, Obs), staffRole(Obs, "doctor").` represents that any doctor `Obs`, who treats a patient `P`, observes and, therefore, learns the patient's gender `gender(P)`. If some `RV` is personal to an entity, i.e., they satisfy the `personal_rv` clause 4.4.3, then by default the party has access to the same, hence they implicitly satisfy the `observe` clause.

Reveal Information can also be revealed (as one of the actions) with a special predicate `reveal`. In that case, the value of a random variable can be revealed to a certain party by someone who knows the secret. This is a special case of the `observe` clause.

Forget In contrast to reveal, some legislation, like the GDPR, provides the right to ask for their personal information to be forgotten. In our language, one can represent this with the clause `forget(Obs, RV)`, representing that the random variable `RV` belonging to the person `Obs` is to be forgotten. For example, if a former patient asks for complete withdrawal of their data from participation in any (or all) medical studies, the medical center will need to forget all their personal information. We cannot formally verify that a pipeline fulfills a `forget` predicate because some random variables can always be stored in physical media outside the pipeline, like human memory.

These are a few examples of predicates that can be included in the declarative language. More exploration in this direction will lead to addressing diverse situations of information exchange and alteration.

6.3 Pufferfish and other privacy frameworks

We assume that the ML algorithms being applied to the dataset here can indeed draw some conclusions about the data entries, and privacy must be specified and

ensured. It is to be noted here that we have thoroughly used differential privacy standards and their privacy parameters to represent, measure, and guarantee privacy in our proposed frameworks. Other privacy mechanisms can be considered that can deal with other aspects of dealing with sensitive information, like Rényi-differential privacy [92].

As a variation of the vanilla differential privacy, the pufferfish privacy [116] metric guarantees epsilon differential privacy for internally correlated data. Sometimes, multiple data entries in the dataset belong to every participant. Pufferfish deals with the internal correlation of individual participant's data in a dataset. In such cases, individuals have full agency to participate or withdraw from a study, and every datapoint associated with that individual is assigned the same. In such cases, the close neighborhood of the data reveals information about the current data under consideration, and hence the correlation might give away some sensitive information. Similarly, our proposed specification and optimization mechanism can be extended to other privacy frameworks in future projects.

6.4 Fairness

Where privacy mostly concerns individuals, there is also fairness property that concerns groups or communities. For example, a correlation between two factors (human races and their average annual income) in a study might show a particular community in a negative light. Fairness allows observing and accepting the existence of such correlations while at the same time avoiding making biased decisions based on such correlations. This can be caused by a biased representation present in the dataset. Fairness ensures not coming to conclusions due to the correlations with variables, based on which we shouldn't discriminate. Fairness promotes making impartial decisions without showing any favoritism to one person or group at the expense of another.

To explain the fairness criteria in privacy and machine learning, let us consider the following example. The German job portal Xing ranks more qualified female candidates lower than less qualified male candidates [85]. Similarly, people with very similar qualifications are ranked far apart. This position bias affects individuals with higher qualifications, like education score and employability, but lower ranks, unfairly. So, the fairness criterion must ensure that no social or ethnic group receives an unfair outcome from any machine learning algorithm. For our current project, fairness could not fit into the scope. But later, we can extend the privacy specification and privacy-parameter optimization into fairness-specification and fairness-parameter optimization.

6.5 Data provenance

Another module worth investigating is how processes control the disclosure (or non-disclosure) of certain sensitive information during an information flow. To

achieve that, we may study data provenance [1] or a derivative of it as a connecting component. This will hopefully enable us to keep track of data from its origin and the changes it goes through over time and action. Here, the question to ask is, "Can a given disclosure possibly reveal a sensitive variable, V?". Only if any information is disclosed that depends on data V, provenance can help to quickly eliminate many cases where the disclosed information is not a function of V and hence cannot reveal V.

We have already mentioned that, as a side-product of our inference, we understand how, by revealing a noisy query response, the privacy of underlying sensitive information can be compromised. However, further in-depth study should be able to step-wise debug and calculate the privacy implications of every action and their effects on the data. This will increase the visibility of how privacy is impacted by the transformation and disclosure of data from the source to the destination. In short, the data flow can be monitored through an operational system with a more in-depth understanding of individual changes and their traces.

6.6 Applications

In many real-world scenarios appearing in various application domains, our specification language and the optimization framework can prove to be very useful. For example, the association of an individual user's internet search history and recommendation of advertising without compromising the user's identity, securing the privacy of cloud-stored sensitive data [95] like patients' vital statistics, medical history, summaries, and predictions made by medical experts, can be such fields where this language can be very productive. Our approach can also be useful in the field of DNA analysis, where we search for DNA markers without revealing the DNA itself.

Anonymizing users while analyzing social network data, voters' histories from ballots, financial analysis of an organization or group, and other data mining tasks [132] can be represented by such a language. It may bridge some gaps in representation in the secure data collection process through a survey or questionnaire. These are a few examples where a lot of work has been done, and a lot of work is still needed to reach automated, optimized, interpretable, transparent, and privacy-preserving solutions.

6.7 Related challenges

This work also leaves us with questions like: 1) How can we deal with more complex, non-linear models like a multi-layer neural network? 2) How can we address the issue of having highly non-linear functions applied to continuous noisy data and simple additive noise causing over/undershooting problems? 3) How can we deal with problems that have non-convex constraints? It will be worth pursuing further investigation to find answers to these questions.

Chapter 7

Discussion and conclusion

In this thesis, we have presented a few solutions for designing privacy-preserving, utility-maximizing, interpretable, and verifiable AI systems. We will have a brief discussion of our work in Section 7.1, followed by a more detailed summary of the individual contributions in Section 7.2.

7.1 Discussion

Generally, isolated tasks are studied in terms of their privacy properties. They try to answer questions like: (a) Is the algorithm private? or (b) Is the model publication private? Many research areas, like distributed machine learning, federated learning, central and local differential privacy, and secure multi-party computation, emphasize answering these questions.

Our goal is not to build a system applicable to every possible scenario, but rather to analyze the privacy of a range of compound systems together. The framework should enable researchers to analyze privacy through the components, characterize their privacy levels, and detect privacy leaks. The language should help in unambiguously specifying privacy requirements, and the inference will aid in optimizing the privacy parameters while automating the entire process.

Such a language should build more privacy-friendly, efficient, and artificially intelligent (AI) systems and make them more human-interpretable. Currently, emerging privacy-friendly AI algorithms can be brought closer to industry applications through such a framework. This can bring more social relevance to privacy concerns. Such a generic framework can model information flow processes and assess privacy concerns in distributed, multi-centric settings.

The language does not only describe the information flow but also addresses the privacy requirements through these iterative computational steps. We mainly used differential privacy as the standard for privacy guarantees, but the framework can adopt other existing privacy-preserving algorithms as well. Reasoning about the privacy of the compound processes can be carried out

within our framework.

Such analysis will explore missing building blocks in privacy-preserving information processing pipelines. Developing those blocks and bridging the gap could be another target. Algorithms can be developed to perform the verification of privacy claims of the information process and explain the privacy properties of the process in non-expert terms. In various real-world application domains, such a framework can prove to be very useful and can provide verified privacy guarantees.

7.2 Summary

Our novel approach is to specify the privacy requirements and the description of a data pipeline first, and then to create a constraint optimization problem that yields a privacy-preserving solution.

Specification
We have designed a declarative language in Chapter 4 that captures the multi-leveled nature of the privacy of a system through specifications. The collected data has multiple layers of privacy requirements for different observers. Some data is public information, some is private to different extents to different parties, and some of it can be the participant's secret. Such an explicit representation of the privacy guarantees facilitates answering user queries about exactly to what extent sensitive data is protected.

The logical Bayesian network-based language represents such privacy clauses in clearly stated, interpretable predicates. This achieves transparency among the parties involved in a study. In the specification of privacy requirements with our language, the user can define sensitive variables and, for each observed variable, some privacy requirements. The user can decide the privacy budget and privacy mechanism, and all the associated variables for a privacy requirement, share the privacy budget.

Inference and constraint problem
Apart from the specification goal, we also performed some inference, and that is primarily represented by a constraint optimization approach. From the different situations described in Chapters 3, 4, and 5, we can say that privacy specifications can be converted into constraint problems, where the privacy requirements are treated as constraints and the objective function can be optimized efficiently. Now, we have encountered that inference can be very diverse, and we have addressed a couple of them in this thesis. Also, this approach can help us answer questions regarding the privacy-preservation guarantees of the system and reason about them.

Combination of constraint problems
The proposed approaches are general concepts and can be molded into different situations depending on the problem and the required solution. Apart from

deciding the sensitive variables, fixing the privacy budget, and choosing the privacy mechanisms, the user can also design multiple approaches using different privacy mechanisms and different computation details.

The superior approach providing the optimum result will be chosen. The optimal solution to the constraint problem can be noise variance or the right place to position those noises in the data pipeline, or some other parameters, like the nature of noise distribution, for situations where more complex non-linear functions are applied to user features. The solution may give us combined privacy parameters for multiple privacy preservation strategies or optimized parameters for the superior one, given the choice.

In our approach, during the design and development phases, the developer can focus on the requirements and not on the implementation choices. The optimal choice will be found automatically by solving the constraint optimization problem.

Tailored noise

In the project of shaping the tailored noise mechanism, we have discussed the unbiased averaging of privatized sensitive features in a distributed setting. In this work, we have discussed how to optimize utility for privatized features computed by applying transformation functions to sensitive attributes. Such transformations can be highly non-linear and, hence, can have high-magnitude gradients or possess points of singularity.

Real world examples

We understand that the concepts discussed in this dissertation are often not very trivial and may pose difficulty in understanding just from their description. So, we introduced one or more examples directly inspired by real-world problems to illustrate the concepts without going too deep into the domain jargon. We were mostly interested in the optimization of utility and privacy parameters in medical system-based applications.

We presented a few examples showing that, in several cases, the translation of privacy requirements to constraint optimization problems is reasonably easy and often yields constraint optimization problems that can be solved efficiently. Of course, this doesn't constitute proof that such a methodology will deliver good results in all cases. An interesting line of future work is to explore more different situations and analyze whether the obtained constraint optimization problems remain tractable and scale well with the problem complexity.

Bibliography

[1] *Data provenence.*

[2] *General data protection regulation (gdpr).*

[3] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, *Deep learning with differential privacy*, in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, 2016, pp. 308–318.

[4] M. S. Alvim, K. Chatzikokolakis, C. Palamidessi, and A. Pazii, *Metric-based local differential privacy for statistical applications*, CoRR, abs/1805.01456 (2018).

[5] M. Andersen, J. Dahl, and L. Vandenberghe, *Cvxopt: Convex optimization*, Astrophysics Source Code Library, (2020), pp. ascl–2008.

[6] S. Asoodeh, J. Liao, F. P. Calmon, O. Kosut, and L. Sankar, *Three variants of differential privacy: Lossless conversion and applications*, CoRR, abs/2008.06529 (2020).

[7] B. Balle, G. Barthe, and M. Gaboardi, *Privacy amplification by subsampling: Tight analyses via couplings and divergences*, Advances in neural information processing systems, 31 (2018).

[8] B. Balle, G. Barthe, M. Gaboardi, J. Hsu, and T. Sato, *Hypothesis testing interpretations and renyi differential privacy*, CoRR, abs/1905.09982 (2019).

[9] B. Balle and Y. Wang, *Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising*, CoRR, abs/1805.06530 (2018).

[10] D. Barth-Jones, *The're-identification'of governor william weld's medical information: a critical re-examination of health data identification risks and privacy protections, then and now*, Then and Now (July 2012), (2012).

[11] G. Barthe, B. Köpf, F. Olmedo, and S. Zanella-Béguelin, *Probabilistic Relational Reasoning for Differential Privacy*, ACM Transactions on Programming Languages and Systems (TOPLAS), 35 (2013).

[12] R. Bassily, A. Smith, and A. Thakurta, *Private empirical risk minimization: Efficient algorithms and tight error bounds*, in 2014 IEEE 55th annual symposium on foundations of computer science, IEEE, 2014, pp. 464–473.

[13] T. Bayes, *Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s*, Philosophical transactions of the Royal Society of London, (1763), pp. 370–418.

[14] M. Becker, A. Malkis, and L. Bussard, *S4P: A Generic Language for Specifying Privacy Preferences and Policies*, Tech. Rep. MSR-TR-2010-32, April 2010.

[15] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*, vol. 6, Athena scientific Belmont, MA, 1997.

[16] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[17] J. K. Blitzstein and J. Hwang, *Introduction to probability*, Crc Press, 2019.

[18] A. Blum, K. Ligett, and A. Roth, *A learning theory approach to non-interactive database privacy*, CoRR, abs/1109.2229 (2011).

[19] D. Bogdanov, P. Laud, and J. Randmets, *Domain-Polymorphic Language for Privacy-Preserving Applications*, in Proceedings of the First ACM Workshop on Language Support for Privacy-Enhancing Technologies, PETShop '13, Association for Computing Machinery, 2013.

[20] D. Boneh and V. Shoup, *A graduate course in applied cryptography*, Draft 0.5, (2020).

[21] L. Bottou, *Large-scale machine learning with stochastic gradient descent*, in Proceedings of COMPSTAT'2010, Physica-Verlag HD, 2010, pp. 177–186.

[22] T. D. Breaux, H. Hibshi, and A. Rao, *Eddy, a formal language for specifying and analyzing data flow specifications for conflicting privacy requirements*, Requirements Engineering, 19 (2014), pp. 281–307.

[23] M. Bun, J. Ullman, and S. Vadhan, *Fingerprinting codes and the price of approximate differential privacy*, in Proceedings of the forty-sixth annual ACM symposium on Theory of computing, 2014, pp. 1–10.

[24] C. L. Canonne, G. Kamath, and T. Steinke, *The discrete gaussian for differential privacy*, in Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin, eds., 2020.

[25] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, *Differentially private empirical risk minimization.*, Journal of Machine Learning Research, 12 (2011).

[26] Y. Chauvin and D. E. Rumelhart, *Backpropagation: Theory, Architectures, and Applications*, Routledge, February 1995.

[27] T. Chen and C. Guestrin, *Xgboost: A scalable tree boosting system*, Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, (2016), pp. 785–794.

[28] K. L. Chung, *On a Stochastic Approximation Method*, The Annals of Mathematical Statistics, 25 (1954), pp. 463 – 483.

[29] G. Cormode, T. Kulkarni, and D. Srivastava, *Constrained differential privacy for count data*, CoRR, abs/1710.00608 (2017).

[30] G. Cormode, T. Kulkarni, and D. Srivastava, *Constrained private mechanisms for count data*, IEEE Trans. Knowl. Data Eng., 33 (2021), pp. 415–430.

[31] C. Cortes and V. Vapnik, *Support-vector networks*, Mach. Learn., 20 (1995), p. 273–297.

[32] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure multiparty computation*, Cambridge University Press, 2015.

[33] S. K. Das, *Deductive Databases and Logic Programming*, Addison-Wesley Longman, 1992.

[34] Y. N. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio, *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*, CoRR, abs/1406.2572 (2014).

[35] P. S. de Laplace, *Mémoire sur les approximations des formules qui sont fonctions de très-grands nombres et sur leur application aux probabilités*, Imprimerie de Baudouin, 1810.

[36] A. De Moivre, *The doctrine of chances: or, A method of calculating the probabilities of events in play*, vol. 200, Chelsea Publishing Company, Incorporated, 1756.

[37] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for machine learning*, Cambridge University Press, 2020.

[38] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester, *A Modern Introduction to Probability and Statistics: Understanding why and how*, vol. 488, Springer, 2005.

[39] L. Deng, *The MNIST database of handwritten digit images for machine learning research [best of the web]*, IEEE Signal Processing Magazine, 29 (2012), pp. 141–142.

[40] I. Dinur and K. Nissim, *Revealing information while preserving privacy*, in Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, 2003, pp. 202–210.

[41] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, J. Mach. Learn. Res., 12 (2011), p. 2121–2159.

[42] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, *Local Privacy and Statistical Minimax Rates*, in 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, 2013, pp. 429–438.

[43] C. Dwork, *Differential privacy: A survey of results*, in International conference on theory and applications of models of computation, Springer, 2008, pp. 1–19.

[44] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, *Our data, ourselves: Privacy via distributed noise generation*, in Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25, Springer, 2006, pp. 486–503.

[45] C. Dwork, F. McSherry, K. Nissim, and A. Smith, *Calibrating noise to sensitivity in private data analysis*, in Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3, Springer, 2006, pp. 265–284.

[46] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum, *Differential privacy under continual observation*, in Proceedings of the forty-second ACM symposium on Theory of computing, 2010, pp. 715–724.

[47] C. Dwork and K. Nissim, *Privacy-preserving datamining on vertically partitioned databases*, in Advances in Cryptology–CRYPTO 2004: 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004. Proceedings 24, Springer, 2004, pp. 528–544.

[48] C. Dwork and A. Roth, *The algorithmic foundations of differential privacy*, vol. 9, Hanover, MA, USA, aug 2014, Now Publishers Inc., p. 211–407.

[49] C. Dwork, G. N. Rothblum, and S. Vadhan, *Boosting and differential privacy*, in 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, IEEE, 2010, pp. 51–60.

[50] C. Dwork and A. Smith, *Differential privacy for statistics: What we know and what we want to learn*, Journal of Privacy and Confidentiality, 1 (2009), pp. 135–154.

[51] U. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, and K. Talwar, *Amplification by Shuffling: From Local to Central Differential Privacy via Anonymity*, in SODA, 2019.

[52] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., *A density-based algorithm for discovering clusters in large spatial databases with noise*, in kdd, vol. 96, 1996, pp. 226–231.

[53] V. Fabian, *On Asymptotic Normality in Stochastic Approximation*, The Annals of Mathematical Statistics, 39 (1968), pp. 1327 – 1332.

[54] D. Fierens, H. Blockeel, M. Bruynooghe, and J. Ramon, *Logical Bayesian Networks and Their Relation to Other Probabilistic Logical Models*, in Inductive Logic Programming, S. Kramer and B. Pfahringer, eds., Berlin, Heidelberg, 2005, Springer Berlin Heidelberg, pp. 121–135.

[55] D. Fierens, J. Ramon, M. Bruynooghe, and H. Blockeel, *Learning directed probabilistic logical models: ordering-search versus structure-search*, Annals of Mathematics and Artificial Intelligence, 54 (2008).

[56] Y. Freund and R. E. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, Journal of computer and system sciences, 55 (1997), pp. 119–139.

[57] J. H. Friedman, *Greedy function approximation: A gradient boosting machine*, in Annals of statistics, JSTOR, 2001, pp. 1189–1232.

[58] B. E. Fristedt and L. F. Gray, *A modern approach to probability theory*, Springer Science & Business Media, 2013.

[59] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. Pierce, *Linear Dependent Types for Differential Privacy*, in 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, ACM, 2013.

[60] C. F. Gauss, *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*, vol. 7, FA Perthes, 1877.

[61] Q. Geng, P. Kairouz, S. Oh, and P. Viswanath, *The staircase mechanism in differential privacy*, IEEE Journal of Selected Topics in Signal Processing, 9 (2015), pp. 1176–1184.

[62] Q. Geng and P. Viswanath, *The optimal noise-adding mechanism in differential privacy*, IEEE Trans. Inf. Theory, 62 (2016), pp. 925–951.

[63] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer, *Learning probabilistic relational models*, Relational data mining, (2001), pp. 307–335.

[64] A. Ghosh, T. Roughgarden, and M. Sundararajan, *Universally utility-maximizing privacy mechanisms*, SIAM J. Comput., 41 (2012), pp. 1673–1693.

[65] M. Goddard, *The EU General Data Protection Regulation (GDPR): European regulation that has a global impact*, International Journal of Market Research, 59 (2017), pp. 703–705.

[66] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[67] H. V. Gupta and H. Kling, *On typical range, sensitivity, and normalization of mean squared error and nash-sutcliffe efficiency type metrics*, Water Resources Research, 47 (2011).

[68] M. Gupte and M. Sundararajan, *Universally optimal privacy mechanisms for minimax agents*, in Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA, J. Paredaens and D. V. Gucht, eds., ACM, 2010, pp. 135–146.

[69] B. Hansen, *Probability and Statistics for Economists*, Princeton University Press, 2022.

[70] W. Hardle and E. Mammen, *Comparing nonparametric versus parametric regression fits*, The Annals of Statistics, (1993), pp. 1926–1947.

[71] M. Hardt and K. Talwar, *On the geometry of differential privacy*, CoRR, abs/0907.3754 (2009).

[72] T. Hastie, J. Friedman, and R. Tibshirani, *The Elements of Statistical Learning*, vol. 1, Springer New York, 2001.

[73] M. Henze, J. Hiller, S. Schmerling, J. H. Ziegeldorf, and K. Wehrle, *CPPL: Compact Privacy Policy Language*, in Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society, WPES '16, New York, NY, USA, 2016, Association for Computing Machinery, p. 99–110.

[74] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural computation, 9 (1997), pp. 1735–1780.

[75] W. Hoeffding, *A Class of Statistics with Asymptotically Normal Distribution*, The Annals of Mathematical Statistics, 19 (1948), pp. 293 – 325.

[76] J. Iyilade and J. Vassileva, *P2U: A privacy policy specification language for secondary data sharing and usage*, in 2014 IEEE Security and Privacy Workshops, IEEE, 2014, pp. 18–22.

[77] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, vol. 112, Springer New York, 2013.

[78] I. T. Jolliffe, *Principal component analysis: a beginner's guide—i. introduction and application*, Weather, 45 (1990), pp. 375–382.

[79] P. Kairouz, S. Oh, and P. Viswanath, *The composition theorem for differential privacy*, in Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015.

[80] O. Kallenberg, *Foundations of modern probability*, vol. 2, Springer, 1997.

[81] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2015.

[82] R. Kohavi et al., *A study of cross-validation and bootstrap for accuracy estimation and model selection*, in Ijcai, vol. 14, Montreal, Canada, 1995, pp. 1137–1145.

[83] B. Kreuter and a. shelat, *Lessons Learned with PCF: Scaling Secure Computation*, in Proceedings of the First ACM Workshop on Language Support for Privacy-Enhancing Technologies, PETShop '13, New York, NY, USA, 2013, Association for Computing Machinery, p. 7–10.

[84] A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images*, tech. rep., Citeseer, 2009.

[85] P. Lahoti, G. Weikum, and K. P. Gummadi, *iFair: Learning Individually Fair Data Representations for Algorithmic Decision Making*, CoRR, abs/1806.01059 (2018).

[86] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, *Object Recognition with Gradient-Based Learning*, Springer-Verlag, 1999.

[87] J. Liu, L. Xiong, and J. Luo, *A privacy framework: indistinguishable privacy*, in Proceedings of the Joint EDBT/ICDT 2013 Workshops, 2013, pp. 131–136.

[88] C. Mckay, *Probability and statistics*, Scientific e-Resources, 2019.

[89] H. B. McMahan, G. Andrew, U. Erlingsson, S. Chien, I. Mironov, N. Papernot, and P. Kairouz, *A general approach to adding differential privacy to iterative training procedures*, arXiv preprint arXiv:1812.06210, (2018).

[90] F. D. McSherry, *Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis*, in Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, SIGMOD '09, Association for Computing Machinery, 2009.

[91] I. Mironov, *On significance of the least significant bits for differential privacy*, in Proceedings of the ACM Conference on Computer and Communications Security, CCS'12, T. Yu, G. Danezis, and V. D. Gligor, eds., ACM, 2012, pp. 650–661.

[92] I. Mironov, *Renyi Differential Privacy*, CoRR, abs/1702.07476 (2017).

[93] S. Muggleton and L. de Raedt, *Inductive logic programming: Theory and methods*, The Journal of Logic Programming, 19-20 (1994), pp. 629–679. Special Issue: Ten Years of Logic Programming.

[94] K. P. Murphy, *Probabilistic Machine Learning: An introduction*, MIT Press, 2022.

[95] M. Naehrig, K. Lauter, and V. Vaikuntanathan, *Can Homomorphic Encryption Be Practical?*, in Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, CCSW '11, Chicago, Illinois, USA, 2011, Association for Computing Machinery.

[96] A. Narayanan and V. Shmatikov, *How to break anonymity of the netflix prize dataset*, 2007.

[97] J. P. Near, D. Darais, C. Abuah, T. Stevens, P. Gaddamadugu, L. Wang, N. Somani, M. Zhang, N. Sharma, A. Shan, and D. Song, *Duet: An Expressive Higher-order Language and Linear Type System for Statically Enforcing Differential Privacy*, Proc. ACM Program. Lang., 3 (2019), pp. 172:1–172:30.

[98] S. Oh and P. Viswanath, *The composition theorem for differential privacy*, CoRR, abs/1311.0776 (2013).

[99] L. Oneto, S. Ridella, and D. Anguita, *Tikhonov, ivanov and morozov regularization for support vector machine learning*, Machine Learning, 103 (2016), pp. 103–136.

[100] P. Paillier, *Public-key cryptosystems based on composite degree residuosity classes*, in Advances in Cryptology—EUROCRYPT'99, Springer, 1999, pp. 223–238.

[101] R. Pardo and D. Le Métayer, *Analysis of Privacy Policies to Enhance Informed Consent*, in Data and Applications Security and Privacy XXXIII, S. N. Foley, ed., Springer International Publishing, 2019.

[102] A. J. Paverd, A. Martin, and I. Brown, *Modelling and automatically analysing privacy properties for honest-but-curious adversaries*, Tech. Rep., (2014).

[103] S.-D. Poisson, *Recherches sur la probabilité des jugements en matière criminelle et en matière civile: précédées des règles générales du calcul des probabilités*, Bachelier, 1837.

[104] J. Ramon and M. Basu, *Interpretable privacy with optimizable utility*, in ECML PKDD 2020 Workshops: Workshops of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2020): SoGood 2020, PDFL 2020, MLCS 2020, NFMCP 2020, DINA 2020, EDML 2020, XKDD 2020 and INRA 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Springer, 2020, pp. 492–500.

[105] A. Rényi, *On measures of entropy and information*, in Proceedings of the fourth Berkeley symposium on mathematical statistics and probability, volume 1: contributions to the theory of statistics, vol. 4, University of California Press, 1961, pp. 547–562.

[106] R. L. Rivest, A. Shamir, and L. M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, 21 (1978), pp. 120–126.

[107] H. Robbins and S. Monro, *A Stochastic Approximation Method*, The Annals of Mathematical Statistics, 22 (1951), pp. 400 – 407.

[108] S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747, (2016).

[109] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *Imagenet large scale visual recognition challenge*, International Journal of Computer Vision, 115 (2015), pp. 211–252.

[110] C. Sabater, A. Bellet, and J. Ramon, *An accurate, scalable and verifiable protocol for federated differentially private averaging*, Machine Learning, 111 (2022).

[111] Y. Sei and A. Ohsuga, *Private true data mining: Differential privacy featuring errors to manage internet-of-things data*, IEEE Access, 10 (2022), pp. 8738–8757.

[112] J. Shoenfield, *Mathematical Logic*, Taylor & Francis, 2001.

[113] R. Shokri, *Privacy games: Optimal user-centric data obfuscation*, Proc. Priv. Enhancing Technol., 2015 (2015), pp. 299–315.

[114] R. Shokri and V. Shmatikov, *Privacy-preserving deep learning*, in Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, 2015, pp. 1310–1321.

[115] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, *Membership inference attacks against machine learning models*, in 2017 IEEE symposium on security and privacy (SP), IEEE, 2017, pp. 3–18.

[116] S. Song, Y. Wang, and K. Chaudhuri, *Pufferfish Privacy Mechanisms for Correlated Data*, in Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17, Chicago, Illinois, USA, 2017, Association for Computing Machinery.

[117] S. Steffen, B. Bichsel, M. Gersbach, N. Melchior, P. Tsankov, and M. Vechev, *Zkay: Specifying and Enforcing Data Privacy in Smart Contracts*, in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19, New York, NY, USA, 2019, Association for Computing Machinery, p. 1759–1776.

[118] T. Steinke and J. Ullman, *Between pure and approximate differential privacy*, arXiv preprint arXiv:1501.06095, (2015).

[119] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[120] J. Tanton, *Encyclopedia of mathematics*, Facts On File, Inc, 2005.

[121] T. Tieleman and G. Hinton, *Lecture 6.5 - RMSprop: Divide the gradient by a running average of its recent magnitude*, Coursera: Neural Networks for Machine Learning, 4 (2012), pp. 26–31.

[122] M. C. Tschantz, D. K. Kaynar, and A. Datta, *Formal Verification of Differential Privacy for Interactive Systems*, CoRR, abs/1101.2819 (2011).

[123] V. Vapnik, *Principles of risk minimization for learning theory*, in Advances in Neural Information Processing Systems, J. Moody, S. Hanson, and R. Lippmann, eds., vol. 4, Morgan-Kaufmann, 1991.

[124] W. Wang, L. Ying, and J. Zhang, *On the relation between identifiability, differential privacy, and mutual-information privacy*, IEEE Transactions on Information Theory, 62 (2016), pp. 5018–5029.

[125] L. Wasserman, *All of statistics: a concise course in statistical inference*, vol. 26, Springer, 2004.

[126] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD thesis, Harvard University, 1988.

[127] D. Witten, G. M. James, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, Springer, 2013.

[128] X. Wu, F. Li, A. Kumar, K. Chaudhuri, S. Jha, and J. Naughton, *Bolt-on differential privacy for scalable stochastic gradient descent-based analytics*, in Proceedings of the 2017 ACM International Conference on Management of Data, 2017, pp. 1307–1322.

[129] L. Xu, *Modular Reasoning about Differential Privacy in a Probabilistic Process Calculus*, in Trustworthy Global Computing, C. Palamidessi and M. D. Ryan, eds., Springer Berlin Heidelberg, 2013.

[130] X. Yan and X. Su, *Linear Regression Analysis: Theory and Computing*, World Scientific, 2009.

[131] J. Yang, K. Yessenov, and A. Solar-Lezama, *A Language for Automatically Enforcing Privacy Policies*, SIGPLAN Not., 47 (2012), p. 85–96.

[132] Z. Yang, S. Zhong, and R. N. Wright, *Privacy-preserving classification of customer data without loss of accuracy*, in Proceedings of the 2005 SIAM International Conference on Data Mining, SIAM, 2005, pp. 92–102.

[133] Z. Zhang, B. I. P. Rubinstein, and C. Dimitrakakis, *On the differential privacy of bayesian inference*, in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16), 2016, pp. 2365–2371.

[134] A. Zheng and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*, O'Reilly Media, Inc., 2018.

[135] H. Zhong and K. Bu, *Privacy-utility trade-off*, 2022.

[136] X. Zhu, *Semi-supervised learning with graphs*, Carnegie Mellon University, 2005.