Technical University of Crete
Department of Electrical and Computer Engineering

Large Differentially Private Data Synthesis

Diploma Thesis

Author: Christos Zacharioudakis (Student ID: 2014030056)
Supervisor: Professor Minos Garofalakis

Thesis Committee:

- Professor Minos Garofalakis
- Professor Antonios Deligiannakis
- Professor Vassilios Samoladas

February 16, 2020

Abstract

In our days, data exists in abundance, it is ever increasing and it finds numerous uses. A most recent use is the training of Machine Learning models, software capable of making their own decisions. However, using data to train said models raises significant privacy concerns, especially when it comes to highly sensitive data such as medical records. A solution to this predicament is the synthetic data generation, the production of "fake" data that resembles the real one. However, synthetic data generation does not provide any privacy guarantees on its own. The need increases for a robust, meaningful, and mathematically rigorous definition of privacy, together with a computationally rich class of algorithms that satisfy this definition. One such definition is Differential Privacy. This thesis attempts to combine the concept of Differential Privacy with various Machine Learning techniques to generate truly private data that can be utilized in place of the real one effectively. The Machine Learning models that will concern us are the Bayesian Networks and the Generative Adversarial Networks.

Acknowledgments

After five years of arduous and yet productive and interesting studies, the time has finally come to present my thesis, which has benefited greatly from the support of many people, some of whom I would sincerely like to thank here.

To begin with, I would like to thank my supervisor *Professor Minos Garofalakis* for pointing me towards exciting and modern topics and for finding the time to guide me and answer my questions, despite his overloaded schedule. I am also grateful to all the professors of the department of Electrical and Computer Engineering for all the knowledge they offered me in these five years.

Finally, I wish to thank my family for their financial and moral support throughout my studies. I owe my deepest gratitude to *my mother* for her constant and unconditional support, patience and encouragement, even in my most pessimistic moments. Last but not least, I would like to thank *my uncle Andreas* for bringing me in contact with computers when I was quite young, thus igniting my interest in them and inspiring me to pursue this field of study.

Contents

1 Introduction

1.1 Data and their privacy

The term "privacy" denotes a socially defined ability of an individual (or organization) to determine whether, when, and to whom personal (or organizational) information is to be released. (Saltzer and Schroeder (3,))

In computing, data is information that has been translated into a form that is efficient for movement or processing. Relative to today's computers and transmission media, data is information converted into binary digital form. Raw data is a term used to describe data in its most basic digital format. The concept of data in the context of computing has its roots in the work of Claude Shannon, an American mathematician known as the father of information theory. He ushered in binary digital concepts based on applying two-value Boolean logic to electronic circuits. Binary digit formats underlie the CPUs, semiconductor memories and disk drives, as well as many of the peripheral devices common in computing today, such as hard and solid-state drives.

The era spanning from the beginning of the 20th century to today is known as the Information Age. This is largely due to the fact that the world's technological capacity to store information has grown from 2.6 exabytes in 1986 to 5 zettabytes in 2014. Also, due to the extended use of the Internet (Social Media, Cloud Storage e.t.c.) and with the growth of the Internet of Things (IoT),[1] 2.5 exabytes (2.5 million terabytes) of data is created every day and the pace is ever increasing. In fact, it was estimated in 2013 that 90% of the data in the world was generated in 2011 and 2012. It is also estimated that the volume of the available data is doubled every three years and by 2020, data is expected to double every 73 days.

Early on, the importance of data in business computing became apparent by the popularity of the terms "data processing" and "electronic data processing," which, for a time, came to encompass the full spectrum of what is now known as information technology. Over the history of corporate computing, specialization occurred, and a distinct data profession emerged along with growth of corporate data processing. Data processing refers to the process of collecting and manipu-

[1]The Internet of Things (IoT) is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction. In other words, IoT is composed of connected "smart" devices that interact with each other and us while collecting all kinds of data. The number of these devices is estimated from 2 billion devices in 2006 to a projected 200 billion by 2020.

lating raw data to yield useful information. In technical terms it is the process of converting raw data to machine-readable form and its subsequent processing such as updating, rearranging, or printing by a computer.

Data processing has numerous applications in many sectors, such as healthcare, customer oriented service (Netflix, Amazon, e.t.c), telecommunication, marketing, commerce, security and many others. As a result, data is considered extremely valuable and nowadays is one of the most important assets a company has. With the rise of the data economy, companies find enormous value in collecting, sharing and using data. Companies such as Google, Facebook, and Amazon have all built empires atop the data economy. However, the usage of data from others aside their owner, such as the aforementioned companies, raises significant *privacy* concerns and risks accidental privacy breaches with serious consequences for both data owners and analysts.

Privacy is a fundamental right, essential to autonomy and the protection of human dignity, serving as the foundation upon which many other human rights are built. As a consequence, the right to privacy is articulated in all of the major international and regional human rights instruments (laws, degrees, agreements e.t.c.). There are many categories of privacy such as personal and defensive privacy. The type of privacy that will concern us most in the current study, though, is known as *information or data privacy*. Information privacy is the relationship between the collection and dissemination of data, technology, the public expectation of privacy, legal and political issues surrounding them (26, 6). The challenge of data privacy is to use data while protecting an individual's privacy preferences and their personally identifiable information. The fields of computer security, data security, and information security design and use software, hardware, and human resources to address this issue.

When one hears the term "data privacy", the term "cryptography" will immediately come to mind. Cryptography is indeed an indispensable tool used to protect information in computing systems. It is used everywhere and by billions of people worldwide on a daily basis. However, although cryptography is extremely useful and has a variety of applications, cryptography is also highly brittle. The most secure cryptographic system can be rendered completely insecure by a single specification or programming error. No amount of unit testing will uncover a security vulnerability in a cryptosystem. Instead, to argue that a cryptosystem is secure, we rely on mathematical modeling and proofs to show that a particular system satisfies the security properties attributed to it. We often

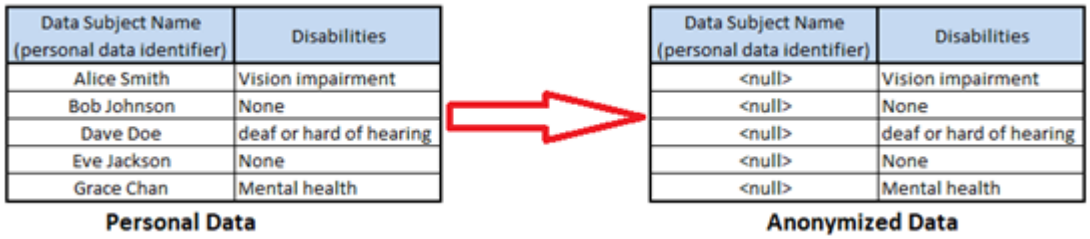
need to introduce certain plausible assumptions to push our security arguments through. That being said, we have to distinguish between security and privacy. Unfortunately, the fact that our data are securely stored today *does not* mean that our privacy is protected; neither today nor in the future.

The problem of privacy-preserving data analysis has a long history spanning multiple disciplines. As electronic data about individuals becomes increasingly detailed, and as technology enables ever more powerful collection and curation of these data, the need increases for a robust, meaningful, and mathematically rigorous definition of privacy, together with a computationally rich class of algorithms that satisfy this definition (*Differential Privacy* (DP), that we will work with, is one such a definition).

The protection of privacy is a complicated and arduous task, especially when it comes to information. Experience has repeatedly shown that when owners of sensitive datasets release derived data, they often reveal more information than intended. Even careful efforts to protect privacy often prove inadequate. Differential Privacy techniques address these problems by only collecting randomized answers from each user, with guarantees of plausible deniability (see Chapter 2), while maintaining the aggregator's ability to build accurate models and predictors by analyzing large amounts of such randomized data. As a result, Differential Privacy is rapidly becoming a golden standard for privacy research. In this study, we attempt to combine the concept of Differential Privacy with data generation techniques, in order to produce synthetic private data that can be used in place of real data.

1.2 Shortcomings of Privacy-Preserving Data Analysis

The effort required for achieving privacy is mostly due to the fact that data cannot be fully anonymized and remain useful. Generally speaking, the richer the data, the more interesting and useful it is. On the contrary, achieving more privacy for the data decreases their usefulness and occasionally requires more computational power and complicated algorithms to accomplish it. Consequently, there is a constant trade-off between data privacy, data richness and computational power. Therefore, we need to make a choice depending on the data in our disposal, the applications we need it for and the privacy budget we have available. Originally, there was the idea of "anonymization" and "removal of personally identifiable information," where the hope is that portions of the data records can be suppressed and the remainder published and used for analysis.



Data Subject Name (personal data identifier)	Disabilities
Alice Smith	Vision impairment
Bob Johnson	None
Dave Doe	deaf or hard of hearing
Eve Jackson	None
Grace Chan	Mental health

Personal Data

Data Subject Name (personal data identifier)	Disabilities
<null>	Vision impairment
<null>	None
<null>	deaf or hard of hearing
<null>	None
<null>	Mental health

Anonymized Data

Figure 1: From personal to 'anonymous' data

In Figure 1, we simply removed the attributes that link the data to a specific person, in other words, the person's name. However, in most cases, this process does not suffice to provide data privacy, because the individuals included in the dataset are still vulnerable to a number of data-breaching attacks. The re-identification of anonymized records not only reveals membership in the data set, but it may also be that the record contains compromising information that, were it tied to an individual, could cause harm. For example, a collection of medical encounter records from a specific urgent care center on a given date may list only a small number of distinct complaints or diagnoses. The additional information that a person visited the facility on the date in question gives a fairly narrow range of possible diagnoses for the person's condition. The fact that it may not be possible to match a specific record to the individual provides him with almost no privacy protection. Some of the most common data-breaching attacks are:

1. **Linkage attacks:** The richness of the data enables "naming" an individual by a sometimes surprising collection of fields, or attributes, such as the combination of zip code, date of birth, and sex, or even the names of three movies and the approximate dates on which an individual watched these movies. This "naming" capability can be used in *a linkage attack* to match "anonymized" records with non-anonymized records in a different dataset. So in addition to this removal, we can add noise drawn from a distribution (e.g Gauss or Laplace) to them, so as to avoid linkage attacks, but still be able reach useful conclusions using them. Methods such as DP neutralize linkage attacks.

2. **Differencing attacks:** Since we aim to protect the privacy of individuals, questions about specific individuals cannot be safely answered with accuracy. Forcing queries to be over large sets is not a real solution either, as shown by the following attack: Suppose the adversary has the following auxiliary information; "Mr. X is in a certain database". And the attacks receive the answers to the two following queries:

    - "How many people in the database have the Y trait?"

- "How many people, not named X, in the database have the Y trait?"

Those queries are allowed by the database administrator, because they are over a large dataset. It is obvious that the two answers indirectly yield the sickle cell status of Mr. X, thus violating that person's privacy.

In response to the aforementioned attacks, one might be tempted to *audit* the sequence of queries and responses, with the goal of interdicting any response if, in light of the history, answering the current query would compromise privacy. For example, the auditor may be on the lookout for pairs of queries that would constitute a differencing attack. However, query auditing has problems of its own:

- It is possible that the very refusal to answer a query may reveal sensitive information to an adversary.

- Query auditing can be computationally infeasible; indeed if the query language is sufficiently rich there may not even exist an algorithmic procedure for deciding if a pair of queries constitutes an attack.

Thus the process is prone to serious errors, that make it impractical for implementing privacy.

Finally, in some cases a particular technique may in fact provide privacy protection for "typical" members of a data set, or more generally, "most" members. In such cases, one often hears the argument that the technique is adequate, as it compromises the privacy of "just a few" participants. Besides the fact that outliers may be precisely those people for whom privacy is most important, the choice of whom to protect creates social and even moral problems, because it discriminates the individuals. That differentiation may cause a great deal of dissent among the individuals and the ones responsible for ensuring privacy. A "just a few" privacy can be achieved by randomly selecting a subset of samples and releasing them in their entirety. The random choice of rows ensures that people from every social group will be protected but it is still a problem for the individuals whose information is revealed. This is especially true in cases where an individual is included in multiple databases, where the "just a few" method is implemented, for the individual is easily made vulnerable to linkage attacks. Differential Privacy provides an alternative when the "just a few" philosophy is rejected.

1.3 What is Differential Privacy?

So after all this talk about Differential Privacy, *what is it?* "Differential Privacy" describes a *promise*, made by a data holder, or curator, to a data subject: "You will not be affected, adversely or otherwise, by allowing your data to be used in any study or analysis, no matter what other studies, data sets, or information sources, are available." At their best, differentially private database mechanisms can make confidential data widely available for accurate data analysis, without resorting to data clean rooms, data usage agreements, data protection plans, or restricted views. Nonetheless, data utility will eventually be consumed: the Fundamental Law of Information Recovery states that overly accurate answers to too many questions will destroy privacy in a spectacular way (1,). In other words, too many queries of a possible adversary on a database inevitably grants information that compromises privacy. So meaningful privacy guarantees come at a price. The goal of algorithmic research on differential privacy is to postpone this inevitability as long as possible.

Differential Privacy addresses the paradox of learning nothing about an individual while learning useful information about a population. A medical database may teach us that smoking causes cancer, affecting an insurance company's view of a smoker's long-term medical costs. Has the smoker been harmed by the analysis? Perhaps he has. For instance, his insurance premiums may rise, if the insurer knows he smokes. But he may also be helped: learning of his health risks, he enters a smoking cessation program. Has the smoker's privacy been compromised? It has since more is known about him after the study than was known before, but was his information "leaked"? Differential Privacy will take the view that it was *not*, with the rationale that the impact on the smoker is the same independent of whether or not he was in the study. It is the *conclusions* reached in the study that affect the smoker, *not his presence or absence in the data set.* Through the medical database, one reaches the conclusion that smoking causes cancer, but should not be able to tell whether a specific individual is included in the database or that he smokes or has cancer. This kind of privacy ensures that the same conclusions, will be reached, independent of whether any individual opts into or opts out of the data set. Specifically, it ensures that any sequence of outputs (responses to queries) is "essentially" equally likely to occur, independent of the presence or absence of any individual. The term "essentially" is captured by a parameter, which we will call ϵ. A smaller ϵ will yield better privacy (but less accurate responses). We will speak of the parameter in more detail in the next chapter.

Last but not least, DP is a **definition**, *not* an **algorithm**. That means it informs one of the conditions that must be fulfilled in order to achieve DP for their data, but gives no instructions on *how* to achieve it. As a result, for a given computational task T and a given value of ϵ there will be many differentially private algorithms for achieving T in an ϵ-differentially private manner. Some will have better accuracy than others. When ϵ is small, finding a highly accurate ϵ-differentially private algorithm for T can be difficult, as much as finding a numerically stable algorithm for a specific computational task can require effort. We devote an entire chapter to Differential Privacy, which constitutes an important portion of the current study.

In conclusion, Differential Privacy promises:

- **Protection against arbitrary risks**
- **Automatic neutralization of linkage attacks**, including all those attempted with all past, present, and future datasets and other forms and sources of auxiliary information.
- **Quantification of privacy loss**: Differential Privacy is not a binary concept, and has a measure of privacy loss. This permits comparisons among different techniques.
- **Composition**: Perhaps most the most important characteristic of DP, the quantification of loss also permits the analysis and control of cumulative privacy loss over multiple computations. Understanding the behavior of differentially private mechanisms under composition enables the design and analysis of complex differentially private algorithms from simpler differentially private building blocks.
- **Group Privacy**: Differential privacy permits the analysis and control of privacy loss incurred by groups, such as families.
- **Closure Under Post-Processing**: Differential Privacy is immune to post-processing: A data analyst, without additional knowledge about the private database, cannot increase privacy loss, no matter what auxiliary information is available.

1.4 What Differential Privacy does *not* promise

However not everything is as perfect as we would wish it to be. While Differential Privacy is an extremely strong guarantee, it does *not* promise unconditional free-

dom from harm. Nor does it create privacy where none previously exists. More generally, DP does not guarantee that what one believes to be one's secrets will remain secret. It merely ensures that one's participation in a survey will not in itself be disclosed, nor will participation lead to disclosure of any specifics that one has contributed to the survey. It is still very possible that conclusions drawn from the survey may reflect statistical information about an individual.

1.5 Implementing Differential Privacy with Machine Learning

The majority of the current thesis is concerned with combining the implementation of Differential Privacy with Machine Learning methods, a subset of Artificial Intelligence, that has received a lot of attention in the recent years and more importantly, it still does! The two methods that will concern us are the Bayesian and the Neural networks.

1.5.1 Bayesian Networks

Bayesian networks are graphical structures for representing the probabilistic relationships among a large number of variables and doing probabilistic inference with those variables. During the 1980's, a good deal of related research was done on developing Bayesian networks, algorithms for performing inference with them and applications that used them. By exploiting conditional independencies entailed by influence chains, we are able to represent a large instance in a Bayesian network using little space, and we are often able to perform probabilistic inference among the features in an acceptable amount of time. In addition, the graphical nature of Bayesian networks gives us a much better intuitive grasp of the relationships among the features of a dataset.

The first and main objective of the current thesis will be to implement and experiment with algorithms that learn Bayesian Networks from real data in a differentiallly private manner and then use them in order to construct a synthetic dataset that a data analyst can freely use for his own purposes without privacy infringements and without the need to access the original data.

1.5.2 Neural Networks

Recent advances in Deep Learning methods (a subset of Machine Learning that uses large neural networks) and the explosion of information collection across a variety of electronic platforms based on artificial neural networks have led to

breakthroughs in long-standing AI tasks such as speech, image, and text recognition, language translation, etc. Companies such as Google, Facebook, and Apple take advantage of the massive amounts of training data collected from their users and the vast computational power of GPU farms to deploy deep learning on a large scale. The unprecedented accuracy of the resulting models allows them to be used as the foundation of many new services and applications, including accurate speech recognition and image recognition that outperforms humans.

The second objective of this thesis will be to implement *Differential Private Generative Adversarial Networks (DPGANs)*, artificial neural networks that produce private synthetic data using real data.

1.6 Thesis Organization & Contributions

In this section we jointly outline the organization of this thesis and its key contributions:

- **Chapter 2**: We present the notion of Differential Privacy. We also enumerate the basic definitions, theorems, and the most common mechanisms that are used in achieving Differential Privacy for our data.
- **Chapter 3**: We will present the definition of Bayesian Networks and the algorithms that we will use to generate private synthetic data using them.
- **Chapter 4**: We make a short introduction to ANNs and Deep Learning and we explain how the GANs operate.
- **Chapter 5**: We will experiment with DPGANs, neural networks capable of generating synthetic image data that do not violate the privacy of the original dataset.

2 Differential Privacy

2.1 The model of computation

2.1.1 Centralized model

In the centralized model, we assume the **existence of a trusted and trustworthy curator (administrator)** who holds the data of individuals in a database D, typically comprised of some number n of rows. Each row contains the data of a single individual and the privacy goal is to simultaneously protect every individual row while permitting statistical analysis of the database as a whole. In the non-interactive, or offline, model the curator produces some kind of object, such as a "synthetic database".

Definition 2.1. *(Synthetic Database) A synthetic database is a multiset drawn from the universe X of possible database rows.*

After this release the curator plays no further role and the original data may be destroyed. The interactive, or online, model permits the data analyst to ask queries [1] adaptively, deciding which query to pose next based on the observed responses to previous queries.

2.1.2 Local model

However, the model that we interests us is the *local model*. In the centralized model of data privacy, we assume the existence of a trusted administrator but what if there is no trusted database administrator? Even if there is a suitable trusted party, there are many reasons not to want private data aggregated by some third party. The very existence of an aggregate database of private information raises the possibility that at some future time, it will come into the hands of an untrusted party, either maliciously (via data theft), or as a natural result of organizational succession. As a result, individuals may be reluctant to share private information with the central data curator. A superior model (from the perspective of the owners of private data) would be a local model, in which agents could (randomly) answer questions in a differentially private manner about their own data, without ever sharing it with *anyone* else. The model we described applies, for instance, in biomedical data analysis, and constitutes a major limitation in biomedical research. Hospitals and other trustworthy entities maintain the clinical records of individuals, but are unable to share and accurately analyze them, due to the risk of privacy breaches. In this model, users randomly perturb their own inputs to provide plausible deniability of their data without the need

[1] A query is a function to be applied to a database

for a trusted party using an instance of a DP algorithm independently but collaboratively with the other users.

The local privacy model was first introduced in the context of learning. The local privacy model formalizes randomized response (see next section): there is no central database of private data. Instead, each individual maintains possession of their own data element (a database of size 1), and answers questions about it only in a differentially private manner. Formally, the database x $\in N^{|X|}$ is a collection of n elements from some domain X and each $x_i \in x$ is held by an individual. This model has been adopted recently by several major technology organizations, including Google, Apple and Microsoft. This model was first suggested by Evfimievski et al (28, 8) and formalized by Kasiviswanathan et al. (19, 9)

2.2 Randomized Response

Differential privacy will provide privacy by process; in particular it will introduce randomness. An early example of privacy by randomized process is randomized response, a technique developed in the social sciences to collect statistical information about embarrassing or illegal behavior, captured by having a property P. Study participants are told to report whether or not they have property P as follows:

1. Flip a coin.

2. If tails, then respond truthfully.

3. If heads, then flip a second coin and respond "Yes" if heads and "No" if tails.

"Privacy" comes from the plausible deniability of any outcome; in particular, if having property P corresponds to engaging in illegal behavior, even a "Yes" answer is not incriminating, since this answer occurs with probability at least 1/4 whether or not the respondent actually has property P . Accuracy comes from an understanding of the noise generation procedure (the introduction of spurious "Yes" and "No" answers from the randomization): The expected number of "Yes" answers is $\frac{1}{4}$ times the number of participants who do not have property P plus $\frac{3}{4}$ the number having property P. Thus, if p is the true fraction of participants having property P, the expected number of "Yes" answers is $(\frac{1}{4})$(1 - p)+ $(\frac{3}{4})$p = $(\frac{1}{4}) + \frac{p}{2}$. Thus, we can estimate p as twice the fraction answering "Yes" minus

$\frac{1}{2}$, that is, 2($\frac{1}{4}$ + $\frac{p}{2}$) - $\frac{1}{2}$. Randomized response was first proposed by Warner in 1965. (27, 7)

2.3 Basic Terms

In this section, we will introduce some necessary definitions, so as to be able to understand the concept of Differential Privacy.

Definition 2.2. *Given a discrete set B, the probability simplex over B, denoted (B) is defined to be:*

$$\Delta(B) = \left\{ x \in \Re^{|B|} : x_i \geq 0 \text{ for all } i \text{ and } \sum_{i=1}^{|B|} x_i = 1 \right\}$$

Definition 2.3. *A randomized algorithm M with domain A and discrete range B is associated with a mapping $M : A \to \Delta(B)$. On input $a \in A$ the algorithm M outputs $M(a) = b$ with probability $(M(a))_b$ for each $b \in B$. The probability space is over the coin flips of the algorithm M.*

We will think of databases x as being collections of records from a universe X. It will often be convenient to represent databases by their histograms: $x \in \mathbb{N}^{|X|}$, in which each entry x_i represents the number of elements in the database x of type $i \in X$.

Definition 2.4. *The ℓ_1 norm of a database x is denoted $||x||_1$ and is defined to be :*

$$||x||_1 = \sum_{i=1}^{|X|} |x_i|$$

The ℓ_1 distance between two databases x and y is $||x - y||_1$ as well a measure of how many records differ between x and y and $||x||_1$ is a measure of the size of a database x (i.e. the number of records it contains).

Using the above definition we define the ℓ_1 sensitivity between two databases x and y that differ in only one element ($||x - y||_1 = 1$).

Definition 2.5. *The ℓ_1 sensitivity of a function $f : \mathbb{N}^{|X|} \to \Re^k$ is :*

$$\Delta f = \max_{x,y \in \mathbb{N}^{|X|}} ||f(x) - f(y)||_1$$

The ℓ_1 sensitivity of a function f captures the magnitude by which a single individual's data can change the function f in the worst case, and therefore, intuitively, the uncertainty in the response that we must introduce in order to hide the participation of a single individual.

2.4 Defining Differential Privacy

Having presented the necessary definitions, we are finally ready to define differential privacy:

Definition 2.6. *A randomized algorithm M with domain* $\mathbb{N}^{|X|}$ *is* (ϵ, δ)*-differentially private if and only if for all* $S \subseteq Range(M)$ *and for all* $x, y \in \mathbb{N}^{|X|}$ *such that* $||x - y||_1 \leq 1$*:*

$$Pr(M(x) \in S) \leq exp(e) * Pr(M(y) \in S) + \delta$$

where the probability space is over the coin ips of the mechanism M. If $\delta = 0$*, we say that M is* ϵ*-differentially private.*

As we mentioned before, we are mostly interested in the local model of data privacy (*Local* Differential Privacy).

Definition 2.7. *A randomized algorithm M satisfies* ϵ*-local differential privacy, where* $\epsilon \geq 0$ *if and only if for all* $S \subseteq Range(M)$ *and for any input x, y, we have:*

$$Pr(M(x) \in S) \leq exp(e) * Pr(M(y) \in S)$$

Local Differential Privacy (LDP) has a stronger privacy model than simple DP, but it entails greater noise.

2.4.1 Parameter ϵ

Since we desire privacy for our data, we want the randomized algorithm M to give similar outputs when implemented to the two databases x and y, so an adversary cannot tell which was given as an input to the algorithm. In other words, we wish for the two probabilities $(Pr(M(x) \in S),\ Pr(M(y) \in S))$ to have similar or even identical values, if possible. For that to occur, we need $\epsilon \to 0$ $(exp(\epsilon) \to 1)$ and $\delta \to 0$. A small ϵ (≤ 1.0) means that the difference of algorithm's output probabilities using x and y at S is small, which indicates high perturbations of ground truth outputs (great amount of noise) and hence high privacy, and vice versa (4,).

It is worth discussing the meaning of the ϵ parameter when applied to real data. If an individual's data is used in a differentially private computation, the probability of any given result changes by at most a factor of $\exp(\epsilon)$, where ϵ is a parameter controlling the trade-off between privacy and accuracy (6,). A small ϵ means higher privacy, but it also means that the utility of the data significantly decreases. In addition to that, more computational power and complex algorithms are required to achieve said privacy.

That is due to another important consideration: ϵ controls how much noise is needed to protect privacy, so it has a direct impact on accuracy. The noise protects the membership of a data point in the dataset. For example, when conducting a clinical experiment, sometimes a person does not want the observer to know that he or she is involved in the experiment. This is due to the fact that observer may link the test result to the appearance/disappearance of certain person and harm the interest of that person. A proper membership protection would ensure that replacing this person with another one will not affect the result too much. This property holds only if the algorithm itself is randomized, i.e. the output is associated with a distribution. And this distribution will not change too much if certain data point is perturbed or even removed. This exactly what the differential privacy tries to achieve. We speak of this further, when we analyze the mathematical mechanisms that we utilize for differential privacy.

Figure 2: An simple example meant to illustrate the trade-off between privacy and accuracy for small -> high ϵ values (7,)

Curiously, despite the importance of the parameter, experimental evaluations of differential privacy, where a concrete choice of ϵ is required, often just pick a value (ranging from 0.01 to 7) with little justification. It is the central parameter controlling strength of the privacy guarantee, and hence the number of queries that can be answered privately as well as the achievable accuracy. But ϵ is also a rather abstract quantity, and it is not clear how to choose an appropriate value in a given situation. A similar concern applies to a second parameter δ in (ϵ, δ)-differential privacy, a standard generalization of differential privacy (6,).

2.4.2 Parameter δ

As is the case with the ϵ parameter, we desire a small value of delta to achieve maximum privacy and preferably we want $\delta = 0$. Typically we are interested in values of δ that are less than the inverse of any polynomial in the size of the database. In particular, values of δ on the order of $\frac{1}{||x||_1}$ are very dangerous: they permit "pre-serving privacy" by publishing the complete records of a small number of database participants — precisely the "just a few" philosophy discussed in Chapter 1.

Even when δ is negligible, however, there are theoretical distinctions between $(\epsilon, 0)$ and (ϵ, δ)-differential privacy. Chief among these is what amounts to a switch of quantification order. $(\epsilon, 0)$-differential privacy ensures that, for every run of the mechanism M(x), the output observed is (almost) equally likely to be observed on every neighboring database, simultaneously. In contrast (ϵ, δ)-differential privacy says that for every pair of neighboring databases x, y, it is extremely unlikely that, ex post facto [1] the observed value M(x) will be much more or much less likely to be generated when the database is x than when the database is y.

The non-private case is given by $\epsilon = \infty$, where δ measures the violation of the "pure" differential privacy. That is, there exists a small output range associated with probability δ such that for some fixed point s in this area, no matter what the value of ϵ is, one can always find a pair of datasets x and y, so that the inequality of the definition holds.

During this study, we will mostly ignore the δ parameter and consider it equal to 0.

2.4.3 Privacy Loss

In this section, we will introduce an important quantity, which allows us to quantify how private our data is. Given an output $\xi \approx M(x)$ it may be possible to find a database y such that ξ is much more likely to be produced on y than it is when the database is x. That is, the mass of ξ in the distribution M(y) may be substantially larger than its mass in the distribution M(x).

[1] An ex post facto law is a law that retroactively changes the legal consequences (or status) of actions that were committed, or relationships that existed, before the enactment of the law.

$$\mathcal{L}^{(\xi)}{}_{M(x)=M(y)} = ln\left(\frac{Pr(M(x)=\xi)}{Pr(M(y)=\xi)}\right)$$

We refer to it as the *privacy loss incurred by observing* ξ. This loss might be positive (when an event is more likely under x than under y) or it might be negative (when an event is more likely under y than under x).

2.5 Useful probabilistic tools

In this section, we will introduce the mechanisms that allow us to implement render our data differentially private.

Definition 2.8. *A privacy mechanism, or simply a mechanism, is an algorithm that takes as input a database, a universe X of data types (the set of all possible database rows), random bits, and, optionally, a set of queries and produces an output string.*

The hope is that the output string can be decoded to produce relatively accurate answers to the queries.

2.5.1 Laplace Mechanism

In this section, we will introduce one of the most commonly used mechanisms for adding noise to our data, the Laplace Mechanism, which naturally lends itself to differential privacy. First, we should introduce the Laplace Distribution:

Definition 2.9. *The Laplace Distribution (centered at 0) with scale b is the distribution with probability density function:*

$$Lap(x|b) = \frac{1}{2b} * exp(-\frac{|x|}{b})$$

The variance of this distribution is $\sigma^2 = 2b$. The Laplace distribution is a symmetric version of the exponential distribution.

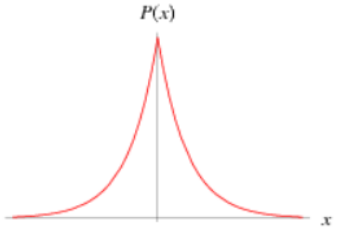


Figure 3: Laplace Probability Density Function (8,)

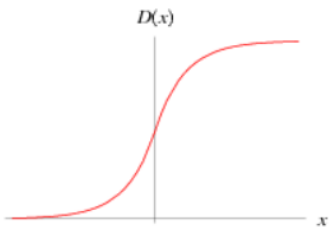


Figure 4: Laplace Cumulative Density Function (8,)

We will now define the Laplace Mechanism. As its name suggests, the Laplace mechanism will simply compute f, and perturb each coordinate with noise drawn from the Laplace distribution. The scale of the noise will be calibrated to the sensitivity of f (divided by ϵ).

Definition 2.10. *Given any function $f : \mathbb{N}^{|X|} \to \Re^k$, the Laplace mechanism is defined as*

$$M_L(x, f(), \epsilon) = f(x) + (Y_1, ..., Y_k)$$

where Y_i are i.i.d. random variables drawn from Lap($\Delta f/\epsilon$)

Theorem 2.1. *The Laplace mechanism preserves $(\epsilon, 0)$-differential privacy*

Proof is in (1,), page 32.

An alternative is the Gaussian Mechanism, which operates similarly to the Laplace Mechanism. Next, we will present some Laplace Mechanism applications on certain types of queries (1,):

Counting Queries: Counting queries are queries of the form "How many elements in the database satisfy Property P ?" We will return to these queries again and again, sometimes in this pure form, sometimes in fractional form ("What fraction of the elements in the databases...?"), sometimes with weights (linear queries), and sometimes in slightly more complex forms (e.g., apply $f : \mathbb{N}^{|X|} \to [0, 1]$ to each element in the database and sum the results). Counting is an extremely powerful primitive. It captures everything learnable in the statistical queries learning model, as well as many standard data mining tasks and basic statistics. Since the sensitivity of a counting query is 1 (the addition or deletion of a single individual can change a count by at most 1), it is an immediate consequence of Theorem 2.1 that $(\epsilon, 0)$-differential privacy can be achieved for counting queries by the addition of noise scaled to $1/\epsilon$, that is, by adding noise drawn from Lap($1/\epsilon$). The expected distortion, or error, is $1/\epsilon$, independent of the size of the database.

Histogram Queries: In the special (but common) case in which the queries are structurally disjoint we can do much better — we don't necessarily have to let the noise scale with the number of queries. An example is the histogram query. In this type of query the universe $\mathbb{N}^{|X|}$ is partitioned into cells, and the query asks how many database elements lie in each of the cells. Because the cells are disjoint, the addition or removal of a single database element can affect the count in exactly one cell, and the difference to that cell is bounded by 1, so histogram queries have sensitivity 1 and can be answered by adding independent draws from

Lap($1/\epsilon$) to the true count in each cell.

An example of a real-world application for the Laplace Mechanism:

Most Common Medical Condition: Suppose we wish to know which condition is (approximately) the most common in the medical histories of a set of respondents, so the set of questions is, for each condition under consideration, whether the individual has ever received a diagnosis of this condition. Since individuals can experience many conditions, the sensitivity of this set of questions can be high. Nonetheless, as we next describe, this task can be addressed using addition of Lap($1/\epsilon$) noise to each of the counts (note the small scale of the noise, which is independent of the total number of conditions). Crucially, the m noisy counts themselves will not be released (although the "winning" count can be released at no extra privacy cost).

An algorithm that can solve the above problem is known as **Report Noisy Max.** We wish to determine which of m counting queries has the highest value. So the algorithm adds independently generated Laplace noise Lap($1/\epsilon$) to each count and return the index of the largest noisy count (we ignore the possibility of a tie). Note the "information minimization" principle at work in the Report Noisy Max algorithm: rather than releasing all the noisy counts and allowing the analyst to find the max and its index, only the index corresponding to the maximum is made public. Since the data of an individual can affect all counts, the vector of counts has high ℓ_1-sensitivity, specifically, $\Delta f = m$, and much more noise would be needed if we wanted to release all of the counts using the Laplace mechanism. The Report Noisy Max algorithm is (ϵ,0)-differentially private.

2.5.2 Exponential Mechanism

The utility [1] of a dataset is directly related to the noise values generated; that is, the popularity of the name or condition is appropriately measured on the same scale and in the same units as the magnitude of the noise.

The exponential mechanism was designed for situations in which we wish to choose the "best" response, but adding noise directly to the computed quantity

[1] In any game (e.g. a contest or an auction), utility represents the motivations of players. A utility function for a given player assigns a number for every possible outcome of the game with the property that a higher number implies that the outcome is more preferred. utility functions may either ordinal in which case only the relative rankings are important, but no quantity is actually being measured, or cardinal, which are important for games involving mixed strategies (9,)

can completely destroy its value, such as setting a price in an auction, where the goal is to maximize revenue, and adding a small amount of positive noise to the optimal price (in order to protect the privacy of a bid) could dramatically reduce the resulting revenue.

Example 2.1. *Suppose we have an abundant supply of pumpkins and four bidders: A, F, I, K, where A, F, I each bid $1.00 and K bids $3.01. What is the optimal price? At $3.01 the revenue is $3.01, at $3.00 and at $1.00 the revenue is $3.00, but at $3.02 the revenue is zero!*

The exponential mechanism is the natural building block for answering queries with arbitrary utilities (and arbitrary non-numeric range), while preserving differential privacy. Given some arbitrary range R, the exponential mechanism is defined with respect to some utility function $u : \mathbb{N}^{|X|}$ x $R \to \Re$, which maps database/output pairs R to utility scores. Intuitively, for a fixed database x, the user prefers that the mechanism outputs some element of R with the maximum possible utility score. Note that when we talk about the sensitivity of the utility score $u : \mathbb{N}^{|X|}$ x $R \to \Re$, we care only about the sensitivity of u with respect to its database argument; it can be arbitrarily sensitive in its range argument:

$$\Delta u = \max_{r \in \Re} \max_{x,y:||x-y||_1 \leq 1} |u(x,r) - u(y,r)|$$

The intuition behind the exponential mechanism is to output each possible $r \in R$ with probability proportional to exp $(\epsilon u(x,r)/\Delta u)$ and so the privacy loss is approximately:

$$ln(\frac{exp(\epsilon u(x,r)/\Delta u}{exp(\epsilon u(y,r)/\Delta u}) = \frac{\epsilon * [u(x,r) - u(y,r)]}{\Delta u} \leq \epsilon$$

Definition 2.11. *The exponential mechanism $M_E(x, u, R)$ selects and outputs an element $r \in R$ with probability proportional to exp* $(\frac{\epsilon u(x,r)}{2 * \Delta u})$.

In contrast to the Laplace Mechanism which is for cases that we wish to calculate numeric values and aggregates (e.g. Counting Queries), the exponential mechanism is appropriate for cases, we wish to receive a discrete set of answers which has some utility. For instance, if we want to find the maximum value of an attribute (Also see Report Noisy Max). However, if we return the value in a deterministic way, there is no privacy due to the fact that we expose the individual to which the maximum value corresponds. If we were to add (e.g. Laplacian) noise to the value of the tuple, its utility will be altered in an undesirable way due to the fact that the noise and the utility are in calculated with different numeric units. So the Laplace Mechanism is not appropriate for such queries. As a consequence, instead of returning the output with the maximum utility, we choose a

random output using the exponential distribution. Each output has a probability given by the distribution and determined by its utility. The greater the utility, the greater the probability of choosing the output. By using this method, we sample the outputs, add randomness to our samples while maintaining a great probability of returning the output with the maximum utility. However, there is still a large enough probability of not doing so, which is desirable and contributes to the privacy we wish to achieve. As a result, we will *almost* always return a result approximately close to the real one due to the nature of the exponential distribution. Using the exponential mechanism, we succeed in having both privacy for our users and a sufficient enough accuracy for our queries, without damaging the utility of the dataset.

The exponential mechanism can define a complex distribution over a large arbitrary domain, and so it may not be possible to implement the exponential mechanism efficiently when the range of u is superpolynomially large in the natural parameters of the problem.

Returning to the pumpkin example, utility for a price p on database x is simply the profit obtained when the price is p and the demand curve is as described by x. It is important that the range of potential prices is independent of the actual bids. Otherwise there would exist a price with non-zero weight in one dataset and zero weight in a neighboring set, violating differential privacy.

Theorem 2.2. *The exponential mechanism preserves (ϵ,0)-differential privacy.*

Proof is in (1,), page 38 - 39.

2.5.3 Composition Theorems

Now that we have several building blocks for designing differentially private algorithms, it is important to understand how we can combine them to design more sophisticated algorithms. In order to use these tools, we would like for the combination of two differentially private algorithms to be differentially private itself. Indeed, as we will see, this is the case. Of course the parameters ϵ and δ will necessarily degrade — consider repeatedly computing the same statistic using the Laplace mechanism, scaled to give ϵ-differential privacy each time. The average of the answer given by each instance of the mechanism will eventually converge to the true value of the statistic, and so we cannot avoid that the strength of our privacy guarantee will degrade with repeated use. This is mathematically proven using the following theorems which are known as the composition theorems.

Theorem 2.3. *Let $M_1 : \mathbb{N}^{|X|} \to R_1$ be an ϵ_1-differentially private algorithm and let $M_2 : \mathbb{N}^{|X|} \to R_2$ be an ϵ_2-differentially private algorithm. Then their combination, defined to be $M_{1,2} : \mathbb{N}^{|X|} \to R_1 \ x \ R_2$ by the mapping: $M_{1,2}(x) = (M_1(x), M_2(x))$ is $\epsilon_1 + \epsilon_2$ differentially private.*

Proof is in (1,), page 42.

Theorem 2.4. *Let $M_i : \mathbb{N}^{|X|} \to R_i$ be an (ϵ_i,δ_i)-differentially private algorithm for $i \in [k]$. Then if $M_{[k]} : \mathbb{N}^{|X|} \to \prod_{i=1}^{k} R_i$ is defined to be $M_{[k]}(x) = (M_1(x), ..., M_k(x))$, then $M_{[k]}$ is ($\sum_{i=1}^{k} \epsilon_i$,$\sum_{i=1}^{k} \delta_i$) differentially private.*

This theorem is a more general version of the previous one. In other words, if we use an algorithm that is composed of two or more differentially private algorithms, then we can safely state that the algorithm is also differentially private, with its ϵ parameter equal to the sum of the ϵ parameters of the algorithms that it is composed of. The same holds true for the δ parameter.

3 DP Data Generation with PrivBayes

3.1 Privacy-preserving data analysis with DP in the distributed model

In this section, we will present a number of methods to learn from data in a private manner using a Machine Learning structure known as Bayesian Network. Before we proceed though, it is necessary to clarify certain terms that we will frequently encounter during this study.

First of all, we must speak of the differences between data owners, data holders and data analysts. Every data field in every database in the organization should be owned by a *data owner*, who is in the authority to ultimately decide on the access to, and usage of, the data. The data owner could be the original producer of the data, one of its consumers, a third party or even the individuals that the dataset refers to. The data owner should be able to fill in or update its value which implies that the data owner has knowledge about the meaning of the field and has access to the current correct value. A *data holder (or custodian)* is an individual, a number of individuals or an organization that is entrusted the possession of the data by its data owner. More often than not, the data owner and the data holder are one and the same. A *data analyst* is someone who collects, processes and performs statistical analyses of data. In the traditional (non-local) differential privacy model that we analyzed in Chapter 2, it is assumed that there exists one data holder, trusted by the data owners also known as the *administrator*. This entity has direct access to the private dataset and analyzes it, ensuring that any output produced by the analysis satisfies differential privacy. Therefore, the data holder and the data analyst is a common (and trusted) entity.

However, our ultimate goal is to process the data in our disposal and receive useful information from it, while minimizing the danger to the data owners' privacy. This process is known as *privacy-preserving data mining*[1]. More specifically, we are interested in making inferences about a population, without compromising the privacy of the individuals (data owners) whose data are used. Given the fact that in the real world applications is horizontally distributed among mutually distrustful parties, we will employ a local model (also see Chapter 2) where we

[1]Data mining is the process of analyzing hidden patterns of data according to different perspectives for categorization into useful information, which is collected and assembled in common areas, such as data warehouses, for efficient analysis, data mining algorithms, facilitating business decision making and other information requirements to ultimately cut costs and increase revenue. Data mining is also known as data discovery and knowledge discovery.

will work with a distributed database. In this model, the data owners/holders are considered reliable with regards to their own data, but not to the data of other holders. Also since the data owners often do not trust the entity that collects and analyzes their data, there is no centralized trusted data administrator and therefore all data analysts are required to access the real data a minimum amount of times and always in a private manner. As a result, the data analyst **(untrusted)** will be considered a separate entity from the data holder(s) **(trusted)**. In particular, each data holder collects its subset of the sensitive data from the data owners and either responds to queries, or performs arbitrary analyses on them; critically, the answers given must satisfy differential privacy (using the standard definition), and the overall privacy budget consumed must meet the privacy requirements. Then the analyst only gets to see these answers and, since differential privacy is immune to post-processing, the owners' privacy is preserved. Our approach will be based on the *PrivBayes* algorithm by Jun Zhang et al (21, 1).

What we seek to accomplish using the PrivBayes algorithm is data publishing with differential privacy. Each data holder constructs a model (e.g. a probabilistic graphical model which in our case is a Bayesian Network) or synopsis (e.g. a histogram) of its dataset, which it then publishes, or uses to generate and publish a synthetic dataset. If the published model/synopsis/dataset satisfies differential privacy, then any analysis performed on it will also guarantee differential privacy, since differential privacy is immune to post-processing. The analyst collects all models/synopses/datasets, merges them, and runs a centralized, non-private algorithm on the merged result.

The main advantage of this approach is that it produces a general and task independent result that can be used for arbitrary analyses. Hence, a vast literature has been developed on data publishing with differential privacy. An important line of work is based on constructing and publishing differentially private synopses of the input dataset. The first connection between differential privacy and probabilistic inference is due to Williams et al. (20, 0); they apply probabilistic inference to the noisy data and, taking into account that the perturbation process is known, they attempt to estimate the parameters of the model that generated the data.

We aim to find efficient methods that are more general compared to older approaches in privacy-preserving data mining and that can be implemented on both non-distributed and distributed data. To that end, we examine three approaches in learning a model (Bayesian Network) that approximates the (high-

dimensional) data distribution as a product of low-order marginal and conditional distributions. This is accomplished by exploiting dependencies that exist between the attributes of the data distribution. Our first approach is an exact approach, in that it requires each data holder to share noisy versions of the algorithm's sufficient statistics. The other two approaches are based on heuristic techniques which usually yield better performance. Finally, once we have the privacy-preserving approximation of the data distribution in hand, we are able to analyze our data without the need of accessing them again. In this study, we will focus on the classification of labeled samples from public datasets.

3.2 Bayes' Theorem

In probability theory and statistics, Bayes' theorem (alternatively Bayes' law or Bayes' rule) describes the probability of an event, based on prior knowledge of conditions that might be related to the event. For example, if cancer is related to age, then, using Bayes' theorem, a person's age can be used to more accurately assess the probability that they have cancer this can be done without knowledge of the person's age. Probability is at the very core of a lot of data science algorithms. In fact, the solutions to so many data science problems are probabilistic in nature. The very fact that we're still learning about it shows how influential his work has been across centuries! Bayes' Theorem enables us to work on complex data science problems and is still taught at leading universities worldwide.

Bayes' theorem is named after Reverend Thomas Bayes, a monk who lived during the eighteenth century, who first used conditional probability to provide an algorithm that uses evidence to calculate limits on an unknown parameter, published as "An Essay towards solving a Problem in the Doctrine of Chances" (1763). In what he called a scholium, Bayes extended his algorithm to any unknown prior cause. Independently of Bayes, Pierre-Simon Laplace in 1774, and later in his 1812 "Théorie analytique des probabilités" used conditional probability to formulate the relation of an updated posterior probability from a prior probability, given evidence. Sir Harold Jeffreys put Bayes's algorithm and Laplace's formulation on an axiomatic basis. Jeffreys wrote that Bayes' theorem "is to the theory of probability what the Pythagorean theorem is to geometry".

Theorem 3.1. ***(Total Probability Theorem) :*** *Let A_1,...,A_n be disjoint events that form a partition of the sample space (each possible outcome is included in exactly one of the events A_1,...,A_n) and assume $P(A_i) > 0$ for all i. Then, for*

any event B, we have :

$$P(B) = P(A_1, B) + ... + P(A_n, B) = P(A_1) * P(B|A_1) + ... + P(A_n) * P(B|A_n)$$

Theorem 3.2. ***(Bayes' Theorem) :*** *Let A_1,...,A_n be disjoint events that form a partition of the sample space, and assume $P(A_i) > 0$ for all i. Then, for any event B such that P(B) > 0, we have :*

$$P(A_i|B) = \frac{P(A_i) * P(B|A_i)}{P(B)} = \frac{P(A_i) * P(B|A_i)}{P(A_1) * P(B|A_1) + ... + P(A_n) * P(B|A_n)}$$

One of the many applications of Bayes' theorem is Bayesian inference, a particular approach to statistical inference. When applied, the probabilities involved in Bayes' theorem may have different probability interpretations. With the Bayesian probability interpretation the theorem expresses how a degree of belief, expressed as a probability, should rationally change to account for availability of related evidence. Bayesian inference is fundamental to Bayesian statistics.

3.3 Bayesian Networks

3.3.1 Introduction

As aforementioned, we consider the attributes of a dataset to be random variables, where dependencies exist between them. These dependencies represent a situation where one feature of an entity has a direct influence on another feature of that entity. An accurate example presenting the need for Bayesian Networks is as follows: (18, 8)

Example 3.1. *The presence or absence of a disease in a human being obviously has a direct influence on whether a test for that disease turns out positive or negative. For decades, Bayes' theorem has been used to perform probabilistic inference in this situation. In this case, a simple application of the Bayes's Theorem would suffice to solve this problem. We would use that theorem to compute the conditional probability of an individual having a disease when a test for the disease came back positive. However, consider the situation where* ***several*** *features are related through inference chains. For example, whether or not an individual has a history of smoking has a direct influence both on whether or not that individual has bronchitis and on whether or not that individual has lung cancer. In turn, the presence or absence of each of these diseases has a direct influence on whether or not the individual experiences fatigue. Also, the presence or absence of lung cancer has a direct influence on whether or not a chest X-ray is positive. In this situation, we would want to do probabilistic inference involving features that are*

not related via a direct influence. We would want to determine, for example, the conditional probabilities both of bronchitis and of lung cancer when it is known an individual smokes, is fatigued, and has a positive chest X-ray. Yet bronchitis has no direct influence (indeed no influence at all) on whether a chest X-ray is positive. Therefore, these conditional probabilities cannot be computed using a simple application of Bayes' theorem. There is a straightforward algorithm for computing them, but the probability values it requires are not ordinarily accessible. Moreover, the algorithm has exponential space and time complexity, which is not acceptable.

Bayesian networks were developed to address these difficulties. By exploiting conditional independencies entailed by influence chains, we are able to represent a large instance in a Bayesian network using little space, and we are often able to perform probabilistic inference among the features in an acceptable amount of time. In addition, the graphical nature of Bayesian networks gives us a much better intuitive grasp of the relationships among the features. The power of Bayesian Networks is twofold. Besides the computational gain they offer, by allowing us to approximate a high-dimensional distribution as a product of low-order conditional and marginal distributions, Bayesian Networks are also highly interpretable models, and can hence be used for knowledge discovery.

i	X_i	Π_i
1	A	$\emptyset$
2	B	{A}
3	C	{A}
4	D	{A}
5	E	{B,D}

Table 1: The attribute-parent pairs in N_1

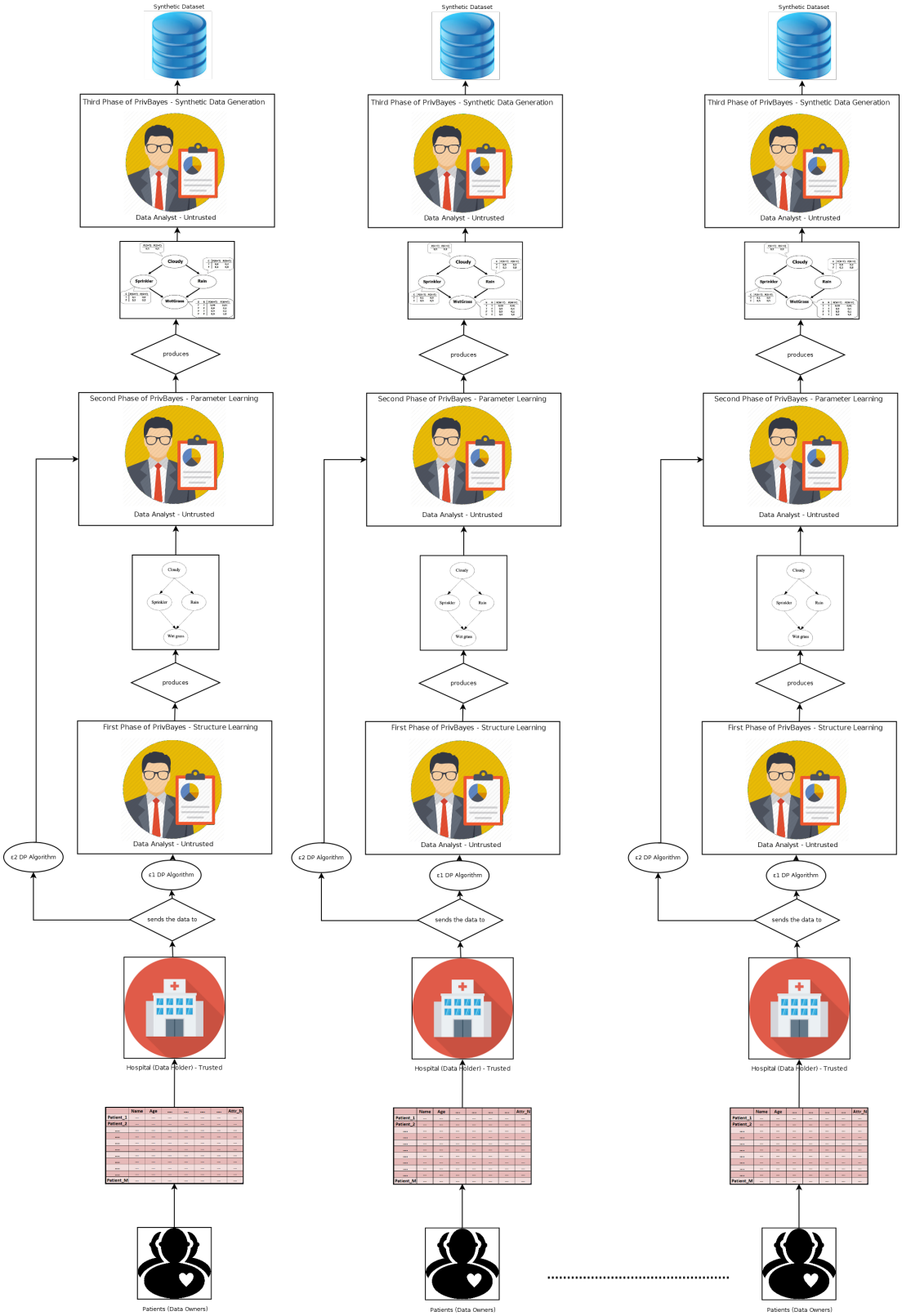


Figure 5: PrivBayes Algorithm with Distributed Data

Before we proceed to the definition of a Bayesian Network, we will present the following fundamental concepts:

Definition 3.1. ***(Conditional Probability):*** *The conditional probability of an event B is the probability that the event will occur given the knowledge that an event A has already occurred. This probability is written P(B|A), notation for the probability of B given A. In the case where events A and B are independent (where event A has no effect on the probability of event B), the conditional probability of event B given event A is simply the probability of event B, that is P(B). From this definition, the conditional probability P(B|A) is easily obtained by dividing by P(A):*

$$P(B|A) = \frac{P(A,B)}{P(A)},\ P(A) \neq 0$$

Definition 3.2. ***(Marginal Probability Mass Function):*** *For discrete random variables, the marginal probability mass function can be written as P(X = X) and it is obtained by:*

$$P(X = x) = \sum_y P(X = x, Y = y) = \sum_y P(X = x|Y = y) * P(Y = y)$$

This definition is an application of the *Law of Total Probability* [1]

3.3.2 Definition

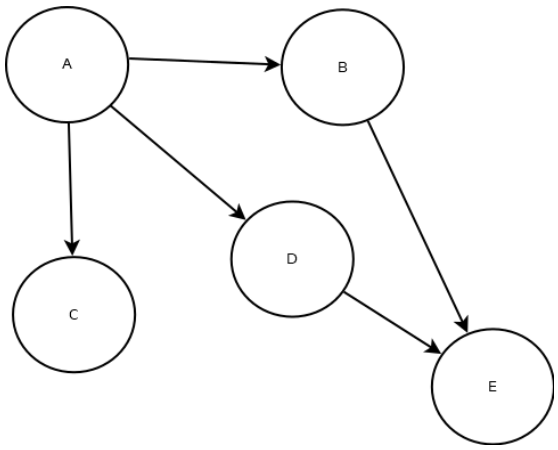


Figure 6: A simple Bayesian Network N_1 over five attributes

[1]The law of total probability is the proposition that if $\{B_n : n = 1, 2, 3, \ldots\}$ is a finite or countably infinite partition of a sample space (in other words, a set of pairwise disjoint events whose union is the entire sample space) and each event B_n is measurable, then for any event A of the same probability space: $\Pr(A) = \sum_n \Pr(A \cap B_n)$ or, alternatively $\Pr(A) = \sum_n \Pr(A \mid B_n) \Pr(B_n)$.

Let A be the set of attributes on a dataset D and N be the size of A. A Bayesian network on A is a way to compactly describe the (probability) distribution of the attributes in terms of other attributes. Formally, a Bayesian network is a directed acyclic graph (DAG [1]) that (i) represents each attribute in A as a node, and (ii) models conditional independence among attributes in A using directed edges. As an example, Figure 6 shows a Bayesian network over a set A of five attributes: A, B, C, D, E. For any two attributes $X, Y \in A$, there exist three possibilities for the relationship between X and Y :

1. **Direct dependence**: There is an edge between X and Y , say, from Y to X. This indicates that for any tuple in D, its distribution on X is determined (in part) by its value on Y . We define Y as a parent of X, and refer to the set of all parents of X as its parent set. For example, in Figure 6, the edge from B to E indicates that the distribution of E depends on B.

2. **Weak conditional independence**: There is a path (but no edge) between Y and X. Assume without loss of generality that the path goes from Y to X. Then, X and Y are conditionally independent given X's parent set. For instance, in Figure 6, there is a two-hop path from A to E, and the parent set of E is B, D. This indicates that, given B and D of an individual, its E and A are conditionally independent.

3. **Strong conditional independence**: There is no path between Y and X. Then, X and Y are conditionally independent given any of X's and Y's parent sets. There are no such attribute pairs in Figure 6.

Definition 3.3. ***(Bayesian Network):*** *A Bayesian Network $\mathcal{N}$ over A is defined as a set of d attribute-parent (AP) pairs, $(X_1, \Pi_1), ..., (X_d, \Pi_d)$, such that:*

1. *Each X_i is a unique attribute in A.*

2. *Each Π_i is a subset of the attributes in $A \setminus \{X_i\}$. We say that Π_i is the parent set of X_i in $\mathcal{N}$.*

3. *For any $1 \leq i < j \leq d$, we have $X_i \notin \Pi_i$, i.e., there is no edge from X_j to X_i in $\mathcal{N}$. This ensures that the network is acyclic, namely, it is a DAG.*

[1] A Directed Acyclic Graph (DAG) is a type of graph in which it's impossible to come back to the same node by traversing the edges. 'Directed' means that the edges of the graph only move in one direction, where future edges are dependent on previous ones. 'Acyclic' means that it is impossible to start at one point of the graph and come back to it by following the edges. Whereas a cycle comes back around to it's original starting point like a circle, an acyclic graph continues moving in a linear direction and never does circle back to the starting point.

Definition 3.4. ***(Degree of Bayesian Network):*** *We define the degree of* $\mathcal{N}$ *as the maximum size of any parent set* Π_i *in* $\mathcal{N}$*.*

For example, Table 1 shows the AP pairs in the Bayesian network $\mathcal{N}_1$ in Figure 6: $\mathcal{N}_1$'s degree equals 2, since the parent set of any attribute in $\mathcal{N}_1$ has a size at most two.

Let $Pr[A]$ denote the full distribution of tuples in database D. The d AP pairs in $\mathcal{N}$ essentially define a way to approximate $Pr[A]$ with d conditional distributions Pr[$X_1|\Pi_1$],...,Pr[$X_d|\Pi_d$].In particular, under the assumption that any X_i and any $X_j \notin \Pi_i$ are conditionally independent given Π_i, we have

$$\Pr[\mathrm{A}] = \Pr[\mathrm{X}_1, X_2, ..., X_d] =>$$

$$\Pr[\mathrm{A}] = \Pr[\mathrm{X}_1] * Pr[X_2|X_1] * Pr[X_3|X_1, X_2].. * Pr[X_d|X_1, ..., X_{d-1}] =>$$

$$\Pr[\mathrm{A}] = \prod_{i=1}^{d} Pr[X_i|\Pi_i]$$

Let $Pr_{\mathcal{N}}[A]$ be the above approximation of $Pr[A]$ defined by the Bayesian Network $\mathcal{N}$. Intuitively, if $\mathcal{N}$ accurately captures the conditional independence among the attributes in A, then $Pr_{\mathcal{N}}[A]$ would be a good approximation of $Pr[A]$. In addition, if the degree of $\mathcal{N}$ is small, then the computation of $Pr_{\mathcal{N}}[A]$ is relatively simple as it requires only N low-dimensional distributions Pr[$X_1|\Pi_1$], ... , Pr[$X_d|\Pi_d$]. Low-degree Bayesian networks are the core of our solution to release high-dimensional data.

3.4 Distributed Bayesian Network Learning with Differential Privacy

3.4.1 Introduction

We will now clarify some assumptions that we will make during the remainder of this study:

- We assume that our distributed database consists of homogeneous records, and each record consists of d attributes X_1,...,X_d , from a set A. In statistical machine learning, each attribute can be viewed as a random variable, and, thus, a record x (also known as data point or a tuple to database administrators) can be viewed as a realization of the random vector X = $[X_1, ..., X_d]^T$. All attributes are observed, so no hidden variables exist. A (relatively small) number of missing values may exist for some attributes. Some of the methods for dealing with such values are:

1. **Encode missing values as -1**: This works reasonably well for numerical features that are predominantly positive in value, and for tree-based models in general. This used to be a more common method in the past when the out-of-the box machine learning libraries and algorithms were not very adept at working with missing data.
2. **Encode missing values as another level of a categorical variable**: This works with tree-based models and other models if the feature can be numerically transformed (one-hot encoding, frequency encoding, etc.).
3. **Deletion of missing data**: Here you simply remove all data points from the dataset that contain missing values. In the case of a very large dataset with very few missing values, this approach could potentially work really well. However, if the missing values are in cases that are also otherwise statistically distinct, this method may seriously skew the predictive model for which this data is used. Another major problem with this approach is that it will be unable to process any future data that contains missing values. If your predictive model is designed for production, this could create serious issues in deployment.
4. **Replace missing values with the mean/median value of the feature in which they occur**: This works for numerical features. The choice of median/mean is often related to the form of distribution that the data has. For imbalanced data, the median may be more appropriate, while for symmetrical and more normally distributed data, the mean could be a better choice.
5. **Run predictive models that impute the missing data**: This should be done in conjunction with some kind of cross-validation scheme in order to avoid leakage. This can be very effective and can help with the final model.

Since missing values are few, we will use the first method.

- We also assume that we deal with discrete data, so for every $i \in \{1, ..., d\}, X_i$ takes values from the discrete alphabet domain (X_i). However continuous attributes do exist in the datasets that we will use, so we either treat them as discrete attributes or we use a discretization technique of our choice. The method we implement is to split the value range of each continuous attribute into sub-intervals (bins). Then we change the value of the attribute to the value corresponding to the bin it belongs to. By having discrete data only, we are able to accurately estimate quantities like the mutual information

I, which is an essential metric for the algorithms that we will implement. On the contrary, estimating such quantities for continuous data is a much more resource-consuming problem.

- We also assume that these distributed datasets are non-overlapping, so each data owner's record is stored by exactly one data holder.

We next introduce some notation in the following table, that will allow us to describe our model.

Notation	**Description**	**Definition**
d	Data dimension - number of attributes per data point	-
A	Set of all attributes	$A = \{X_1, ..., X_d\}$
X_i	i-th attribute of set A where $i \in \{1, ...d\}$	-
X or X_A	The random vector that consists of all N attributes in A	-
x or x_A	A realization of X (a data point or in other words a tuple of the dataset)	-
A'	A subset of the attribute set A	$A' = \{X_i, X_j, ...\} \subseteq A$ for some $j \in \{1, ..., d\}$ and $j \neq i$
$X_{A'}$	The random vector that consists of all attributes in A'	-
$x_{A'}$	A realization of $X_{A'}$ (a tuple that consists of the values that the attributes in A take in data point x.)	-
$d_{A'}$	The Cartesian product of the domains of the attributes in A'. This set includes all possible tuples that can exist using all the attributes in A'. We consider this set ordered	$d_{A'}$ = domain(X_i) x domain(X_j) x ...
$d_{A'}[k]$	Represents the k-th element in $d_{A'}$	$d_{A'}[k] = (x_i, x_j, ...)$ for some $k \in \{1, ..., d\}$ where d = $\lvert d_{A'} \rvert$
D	The (centralized) dataset	-
M	Number of data holders (degree of distribution)	-
D_j	Dataset of holder j ($j \in \{1, ..., M\}$)	-
n_j	Size of dataset of holder j	$n_j = \lvert D_j \rvert$
n	Size of total dataset	n = $\sum_{j=1}^{M} n_j = \lvert D \rvert$

$c_{X_{A'}}$ or $c_{A'}$	The joint frequency distribution of the attributes in A'	$c_{A'} = [c_1 \ ... \ c_d]^T$
c_k	The number of times the k-th element of $d_{A'}$ ($d_{A'}[k]$) is included in the database D (COUNT)	$c_k = \sum_{x \in D} 1 * f(X, k)$
$f(X, k)$	It has a value of one if $d_{A'}[k]$ is equal to the tuple of D we currently examine and zero if it is not	$f(X, k) = 1$ if $X_{A'} == d_{A'}[k]$ or else it is 0
$p_{X_{A'}}$ or $p_{A'}$	The joint probability distribution of the attributes in A'	$p_{A'} = [p_1 \ ... \ p_d]^T$
p_k	The probability of the k-th element of $d_{A'}$ ($d_{A'}[k]$) to be equal to the $X_{A'}$ (tuple) we currently examine	$p_k = \Pr(X_{A'} = d_{A'}[k])$
$\hat{p}_{A'}$	The maximum likelihood (empirical) estimate of $p_{A'}$ is computed from $c_{A'}$ using the classical definition of probability	$\hat{p}_{A'} = \frac{1}{\|D\|} * c_{A'}$

A simple example where will we use some of the aforementioned notation:

Example 3.2. *We seek to study the following database D :*

$Attr_1$	$Attr_2$
1	0
0	1
1	1
1	0
1	1
0	0
0	1
1	0

where the set of attributes is : $\boldsymbol{A} = \boldsymbol{\{}Attr_1\boldsymbol{,}Attr_2\boldsymbol{\}}$ *and their realizations* $\boldsymbol{\{}attr_1, attr_2\boldsymbol{\}} \in \{0, 1\}$ *(binary attributes). Calculate the variables:*

1. $\boldsymbol{N}$ *(Data dimension)*
2. $\boldsymbol{M}$ *(Data holders)*
3. $\boldsymbol{n}$ *(Size of dataset)*
4. d_A *(The Cartesian product of the domains of the attributes in A)*

5. c_A *(The joint probability distribution of the attributes in A)*

6. $\hat{p}_A$ *(The maximum likelihood estimate of joint probability distribution of the attributes in A)*

Solution:

1. *We have 2 attributes, so the data dimension is* $\boldsymbol{N = 2}$

2. *We have only one dataset and one data holder, so* $\boldsymbol{M = 1}$

3. *We have 8 tuples, so the size of the total dataset is* $\boldsymbol{n = 8}$

4. *We consider an arbitrarily ordered* d_A *equal to* ***{{0,0},{0,1},{1,0},{1,1}}***.

5. *The joint frequency distribution of the attributes* c_A *is equal to* ***[1, 2, 3, 2]***.

6. *The maximum likelihood estimate of joint probability distribution of the attributes* $\hat{p}_A$ *is equal to* ***[1/8, 2/8, 3/8, 2/8]***

3.4.2 Learning Bayesian Networks from data

In real-world applications, we assume that data follows a distribution p over a set of attributes, that can be encoded by a Bayesian Network. Sometimes the Bayesian Network structure is known; for example, it may be given by an expert, or it may be determined by the physical properties of the application. However, our focus is on the case that the structure is *unknown*, and our objective is to find both the structure and the parameters of the Bayesian Network that best fits our dataset D. The process through which we learn a model from data is known as *Structure Learning*.

There are many approaches to learning Bayesian Networks of unknown structure from data. The most common is a *score-based* approach where the Bayesian Network learning problem is viewed as a model selection problem and is solved using an optimization method. The first step in score-based Bayesian Network learning is to assign each possible Bayesian Network $\mathfrak{B}$ a score. A natural choice for scoring function is the likelihood function $\mathfrak{L}$, which measures the probability of observing the data in D (assuming that individual data points are i.i.d.), given a model. In our case, the model is a Bayesian Network $\mathfrak{B} = (\mathfrak{G},\Theta)$ (where $\mathfrak{G}$ is the graph of the Bayesian Network and Θ its conditional probability distributions[1]), so, denoting by $p_{\mathfrak{B}}$ the distribution encoded by $\mathfrak{B}$, we conclude that the score of

[1]We denote the set of all conditional probability tables by : $\Theta = \{\Theta_1, ..., \Theta_d\}$ where $\Theta_i = Pr[X_i|\Pi_i]\ \forall i \in \{1, ..., N\}$

$\mathfrak{B}$ is:

$$\mathfrak{L}(\mathfrak{B}, D) = \prod_{x \in D} p_{\mathfrak{B}}(x) = \prod_{x \in D} \prod_{i=1}^{d} Pr_{X_i|\Pi_i}[X_i, \Pi_i] = \prod_{i=1}^{d} \prod_{x \in D} \Theta_i(x_i, x_{\Pi_i})$$

which illustrates the decomposability of the global likelihood into local likelihoods (one for each parameter Θ_i), based on the Bayesian Network structure. If we instead use the logarithm of $\mathfrak{L}$ as scoring function, it turns out that:

$$log\mathfrak{L}(\mathfrak{B}, D) = n * \left(\sum_{i=1}^{d} \hat{I}(X_i, \Pi_i) - \sum_{i=1}^{d} \hat{H}(X_i) \right)$$

where $\hat{H}$ is the empirical entropy:

$$\hat{H}(X) = - \sum_{x \in dom(X)} Pr[X = x] * log(Pr[X = x])$$

[2]

and $\hat{I}$ is the mutual information between two variables:

$$\hat{I}(X, \Pi) = \sum_{x \in dom(X)} \sum_{\Pi \in dom(\Pi)} Pr[X = x, \Pi = \pi] * log\left(\frac{Pr[X=x, \Pi=\pi]}{Pr[X=x] * Pr[\Pi=\pi]}\right)$$

The mutual information metric has some very useful properties that will assist us during the experimental evaluation of the algorithms that we will use. Some of them are:

1. I(X,Y) ≥ 0
2. Mutual information is symmetric: I(X,Y) = I(Y,X)
3. Mutual information is additive for independent variables: I(X,W,Y,Z) = I(X,Y) + I(W,Z).
4. $I(X, Y, Z) \geq I(X, Y)$

The next step is to find the Bayesian Network structure $\mathfrak{G}$ that achieves the highest score. Noting that the only term in the log-likelihood score that depends on $\mathfrak{G}$ is the empirical mutual information, gives that:

$$\max_{\mathfrak{B}}(log\mathfrak{L}(\mathfrak{B}, D)) = \max_{\mathfrak{G}} \max_{\Theta}(log\mathfrak{L}(\mathfrak{G}, \Theta, D)) = \max_{\mathfrak{G}}(log\mathfrak{L}(\mathfrak{G}, \Theta, D)) =>$$

$$\max_{\mathfrak{B}}(log\mathfrak{L}(\mathfrak{B}, D)) = \max_{\mathfrak{G}} \sum_{i=1}^{d} \hat{I}(X_i, \Pi_i)$$

[2]The domain of a random variable X is a sample space (dom(X)), which is interpreted as the set of possible outcomes of a random phenomenon. For example, in the case of a coin toss, only two possible outcomes are considered, namely heads or tails.

which demonstrates that the structure G* that maximizes the log-likelihood score is the one that has the maximum sum over all attributes, of the mutual information between each attribute and its parents in G*, when the maximum likelihood parameters $\hat{\Theta}$ are used for G*.

Solving the aforementioned optimization problem turns out to be hard in general. Specifically, taking into account that our graph consists of d nodes (since there are d attributes), there are $O(2^{2^d})$ potential structures that form our search space (super-exponential in the number of attributes). Consequently, using brute-force methods is out of the question and therefore heuristic (local-search) algorithms are employed in practice. Nevertheless, **for the special case that k = 1 (Bayesian Networks with degree 1)**, the well-known Chow-Liu algorithm (23, 3) allows us to greedily find the optimal structure.

An important limitation of the likelihood score is that it favors more complex structures over simpler ones. In fact, if we do not artificially constrain the number of parents allowed for each attribute (as in the case of the Chow-Liu algorithm), then any algorithm using the likelihood score will almost always return a fully connected structure. This stems from the fact that adding an additional parent to any attribute will almost always increase the score (Check property 4 of mutual information). Therefore, in practice, we learn fixed-degree Bayesian Networks as we do in PrivBayes, where k is a parameter. There are methods with which the algorithm can choose an appropriate value for k. We will speak of them in a later section.

Algorithm 1: Chow-Liu Algorithm

Input: Dataset D
Output: Bayesian Network (Graph) $\mathfrak{G}$

```
Initialize 𝔊_0 to a fully connected (undirected) graph
for i = 1 to N do
    Estimate and store p̂_{X_i} from D
    for j = 1 to i-1 do
        Estimate and store p̂_{X_j,X_i} from D
        Compute weight Î(X_j, X_i) of edge (X_j, X_i) in 𝔊_0
𝔊 = MaximumWeightSpanningTree (𝔊_0)
Give directions to edges in 𝔊
Return 𝔊
```

[1]

[1] A spanning tree is a subset of Graph G, which has all the vertices covered with minimum

Once the Bayesian Network structure G is known, we have to estimate the parameters in Θ. A common approach to achieve this is to (once again) use the maximum likelihood principle. The decomposability of the likelihood function, which we demonstrated earlier, allows us to maximize each local likelihood $\mathfrak{L}_i(\Theta_i, D) = \prod_{x \in D} \Theta_i(x_i, x_{\Pi_i})$ separately and then combine the solutions to get the (global) maximum likelihood estimate $\hat{\Theta}$ for Θ :

$$\hat{\Theta} = \{\hat{\Theta}_1, ..., \hat{\Theta}_N\} \; where \; \hat{\Theta}_i = argmax_{\Theta_i}(\mathfrak{L}_i(\Theta_i, D))$$

We described a purely frequentist approach for learning Bayesian Networks from data, in that we utilize the likelihood score to learn the structure and then estimate the parameters that maximize the likelihood function. The process of estimating the parameters of the model is known as *Parameter Learning*. An alternative path would be to adopt a Bayesian approach and treat the candidate structures (in the Structure Learning phase) or the parameters (in the Parameter Learning phase) as random; we would assign them prior distributions, and we would maximize the posterior distribution of the data, instead of the likelihood. We do not further discuss the Bayesian approach here, but we mention that its main advantage over the frequentist approach is that it avoids overfitting. Finally we will present a simple example of finding a 1-degree Bayesian network from data using Algorithm 1:

Example 3.3. ***(Chow-Liu Algorithm for 1-degree BN):***

We are given the dataset:

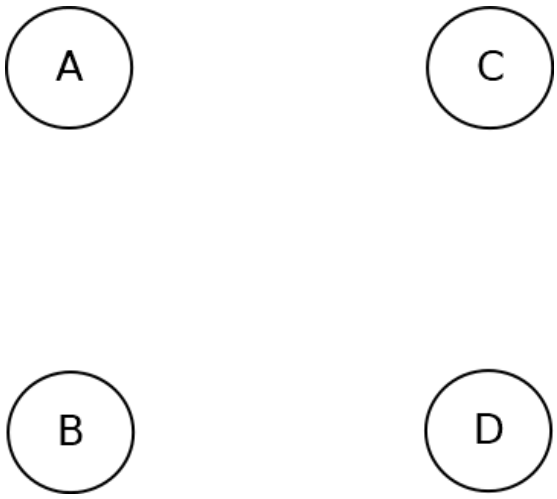
A	**B**	**C**	**D**
1	1	1	0
0	0	0	0
1	1	0	0
1	1	1	0
1	1	0	0
0	0	0	0
1	1	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Build an 1-degree Bayesian Network that represents the given dataset.

possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected. By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices. In this algorithm, we need the spanning tree with the maximum weights, so that we have the maximum mutual information $\hat{I}$ between the attributes.

Solution:

1. *We start from a random attribute. We choose A in this example. So our Bayesian Network will look like this:*



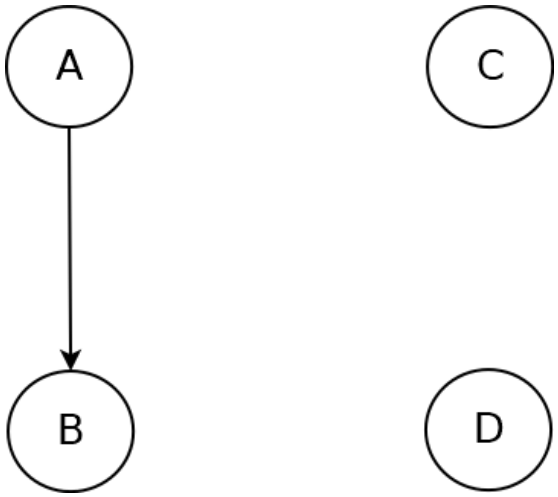
2. *We select the next tree edge by its mutual information $\hat{I}$. We calculate the mutual information between A and the other attributes and we pick the edge with the highest value.*

   (a) ***I(B,A)*** *= P(A = 0, B = 0) * log (P(A = 0, B = 0)/(P(A = 0) * P(B = 0))) + P(A = 1, B = 0) * log (P(A = 1, B = 0)/(P(A = 1) * P(B = 0))) + P(A = 0, B = 1) * log (P(A = 0, B = 1)/(P(A = 0) * P(B = 1))) + P(A = 1, B = 1) * log (P(A = 1, B = 1)/(P(A = 1) * P(B = 1))) = 1*

   (b) ***I(C,A)*** *= P(A = 0, C = 0) * log (P(A = 0, C = 0)/(P(A = 0) * P(C = 0))) + P(A = 1, C = 0) * log (P(A = 1, C = 0)/(P(A = 1) * P(C = 0))) + P(A = 0, C = 1) * log (P(A = 0, C = 1)/(P(A = 0) * P(C = 1))) + P(A = 1, C = 1) * log (P(A = 1, C = 1)/(P(A = 1) * P(C = 1))) = 0.16 + 0 − 0.12 + 0.4 ≃ 0.4*

   (c) ***I(D,A)*** *= P(A = 0, D = 0) * log (P(A = 0, D = 0)/(P(A = 0) * P(D = 0))) + P(A = 1, D = 0) * log (P(A = 1, D = 0)/(P(A = 1) * P(D = 0))) + P(A = 0, D = 1) * log (P(A = 0, D = 1)/(P(A = 0) * P(D = 1))) + P(A = 1, D = 1) * log (P(A = 1, D = 1)/(P(A = 1) * P(D = 1))) = 0*

   So the edge with the highest mutual information is $A \rightarrow B$ and the Bayesian Network will look like this:



3. *We select the next tree edge. The candidate edges are $A \to C$, $A \to D$ which remain from the previous step and since we connected A to B we also have $B \to C$ and $B \to D$. We compare the mutual information of these edges and it occurs that:*

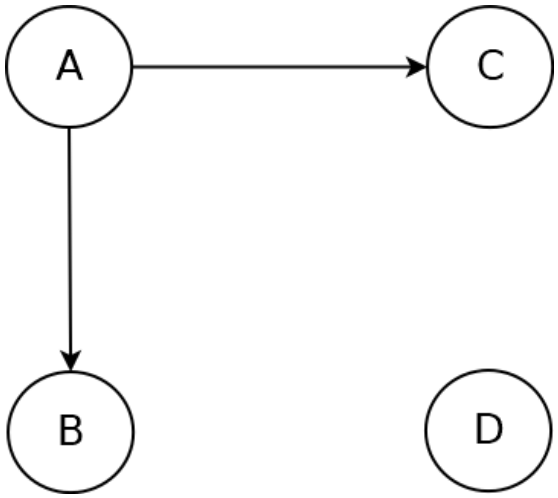
   (a) $\boldsymbol{I(C,A)} \simeq 0.4$

   (b) $\boldsymbol{I(D,A)} = 0$

   (c) $\boldsymbol{I(C,B)} = 0$

   (d) $\boldsymbol{I(D,B)} = 0.2$

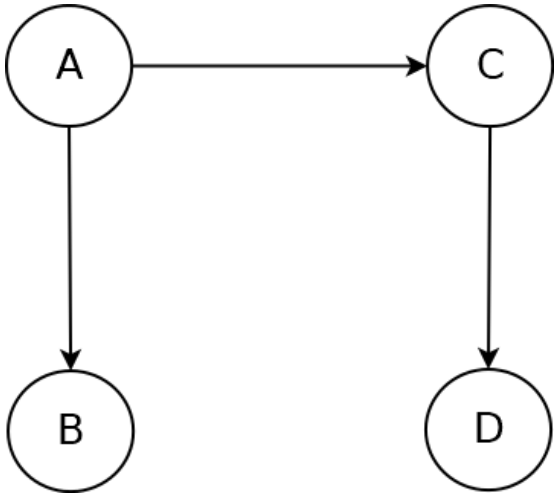
   So the edge with the highest mutual information is $A \to C$ and the Bayesian Network will look like this:



4. *We select the next tree edge. The candidate edges are $A \to D$, $B \to C$ and $B \to D$ from the previous step and since we connected A to C, we also have $C \to D$. We compare the mutual information of these edges* [1] *and it*

[1] It is worth mentioning that if a term of the mutual information sum contains a zero probability, we set the term equal to 0. In other words, we ignore it.

occurs that the edge with the highest mutual information is $C \rightarrow D$. Now all nodes (except the node A from which we began) have 1 parent each, so the 1-degree Bayesian network is complete:



3.5 PrivBayes

3.5.1 Introduction

This section presents *PrivBayes*, the state of the art solution in data publishing first introduced by Zhang et al (21, 1) for releasing a high-dimensional dataset D in an ϵ-differentially private manner. The authors identify that the main problem in publishing high-dimensional data (that consist of a relatively large number of attributes d = |A|) with differential privacy is that the perturbation required inevitably overlaps the signal in the data, rendering it useless in the process. The proposed solution, namely PrivBayes, is inspired from the theory of probabilistic graphical models, and is based on learning the Bayesian Network (directed graphical model) that best fits the data, while satisfying differential privacy. The learned Bayesian Network provides an approximation of the high-dimensional data distribution as a product of low-order conditional and marginal distributions; these low-order distributions contain much more compact signal that is not severely damaged by the required perturbation. Finally, a synthetic dataset is published by sampling tuples (data points) from the approximate distribution. Unlike our approach, PrivBayes uses the non-local definition of Differential Privacy and is applied on one dataset only (non-distributed data). However, the algorithms that we will use are based on PrivBayes and therefore it is important to present it in this thesis. Finally, a synthetic dataset is published by sampling tuples (data points) from the approximate distribution. PrivBayes runs in three phases :

1. **Structure learning phase**: During this phase, the analyst accesses the private data in dataset D and uses an ϵ_1-differentially private algorithm to construct a k-degree Bayesian network over the attributes in D that accurately encodes the conditional independencies that are present in the underlying data distribution (k is a small value that can be either chosen automatically by PrivBayes or by the user).

2. **Parameter learning phase**: During this phase, the analyst utilizes the learned structure and again accesses the sensitive data in D. He uses an ϵ_2-differentially private algorithm to generate the parameters of the learned Bayesian Network, a set of conditional distributions of D, such that for each AP pair (X_i,Π_i) in the network, we have a noisy version of the conditional distribution $\Pr[X_i|\Pi_i]$.

3. **Synthetic data generation phase**: During this phase, no access to the sensitive data is performed. The analyst uses the Bayesian network (constructed in the first phase) and the d noisy conditional distributions (constructed in the second phase) to derive an approximate distribution of the tuples in D and then sample n' tuples (data points) from the approximate distribution (encoded by the Bayesian Network) to generate a synthetic dataset D' of size n'. Since (as we argue in Theorem 3.3) the output of the first two phases satisfies differential privacy, so does the result of the third phase, since no additional access to the data is required) and no additional perturbation is required.

In short, PrivBates utilizes a low-degree Bayesian network to generate a synthetic dataset D' that approximates the high dimensional input data D. The construction of the Bayesian Network is highly non-trivial, as it requires carefully selecting AP pairs and the value of k to derive a close approximation of D without violating differential privacy. By contrast, the second and third phases of PrivBayes are relatively straightforward, especially the third phase. Next, we will clarify the details of these phases, and prove the privacy guarantee of PrivBayes for each of its first two phases using the following theorem:

Theorem 3.3. *Let ϵ_1 and ϵ_2 be the privacy budget consumed during each of the first two PrivBayes respectively. Then, the overall algorithm satisfies (ϵ_1 + ϵ_2)-differential privacy*

Theorem 3.3 is easy to prove using the composition theorem of differential privacy. The choice of ϵ_1 and ϵ_2 determines the balance between the quality of

the learned structure and the learned parameters. The smaller the ϵ parameter, the greater the noise required is. Obviously, greater noise introduces greater bias.[1]

3.5.2 First Phase: Structure Learning

The structure learning phase of PrivBayes is based on a greedy extension of the Chow-Liu algorithm (Algorithm 1) for higher degree Bayesian Networks, which we present in Algorithm 3.

Before we present the private version of the algorithm (Algorithm 3), we describe the non-private version of the algorithm, which can also be found in the PrivBayes paper (21, 1) to help us understand the more advanced algorithms better.

Algorithm 2: GreedyBayes

Input: Dataset D, Bayesian Network degree k

Output: Bayesian Network (Graph) $\mathfrak{G}$

Initialize $\mathfrak{G} = \emptyset$ and $V = \emptyset$
Randomly select an attribute X_1 from A
Add (X_1,$\emptyset$) to $\mathfrak{G}$
Add X_1 to V
for $i = 2$ **to** d **do**
    Initialize $\Omega = \emptyset$
    for *each* $X \in A \setminus V$ *and each* $\Pi \in \binom{V}{k}$ [a] **do**
        Add (X, Π) to Ω
    Select a pair (X_i, Π_{X_i}) from Ω with the maximum mutual information $\hat{I}(X_i, \Pi_{X_i})$
    Add (X_i, Π_{X_i}) to $\mathfrak{G}$ and X_i to V
Return $\mathfrak{G}$

[a] This binominal denotes the set of all subsets of V with size equal to min(k,|V|)

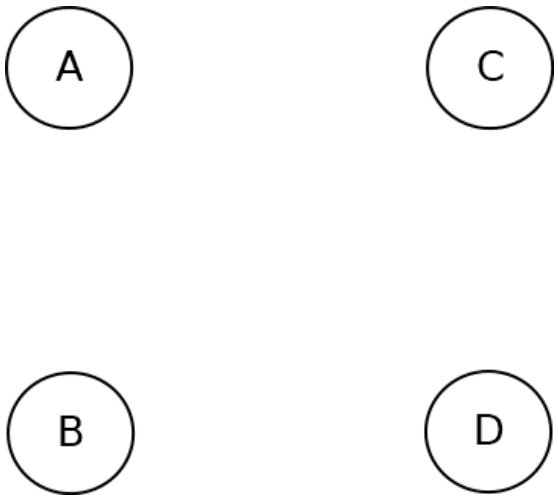
Algorithm 2 aims to build a Bayesian Network from data. Therefore, it is a Structure Learning algorithm. Next, we will present an example where we use Algorithm 2:

Example 3.4. *We are given the same dataset as the previous example. Build a k-degree Bayesian Network that represents the dataset.*

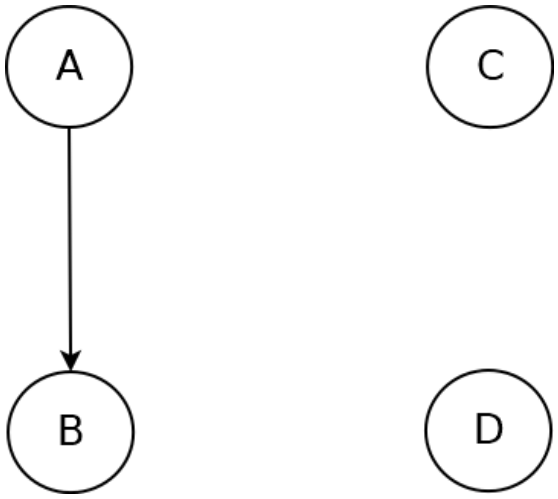
[1] Assuming that the total available privacy budget is ϵ, and by setting $\epsilon_1 = \beta * \epsilon$ and $\epsilon_2 = (1 - \beta) * \epsilon$, the authors of PrivBayes experimentally conclude that the optimal split is achieved for some $\beta \in [0.2, 0.5]$.

Solution:

1. *We start from a random attribute. We choose A in this example. So V=*$\{A\}$ *and our Bayesian Network will look like this:*



2. *We have* $\Omega = \emptyset$ *and then we add (B,A), (C,A) and (D,A) to it, so* $\Omega = \{(B, A), (C, A), (D, A)\}$*. Then we select one of the pairs in* Ω*, the one with the highest mutual information* $\hat{I}$*. This step is the same with the previous example, so the edge with the highest* $\hat{I}$ *is* A $(\Pi_2) \rightarrow B$ (X_2)*,* $V = \{A, B\}$ *and the Bayesian Network looks like:*

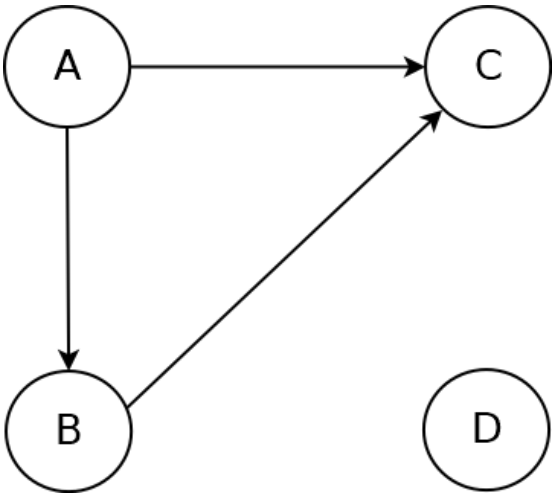


3. *We have* $\Omega = \emptyset$ *and then since* $V = \{A, B\}$*, we add (C,{A,B}), (D,{A,B}) to it. We calculate the mutual information of each pair:*

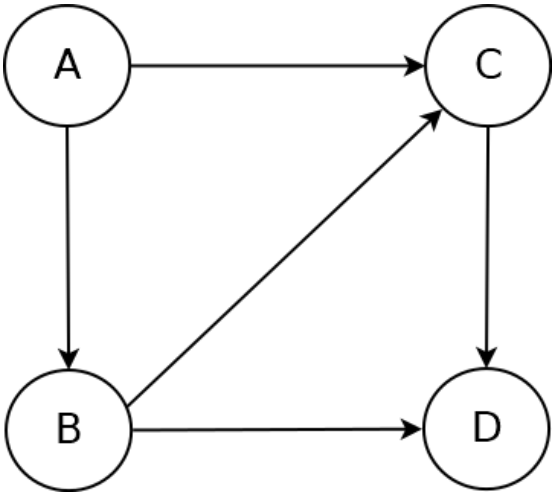
   (a) *I(C,{A,B}) = I(C,A) + I(C,B) = 0.4*

   (b) *I(D,{A,B}) = I(D,A) + I(D,B) = 0.2*

   So we choose (C,{A,B}), $V = \{A, B, C\}$ *and the Bayesian Network looks like this:*



4. *We have* $\Omega = \emptyset$ *and then since* $V = \{A, B, C\}$*, we add (D,{A,C}), (D,{A,B}), (D,{B,C}) to it. We calculate the mutual information of each pair and as a result, we choose (D,{B,C}). So* $V = \{A, B, C, D\}$ *and the (final) Bayesian Network looks like this:*



Now we are ready to present the *private* version of *GreedyBayes*, which uses the exponential mechanism (see Chapter 2) to achieve differential privacy:

Algorithm 3: Private Greedy Bayesian Network Structure Learning

Input: Dataset D, Bayesian Network degree k, Privacy budget ϵ_1

Output: Bayesian Network (Graph) $\mathfrak{G}$

1 Initialize $\mathfrak{G} = \emptyset$ and $V = \emptyset$
2 **for** *each* $A' \subseteq A$ *such that* $|A'| \leq k+1$ **do**
3 Estimate and store $\hat{p}_{A'}$ from D
4 Arbitrarily select an attribute X from A
5 Add (X,$\emptyset$) to $\mathfrak{G}$ and X to V
6 **for** $i = 1$ **to** *d-1* **do**
7 Initialize $\Omega = \emptyset$
8 **for** *each* $X \in A \setminus V$ **do**
9 **for** *each* $\Pi_X \subseteq V$ *such that* $|\Pi_X| \leq k$ **do**
10 Calculate $\hat{I}(X, \Pi_X)$ using $\hat{p}_X, \hat{p}_{\Pi_X}$ and $\hat{p}_{X,\Pi_X}$
11 Add $(X, \Pi_X, \hat{I}(X, \Pi_X))$ to Ω
12 Sample a tuple $(X, \Pi_X, \hat{I}(X, \Pi_X))$ from Ω with probability $\propto \frac{\hat{I}(X,\Pi_X)*\epsilon_1}{2(d-1)*\Delta\hat{I}}$
13 Add (X, Π_X) to $\mathfrak{G}$ and X to V
14 Return $\mathfrak{G}$

The output $\mathfrak{G}$ of Algorithm 3 consists of the d - 1 attribute-parent pairs each of which was added using the exponential mechanism (Line 12) with the empirical mutual information as scoring function and with privacy budget $\frac{\epsilon_1}{d-1}$. Thus, Theorem 3.4 directly follows (by the composition theorem):

Theorem 3.4. *Algorithm 3 satisfies ϵ_1-differential privacy.*

Also to implement the algorithm, we need a mathematical formula for the sensitivity of mutual information $\hat{I}$, which we derive from the following lemma:

Lemma 3.1. *For any random variables X and Y, the sensitivity of $\hat{I}(X, Y)$ is :*

$$\Delta\hat{I} = \begin{cases} \frac{1}{n}\log n + \frac{n-1}{n}\log\frac{n}{n-1} & \text{if } X \text{ or } Y \text{ is binary} \\ \frac{2}{n}\log\frac{n+1}{2} + \frac{n-1}{n}\log\frac{n+1}{n-1} & \text{otherwise} \end{cases}$$

Lemma 3.1 quantifies the sensitivity of the empirical mutual information; the proof can be found in the full PrivBayes paper (21, 1). We remark that the sensitivity is computed using the definition of adjacent datasets that leads to

bounded differential privacy, so $\Delta\hat{I}$ expresses the maximum change in the empirical mutual information that is caused by changing the value of one data point.

However, we are interested in applying the PrivBayes algorithm on distributed data. As a result, instead of having one dataset D and one data holder/owner, we have many datasets $\{D_1, ..., D_M\}$ and M respective data holders/owners, as we can also see in Figure 5. We also need to alter our existing algorithms, so that they operate correctly in the distributed model. In order for us to do so, we must first re-formulate the optimization problem of finding a k-degree Bayesian Network, that represents the multiple datasets (distributed among an equal number of data owners) in a greedy manner :

$$\max_{\mathfrak{B}}(log(\mathfrak{L}(\mathfrak{B}, D_1, ..., D_M))) = \max_{\mathfrak{G}} \sum_{i=1}^{d} \hat{I}(X_i, \Pi_i) =$$

$$\max_{\mathfrak{G}} \sum_{i=1}^{d} \sum_{x \in d_{X_i}, p \in d_{\Pi i}} \frac{\sum_{j=1}^{M} c^{(j)}_{X_i,\Pi_i}(x,p)}{n} * log_2 \frac{n * \sum_{j=1}^{M} c^{(j)}_{X_i,\Pi_i}(x,p)}{\sum_{j=1}^{M} c^{(j)}_{X_i}(x) * \sum_{j=1}^{M} c^{(j)}_{\Pi_i}(p)}$$

The key question that needs to be answered is what information each data owner shares with the analyst. Furthermore, depending on the answer to this question, we need to figure out how to combine the information shared by different data holders. We attempt to answer this question with the following approaches:

First Approach: Sharing the Noisy Sufficient Statistics

The first approach is based on asking each data holder to share its part of the sufficient statistics, that is, all (k + 1)-dimensional frequency distributions. In doing so, the analyst is able to compose the global sufficient statistics (since the empirical frequency distribution is composable by simply summing the counts), and evaluate the scoring function for all candidate structures. In that sense, this approach, which we call Sharing the Noisy Sufficient Statistics (Algorithm 4), is an exact approach. We will also refer to this algorithm as **Structure Learning 1** during the experimental evaluation.

Algorithm 4: Sharing the Noisy Sufficient Statistics

Input: Datasets $D_1, ..., D_M$, Bayesian Network degree k
Output: Bayesian Network (Graph) $\mathfrak{G}$

Initialize $\mathfrak{G} = \emptyset$ and $V = \emptyset$
for *each* $A' \subseteq A$ *such that* $|A'| \leq k+1$ **do**
  for $j = 1$ **to** M **do**
    QUERY(D_j): retrieve local $\tilde{c}^{(j)}_{A'}$
  Estimate global $\tilde{p}_{A'} = \frac{1}{n_{noisy}} \sum_{j=1}^{M} \tilde{c}^{(j)}_{A'}$
Arbitrarily select an attribute X from A
Add (X,$\emptyset$) to $\mathfrak{G}$ and X to V
for $i = 1$ **to** *d-1* **do**
  Initialize $\Omega = \emptyset$
  for *each* $X \in A \setminus V$ **do**
    for *each* $\Pi_X \subseteq V$ *such that* $|\Pi_X| \leq k$ **do**
      Calculate $\hat{p}_{X,\Pi_X}$ using the classical definition of probability and $\hat{p}_X$, $\hat{p}_{\Pi_X}$ by marginalizing the proper distributions
      Calculate $\hat{I}(X, \Pi_X)$ using $\hat{p}_X, \hat{p}_{\Pi_X}$ and $\hat{p}_{X,\Pi_X}$
      Add $(X, \Pi_X, \hat{I}(X, \Pi_X))$ to Ω
  Select (X, Π_X) with the highest $\hat{I}(X, \Pi_X)$
  Add (X, Π_X) to $\mathfrak{G}$ and X to V
Return $\mathfrak{G}$

Advantages :

1. Once the analyst collects the frequency distributions, it does not need to access the data again in any of the phases.

2. Once the Structure Learning phase is complete, the parameter learning Algorithm 8 utilizes the already-retrieved distributions and **hence the entire privacy budget can be consumed in the Structure Learning phase.**

Disadvantages :

1. Each data holder has to share $\binom{d}{k+1}$ frequency distributions, which may be prohibitive for high-degree Bayesian Networks, in terms of both perturbation and communication cost, as the amount noise required also significantly damages the utility of the data and the bandwidth required to share the distributions is quite large.

Before we proceed to the second approach, we need to clarify some important points about Algorithm 4.

First of all, when we calculate the probabilities of the (k+1)-sized subsets A' of the attribute set A, we divide the sum of counts by a value we call n_{noisy} instead of

the dataset size n, that one might expect. However doing so is necessary so that the probabilities are properly normalized. For each $A' \subseteq A$, such that |A|= k + 1, each data holder computes the (local) joint frequency distribution of the attributes (counts) in A'. Therefore, if $\mathbf{c}^{(j)}$ is the distribution that holder $j \in \{1, ..., M\}$ computes, then, assuming that $|d_{A'}| = z$, holder j shares the following z-dimensional **vector**:

$$\tilde{\mathbf{c}}^{(j)} = \mathbf{c}^{(j)} + \boldsymbol{\eta}^{(j)} = [c_1^{(j)} \ ... \ c_z^{(j)}] + [\eta_1^{(j)} \ ... \ \eta_z^{(j)}] = [\tilde{c}_1^{(j)} \ ... \ \tilde{c}_z^{(j)}]$$

where $\mathbf{n}$ is the Laplacian noise. Then, the analyst collects the noisy vectors $\tilde{\mathbf{c}}^{(1)}$, ... , $\tilde{\mathbf{c}}^{(M)}$ (one vector for each data holder) and merges them as:

$$\tilde{\mathbf{c}} = \sum_{j=1}^{M} \tilde{\mathbf{c}}^{(j)} = \sum_{j=1}^{M} (\mathbf{c}^{(j)} + \boldsymbol{\eta}^{(j)}) = \mathbf{c} + \sum_{j=1}^{M} \boldsymbol{\eta}^{(j)}$$

As a consequence, in order to *estimate* the corresponding probability distribution correctly, we use:

$$\tilde{\mathbf{p}} = \frac{1}{n_{noisy}} * \tilde{\mathbf{c}}$$

where n_{noisy} is :

$$n_{noisy} = \sum_{i=1}^{d} \sum_{j=1}^{M} \tilde{c}_i^{(j)} = \sum_{i=1}^{d} \sum_{j=1}^{M} (c_i^{(j)} + \eta_i^{(j)}) = n + \sum_{i=1}^{d} \sum_{j=1}^{M} \eta_i^{(j)}$$

So n_{noisy} is the sum of the actual total dataset size n, plus the sum of z x M zero-mean Laplace random variables and, hence, it is also random with E[n_{noisy}] = n.

Finally, we need to quantify the amount of noise we will add to the counts. In order to ensure ϵ-Differential Privacy, the noise will be Laplacian with mean equal to 0 and scale b equal to $\frac{2*\binom{d}{k+1}}{\epsilon}$. This is indicated by the following theorem:

Theorem 3.5. *Let* $b = \frac{2*\binom{d}{k+1}}{\epsilon}$. *If* $\forall A' \subseteq A$ *such that* $|A'| = k + 1$, *each data holder shares* $\tilde{\mathbf{c}}_{A'} = \mathbf{c}_{A'} + \boldsymbol{\eta}$ *where* $\boldsymbol{\eta} = [\eta_1 \ ... \ \eta_z]$ *is a random vector of i.i.d. Laplace(0,b) entries, then Algorithm 4 preserves* ϵ*-differential privacy for any dataset* D_j $(j \in \{1, ..., M\})$.

Proof is in (29, 9), page 26.

Second Approach: Noisy Majority Voting

Given the serious disadvantage of the previous method, we propose a second answer (Algorithm 5), which is based on the notion of majority voting from the machine learning literature. Specifically, each data holder incrementally reports the highest mutual information attribute-parent pair that it would add to the Bayesian Network. The analyst collects all votes, and adds the most-voted pair to the structure. We will also refer to this algorithm as **Structure Learning 2** during the experimental evaluation.

Algorithm 5: Noisy Majority Voting

Input: Datasets $D_1, ..., D_M$, Bayesian Network degree k

Output: Bayesian Network (Graph) $\mathfrak{G}$

```
1  Initialize 𝔊 = ∅ and V = ∅
2  Arbitrarily select an attribute X from A
3  Add (X,∅) to 𝔊
4  Add X to V
5  for i = 1 to d-1 do
6      Initialize multi-set votes = ∅
7      for j = 1 to M do
8          QUERY(D_j): Select (X, Π_X) with the highest Î(X, Π_X), subject to
             X ∈ A \ V and Π_i ⊆ V
9          Add (X, Π_X) to multi-set votes
10     Find most-voted (X, Π_X) in multi-set votes (break ties arbitrarily)
11     Add (X, Π_X) to 𝔊 and X to V
12 Return 𝔊
```

Each data holder responds to the queries on its dataset using the exponential mechanism with privacy budget $\frac{\epsilon_1}{d-1}$. The probability with which a user chooses the pair with the "highest" mutual information I is exactly the one we use in Algorithm 3 (Line 12).

Once we have received the Bayesian Network, Algorithm 8 is used to learn its parameters. If we use Algorithm 5 during the Structure Learning Phase, the analyst now has to retrieve the d required frequency distributions to estimate the parameters (*retrieved* == False), and the data holders respond to its queries using the Laplace mechanism, with privacy budget $\frac{\epsilon_2}{d}$. If we use Algorithm 4, we are already in possession of the required distributions and retrieving them during Parameter Learning is unnecessary (*retrieved* == True) and no further privacy budget needs to be consumed.

Theorem 3.6. *Let ϵ_1 and ϵ_2 be the total privacy budget that each data holder uses in responding to the analyst's queries, during the structure learning phase (Algorithm 5) and the parameter learning phase (Algorithm 8 with input retrieved = False) respectively. Then, the overall algorithm satisfies $(\epsilon_1+\epsilon_2)$-differential privacy for any dataset D_j $(j \in \{1, ..., M\})$.*

Third Approach: Sharing the Noisy Model

Finally, each data holder locally executes PrivBayes on its dataset. In other words, each user uses PrivBayes (Algorithm 3 for Structure Learning and Algorithm 7 for Parameter Learning) to build a Bayesian Network and learn its parameters. The analyst then generates, for each dataset, a synthetic dataset using the Prior Sampling technique.

Each dataset is of size proportional to that of the local dataset that was used to learn the local model ($n'_j = n_j$). Then the analyst merges the smaller datasets to produce the final synthetic dataset with size of n. We will also refer to this algorithm as **Structure Learning 3** during the experimental evaluation.

Algorithm 6: Sharing the Noisy Model

Input: Datasets $D_1, ..., D_M$, Bayesian Network degree k

Output: Bayesian Network (Graph) $\mathfrak{G}$

```
Initialize D_synth = ∅
for k = 1 to M do
    QUERY(D_j): get local model structure G_(j), parameters Θ_j and dataset
      size n_j
    Generate D_j,synth using Prior Sampling such that |D_j,synth| = n_j
    Add D_j,synth to D_synth
Return D_synth
```

Each data holder constructs its local model using Algorithms 3 and 7, which were shown to jointly satisfy differential privacy (Theorem 3.3). The privacy budget for Structure Learning is ϵ_1 and for Parameter Learning is ϵ_2 ($\epsilon_1 + \epsilon_2 = \epsilon$). A disadvantage of this method is that much is required from the data holders. However, it can be considered much safer than the others, given that the analyst never accesses the real data or its frequency distributions, as it did in previous methods. Since differential privacy is immune to post-processing, we have the following theorem:

Theorem 3.7. *Assume that each data holder shares an ϵ-differentially private local model. Then, the global model that results by aggregating the local models according to Algorithm 7, also also satisfies ϵ-differential privacy for any dataset D_j ($j \in \{1, ..., M\}$).*

3.5.3 Second Phase: Parameter Learning

Algorithm 7: Bayesian Network Parameter Learning

Input: Dataset D, Bayesian Network graph $\mathfrak{G}$, Privacy budget ϵ_2

Output: Bayesian Network parameters Θ

1 Initialize $\Theta_i = \emptyset$, $\forall i \in \{1, ..., N\}$
2 **for** $i = 1$ **to** d **do**
3 Estimate $\hat{p}_{X_i,\Pi_{X_i}}$
4 Calculate noisy $\tilde{p}_{X,\Pi_{X_i}} = \hat{p}_{X,\Pi_{X_i}} + \text{Laplace}(0, \frac{2*d}{n*\epsilon_2})$
5 Set negative values in $\tilde{p}_{X_i,\Pi_{X_i}}$ to 0 and normalize
6 Calculate $\tilde{\Theta}_i = \tilde{p}_{X_i|\Pi_{X_i}}$ by marginalizing $\tilde{p}_{X_i,\Pi_{X_i}}$
7 Add $\tilde{\Theta}_i$ to $\tilde{\Theta}$
8 Return $\tilde{\Theta}$

The output of Algorithm 7 consists of the d conditional probability tables $\tilde{\Theta}_i$, $i \in \{1, ..., d\}$, each of which represents the set of conditional distributions of an attribute, given all realizations of its parents. Notice that each $\tilde{\Theta}_i$ is constructed using a noisy version of the maximum likelihood estimate of the joint probability distribution of attribute X_i and its parents. In total, we estimate d noisy distributions using the Laplace mechanism (Line 4), each with privacy budget $\frac{\epsilon_2}{d}$, so (again) by the composition theorem:

Theorem 3.8. *Algorithm 7 satisfies ϵ_2-differential privacy.*

As we did before during the first phase, we must adjust the second phase of PrivBayes (Algorithm 7) to the distributed model. As we mentioned before, depending on the structure learning approach used, the analyst may need or need not re-access the data to estimate the Bayesian Network parameters, depending on whether it has already retrieved the required local distributions or not. Therefore, we introduce the boolean parameter *retrieved* that indicates whether the analyst already possesses the distributions $\tilde{p}_{X_i}, \tilde{p}_{\Pi_i}$ *and* $\tilde{p}_{X_i|\Pi_i}$ $\forall i \in \{1, ..., d\}$ or not.

Algorithm 8: Distributed Bayesian Network Parameter Learning

Input: Datasets $D_1, ..., D_M$, Bayesian Network graph $\mathfrak{G}$, Boolean Variable *retrieved*

Output: Bayesian Network parameters Θ

Initialize $\Theta_i = \emptyset$, $\forall i \in \{1, ..., d\}$
for $i = 1$ **to** d **do**
  if *retrieved == False* **then**
    for $j = 1$ **to** M **do**
      QUERY(D_j): retrieve local $\tilde{c}^{(j)}_{X_i,\Pi_i}$
    Estimate $\tilde{p}_{X_i,\Pi_i} = \frac{1}{n}\sum_{j=1}^{M} \tilde{c}^{(j)}_{X_i,\Pi_i}$
  Calculate $\tilde{\Theta}_i = \tilde{p}_{X_i|\Pi_{X_i}}$ by marginalizing $\tilde{p}_{X_i,\Pi_{X_i}}$
  Add $\tilde{\Theta}_i$ to $\tilde{\Theta}$
Return $\tilde{\Theta}$

Notice that since the analyst is not trusted, it is **the data holders' responsibility** to properly perturb their local frequency distributions, and handle any negative frequencies that may appear, prior to sharing them. This is exactly what was done in the distributed Structure Learning Algorithm 4.

3.5.4 Third Phase: Synthetic Data Generation

Since we now have the k-degree Bayesian Network G and its parameters $\tilde{\Theta}_i$ (conditional probabilities), we are finally ready to generate our synthetic dataset. The size n' of the new dataset D' will be determined by the user, but it usually is equal to the size of the original dataset. We can accomplish this by using a variety of simple but useful *sampling techniques*. Before we speak of the techniques that will concern us, we will speak of *what sampling is* and will present an example.

Sampling is not an easy problem. Our computers can only generate samples from very simple distributions, such as the uniform distribution over [0,1). Even those samples are not truly random. They are actually taken from a deterministic sequence whose statistical properties (e.g. running averages) are indistinguishable from a truly random one. We call such sequences *pseudorandom*. All sampling techniques involve calling some kind of simple subroutine multiple times in a clever way. In our case, we may reduce sampling from a multinomial variable to sampling a single uniform variable by subdividing a unit interval into k regions with region i having size θ_i. We then sample uniformly from [0,1] and return the value of the region in which our sample belongs to.

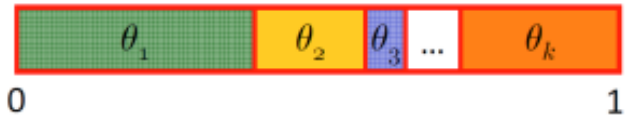


Figure 7: Sampling from the uniform distribution over [0,1)

We can also express the above process as an algorithm:

Algorithm 9: Sampling from a given distribution

Input: A distribution over a number of random variables

Output: Sample

1 Get a sample z from the uniform distribution over [0,1]

2 Convert this sample u into an outcome for the given distribution by having each outcome i associated with a sub-interval of [0,1) (θ_i) with sub-interval size equal to probability of the outcome.

Example 3.5. *The following probability distribution is given:*

W	P(W)
Rain	0.3
Sunny	0.4
Cloudy	0.2
Foggy	0.1

We implement Algorithm 9 and get a sample from the uniform distribution equal to $z = 0.67$. We can receive a sample from interval [0,1) by running the random() function using any programming language. Next, we compute the sub-intervals based on the given distribution:

- $[0, 0.3) \rightarrow W = Rain$
- $[0.3, 0.7) \rightarrow W = Sunny$
- $[0.7, 0.9) \rightarrow W = Cloudy$
- $[0.9, 1) \rightarrow W = Foggy$

Finally, since $z = 0.67$, we have $z \in [0.3, 0.7)$, so our sample is ***W=Sunny****. If we wish to sample more times, we simply repeat the first and the final step.*

Before we proceed to the sampling methods, we will introduce the *Big-O notation* as described by Cormen et al (24, 4):

Definition 3.5. *For a given function g(n), we denote by O(g(n)) the set of functions:* $O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c * g(n) \text{ for all } n \geq n_0\}$

We use O-notation to give an upper bound on a function, to within a constant factor and we write We write f(n) = O(g(n)) to indicate that a function f(n) is a member of the set O(g(n)).

The sampling methods that will concern us are:

1. **Prior (or Forward) Sampling**: Given a probability $p(x_1, ..., x_n)$ specified by a Bayes net, we sample variables in topological order. We start by sampling the variables with no parents; then we sample from the next generation by conditioning these variables' CPDs to values sampled at the first step. We proceed like this until all n variables have been sampled. Importantly, in a Bayesian network over n variables, forward sampling allows us to sample from the joint distribution $x \sim p(x)$ in linear O(n) time by taking by taking exactly 1 multinomial sample from each CPD. "Forward sampling" can also be performed efficiently on undirected models if the model can be represented by a clique tree with a small number of variables per node. Calibrate the clique tree, which gives us the marginal distribution over each node, and choose a node to be the root. Then, marginalize over variables in the root node to get the marginal for a single variable. Once the marginal for a single variable x $\sim p(X_1|E = e)$ has been sampled from the root node, the newly sampled value $X_1 = x_1$ can be incorporated as evidence. Finish sampling other variables from the same node, each time incorporating the newly sampled nodes as evidence, i.e. $x \sim p(X_2 = x_2|X_1 = x_1, E = e)$ and $x \sim p(X_3 = x_3|X_1 = x_1, X_2 = x_2, E = e)$ and so on. When moving down the tree to sample variables from other nodes, each node must send an updated message containing the values of the sampled variables.

2. **Rejection Sampling**: It is is a basic technique used to generate observations from a distribution. It is also commonly called the acceptance-rejection method or "accept-reject algorithm" and is a type of exact simulation method. The key idea is to reject a sample once an evidence variable has been sampled to take on a value inconsistent with the evidence of the query. The method works for any distribution in $\mathbb{R}^m$ with a density. Rejection sampling is based on the observation that to sample a random variable in one dimension, one can perform a uniformly random sampling of the two-dimensional Cartesian graph, and keep the samples in the region under the graph of its density function. This property can be extended to N-dimension functions. The rejection sampling method generates sampling values from a target distribution X with arbitrary probability density function f(x) by using a proposal distribution Y with probability density g(x). There are a number of extensions to this algorithm, such as the Metropolis algorithm and the combination with ratio-of-uniforms approach. If we know the query we wish to perform on the data in advance, this is a more efficient method.

3. **Likelihood Weighting**: Instead of creating a sample and then rejecting it,

it is possible to mix sampling with inference to reason about the probability that a sample would be rejected. In importance sampling methods, each sample has a weight, and the sample average is computed using the weighted average of samples. Likelihood weighting is a form of importance sampling where the variables are sampled in the order defined by a belief network, and evidence is used to update the weights. The weights reflect the probability that a sample would not be rejected. So rather than sampling the evidence variables, force them to be consistent with the evidence and the re-weight the sample to account for the CPTs [1] of the evidence variables. Like with Rejection Sampling, this method is more efficient, if we know the query in advance. The process is described by the following algorithm:

Algorithm 10: Likelihood Weighting

Input: Bayesian Network graph $\mathfrak{G}$, Evidence e, Query variable Q, Number n of samples to generate

Output: Posterior Distribution over Q

```
Initialize arrays sample and counts to 0
for i = 1 to n do
    sample = ∅
    weight = 1
    for each variable X ∈ B, in order do do
        if X = V is in e then
            sample[X] = v
            weight = weight * P(X = V)
        else
            sample[X] = a random sample from P(X/Π_X)
    v = sample[Q]
    counts[v] = counts[v] + weight
Return counts / (∑_v counts[v])
```

To implement the aforementioned sampling techniques, we must first sort the nodes of the generated Bayesian Network using the *Topological Sorting* definition (25, 5):

Definition 3.6. *In computer science, a topological sort or topological ordering of a*

[1] In statistics, the conditional probability table (CPT) is defined for a set of discrete and mutually dependent random variables to display conditional probabilities of a single variable with respect to the others (i.e., the probability of each possible value of one variable if we know the values taken on by the other variables). For example, assume there are three random variables x_1, x_2, x_3 where each has K states. Then, the conditional probability table of x_1 provides the conditional probability values $P(x_1 = a_k \mid x_2, x_3)$ – where the vertical bar $\mid$ means "given the values of" – for each of the K possible values a_k of the variable x_1 and for each possible combination of values of x_2, x_3. This table has K^3 cells. In general, for M variables $x_1, x_2, \ldots, x_M x_1, x_2, \ldots, x_M$ with K_i states for each variable x_i, the CPT for any one of them has the number of cells equal to the product $K_1 K_2 \cdots K_M$. The tables 2, 3 and 4 of Example 3.2 are CPTs.

directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering.

Topological sorting arises as a subproblem in most algorithms on directed acyclic graphs. Topological sorting orders the vertices and edges of a DAG in a simple and consistent way and hence plays the same role for DAGs that a depth-first search does for general graphs. Topological sorting can be used to schedule tasks under precedence constraints. Suppose we have a set of tasks to do, but certain tasks have to be performed before other tasks. These precedence constraints form a directed acyclic graph, and any topological sort (also known as a linear extension) defines an order to do these tasks such that each is performed only after all of its constraints are satisfied.

Some important facts about topological sorting are:

1. Only DAGs (like the Bayesian Networks we utilize) can be topologically sorted, since any directed cycle provides an inherent contradiction to a linear order of tasks.
2. Every DAG can be topologically sorted, so there must always be at least one schedule for any reasonable precedence constraints among jobs.
3. DAGs can often be topologically sorted in many different ways, especially when there are few constraints. Consider n unconstrained jobs. Any of the n! permutations of the jobs constitutes a valid topological ordering.

Now we will present an example for each of the sampling techniques mentioned:

Example 3.6. ***(Prior Sampling):*** *We are given the following Bayesian Network:*

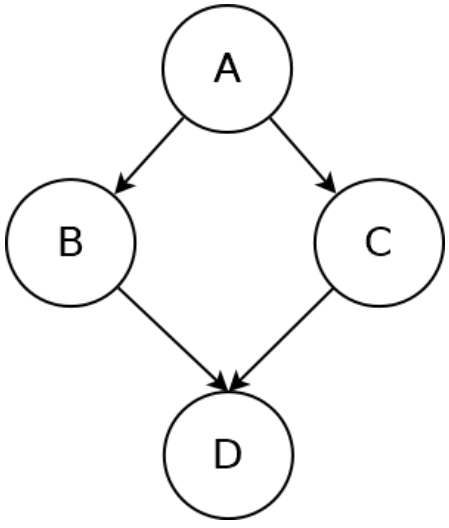


Figure 8: 2-degree Bayesian Network with 4 states (attributes)

and the following values for its parameters (conditional probabilities):

A	P(A)
1	0.8
0	0.2

A	B	P(B|A)
1	1	0.8
1	0	0.2
0	1	0.5
0	0	0.5

A	C	P(C|A)
1	1	0.7
1	0	0.3
0	1	0.1
0	0	0.9

B	C	D	P(D|B,C)
1	1	1	0.3
1	1	0	0.7
1	0	1	0.1
1	0	0	0.9
0	1	1	0.2
0	1	0	0.8
0	0	1	0.9
0	0	0	0.1

All the attributes of the example have binary values.

1. *Generate a synthetic dataset of n' = 10 tuples.*
2. *Calculate* ***P(D = 1), P(A = 0, C = 1), P(A = 1, B = 0, C = 1, D = 0), P(B = 0 | C = 1) and P(D = 0 | A = 0, C = 1)*** *based on the generated dataset.*

Solution:

1. *First we need to decide the (topological) order with which we will calculate the attributes. There are two possible orders: i){A,B,C,D} ii){A,C,B,D}. Both are correct, but in this example we will work with the former. Next, as we have 4 attributes and we need 10 tuples, we will get 40 samples from the uniform distribution over [0,1). The samples that we received are {0.59,0.11,0.7,0.07 ...}. Now that we have our samples, we can begin generating the tuples:*
   (a) *We begin with attribute A of the first tuple (due to the {A,B,C,D} order). In order to sample for this attribute, we turn to the P(A) table, where the attribute is the query.*[1] *As a result, we use the first table. As we did in the previous example, we split interval [0,1) to sub-intervals according to the distribution table. So we have:*
      - $[0, 0.8) \rightarrow A = 1$
      - $[0.8, 1) \rightarrow A = 0$

[1] If we have the conditional probability P(A|B,C), then we define *query* = *A* and *evidence* = *(B,C)*

Since our first sample is $0.59 \in [0, 0.8) \rightarrow A = 1$

(b) *For attribute B, we use the P(B|A) table and we partition the interval [0,1) in two ways, one for each value the evidence A takes. So for A = 1 we have:*

- $[0, 0.8) \rightarrow B = 1$
- $[0.8, 1) \rightarrow B = 0$

and for A = 0 :

- $[0, 0.5) \rightarrow B = 1$
- $[0.5, 1) \rightarrow B = 0$

Since A = 1 for the current tuple, we use the first partitions. The second sample is $0.11 \in [0, 0.8) \rightarrow B = 1$

(c) *For attribute C, we use the P(C|A) table and we partition the interval [0,1) in two ways, one for each value the evidence A takes. So for A = 1 we have:*

- $[0, 0.7) \rightarrow C = 1$
- $[0.7, 1) \rightarrow C = 0$

and for A = 0 :

- $[0, 0.1) \rightarrow C = 1$
- $[0.1, 1) \rightarrow C = 0$

Since A = 1 for the current tuple, we use the first partitions. The third sample is $0.7 \in [0.7, 1) \rightarrow C = 0$

(d) *For attribute D, we use the P(D|B,C) table and we partition the interval [0,1) in four ways, one for each value the evidence (B,C) takes. So for (B,C) = (1,1) we have:*

- $[0, 0.3) \rightarrow D = 1$
- $[0.3, 1) \rightarrow D = 0$

for (B,C) = (1,0) :

- $[0, 0.1) \rightarrow D = 1$
- $[0.1, 1) \rightarrow D = 0$

for (B,C) = (0,1) :

- $[0, 0.2) \rightarrow D = 1$
- $[0.2, 1) \rightarrow D = 0$

and for (B,C) = (0,0) :

- $[0, 0.9) \rightarrow D = 1$
- $[0.9, 1) \rightarrow D = 0$

Since (B,C) = (1,0) for the current tuple, we use the second partitions. The fourth sample is $0.07 \in [0, 0.1) \rightarrow D = 0$.

Consequently, the first generated tuple is (A,B,C,D) = (1,1,0,1). We continue the process by calculating the attribute A of the second tuple using the fifth sample and so on, until we have our 10 tuples.

The final result is:

A	B	C	D
1	1	0	1
1	0	0	0
1	0	1	1
1	0	1	0
0	1	1	0
0	1	0	0
0	1	1	1
0	0	0	0
1	1	0	1
0	0	0	0

2. (a) $\boldsymbol{P(D = 1)} = \frac{Tuples\ with\ D{=}1}{Tuples} = \frac{4}{10}$

   (b) $\boldsymbol{P(A = 0,\ C = 1)} = \frac{Tuples\ with\ (A,C){=}(0,1)}{Tuples} = \frac{2}{10}$

   (c) $\boldsymbol{P(A = 1,\ B = 0,\ C = 1,\ D = 0)} = \frac{Tuples\ with\ (A,B,C,D){=}(1,0,1,0)}{Tuples} = \frac{1}{10}$

   (d) $\boldsymbol{P(B = 0 \mid C = 1)} = \frac{Tuples\ with\ (B,C){=}(0,1)}{Tuples\ with\ C{=}1} = \frac{2}{4}$

   (e) $\boldsymbol{P(D = 0 \mid A = 0,\ C = 1)} = \frac{Tuples\ with\ (A,C,D){=}(0,1,0)}{Tuples\ with\ (A,C){=}(0,1)} = \frac{1}{2}$

Example 3.7. ***(Rejection Sampling):*** *We are given the Bayesian Network and the parameters of the previous example. Perform the Rejection Sampling technique for the query:* $\boldsymbol{P(D = 0 \mid B = 1)}$.

Solution:

First we need to decide the (topological) order with which we will calculate the attributes. There are two possible orders: i){A,B,C,D} ii){A,C,B,D}. Both are correct, but in this example we will work with the former. Next, we once again get samples from the uniform distribution over [0,1). The samples that we received are {0.14,0.87,0.69, ...}. Now that we have our samples, we can begin generating the tuples:

1. *We begin with attribute A of the first tuple (due to the {A,B,C,D} order). In order to sample for this attribute, we turn to the P(A) table, where the attribute is the query. As a result, we use the first table. We split interval [0,1) to sub-intervals according to the distribution table. So we have:*

   - $[0, 0.8) \rightarrow A = 1$
   - $[0.8, 1) \rightarrow A = 0$

Since our first sample is $0.14 \in [0, 0.8) \rightarrow A = 1$

2. *For attribute B, we use the P(B|A) table and we partition the interval [0,1) in two ways, one for each value the evidence A takes. So for A = 1 we have:*

   - $[0, 0.8) \rightarrow B = 1$
   - $[0.8, 1) \rightarrow B = 0$

   and for A = 0 :

   - $[0, 0.5) \rightarrow B = 1$
   - $[0.5, 1) \rightarrow B = 0$

   Since A = 1 for the current tuple, we use the first partitions. The second sample is $0.87 \in [0.8, 1) \rightarrow B = 0$. *However, B = 1 in the query, so we reject this tuple and begin working on a new one, starting again with attribute A and using the 0.69 sample.*

In other words, we follow the same algorithm as in Example 3.6, except for the fact that we will reject tuples that do not agree with the evidence of the query we are given, tuples who have B = 0.

Example 3.8. ***(Likelihood Weighting):*** *We are given the Bayesian Network and the parameters of the previous examples. Perform the Likelihood Weighting technique for the query:* ***P(D = 1 | A = 0, B = 1)***.

Solution:

First we need to decide the (topological) order with which we will calculate the attributes. There are two possible orders: i){A,B,C,D} ii){A,C,B,D}. Both are correct, but in this example we will work with the former. Next, we once again get samples from the uniform distribution over [0,1). The samples that we received are {0.98,0.52, ...}. Now that we have our samples, we can begin generating the tuples:

1. *We begin with attribute A of the first tuple (due to the {A,B,C,D} order). In order to sample for this attribute, we turn to the P(A) table, where the attribute is the query. As a result, we use the first table. However, since attribute A is included in the query evidence, we force the tuple to have the value of the query for this attribute. As a result,* $\boldsymbol{A = 0}$. *We also set a weight for this attribute (for the current tuple) equal to 0.2, because P(A = 0) = 0.2 according to the P(A) table.*

2. *For attribute B, we use the P(B|A) table. Again, since attribute B is included in the query evidence, we force the tuple to have the value of the query for this*

attribute. As a result, $\boldsymbol{B = 1}$. *We also set a weight for this attribute (for the current tuple) equal to 0.5, because P(B = 1 |A = 0) = 0.5 according to the P(B|A) table.*

3. *C and D are not evidence variables, so we use our samples (0.98 and 0.52 respectively) to decide their values, like we did in Example 3.2. We set a weight equal to 1 to these attributes. So the total weight of this tuple is the product of the weights of all its attributes: W = 0.2 * 0.5 * 1 * 1 = 0.1.*

4. *We continue generating tuples, using the same steps.*

Example 3.9. ***(Answering probabilistic queries from weighted samples):*** *We are given the following weighted samples:*

A	**B**	**C**	**D**	Weight
1	0	0	1	0.5
0	0	0	0	0.2
0	1	0	1	0.3
1	0	0	0	0.4
0	1	1	0	0.1
0	1	0	1	0.2
0	0	0	1	0.1
0	0	0	0	0.7
1	0	0	1	0.5
0	0	0	0	0.6

Answer the queries: P(C = 0), P(A = 0, B = 0, C = 1,D = 0) and P(D = 0 | A = 1, C= 0)

Solution:

1. $\boldsymbol{P(C = 0)} = \frac{\textit{Sum of weights that have } C=0}{\textit{Sum of weights of all samples}} = \frac{0.5+0.2+0.3+0.4+0.2+0.1+0.7+0.5+0.6}{0.5+0.2+0.3+0.4+0.2+0.1+0.7+0.5+0.6+0.1} = \frac{3.5}{3.6} = 0.97$

2. $\boldsymbol{P(A = 1, B = 0, C = 0,D = 1)} = \frac{\textit{Sum of weights that have } (A,B,C,D)=(1,0,0,1)}{\textit{Sum of weights of all samples}} = \frac{0.5+0.5}{0.5+0.2+0.3+0.4+0.2+0.1+0.7+0.5+0.6+0.1} = \frac{1}{3.6} = 0.27$

3. $\boldsymbol{P(D = 0 \mid A = 1, C = 0)} = \frac{\textit{Sum of weights that have } (A,C,D)=(1,0,0)}{\textit{Sum of weights of samples that have } (A,C)=(1,0} = \frac{0.4}{0.5+0.4+0.5} = \frac{0.4}{1.4} = 0.28$

3.6 Experimental Evaluation

Now that we have completed the explanation of the algorithms that interest us, we are finally ready to perform a series of experiments that illustrate what we have accomplished. We remind you at this point that our ultimate goal is to produce synthetic data, which a data analyst may use as they see fit without compromising the privacy of any real individual. We achieve that by building a model (Bayesian Network) of the real

data and by learning its parameters while we ensure the privacy of the data by accessing it the minimum possible amount of times and always in a differentially private manner. To conduct our experiments, we implement the aforementioned algorithms using the Python programming language. We also import certain libraries/packages to assist us in our implementation. The most important of them are:

1. SciPy
2. NumPy
3. Scikit-learn
4. NetworkX
5. MultiProcessing

and various others.

3.6.1 Datasets

For our experiments, we use *real-world* datasets from UCI Machine Learning Repository. All of them are *labeled* and that means that each tuple belongs to a particular (and known to us) category (*class*). The type of Machine Learning that uses labeled samples is known as *Supervised Learning*. For more information on Supervised Learning, check Chapter 5. The datasets are:

1. Heart Disease Dataset
2. Poker Hand Dataset
3. Adult Dataset

Our program infers the domain of each attribute i.e. the values it takes, while reading the dataset.

Heart Disease: This dataset contains 303 instances of patients and it was created by V.A. Medical Center, Long Beach and Cleveland Clinic Foundation. It contains 13 attributes for each patient which are:

Name	Type	Values	Meaning
Age	Continuous	Real	-
Sex	Categorical	0,1	Female,Male
Chest Pain	Categorical	0,1,2,3,4	No pain, ... , A lot of pain
Blood Pressure	Continuous	Real	-
Cholesterol	Continuous	Real	-
Blood Sugar	Categorical	0,1	Less than 120 mg/dl,More than 120 mg/dl
Cardiogram Results	Categorical	0,1,2	Normal, Serious, Very serious

Max Pulses	Continuous	Real	-
Exercise Induced Angina	Categorical	0,1	No, Yes
Oldpeak	Continuous	Real	-
Slope	Categorical	1,2,3	-
Number of major vessels	Continuous	Real	-
Thal	Categorical	3,6,7	Normal, Stable damage, Reversible damage

Each patient in the dataset belongs to a category/class according to the diagnosis based on the 13 attributes. These are:

Name	Meaning
Class 0	No heart disease
Class 1	Not likely to have a heart disease
Class 2	More likely to have a heart disease than class 1
Class 3	More likely to have a heart disease than class 2
Class 4	Most likely to have a heart disease

We applied additional pre-processing to the dataset, so as to make it suitable to our purposes:

1. We encoded the (few) missing values (?) as "-1".
2. We converted the continuous attributes to categorical using the method described in Section 3.4.1.

Poker Hand: This dataset contains 1025010 instances and it was created by Franz Oppacher, Carleton University, Department of Computer Science Intelligent Systems Research Unit. Each record is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes (suit and rank), for a total of 10 predictive attributes. The order of cards is important, which is why there are 480 possible Royal Flush hands as compared to 4. More specifically we have:

Name	**Type**	**Values**	**Meaning**
Suit of card 1	Categorical	1,2,3,4	Hearts,Spades,Diamonds,Clubs
Rank of card 1	Categorical	1,2,3,4,5,6,7,8,9,10,11,12,13	Ace,2,3,...,Queen,King
Suit of card 2	Categorical	1,2,3,4	Hearts,Spades,Diamonds,Clubs
Rank of card 2	Categorical	1,2,3,4,5,6,7,8,9,10,11,12,13	Ace,2,3,...,Queen,King
Suit of card 3	Categorical	1,2,3,4	Hearts,Spades,Diamonds,Clubs
Rank of card 3	Categorical	1,2,3,4,5,6,7,8,9,10,11,12,13	Ace,2,3,...,Queen,King
Suit of card 4	Categorical	1,2,3,4	Hearts,Spades,Diamonds,Clubs
Rank of card 4	Categorical	1,2,3,4,5,6,7,8,9,10,11,12,13	Ace,2,3,...,Queen,King

Suit of card 5	Categorical	1,2,3,4	Hearts,Spades,Diamonds,Clubs
Rank of card 5	Categorical	1,2,3,4,5,6,7,8,9,10,11,12,13	Ace,2,3,...,Queen,King

Each poker player hand in the dataset belongs to a category/class based on the 10 attributes. These are:

Name	Meaning
Class 0	Nothing in hand; not a recognized poker hand
Class 1	One pair; one pair of equal ranks within five cards
Class 2	Two pairs; two pairs of equal ranks within five cards
Class 3	Three of a kind; three equal ranks within five cards
Class 4	Straight; five cards, sequentially ranked with no gaps
Class 5	Flush; five cards with the same suit
Class 6	Full house; pair + different rank three of a kind
Class 7	Four of a kind; four equal ranks within five cards
Class 8	Straight flush; straight + flush
Class 9	Royal flush; Ace, King, Queen, Jack, Ten + flush

There is no need for pre-processing, because:

1. There are no missing values
2. All attributes are already categorical

Adult This dataset contains 48842 instances and it was created by Ronny Kohavi and Barry Becker, Data Mining and Visualization Silicon Graphics department. Each record refers to an adult individual and categorizes them in two classes, depending whether they earn more or less than 50000 $ every year. 14 attributes describe each adult:

Name	Type	Values	Meaning
Age	Continuous	Real	
Workclass	Categorical	0,1,2,3,4,5,6,7	Private, Self-emp-not-inc, Self-emp-inc, Federalgov,Local-gov, State-gov, Without-pay, Never-worked
Fnlwgt	Continuous	Real	
Education	Categorical	0,1,2,3,4,...,15	Bachelors, Some-college,11th,HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th,7th-8th, 12th, Masters,1st-4th, 10th,Doctorate,5th-6th,Preschool
Education number	Continuous	Real	

Marital Status	Categorical	0,1,2,3,4,5,6	Married-civ-spouse, Divorced,Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse
Occupation	Categorical	0,1,2,3,4,...,13	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces
Relationship	Categorical	0,1,2,3,4,5	Wife, Own-child, Husband,Not-in-family, Other-relative,Unmarried
Race	Categorical	0,1,2,3,4	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black
Sex	Categorical	0,1	Female,Male
Capital gain	Continuous	Real	
Capital loss	Continuous	Real	
Hours per week	Continuous	Real	
Native country	Categorical	0,1,2,3,...,40	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador Trinadad&Tobago, Peru, Hong, Holand-Netherlands

Each adult in the dataset belongs to a category/class based on the 14 attributes.

These are:

Name	Meaning
Class 0	earns $>$ 50000 $ per year
Class 1	earns $\leq$ 50000 $ per year

We applied additional pre-processing to the dataset, so as to make it suitable to our purposes:

1. We encoded the (few) missing values (?) as "-1".
2. We converted the continuous attributes to categorical using the method described in Section 3.4.1.
3. The categorical attributes of the original dataset had string values (Check fourth column of the attribute table). To be able to use the built-in classifiers that Python offers, we encoded these values as integers as shown in the third column of the attribute table.

3.6.2 Hyperparameters and classifiers

In our experiments, we will use the real dataset to generate a synthetic one. Then we split the real dataset into a training set (80% of the dataset) and a testing set (20% of the dataset). The training set is used to "train" our model to recognize samples and place them in the correct class. The testing set is composed of samples that the model has never seen before and is used to "test" how well trained the model is by having it to place the test samples in classes. This process is known as *classification*. Then by using the whole synthetic dataset as a training set and the testing set from the real data, we perform classification again. If classification using the synthetic data as a training set performs as well as classification using the real data does, then we have succeeded in generating data that can be used in the place of real data. We perform this experiment for many different cases by adjusting the following (hyper)parameters:

Name	Use	Values	Meaning
dataset_choice	Dataset that we will use	1,2,3	Heart Disease, Poker Hand, Adult
M	Number of data holders (sub-datasets)	1, 5, splitting method 3 or 4 choice	
split_choice	Method with which we split the original dataset into many	0,1,2,3,4	One centralized dataset, Equal-sized datasets, Random-sized datasets, Split by class, Split by random attribute

k	degree of generated Bayesian Network	1,2	
str_choice	Structure Learning method	1,2,3	Algorithm 4, Algorithm 5, Algorithm 6
ϵ	Parameter that determines the level of DP	0.01,0.02,0.05, 0.1,0.2,0.5,1, 5,10,20	High Privacy,...,Low Privacy

Next we will speak of the classifiers that we will use:

1. **Decision Tree Algorithm (CART)**: A decision tree is a tree where each node represents a feature (attribute), each link (branch) represents a decision (rule) and each leaf represents an outcome (categorical value). When training a dataset to classify a variable, the idea of the Decision Tree is to divide the data into smaller datasets based on a certain feature value until the target variables all fall under one category. A computer splits the dataset based on the maximum information gain [1]. Every tree starts with a root node, i.e. the first split. The decision tree will then consider all the possible splits and choose the one with the highest information gain. This process will repeat itself until the nodes can be split no more or some user-defined criteria are fullfiled. The new samples are classified by placing them in their respective leaf nodes according to the splits.

2. **AdaBoost Algorithm**: AdaBoost, short for "Adaptive Boosting", is the first practical boosting algorithm proposed by Freund and Schapire in 1996. It focuses on classification problems and aims to convert a set of weak classifiers into a strong one. It places weights on the training samples, weights which are originally equal to the inverse the size of the training set. These weights are constantly updated during the training process and finally we have a prediction for each sample from each individual classifier. The final prediction is the one with the most votes, the one chosen by the majority of classifiers.

3. **XGBoost Algorithm (XGB)**: XGBoost is a decision-tree-based ensemble [2] Machine Learning algorithm that uses a gradient boosting framework. XGBoost algorithm was developed as a research project at the University of Washington. This algorithm tends to surpass all other classification algorithms in problems with structured data (images, text, e.t.c.). The reason for that is that it improves the framework of Gradient Boosting with optimizations like Parallelization and

[1] The information gain is based on the decrease in entropy after a data-set is split on an attribute.

[2] An ensemble is just a collection of predictors which come together (e.g. mean of all predictions) to give a final prediction. The reason we use ensembles is that many different predictors trying to predict same target variable will perform a better job than any single predictor alone.

Tree Pruning and techniques like Regularization, Sparsity Awareness, Weighted Quantile Sketch, Cross-Validation and others.

4. **Random Forest Algorithm**: Random Forest is a learning method that operates by constructing multiple decision trees. The final decision is made based on the majority of the trees and is chosen by the random forest. The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is that a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The low correlation between models is the key. Uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

5. **Support Vector Algorithm (SVC)**: "Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane and so on. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier.

6. **Linear Support Vector Algorithm (Linear SVC)**: A version of the support vector algorithms where the margins of the hyperplane have linear form.

7. **Gradient Boosting Algorithm**: Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

8. **Gaussian Naive Bayes Algorithm (GNB)**: This algorithm calculates an apri-

ori probability P(c_i) = Number of Instances in c_i class/Total number of instances for every class. Next for every sample x and class c_i, we calculate the aposteriori probability $P(X|c_i) = P(X_1|c_i) * P(X_2|c_i) * ...P(X_N|c_i)$, where X_i are the attributes of the sample. Each one of the aposteriori probabilities is drawn from the Gaussian distribution, after we calculate the mean and the variance for each attribute given the respective class c_i. Finally we calculate the probability $P(c_i|X)$ for each sample X and class c_i. So the sample X belongs to class c_i which has the largest probability $P(c_i|X)$.

9. **Linear Discriminant Analysis Algorithm (LDA)**: Linear Discriminant Analysis or Normal Discriminant Analysis or Discriminant Function Analysis is a dimensionality reduction technique which is commonly used for the supervised classification problems. It is also used for modeling differences in groups i.e. classification. It is used to project the features in higher dimension space into a lower dimension space. It calculates the mean and the co-variance matrices for each class and uses the Bayes Theorem to calculate the probability that a sample X belongs to class C for every X,C. As in Gaussian Naive Bayes, the X is classified to the class C with the largest probability P(C|X).

10. **Quadratic Discriminant Analysis Algorithm (QDA)**: An extended version of Linear Discriminant Analysis, where classes are no longer assumed to have the same co-variance and the decision boundary is not necessarily linear. As a result, QDA tends to be more flexible than LDA.

11. **Multi-layer Perceptron Algorithm (MLP)**: Multi-layer Perceptron is a fully connected feed-forward artificial neural network. MLP is composed of at least three layers. The output layer are has as many nodes as the number of classes. For more details on neural networks, see Chapter 4.

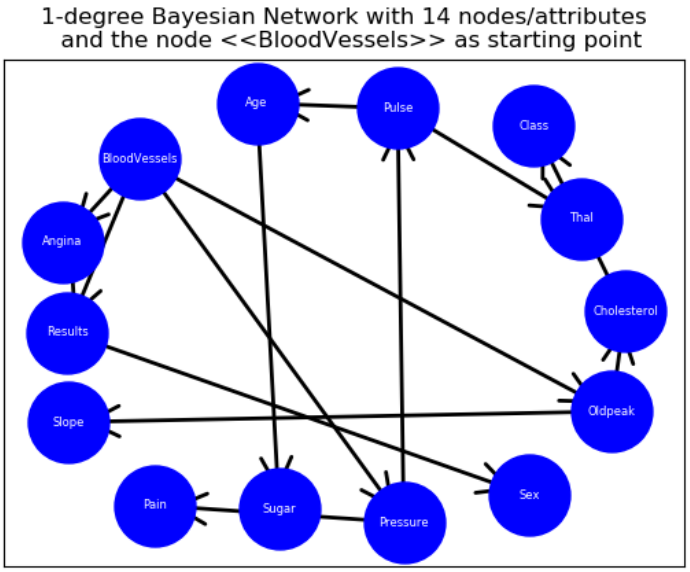
12. **K-Nearest Neighbors Algorithm (KNN)**: The k-nearest neighbors algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. The training examples are vectors in a multidimensional feature space, each with a class label. As a result, this algorithm believes that an unlabeled sample belongs to the same class that the k-nearest neighbouring training samples belong to (k is a user-defined variable). To find the neighbours of a sample, we use distance metrics like the Euclidean distance [1].

[1] In mathematics, the Euclidean distance or Euclidean metric is the "ordinary" straight-line distance between two points in Euclidean space.

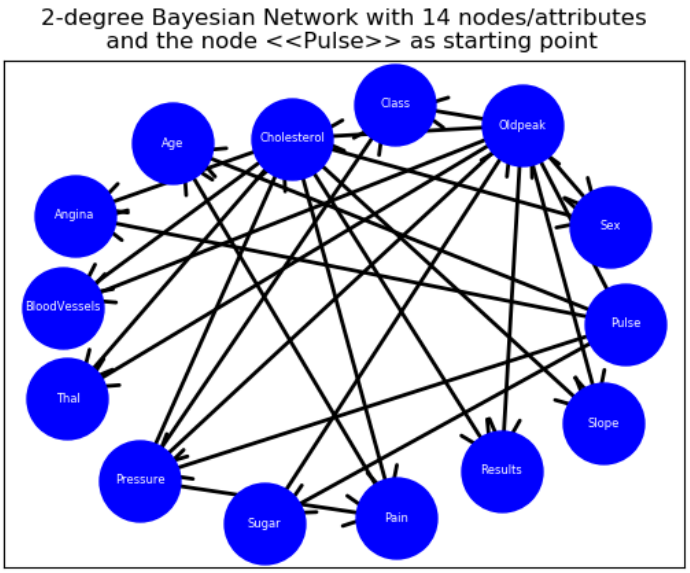
3.6.3 Experimental Evaluation

Examples of generated Bayesian Networks

We have executed our code for 767 different combinations of the aforementioned parameters and we have received the prediction accuracy of each classifier when used to classify the respective synthetic data. First, we present some examples of generated Bayesian Networks (using the NetworkX and Matplotlib libraries):



(a) Dataset 1, k = 1



(b) Dataset 1, k = 2

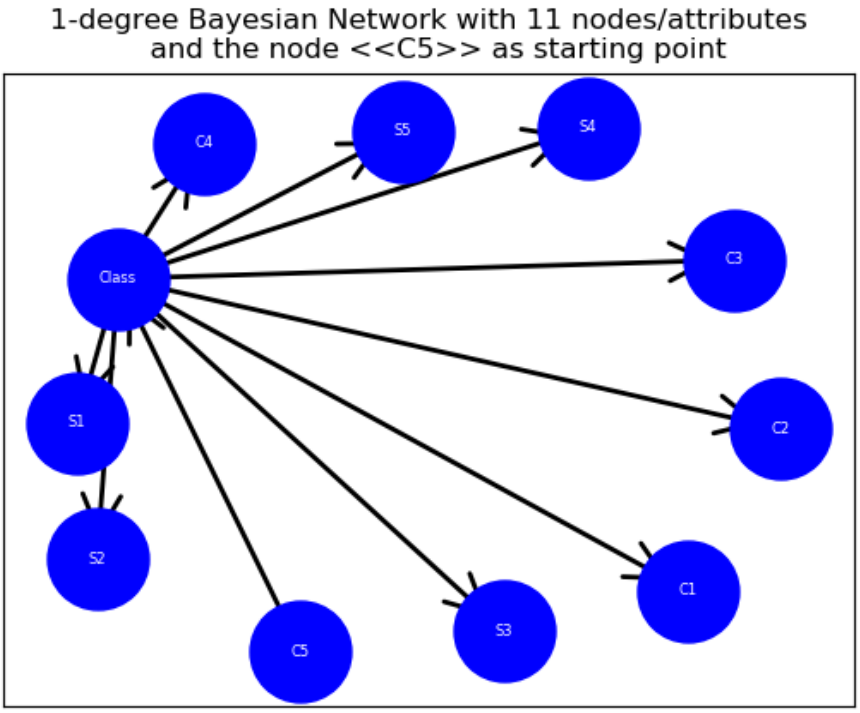


Figure 10: Dataset 2, k = 1

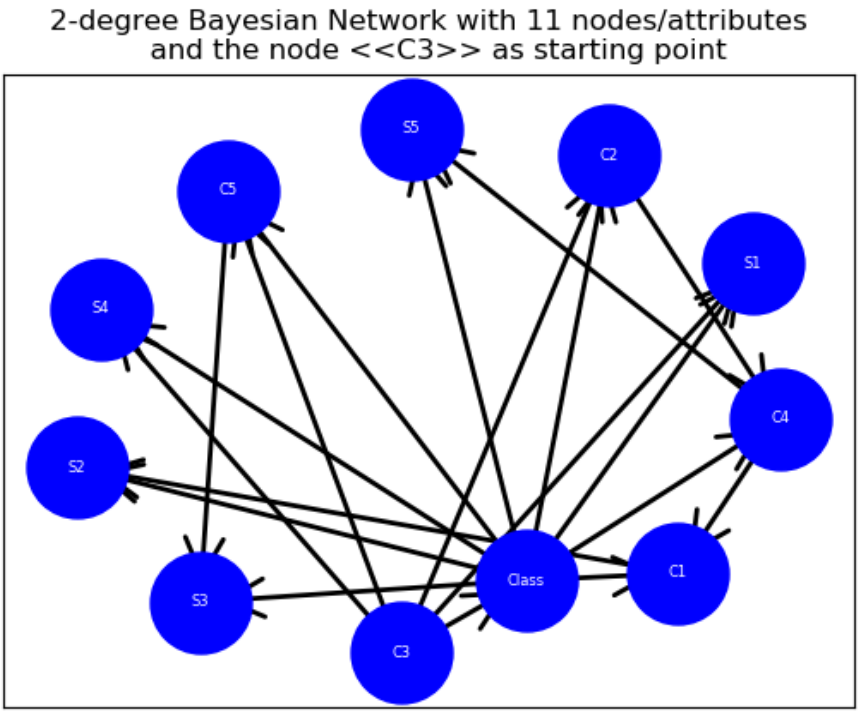


Figure 11: Dataset 2, k = 2

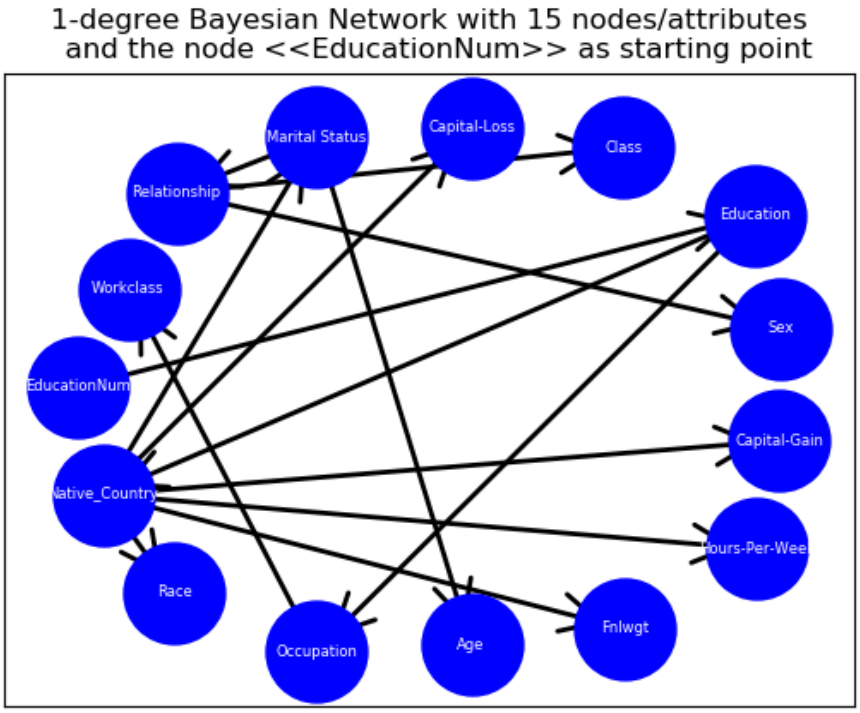


Figure 12: Dataset 3, k = 1

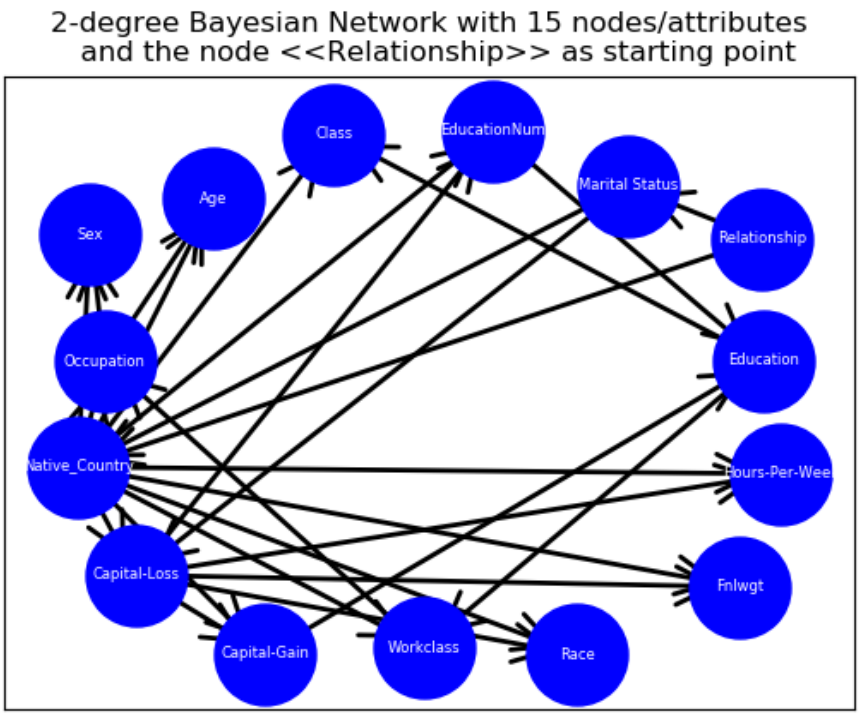


Figure 13: Dataset 3, k = 2

Finding the best classifier

Secondly, we seek to find the best classifier for each dataset. To accomplish that we examine for each dataset (dataset_choice) and each Structure Learning method (str_choice), the cases which yield the *highest average accuracy*. These are (check next 3 pages):

ϵ	XGB	CART	KNN	Linear SVC	SVC	Random Forest	MLP	AdaBoost	Gradient Boosting	GNB	LDA	QDA	Average Accuracy (Synthetic Data)	Best classifier		
10	68,97	55,17	48,28	62,07	62,1	65,52	62,1	62,07	65,52	65,52	68,97	65,52	62,64	XGB	LDA	
1	68,97	51,72	55,17	62,07	65,5	51,72	72,4	62,07	62,07	68,97	68,97	62,07	62,64	MLP		
10	68,97	51,72	55,17	51,72	58,6	68,97	69	65,52	58,62	65,52	65,52	65,52	62,07	XGB	Random Forest	
10	65,52	48,28	51,72	68,97	62,1	72,41	58,6	62,07	62,07	62,07	62,07	65,52	61,78	Random Forest		
5	62,07	44,83	55,17	62,07	62,1	58,62	65,5	62,07	65,52	55,17	68,97	65,52	60,63	LDA		
10	65,52	62,07	65,52	37,93	62,1	58,62	62,1	62,07	62,07	51,72	58,62	75,86	60,34	QDA		
10	65,52	58,62	48,28	72,41	65,5	62,07	72,4	65,52	72,41	65,52	65,52	58,62	64,36	Linear SVC	MLP	Gradient Boosting
20	62,07	58,62	58,62	65,52	62,1	58,62	65,5	51,72	68,97	65,52	65,52	51,72	61,2	Gradient Boosting		
10	62,07	55,17	62,07	62,07	62,1	65,52	55,2	48,28	62,07	65,52	72,41	58,62	60,92	LDA		
10	62,07	55,17	51,72	62,07	62,1	62,07	62,1	44,83	68,97	62,07	62,07	65,52	60,05	Gradient Boosting		
1	68,97	65,52	37,93	62,07	62,1	62,07	65,5	58,62	62,07	62,07	62,07	48,28	59,77	XGB		
20	58,62	51,72	58,62	55,17	65,5	58,62	62,1	55,17	55,17	65,52	62,07	55,17	58,62	SVC		
20	62,07	58,62	48,28	68,97	62,1	65,52	65,5	48,28	58,62	65,52	65,52	58,62	60,63	Linear SVC		
20	58,62	58,62	58,62	55,17	62,1	65,52	58,6	58,62	58,62	65,52	62,07	62,07	60,34	Random Forest	GNB	
20	65,52	51,72	62,07	41,38	62,1	51,72	58,6	41,38	65,52	62,07	62,07	62,07	57,18	XGB	Gradient Boosting	
20	62,07	51,72	51,72	10,34	62,1	65,52	58,6	37,93	55,17	65,52	65,52	58,62	53,73	Random Forest	LDA	GNB
20	48,28	44,83	62,07	51,72	62,1	62,07	48,3	51,72	44,83	55,17	55,17	48,28	52,87	KNN	SVC	Random Forest
10	55,17	37,93	37,93	3,45	62,1	58,62	62,1	51,72	44,83	65,52	62,07	62,07	50,28	GNB		

Table 7: Best accuracy results for every structure learning method (Dataset 1)

ϵ	XGB	CART	KNN	Linear SVC	SVC	Random Forest	MLP	AdaBoost	Gradient Boosting	GNB	LDA	QDA	Average Accuracy (Synthetic Data)	Best classifier
10	54,96	45,16	49,16	49,4	51,7	50,96	59,2	45,92	55,8	49,4	49,36	53,56	51,21	MLP
5	54,88	43,2	48,08	49,4	52,1	49,92	58,8	49,28	56,44	49,4	49,4	52,52	51,12	MLP
10	55,36	43,92	50,24	48,96	51,8	49,96	55,8	47,68	55,56	49,44	49,8	51,64	50,84	MLP
10	55,44	45,04	49,52	49,4	53	49,72	56,4	43,76	55,04	49,4	49,4	52,2	50,69	MLP
1	53,72	42,72	48,96	49,4	50	49,92	57,5	49,04	54,48	49,12	49,28	53,44	50,63	MLP
5	54,56	40,04	49,04	49,48	51,8	49,6	56,8	49,28	54,8	49,08	49,24	52,24	50,50	MLP
10	52,04	37,76	47,4	49,4	47,1	47,92	50	49,4	51,4	49,36	49,4	50,76	48,49	XGB
20	51	39,36	47,28	49,4	48,8	48,04	50,9	49,08	49,8	49,4	49,4	48,2	48,38	XGB
10	50,52	40,44	47,12	49,32	47,8	46,88	50,4	49,32	50,36	49,4	49,4	49,52	48,38	XGB
2	50,68	42,12	46,44	49,68	48,2	47,12	48	49,24	49,6	49,4	49,4	50	48,33	XGB
20	49,92	41,32	49,52	44,16	49,4	46,76	49,6	48,36	49,12	49,68	49,6	49,16	48,05	XGB
20	49,8	39,72	46,8	47,84	46,9	47,76	50,1	49,28	49,76	49,4	49,4	48,4	47,93	MLP
20	54,28	43,88	48,28	49,08	51,1	49,44	55,3	48,88	54,12	49,48	49,36	51,24	50,37	MLP
20	52,84	42,28	48,88	49,52	51,4	51	53,3	48,68	53,68	49,32	49,24	52,48	50,22	Gradient Boosting
10	52,96	43,64	49,92	48,88	50	50,48	53,7	48,6	53,04	48,64	48,92	51,6	50,03	MLP
10	50,52	35,4	49,8	43,16	52,2	48,48	50,3	49,48	51,92	49,4	49,4	51,2	48,44	SVC
20	50,12	41,72	46,64	49,2	48,3	46,84	49,4	49,48	49,72	49,4	49,4	49,48	48,31	XGB
20	51,76	33,44	46,68	49,48	49,1	48,04	48	48,6	51,6	49,4	49,4	51,4	48,07	XGB

Table 8: Best accuracy results for every structure learning method (Dataset 2)

ϵ	XGB	CART	KNN	Linear SVC	SVC	Random Forest	MLP	AdaBoost	Gradient Boosting	GNB	LDA	QDA	Average Accuracy (Synthetic Data)	Best classifier	
10	82,37	75,26	78,4	76,32	76,4	78,9	76,4	82,26	82,22	77,1	81,97	81,57	79,096667	XGB	
0,05	81,25	77,85	79,61	79,65	76,4	81,17	77,4	80,09	81,33	76,89	76,72	76,85	78,761667	Gradient Boosting	
1	80,15	73,51	77,51	80,08	76,4	77,03	78,8	80,11	80,09	79,79	79	79,87	78,525833	XGB	
10	80,11	73,02	77,43	79,97	76,4	76,64	79,1	80,11	80,1	79,52	79,03	79,54	78,415833	XGB	
5	80,18	73,53	77,39	80,03	76,4	76,66	78,7	80,2	80,14	79,39	78,76	79,43	78,398333	AdaBoost	
10	80,12	73,61	77,59	76,86	76,4	77,2	78,7	80,14	80,11	79,47	78,71	79,45	78,198333	AdaBoost	
10	83,96	76,89	80,02	79,31	76,4	80,51	78,1	83,55	83,96	77,14	80,68	80,48	80,079167	XGB	Gradient Boosting
20	82,36	74,85	77,3	75,04	76,4	78,26	78,4	82,24	82,27	76,9	80,33	80,28	78,714167	XGB	
2	83,48	72,76	78,06	77,64	76,4	79,05	77,6	82,52	83,58	76,89	76,99	77,24	78,518333	Gradient Boosting	
20	82,31	74,76	77,25	75,29	76,4	78,64	75,2	82,21	82,24	76,75	80,22	79,87	78,428333	XGB	
20	80,19	73,15	77,43	78,26	76,4	77,23	77,5	80,22	80,15	79,34	78,91	79,39	78,181667	AdaBoost	
1	82,36	73,53	78,36	76,63	76,4	78,75	76,2	80,01	82,22	76,9	76,59	76,94	77,9025	XGB	
20	84,87	79,27	81,1	80,35	76,4	81,65	79,3	84,64	84,85	79,34	82,19	81,74	81,31	XGB	
10	84,58	79,55	81,14	76,59	76,4	82,36	78,9	84,36	84,59	78,26	81,55	80,36	80,718333	Gradient Boosting	
20	83,77	77	80,74	77,61	76,4	80,35	78,6	83,48	83,77	77,12	82,21	82,29	80,279167	XGB	
20	83,95	77,23	79,99	80,3	76,4	80,52	79,3	83,94	83,95	77,14	80,5	80,12	80,278333	XGB	Gradient Boosting
20	83,9	74,71	79,94	79,52	76,4	79,93	76,7	83,76	83,81	77,05	79,24	80,02	79,58	XGB	
20	82,89	79,66	79,73	77,2	76,4	81,64	76,5	82,46	82,94	76,4	78,23	77,5	79,296667	Gradient Boosting	

Table 9: Best accuracy results for every structure learning method (Dataset 3)

Based on these results, we make the following observations and remarks:

- Dataset 3 yields much better accuracy for both real and synthetic data compared to the other two. We attribute this fact to the statistical properties of the dataset as well as to its usage of only two classes. As a result, the worst accuracy we can expect from this dataset is 50 %. It is also worth noting that we cannot expect high accuracy from the first dataset, since it has high dimensionality (a great number of attributes), a small number of samples and uses five classes. This will be made more obvious in our next experiment.
- As we anticipated, most of the cases that yield great accuracy correspond to high values of ϵ. However, these values will not serve our purposes, since almost no privacy is guaranteed.
- The best classifiers (by majority voting) are:
  - **Dataset 1**: 1.XGBoost, Random Forest
  - **Dataset 2**: 1.MLP Classifier, 2.XGBoost
  - **Dataset 3**: 1.XGBoost, 2.Gradient Boost
  - **All Datasets**: 1.XGBoost, 2.MLP Classifier, Gradient Boost

  It worth noting that the most of the best classifiers are also *ensemble* [2] classifiers.

Finding an appropriate value of ϵ

For our next experiment, we aim to find an acceptable ϵ value for each of our datasets. Since we require privacy, we will examine $\epsilon \leq 1$ values. To judge whether a case has acceptable accuracy, we will calculate the difference between the average accuracy of the real and synthetic data. The closer this result is to 0, the closer the performance of the synthetic data is to that of the real data. A value equal to 0 means that the synthetic data performs just as well as the real data does. In rare cases, where the synthetic data performs even better that the real one, the value will be negative. We will consider as acceptable differences the ones which have values $\leq 10\%$. The first ϵ that corresponds to such a difference will be our choice. The results that we will examine are those with the smallest difference of accuracy for each dataset and Structure Learning method and with $\epsilon \leq 1$. These are:

[2]Ensemble learning is a way of generating various base classifiers from which a new classifier is derived which performs better than any constituent classifier.

Structure Learning	k	ϵ	Average Accuracy (Real Data) (%)	Average Accuracy (Synthetic Data) (%)	Accuracy (Real Data) - Accuracy (Synthetic Data) (%)
1	1	0,01	59,19583333	43,39083	15,805
1	1	0,01	59,19583333	42,24083	16,955
1	1	0,01	59,19583333	37,3575	21,83833
1	2	0,01	59,19583333	25,28667	33,90917
1	1	0,01	59,19583333	24,99917	34,19667
1	2	0,01	59,19583333	21,26333	37,9325
3	2	0,02	59,19583333	40,80417	18,39167
1	1	0,02	59,19583333	33,33167	25,86417
1	1	0,02	59,19583333	33,33167	25,86417
2	1	0,02	59,19583333	33,045	26,15083
2	2	0,02	59,19583333	30,4575	28,73833
1	2	0,02	59,19583333	29,02333	30,1725
1	2	0,05	59,19583333	45,4025	13,79333
1	1	0,05	59,19583333	37,07	22,12583
1	1	0,05	59,19583333	29,59667	29,59917
1	1	0,05	59,19583333	27,8725	31,32333
1	2	0,05	59,19583333	24,71167	34,48417
1	1	0,05	59,19583333	18,10333	41,0925
2	1	0,1	59,19583333	50,575	8,620833
1	1	0,1	59,19583333	49,13833	10,0575
1	2	0,1	59,19583333	46,83833	12,3575
1	1	0,1	59,19583333	43,39083	15,805
3	2	0,1	59,19583333	43,1025	16,09333
3	2	0,1	59,19583333	42,52833	16,6675
2	1	0,2	59,19583333	54,5975	4,598333
1	1	0,2	59,19583333	54,30917	4,886667
1	1	0,2	59,19583333	54,30917	4,886667
1	1	0,2	59,19583333	52,2975	6,898333
1	1	0,2	59,19583333	52,2975	6,898333
2	2	0,2	59,19583333	41,37833	17,8175
1	1	0,5	59,19583333	56,61	2,585833
1	2	0,5	59,19583333	54,02333	5,1725
1	2	0,5	59,19583333	52,87333	6,3225
1	1	0,5	59,19583333	52,01083	7,185
1	1	0,5	59,19583333	50,28667	8,909167
1	1	0,5	59,19583333	48,85	10,34583

1	2	1	59,19583333	62,64417	-3,44833
2	1	1	59,19583333	59,77167	-0,57583
1	2	1	59,19583333	55,46083	3,735
1	1	1	59,19583333	54,88417	4,311667
2	1	1	59,19583333	54,59667	4,599167
1	1	1	59,19583333	54,31167	4,884167

Table 10: Best accuracy results for $\epsilon \leq 1$ (Dataset 1)

Split Method	Str Learning Method	k	ϵ	Average Accuracy (Real Data) (%)	Average Accuracy (Synthetic Data) (%)	Accuracy (Real Data) - Accuracy (Synthetic Data)
2	1	1	0,01	53,85333333	42,45333	11,4
4	1	1	0,01	53,85333333	42,18667	11,66667
3	1	2	0,01	53,85333333	31,00667	22,84667
1	1	1	0,01	53,85333333	26,92333	26,93
3	1	1	0,01	53,85333333	23,21667	30,63667
4	1	2	0,01	53,85333333	21,50667	32,34667
1	1	1	0,02	53,85333333	44,91	8,943333
1	1	1	0,02	53,85333333	44,91	8,943333
4	1	1	0,02	53,85333333	44,06333	9,79
4	1	1	0,02	53,85333333	44,06333	9,79
2	1	1	0,02	53,85333333	41,55	12,30333
2	1	1	0,02	53,85333333	41,55	12,30333
4	1	1	0,05	53,85333333	47,19	6,663333
2	1	1	0,05	53,85333333	46,28	7,573333
1	1	2	0,05	53,85333333	45,14667	8,706667
1	1	1	0,05	53,85333333	43,92667	9,926667
3	1	1	0,05	53,85333333	42,02	11,83333
4	1	2	0,05	53,85333333	41,17333	12,68
2	1	2	0,1	53,85333333	48,09	5,763333
2	1	1	0,1	53,85333333	47,26	6,593333
4	1	2	0,1	53,85333333	47,09667	6,756667
4	2	1	0,1	53,85333333	46,46	7,393333
4	1	1	0,1	53,85333333	46,24667	7,606667
1	1	1	0,1	53,85333333	46,22333	7,63
2	2	1	0,2	53,85333333	47,58667	6,266667
1	1	1	0,2	53,85333333	47,53	6,323333
1	1	1	0,2	53,85333333	47,53	6,323333

1	2	1	0,2	53,85333333	46,99667	6,856667
3	2	1	0,2	53,85333333	46,97667	6,876667
4	1	2	0,2	53,85333333	46,58	7,273333
4	1	2	0,5	53,85333333	50,42333	3,43
2	1	2	0,5	53,85333333	49,78667	4,066667
1	1	2	0,5	53,85333333	48,31	5,543333
2	1	1	0,5	53,85333333	47,54	6,313333
3	1	2	0,5	53,85333333	47,42667	6,426667
3	1	1	0,5	53,85333333	47,26667	6,586667
1	1	2	1	53,85333333	50,63333	3,22
2	1	2	1	53,85333333	50,31	3,543333
4	1	2	1	53,85333333	50,20333	3,65
3	1	2	1	53,85333333	49,19	4,663333
1	1	1	1	53,85333333	48,30333	5,55
2	1	1	1	53,85333333	47,80667	6,046667

Table 11: Best accuracy results for $\epsilon \leq 1$ (Dataset 2)

Split Method	Str Learning Method	k	ϵ	Average Accuracy (Real Data) (%)	Average Accuracy (Synthetic Data) (%)	Accuracy (Real Data) - Accuracy (Synthetic Data)
4	1	1	0,01	81,84666667	76,67917	5,1675
4	1	2	0,01	81,84666667	76,5425	5,304167
2	1	1	0,01	81,84666667	73,92417	7,9225
1	1	1	0,01	81,84666667	73,51	8,336667
2	1	2	0,01	81,84666667	72,3225	9,524167
3	1	1	0,01	81,84666667	71,12583	10,72083
2	1	2	0,02	81,84666667	77,1925	4,654167
2	1	2	0,02	81,84666667	77,1925	4,654167
1	1	2	0,02	81,84666667	76,46167	5,385
1	1	2	0,02	81,84666667	76,46167	5,385
1	2	1	0,02	81,84666667	74,8125	7,034167
2	2	1	0,02	81,84666667	74,13083	7,715833
2	1	2	0,05	81,84666667	78,76167	3,085
4	1	2	0,05	81,84666667	74,28333	7,563333
1	1	1	0,05	81,84666667	74,14333	7,703333
3	1	1	0,05	81,84666667	73,96083	7,885833
4	1	1	0,05	81,84666667	73,52083	8,325833
1	1	2	0,05	81,84666667	71,1975	10,64917

2	1	2	0,1	81,84666667	77,88167	3,965
1	2	1	0,1	81,84666667	77,09083	4,755833
1	1	2	0,1	81,84666667	76,9275	4,919167
2	2	1	0,1	81,84666667	75,41833	6,428333
3	1	2	0,1	81,84666667	75,035	6,811667
4	1	2	0,1	81,84666667	74,81	7,036667
2	1	2	0,2	81,84666667	78,8525	2,994167
2	1	2	0,2	81,84666667	78,8525	2,994167
2	2	2	0,2	81,84666667	76,0025	5,844167
1	1	2	0,2	81,84666667	75,02833	6,818333
1	1	2	0,2	81,84666667	75,02833	6,818333
3	1	1	0,2	81,84666667	75,00667	6,84
3	1	1	0,5	81,84666667	78,0075	3,839167
1	2	1	0,5	81,84666667	76,40083	5,445833
1	2	1	0,5	81,84666667	76,40083	5,445833
2	1	2	0,5	81,84666667	75,115	6,731667
4	1	1	0,5	81,84666667	74,965	6,881667
3	2	1	0,5	81,84666667	74,93667	6,91
1	1	1	1	81,84666667	78,52583	3,320833
3	1	1	1	81,84666667	78,09917	3,7475
2	2	2	1	81,84666667	77,9025	3,944167
3	3	1	1	81,84666667	76,90167	4,945
1	2	2	1	81,84666667	76,54	5,306667
4	3	1	1	81,84666667	75,785	6,061667

Table 12: Best accuracy results for $\epsilon \leq 1$ (Dataset 3)

The connection between accuracy and the ϵ parameter is made clearer by observing the following plots where split_choice = 2 (random-sized datasets), since this value frequently corresponds to real-life conditions.

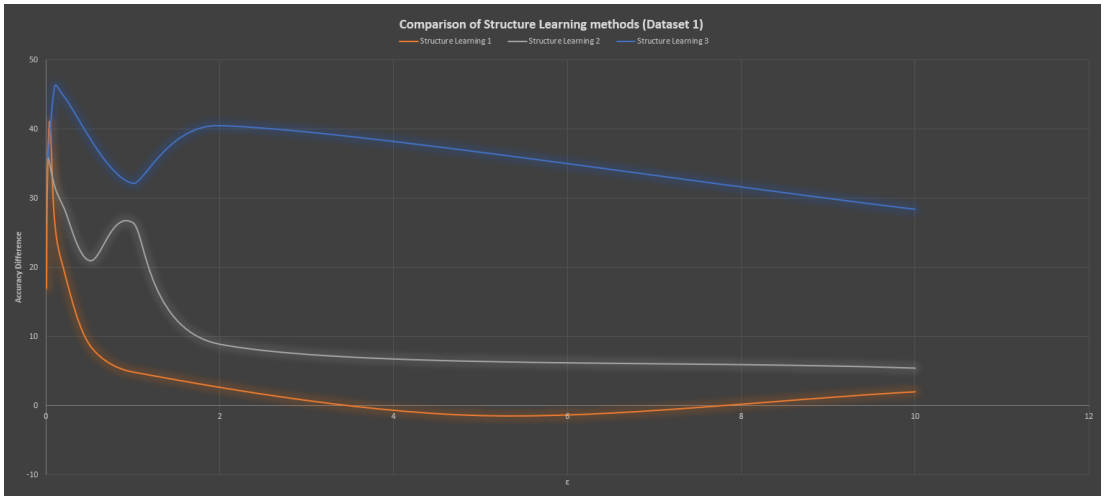


Figure 14: Dataset 1, k = 1

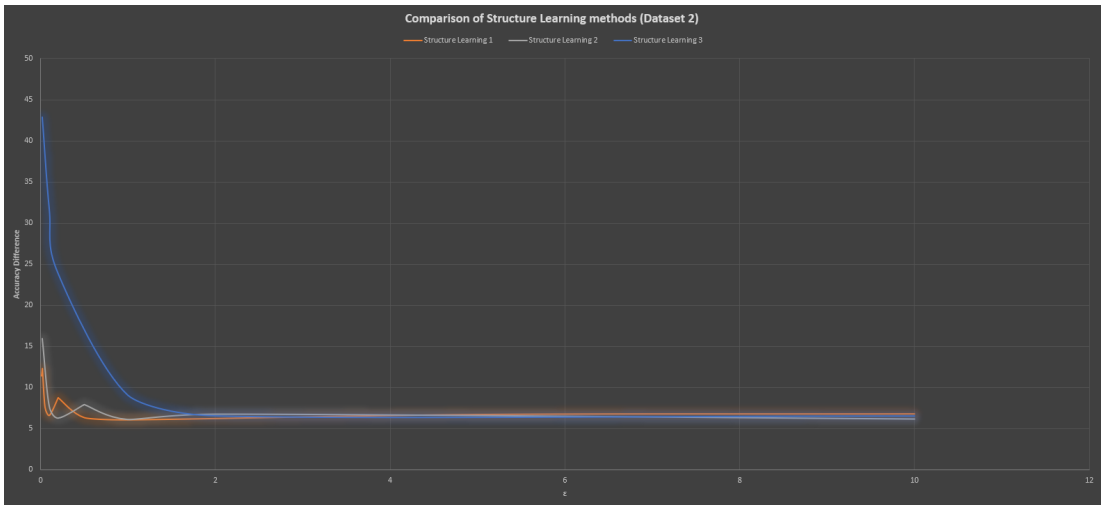


Figure 15: Dataset 2, k = 1

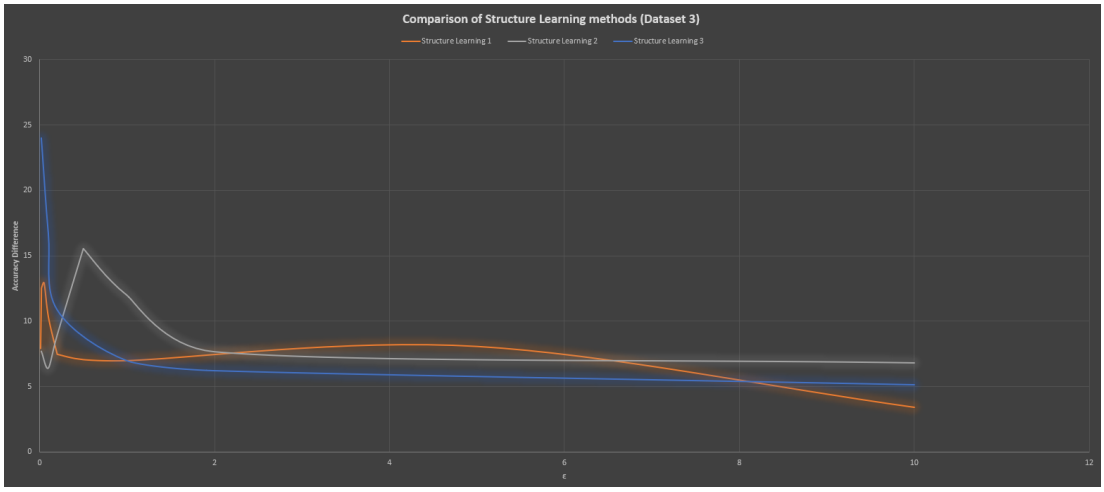


Figure 16: Dataset 3, k = 1

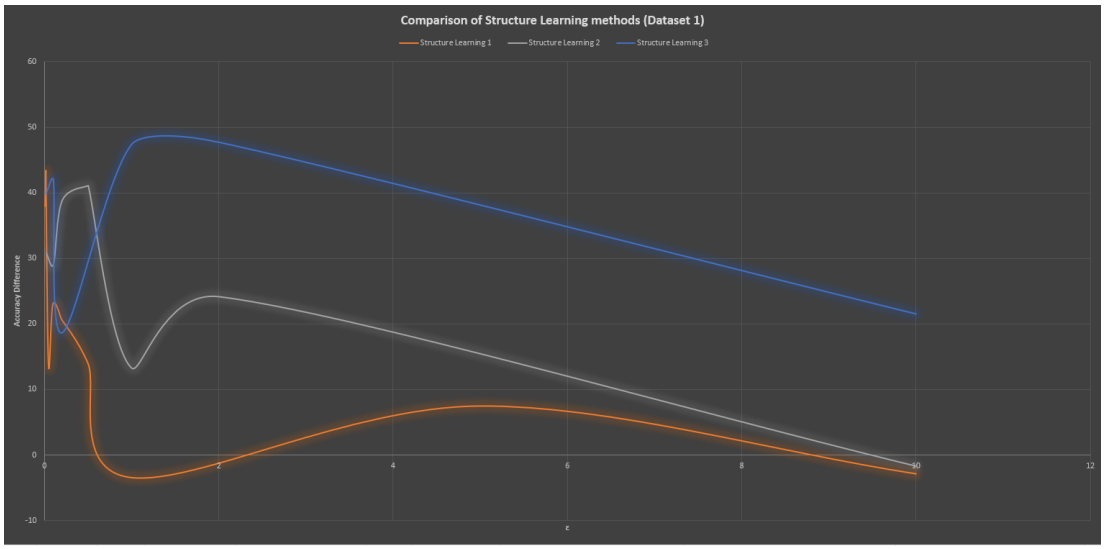


Figure 17: Dataset 1, k = 2

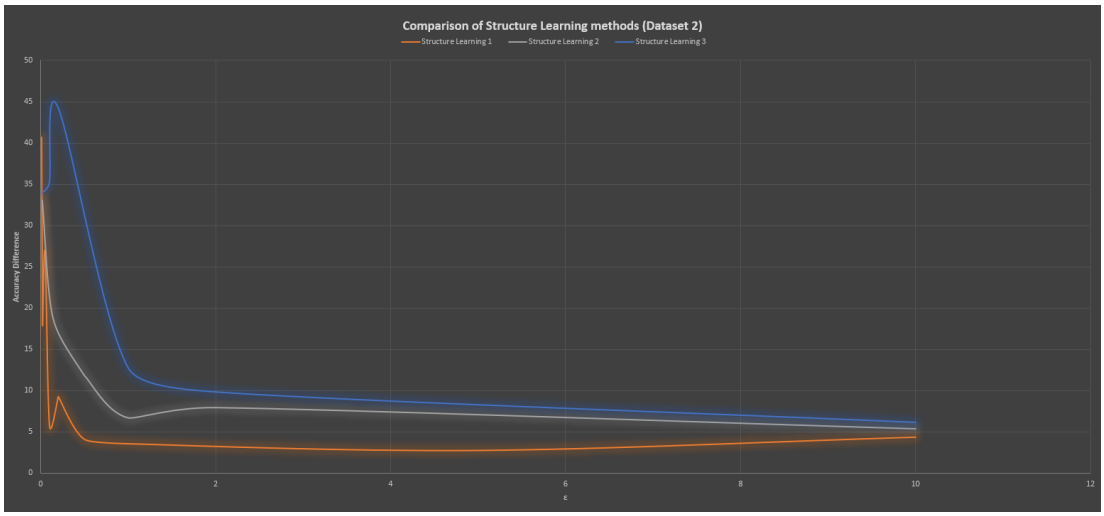


Figure 18: Dataset 2, k = 2

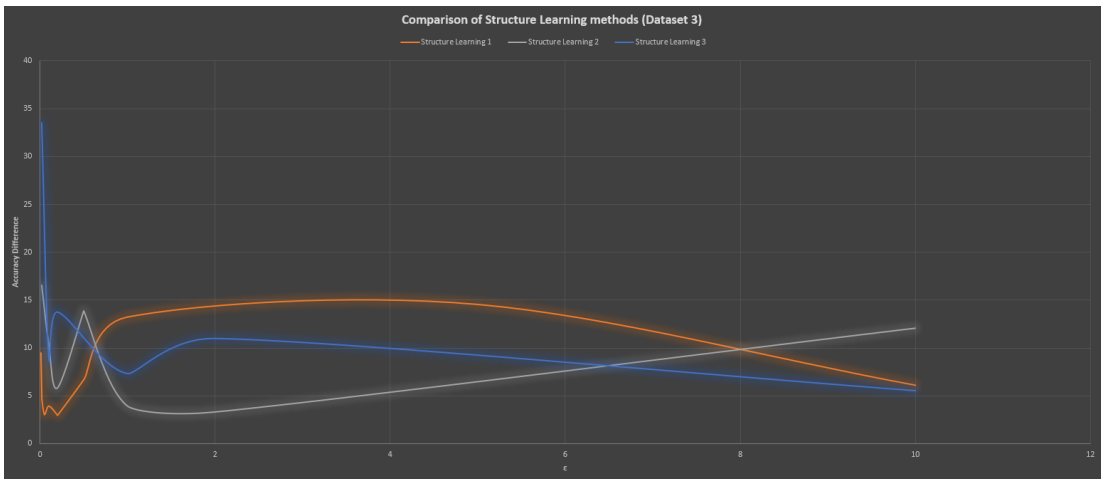


Figure 19: Dataset 3, k = 2

According to these plots, we make the following observations with regards to the Structure Learning algorithms:

- Accuracy difference decreases while ϵ increases, due to the fact that less amount of noise is added to the probability distributions.
- In dataset 1, for both k values, Structure Learning 1 clearly outperforms the other two and Structure Learning 2 outperforms Structure Learning 3. So the best method is Structure Learning 1.
- In dataset 2, for k = 1 , Structure Learning 1 and Structure Learning 2 have the same accuracy and outperform Structure Learning 3. However, it is worth noting that Structure Learning 3 has almost the same performance as the other methods for $\epsilon \geq 1.7$. So the best methods are Structure Learning 1 and 2.
- In dataset 2, for k = 2, the methods perform as they do for dataset 1 and k = 1, but the difference in their performance is much smaller and tends to become zero for $\epsilon \geq 8$. So the best method is Structure Learning 1.
- In dataset 3, for k = 1, Structure Learning 2 has the best performance for $\epsilon \in [0.01, 0.2)$ and the worst for $\epsilon \in [0.2, 2]$. Also Structure Learning 3 performs better than Structure Learning 1 for $\epsilon \in [1, 8]$ and worse for $\epsilon \in [0.01, 1)$. For $\epsilon \in [0.01, 1)$, the difference in the performance is negligible, since the difference in accuracy is below 10%. Since we are mostly interested in small values of ϵ, the best methods are Structure Learning 1 and 2.
- In dataset 3, for k = 2, Structure Learning 1 has the best performance for $\epsilon \in [0.01, 0.6)$ and Structure Learning 2 has the best performance for $\epsilon \in [0.6, 6.5)$. So the best methods are Structure Learning 1 and 2.

Based on these observations, the results we presented in the tables before and by using the aforementioned criterion for the accuracy difference, we find that the best ϵ value for each dataset is:

- Dataset 1: $\epsilon \simeq 0.5$
- Dataset 2: $\epsilon \simeq 0.1$
- Dataset 3: $\epsilon \simeq 0.02$

Once again, we notice that the third dataset performs the best, since it achieves high accuracy (low accuracy difference) using a small value of ϵ.

Finding the best Structure Learning method

Using these values for ϵ, we will attempt to find the optimal Structure Learning method using the samples of each dataset with the smallest accuracy difference. These are:

split_choice	str_choice	k	ϵ	Average Accuracy (Real Data)	Average Accuracy (Synthetic Data)	Average Accuracy (Difference)
3	1	1	0,5	59,19583	56,61	2,585833
1	1	2	0,5	59,19583	54,02333	5,1725
3	1	2	0,5	59,19583	52,87333	6,3225
1	1	1	0,5	59,19583	52,01083	7,185
2	1	1	0,5	59,19583	50,28667	8,909167
4	1	1	0,5	59,19583	48,85	10,34583

Table 13: Best accuracy dfference samples of dataset 1 with $\epsilon = 0.5$

split_choice	str_choice	k	ϵ	Average Accuracy (Real Data)	Average Accuracy (Synthetic Data)	Average Accuracy (Difference)
2	1	2	0,1	53,85333	48,09	5,763333
2	1	1	0,1	53,85333	47,26	6,593333
4	1	2	0,1	53,85333	47,09667	6,756667
4	2	1	0,1	53,85333	46,46	7,393333
4	1	1	0,1	53,85333	46,24667	7,606667
1	1	1	0,1	53,85333	46,22333	7,63

Table 14: Best accuracy dfference samples of dataset 2 with $\epsilon = 0.1$

split_choice	str_choice	k	ϵ	Average Accuracy (Real Data)	Average Accuracy (Synthetic Data)	Average Accuracy (Difference)
2	1	2	0,02	81,84667	77,1925	4,654167
2	1	2	0,02	81,84667	77,1925	4,654167
1	1	2	0,02	81,84667	76,46167	5,385
1	1	2	0,02	81,84667	76,46167	5,385
1	2	1	0,02	81,84667	74,8125	7,034167
2	2	1	0,02	81,84667	74,13083	7,715833

Table 15: Best accuracy dfference samples of dataset 3 with $\epsilon = 0.02$

However, the results are still inconclusive. The optimal value for str_choice seems to be 1 in most cases, followed by Structure Learning 2. However, as we mentioned before, Algorithm 4 does indeed perform quite well, but only in low noise conditions. Even though the ϵ parameter guarantees a certain amount of privacy, our experiments so far use low values of k. For greater values of k, the noise used in Algorithm 4 may increase dramatically, thus ruining its performance. Thus, we cannot yet say for certain which method is the optimal one.

In an another attempt to determine which of the methods performs the best, we will use the Kullback–Leibler divergence metric. Kullback–Leibler divergence is defined as:

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) * log\left(\frac{P(x)}{Q(x)}\right)$$

The KL-divergence is a measure of how one probability distribution P is different from a second, reference probability distribution Q. In our case, P is the probability distribution of the real data and Q is the probability distribution of the synthetic data. Q was created after we added Laplacian noise to the frequency distribution C or to the probability distribution P of the real data. A Kullback-Leibler divergence of 0 indicates that the two distributions in question are identical. In other words, the smaller the divergence, the more alike are the two distributions and their respective datasets. As a result, we desire small values of KL-divergence. By observing the following plots:

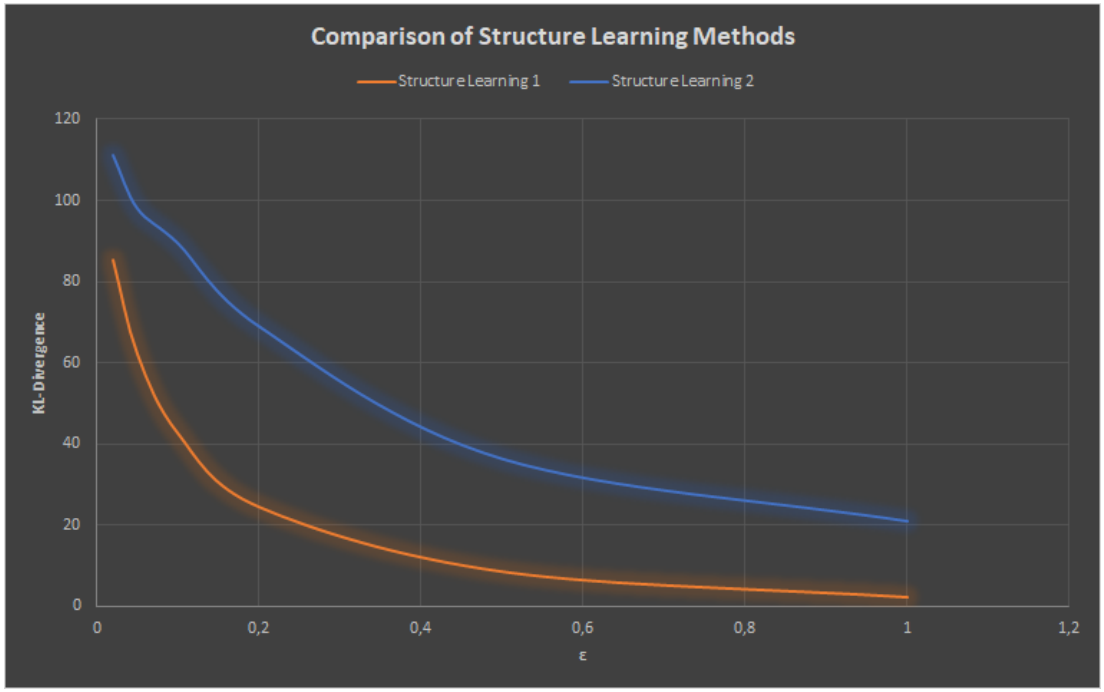


Figure 20: Dataset 1, k = 1

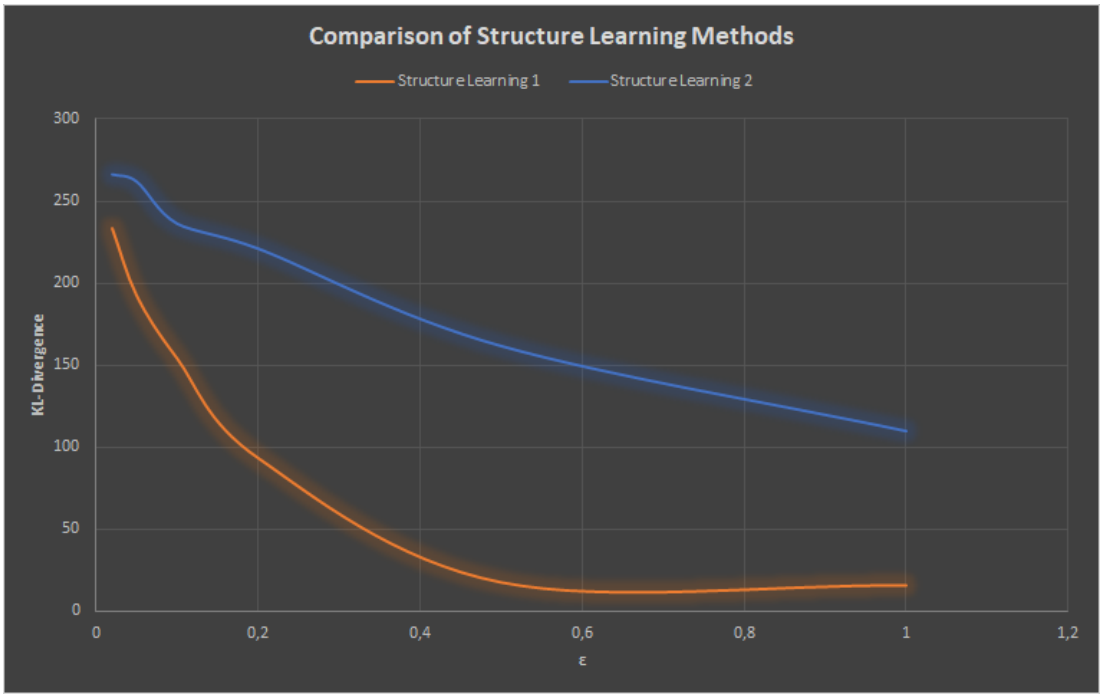


Figure 21: Dataset 1, k = 2

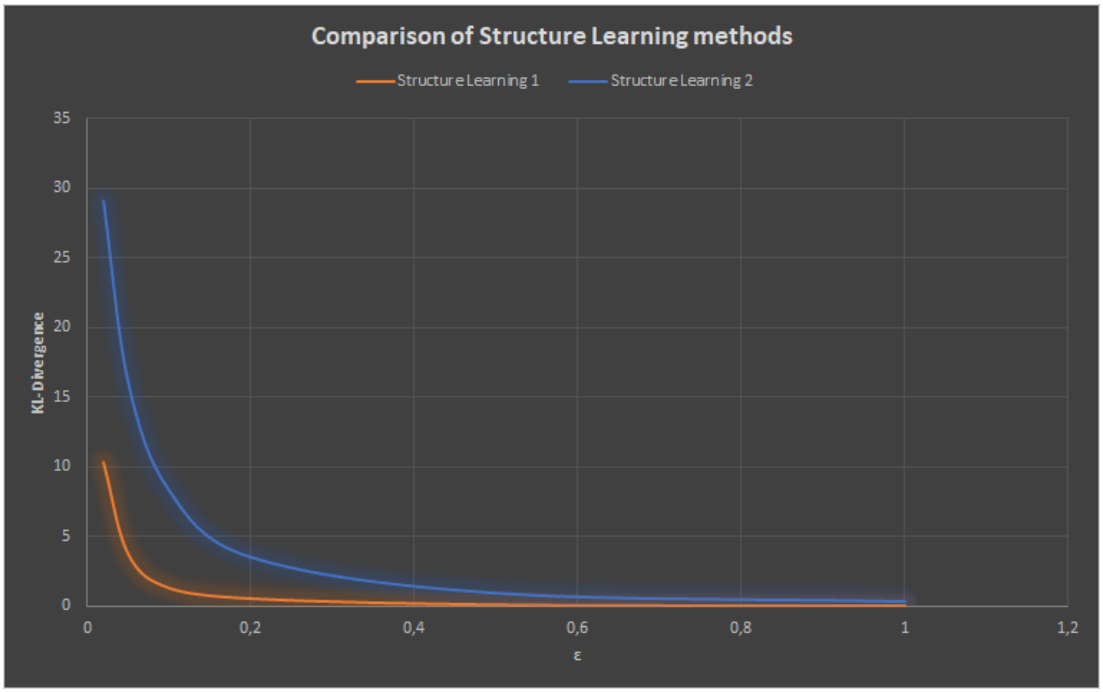


Figure 22: Dataset 2, k = 1

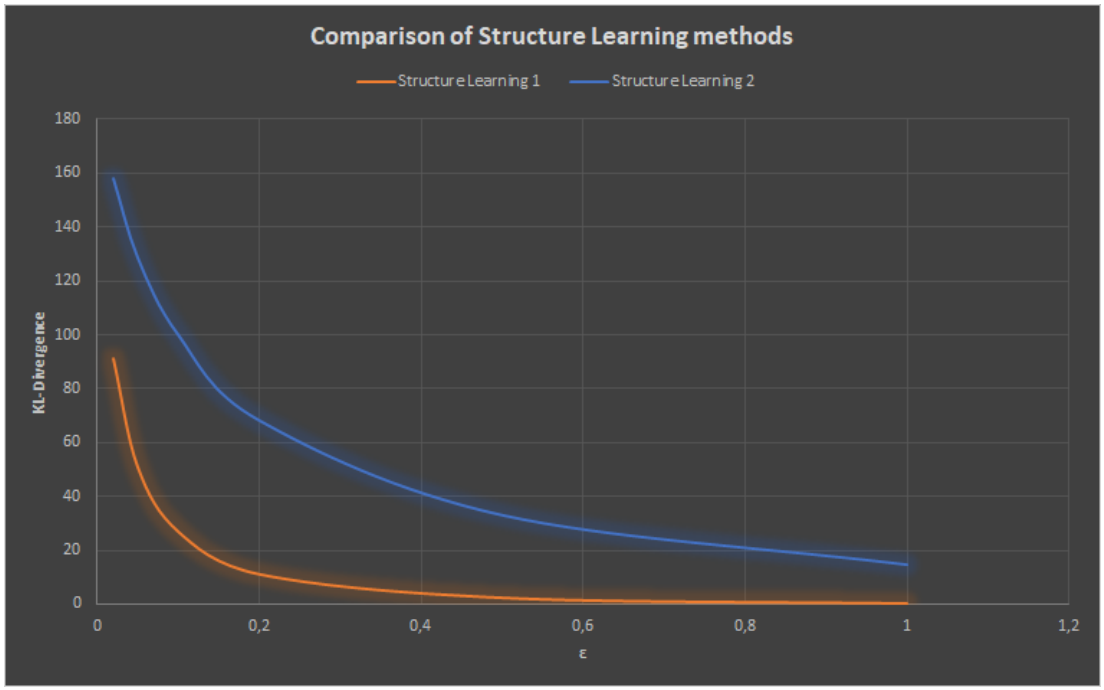


Figure 23: Dataset 2, k = 2

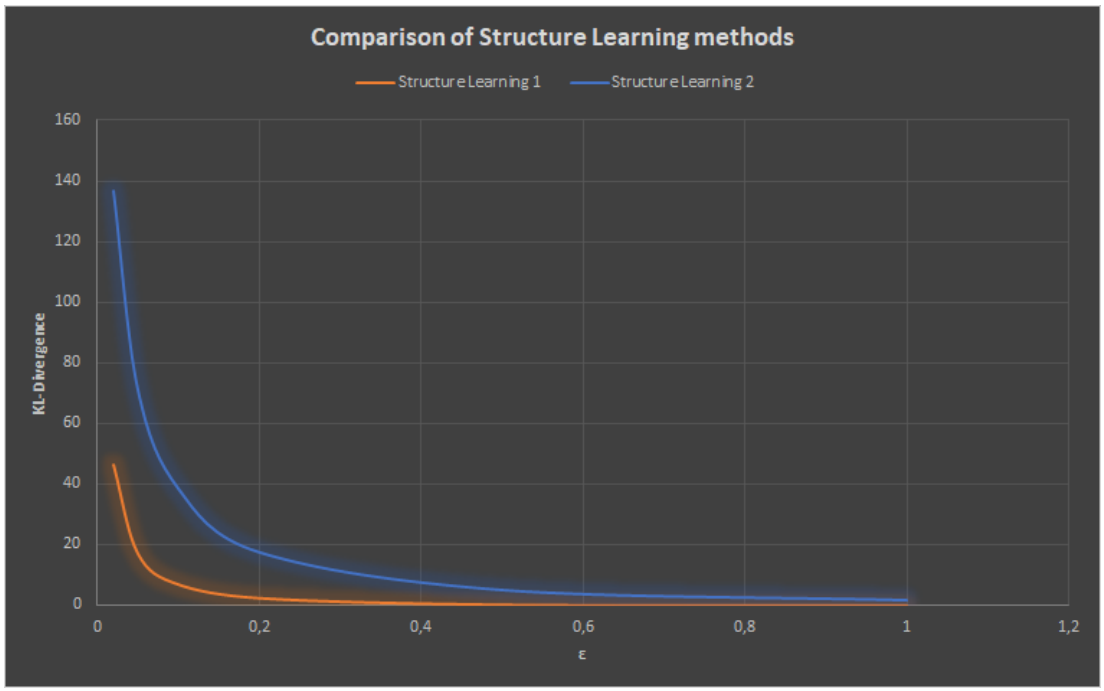


Figure 24: Dataset 3, k = 1

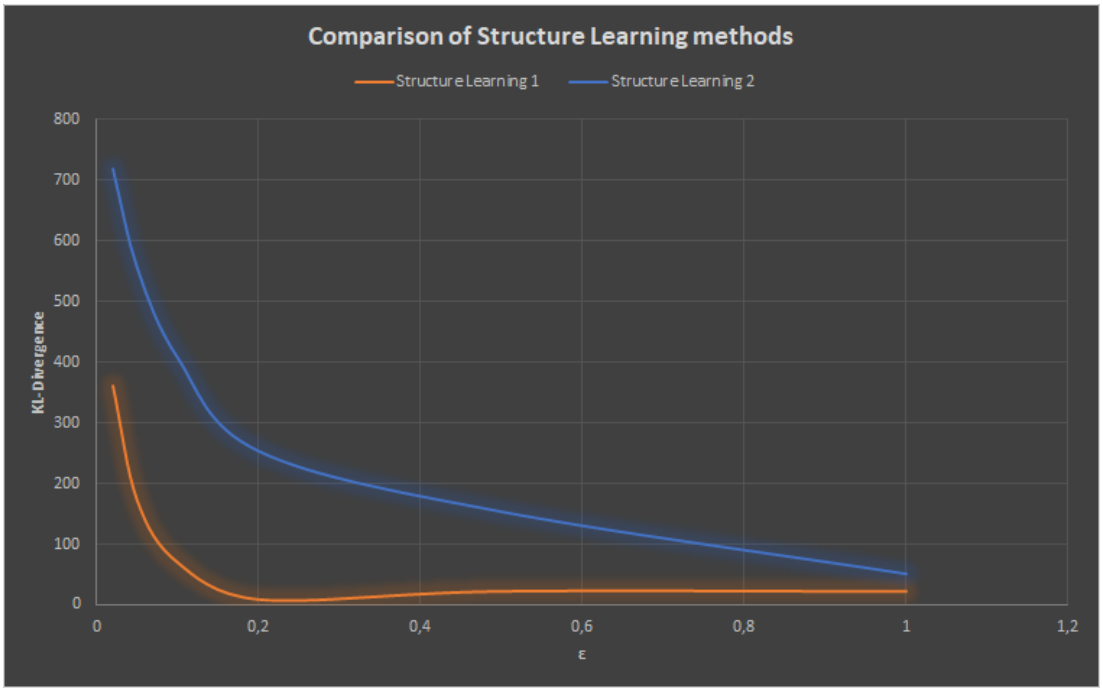


Figure 25: Dataset 3, k = 2

we can make certain remarks:

- Using KL-divergence, the performance superiority of Structure Learning 1 is clear in all cases, since it always yields a smaller KL-divergence between the original and noisy probability distributions.
- It is also worth noting that the KL-divergence of both methods for $\epsilon = 0.01$ (starting point) is always far greater for k=2 than it is for k=1, due to having larger probability distributions.
- KL-divergence decreases as ϵ increase, due to the less amount of noise required. This was to be expected.
- Our earlier choice for the values of the ϵ parameter is justified further, if we observe Structure Learning 1 for k = 1 for each dataset. If we do so, we will see that KL-divergence drops below 10 approximately when ϵ takes the value we chose before.
- Even though inferior to Structure Learning 1, Structure Learning 2 also yields very good results in datasets 2 and 3 for k = 1.

Comparison of distributed and centralized splitting methods

Until now, we used either the value 2 for split_choice or any of the distributed dataset values (1,2,3,4) for our experiments. Using both accuracy and KL-divergence,

we will determine whether split_choice = 2 is truly the optimal value and how well the distributed model performs compared to a centralized one (split_choice = 0). First, we will compare the accuracy difference for every dataset, structure learning method, k value and splitting method. Dataset 1 does not yield helpful results, so it is excluded from this part of the experiment.

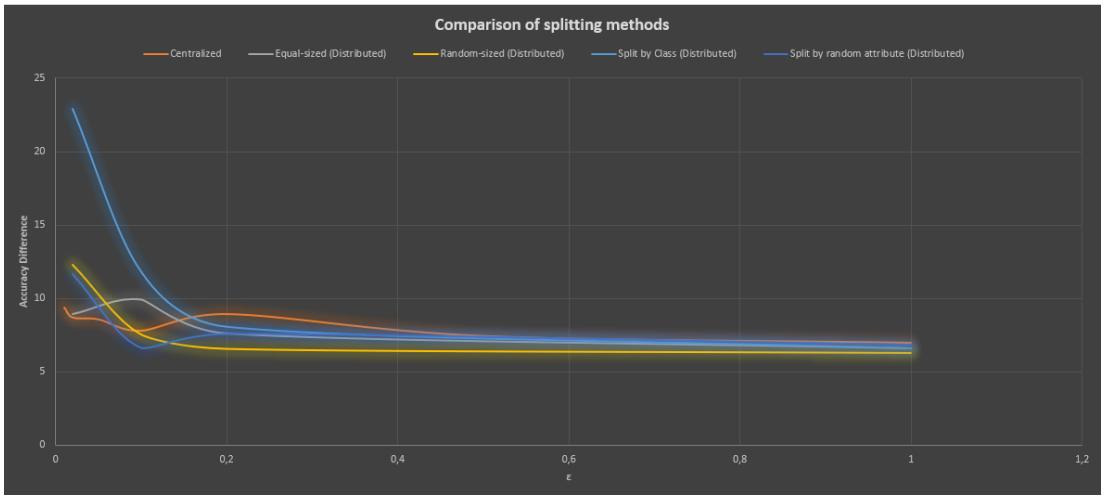


Figure 26: Dataset 2, k = 1 , Structure Learning 1

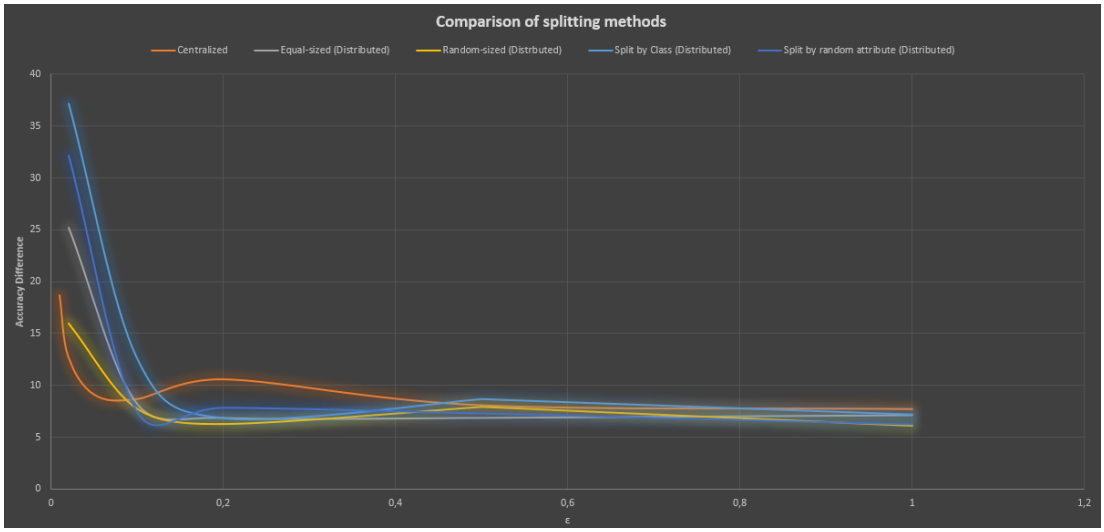


Figure 27: Dataset 2, k = 1 , Structure Learning 2

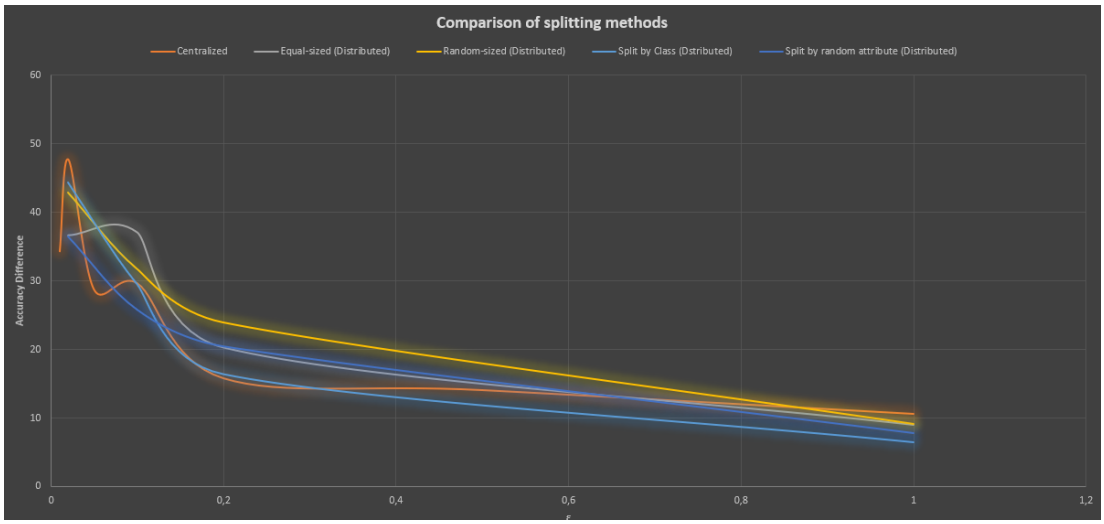


Figure 28: Dataset 2, k = 1 , Structure Learning 3

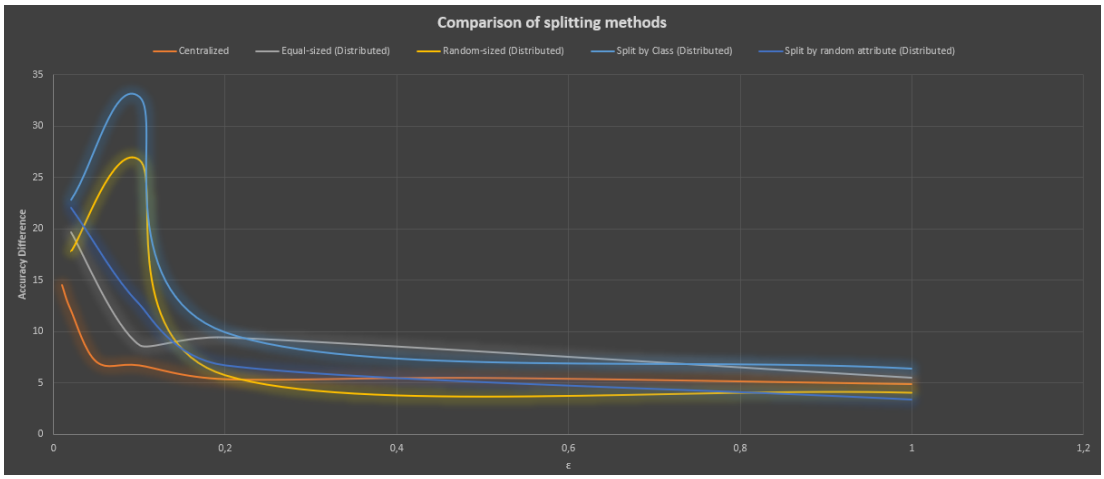


Figure 29: Dataset 2, k = 2 , Structure Learning 1

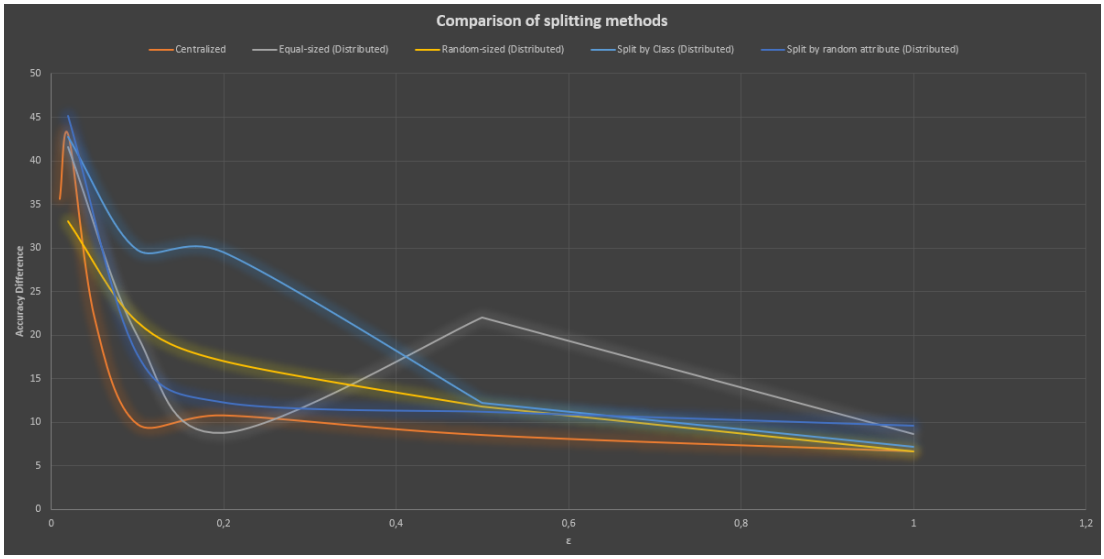


Figure 30: Dataset 2, k = 2 , Structure Learning 2

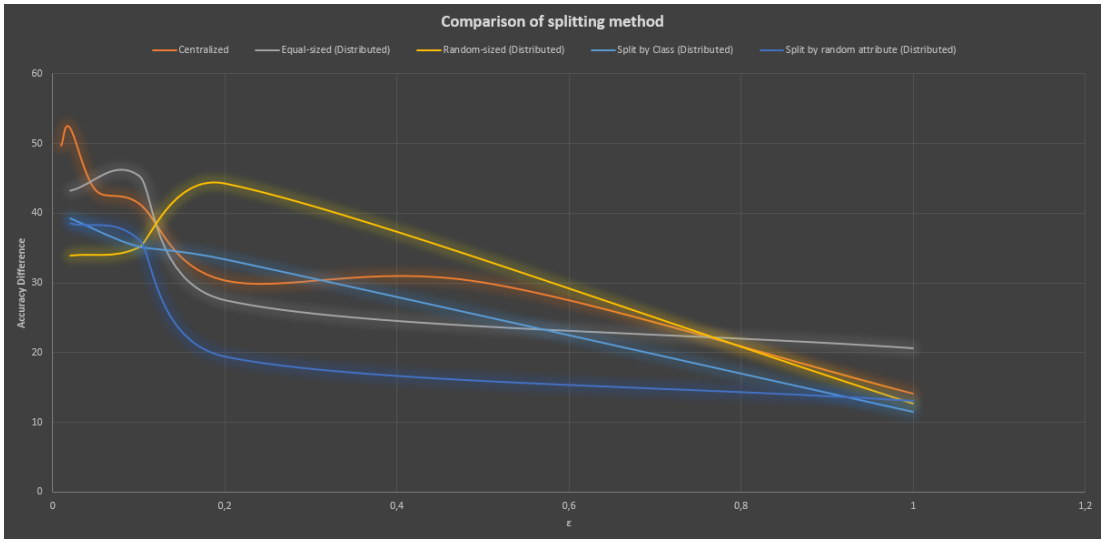


Figure 31: Dataset 2, k = 2 , Structure Learning 3

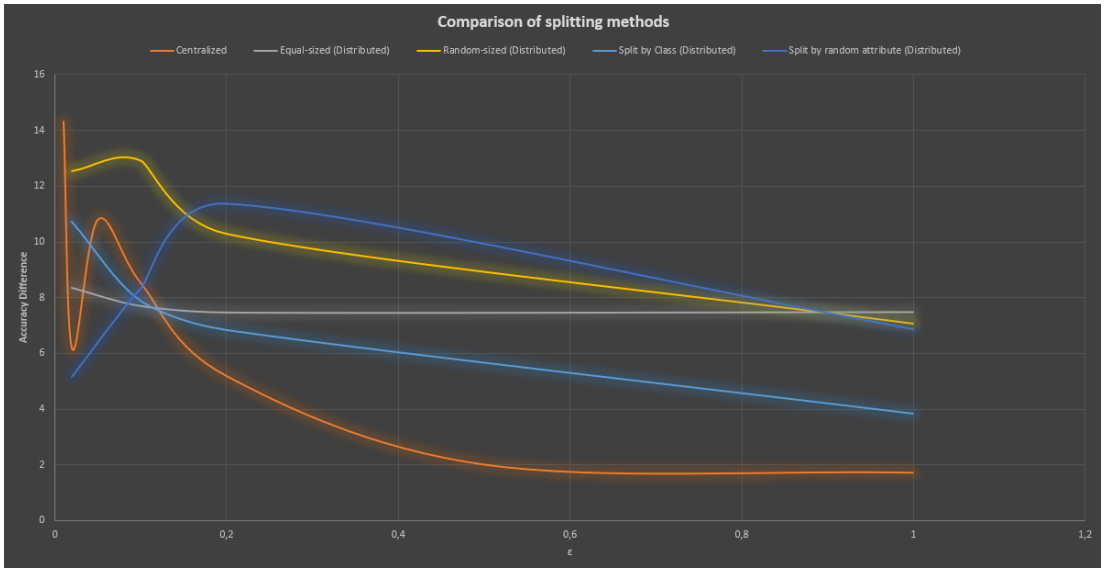


Figure 32: Dataset 3, k = 1 , Structure Learning 1

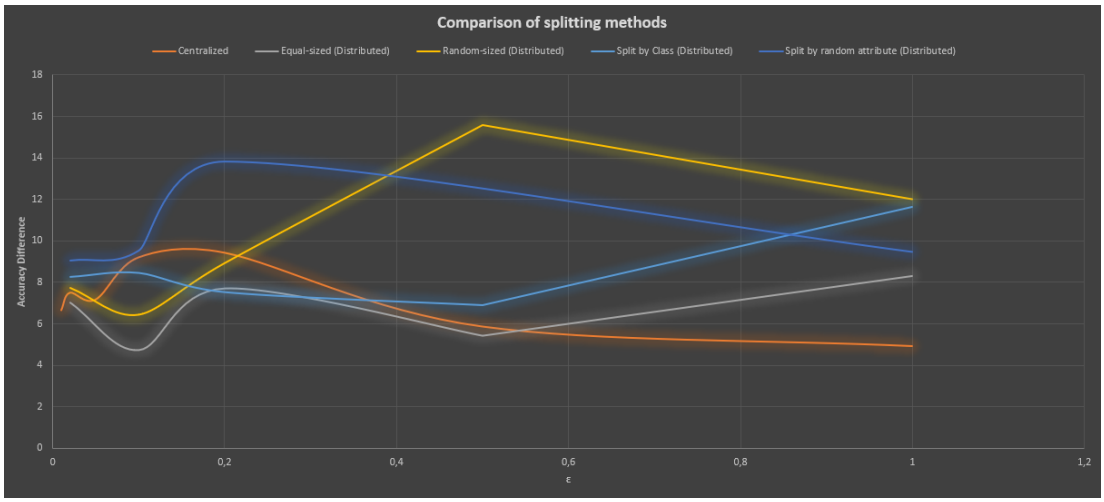


Figure 33: Dataset 3, k = 1 , Structure Learning 2

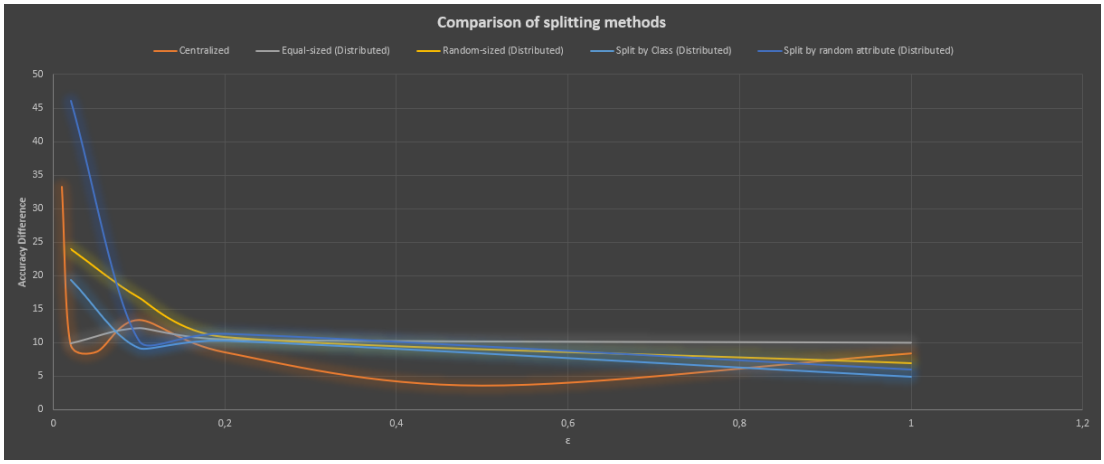


Figure 34: Dataset 3, k = 1 , Structure Learning 3

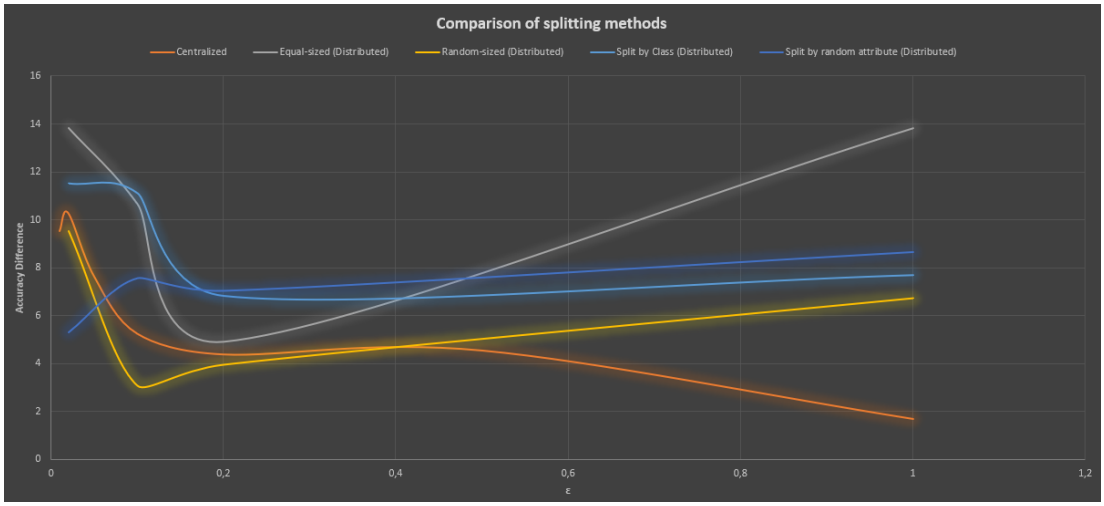


Figure 35: Dataset 3, k = 2 , Structure Learning 1

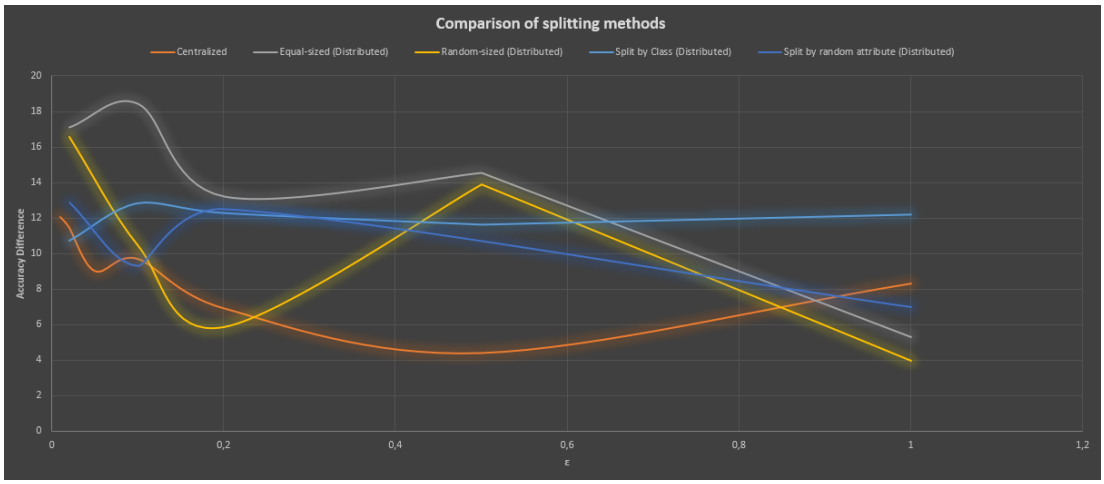


Figure 36: Dataset 3, k = 2 , Structure Learning 2

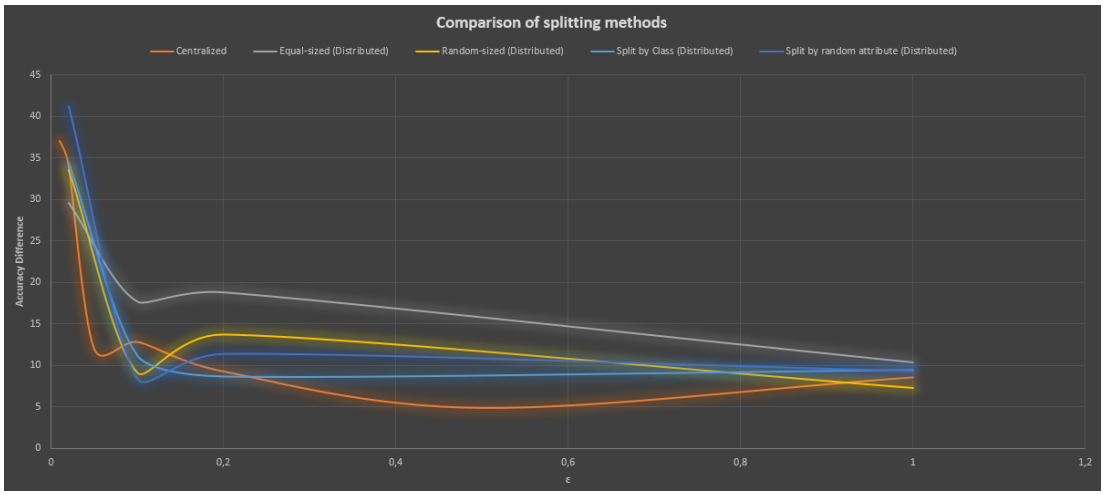


Figure 37: Dataset 3, k = 2 , Structure Learning 3

Remarks:

- In most of the examined cases, all splitting methods yield an accuracy difference that decreases when epsilon increases (With small fluctuations around $\epsilon = 1$).
- In most of the examined cases, the difference in the performance of the splitting methods is quite small.
- Usually, it is the centralized version that performs best. An exception is for Dataset = 2, k = 2 and Structure Learning 3 (Figure 31).

The centralized version seems to be the best of the splitting methods. However, since we are interested in the distributed model and the difference in performance between the methods is small ($\leq 10\%$), we will continue to consider 2 (Random-sized) to be the optimal value. We will now perform the same comparison using KL-divergence (dataset 1 included):

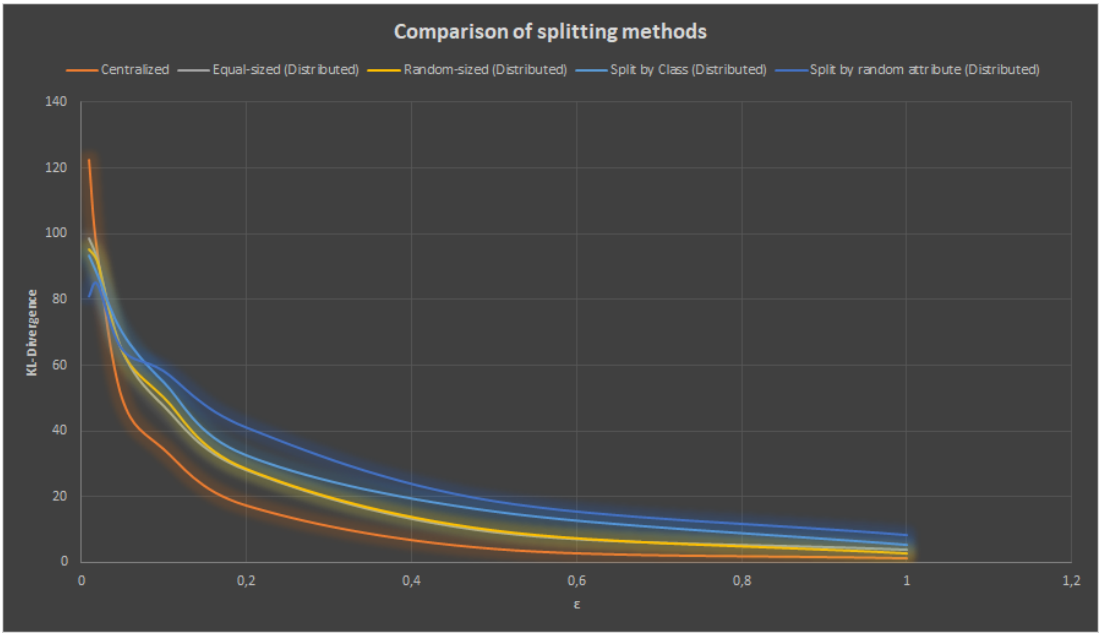


Figure 38: Dataset 1, k = 1 , Structure Learning 1

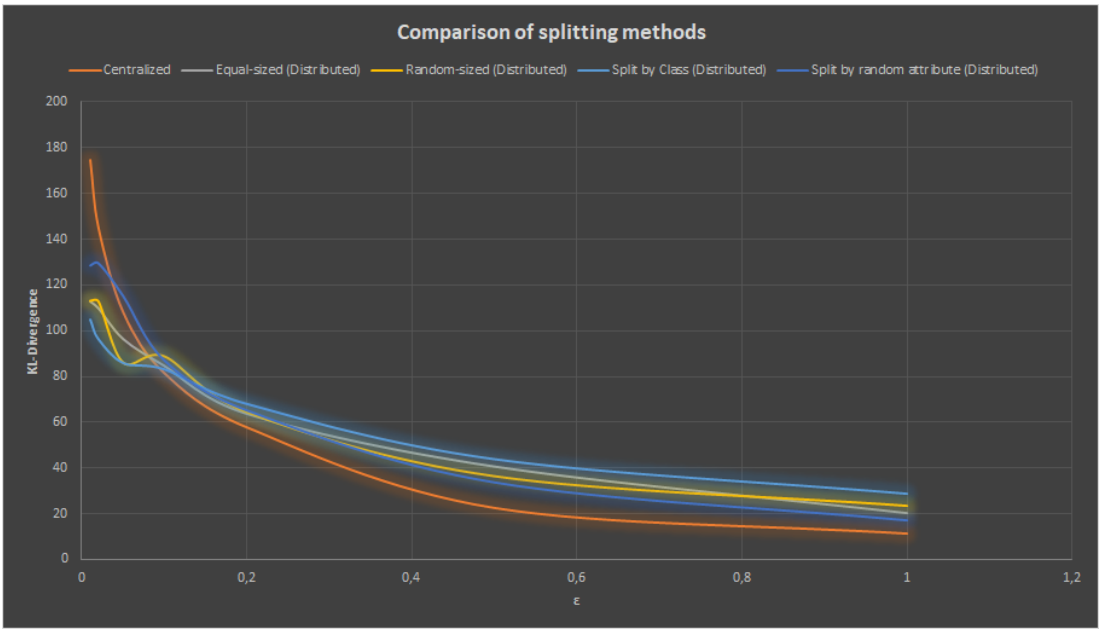


Figure 39: Dataset 1, k = 1 , Structure Learning 2

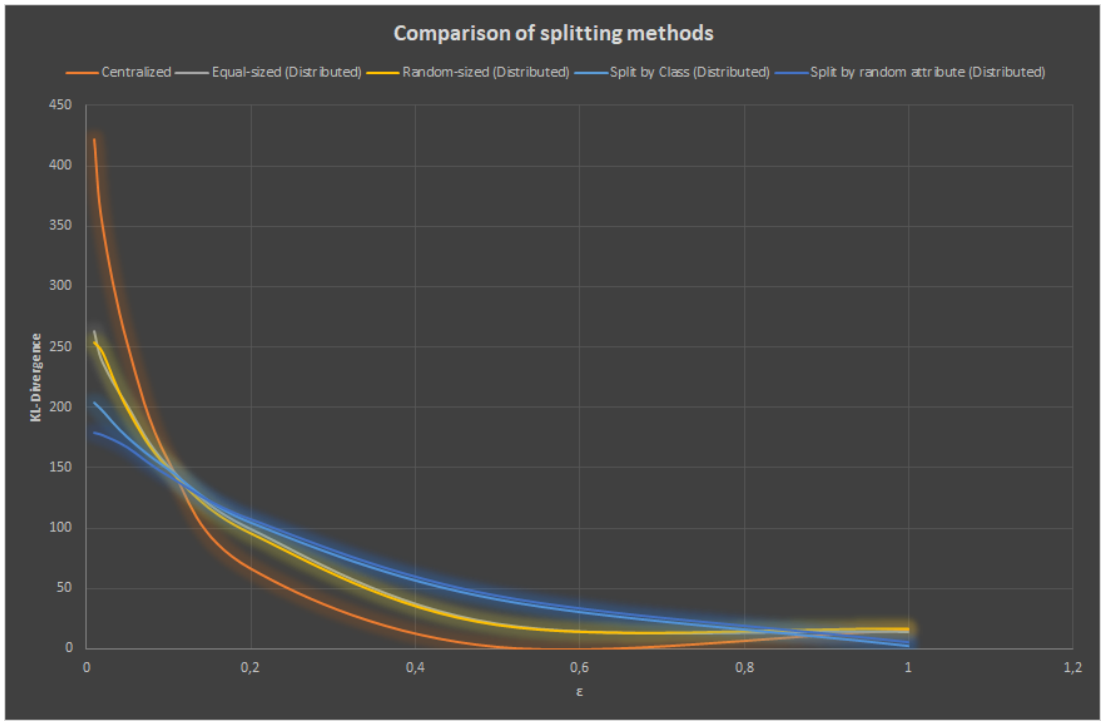


Figure 40: Dataset 1, k = 2 , Structure Learning 1

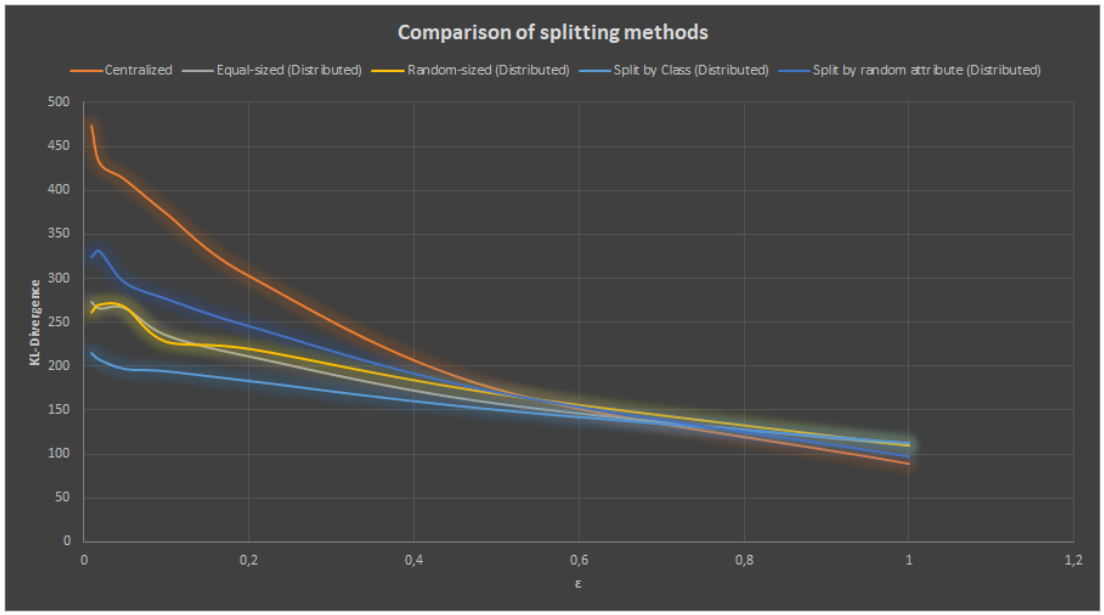


Figure 41: Dataset 1, k = 2 , Structure Learning 2

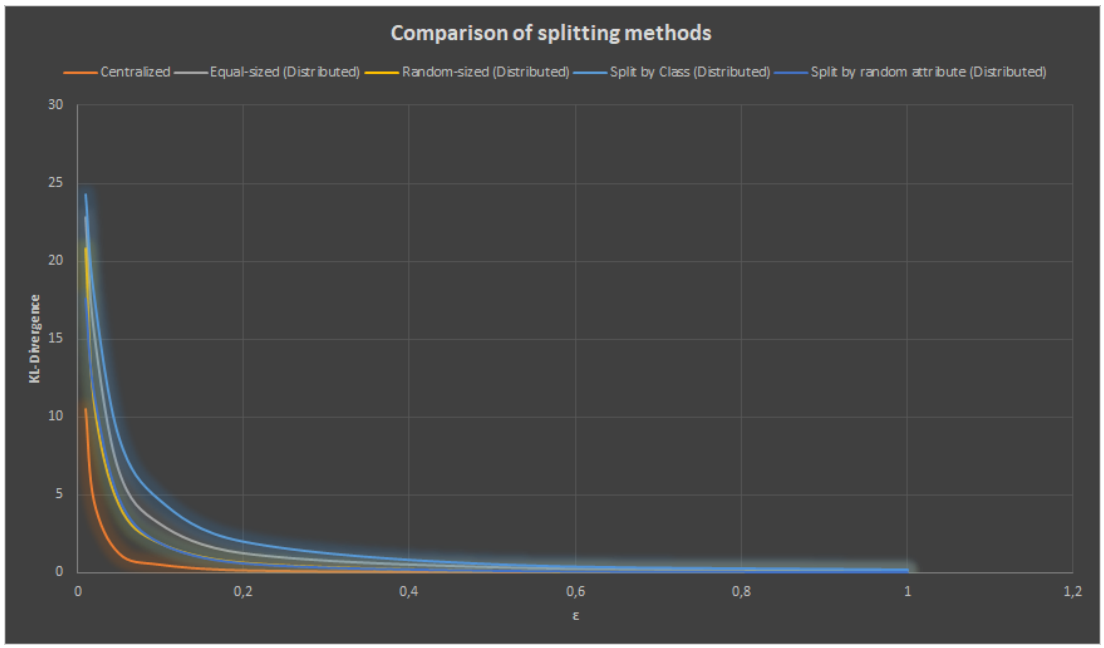


Figure 42: Dataset 2, k = 1 , Structure Learning 1

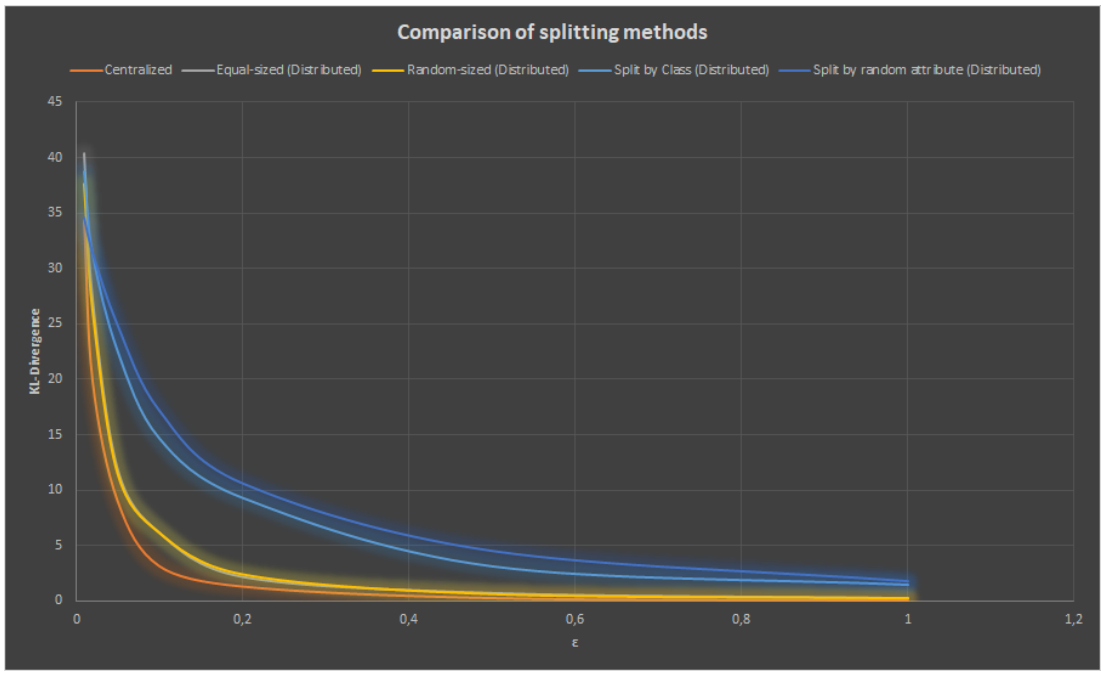


Figure 43: Dataset 2, k = 1 , Structure Learning 2

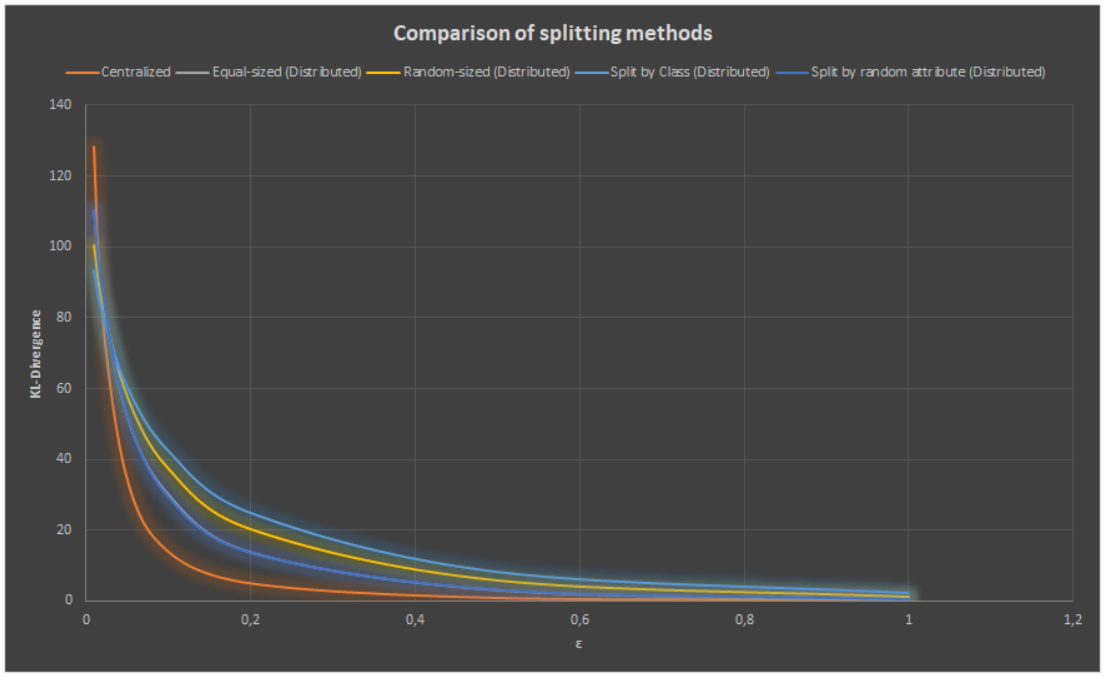


Figure 44: Dataset 2, k = 2 , Structure Learning 1

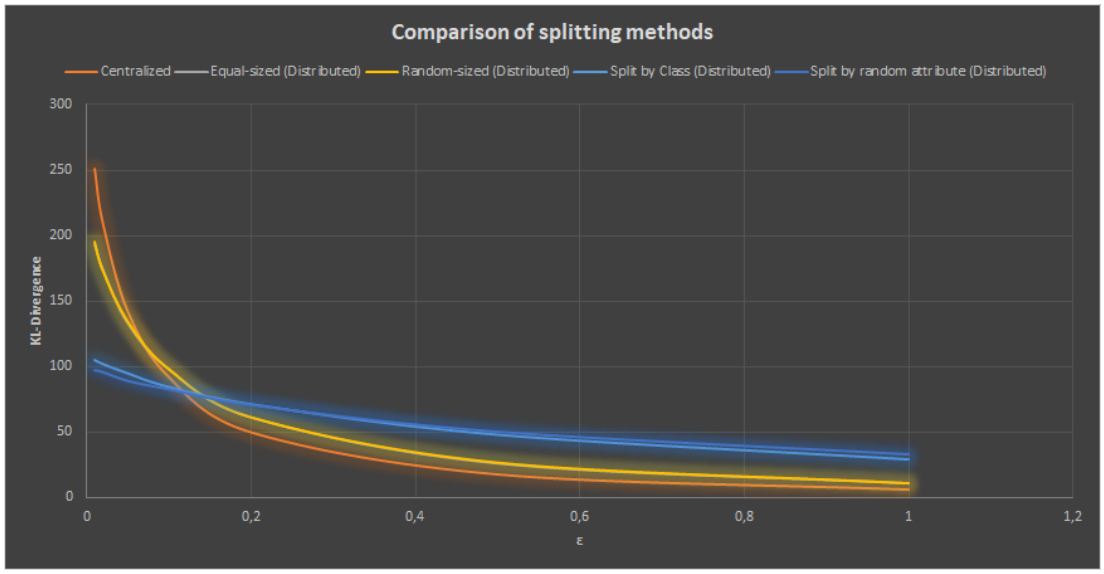


Figure 45: Dataset 2, k = 2 , Structure Learning 2

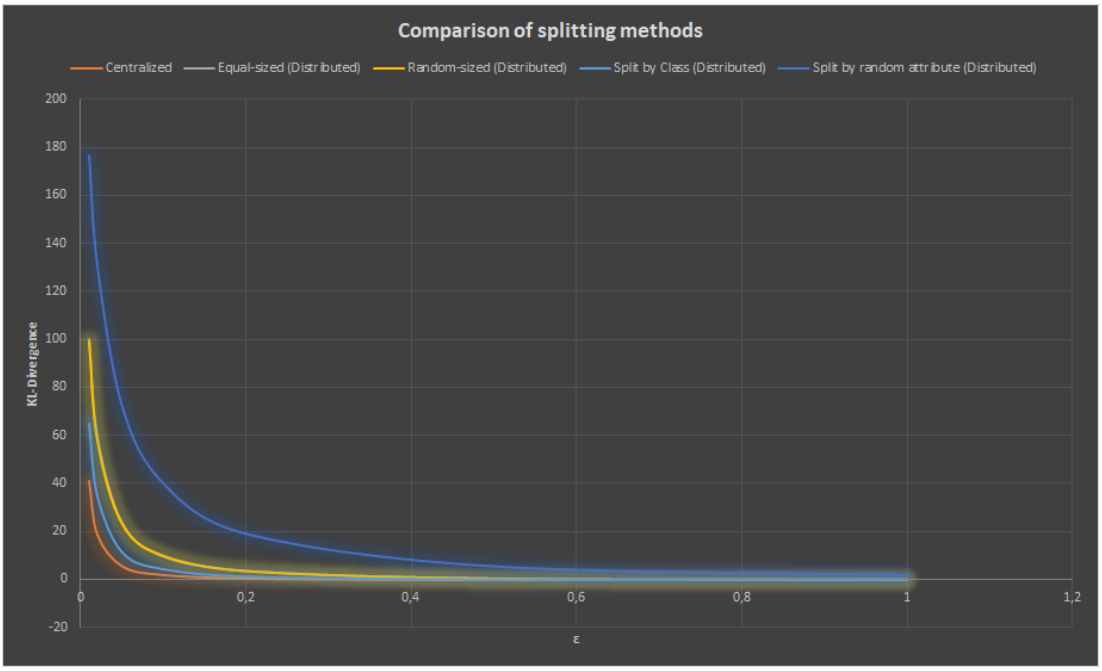


Figure 46: Dataset 3, k = 1 , Structure Learning 1

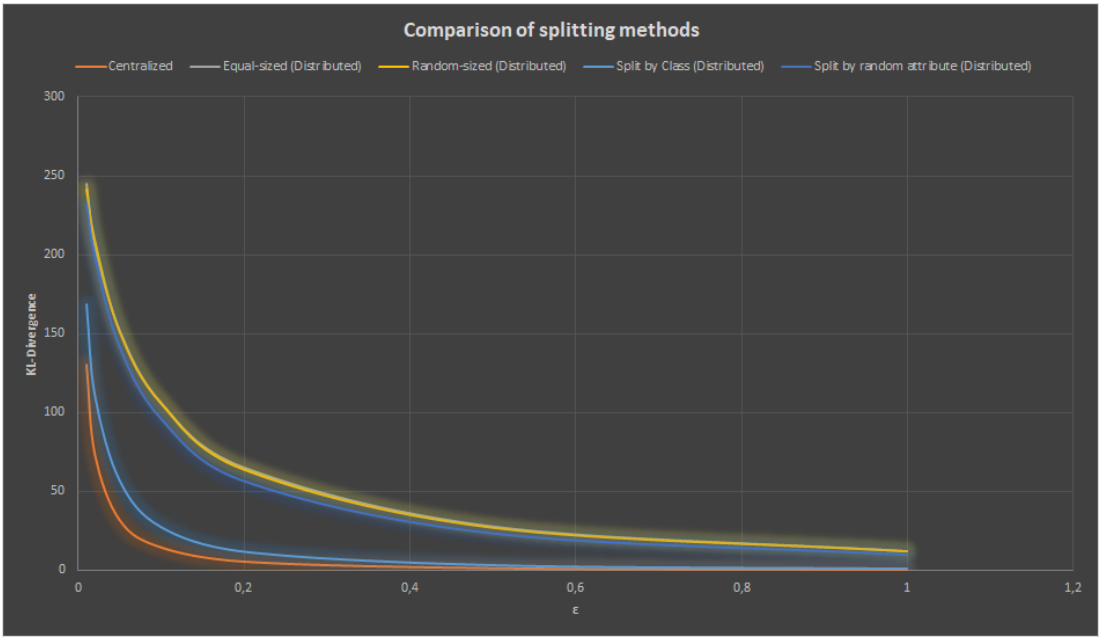


Figure 47: Dataset 3, k = 1 , Structure Learning 2

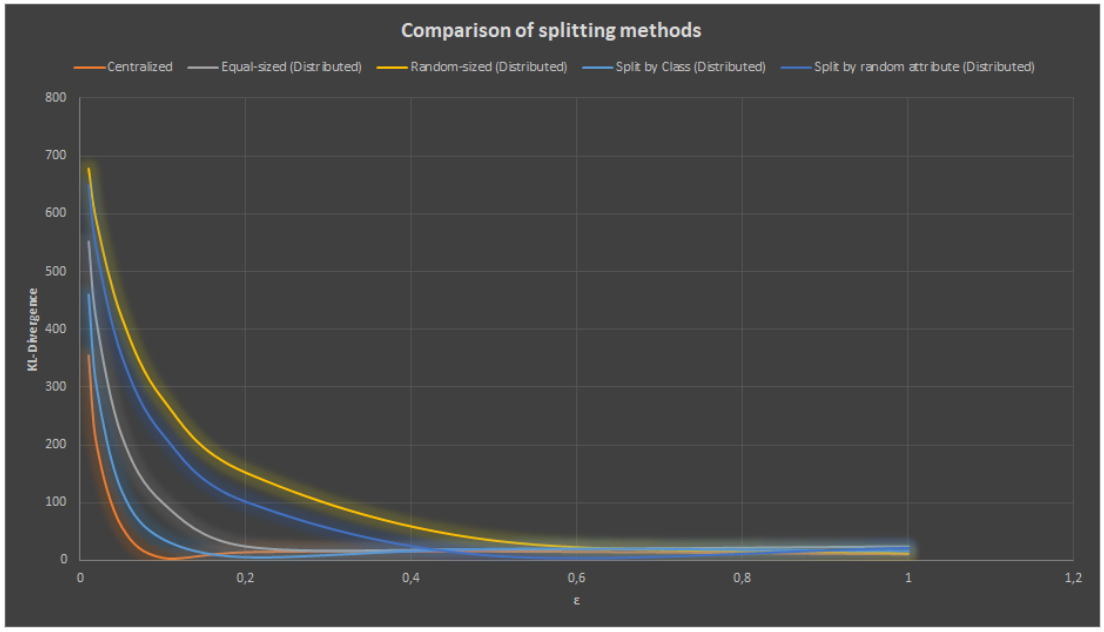


Figure 48: Dataset 3, k = 2 , Structure Learning 1

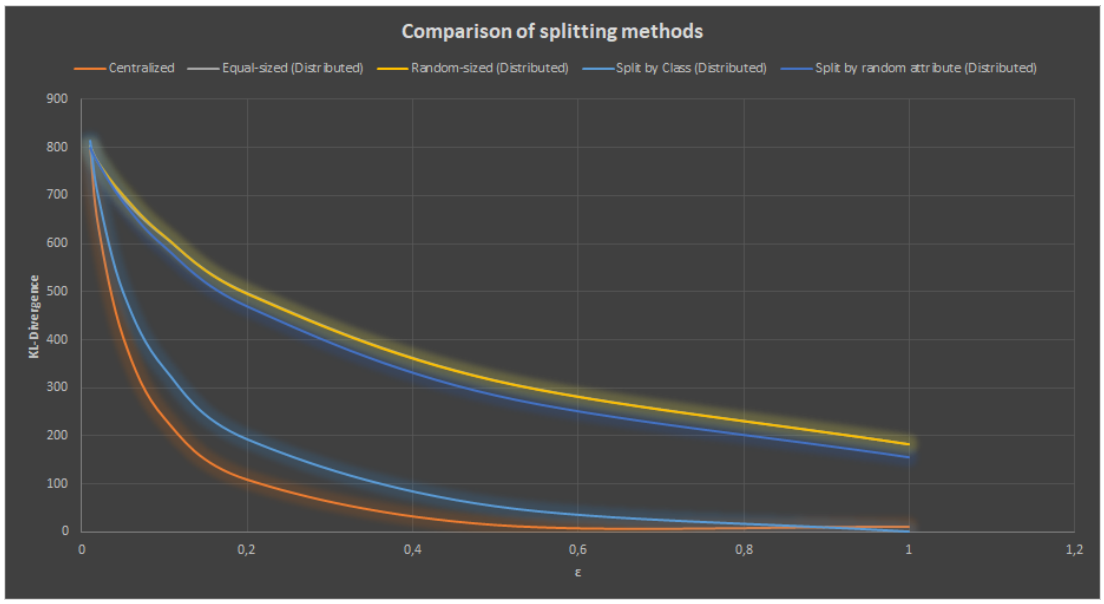


Figure 49: Dataset 3, k = 2 , Structure Learning 2

Remarks:

- As we noticed the last time we used KL-divergence:
  - As ϵ increases, KL-divergence decreases due to the smaller amount of noise
  - KL-divergence values (especially for small ϵ values) are greater for k=2 that they are for k=1
- The centralized version seems to be superior once again for all cases. One exception is for Dataset 1, k = 2 , Structure Learning 2 (Figure 41) and $\epsilon \leq 0.5$
- The distributed splitting methods once again seem to perform the approximately the same. Since we cannot claim which of them is the best, we will keep considering 2 (Random-sized) as the optimal value of the split_choice parameter.

3.7 Conclusions & Future Work

In this chapter, we introduced Bayesian Networks and discussed various methods on how to utilize them to generate differentially private synthetic data from real data on the distributed model. Next after implementing these methods in code, we performed a detailed experimental evaluation in an attempt to grade our model using classifier accuracy and the KL-divergence metric. Another goal we accomplished is to choose optimal, or at the very least, appropriate values for the hyperparameters of our model. With regards to the Structure Learning methods, Algorithm 4 seems to perform the best (at least for small values of k), followed closely by Algorithm 5. We also performed a comparison of the centralized and distributed model and found out that the centralized model outperforms the distributed ones, but the difference is in most cases inconsequential.

However, there are still improvements and additions that could be done. Those that we would suggest are:

1. To grade our model, we experimented with different k values, but we did not choose an optimal one. In section 4.6 of the PrivBayes paper (21, 1), there is an idea about using the definition of θ-usefulness to choose an optimal value for k.
2. Instead of using mutual information I as a scoring function for the generated Bayesian Networks, there is an improved solution suggested in section 4.3 and 4.4 of the PrivBayes paper, a F function that apparently yields better results than I as showed by Figure 2 of the same paper.
3. One could improve the performance of the algorithms we used and suggest alternative ones.

4 Neural networks

4.1 Introduction

Our imagination has long been captivated by visions of machines that can learn and imitate human intelligence. Nowadays, software programs that can acquire new knowledge and skills through experience are becoming increasingly common. We use such machine learning programs to discover new music that we enjoy, and to quickly find the exact shoes we want to purchase online. Machine learning programs allow us to dictate commands to our smartphones and allow our thermostats to set their own temperatures. Machine learning programs can decipher sloppily-written mailing addresses better than humans and guard credit cards from fraud more vigilantly. From investigating new medicines to estimating the page views for versions of a headline, machine learning software is becoming central to many industries. Machine learning has even encroached on activities that have long been considered uniquely human, such as writing columns in a newspaper.

When most people hear "Machine Learning," they picture some kind of highly-advanced robot. But Machine Learning is not just a futuristic fantasy, it's already here. In fact, it has been around for decades in some specialized applications, such as Optical Character Recognition (OCR). But the first ML application that really became mainstream, improving the lives of hundreds of millions of people, took over the world back in the 1990s: it was the spam filter. By observing thousands of emails that have been previously labeled as either spam or ham, spam filters learn to classify new messages. It was followed by hundreds of ML applications that now quietly power hundreds of products and features that you use regularly, from better recommendations to voice search. Machine learning is the design and study of software artifacts that use past experience to make future decisions; it is the study of programs that learn from data. The fundamental goal of machine learning is to generalize, or to induce an unknown rule from examples of the rule's application.

4.2 Machine Learning Tasks

Machine learning tasks can be split into two major categories, supervised and unsupervised learning.

The majority of practical machine learning uses *supervised learning*. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output (Y = f(x)). The goal is to approximate the mapping function so well that when you have

new input data (x) that you can predict the output variables (Y) for that data. It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance. In Supervised learning, you train the machine using data which is well "labeled." It means some data is already tagged with the correct answer. A supervised learning algorithm learns from labeled training data, helps you to predict outcomes for unforeseen data. Supervised learning can be split into two main subcategories:

- Classification: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
- Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Unsupervised learning is where you only have input data (X) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. These are called unsupervised learning because unlike supervised learning there are no correct answers and there is no teacher. Algorithms are left to their own devices to discover and present the underlying structure in the data. Unsupervised learning can be split into two main subcategories:

- Clustering: It mainly deals with finding a structure or pattern in a collection of uncategorized data. Clustering algorithms will process your data and find natural clusters(groups) if they exist in the data. You can also modify how many clusters your algorithms should identify. It allows you to adjust the granularity of these groups.
- Association: This unsupervised technique is about discovering interesting relationships between variables in large databases. For example, people that buy a new home most likely to buy new furniture.

For this study, we are interested in classification problems.

4.3 Training data and test data

The observations in the *training set* comprise the experience that the algorithm uses to learn. In supervised learning problems, each observation consists of an observed response variable and one or more observed explanatory variables.

The *test set* is a similar collection of observations that is used to evaluate the performance of the model using some performance metric. It is important that no observations from the training set are included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it. A program that generalizes well will be able to effectively perform a task with new data. In contrast, a program that memorizes the training data by learning an overly complex model could predict the values of the response variable for the training set accurately, but will fail to predict the value of the response variable for new examples.

Memorizing the training set is called *overfitting*. A program that memorizes its observations may not perform its task well, as it could memorize relations and structures that are noise or coincidence. Balancing memorization and generalization, or over-fitting and under-fitting, is a problem common to many machine learning algorithms.

In addition to the training and test data, a third set of observations, called a *validation or hold-out set*, is sometimes required. The validation set is used to tune variables called *hyperparameters* (learning rate, batch size e.t.c.), which control how the model is learned. The program is still evaluated on the test set to provide an estimate of its performance in the real world; its performance on the validation set should not be used as an estimate of the model's real-world performance since the program has been tuned specifically to the validation data. It is common to partition a single set of supervised observations into training, validation, and test sets. There are no requirements for the sizes of the partitions, and they may vary according to the amount of data available. It is common to allocate 50% or more of the data to the training set, 25% to the test set, and the remainder to the validation set.

4.4 Introduction to Neural Networks

The most successful model in the context of pattern recognition is the feed-forward neural network. The term 'neural network' has its origins in attempts to find mathematical representations of information processing in biological systems (McCulloch and Pitts, 1943; Widrow and Hoff, 1960; Rosenblatt, 1962; Rumelhart et al., 1986). Indeed, it has been used very broadly to cover a wide range of different models, many of which have been the subject of exaggerated claims regarding their biological plausibility. From the perspective of practical applications of pattern recognition, however, biological realism would impose entirely unnecessary constraints.

Birds inspired us to fly, burdock plants inspired velcro, and nature has inspired

many other inventions. It seems only logical, then, to look at the brain's architecture for inspiration on how to build an intelligent machine. This is the key idea that inspired artificial neural networks (ANNs). However, although planes were inspired by birds, they don't have to flap their wings. Similarly, ANNs have gradually become quite different from their biological counterparts. Some researchers even argue that we should drop the biological analogy altogether (e.g., by saying "units" rather than "neurons"), lest we restrict our creativity to biologically plausible systems

ANNs are at the very core of Deep Learning. They are versatile, powerful and scalable, making them ideal to tackle large and highly complex Machine Learning tasks, such as classifying billions of images (e.g. Google Images), powering speech recognition services (e.g. Apple's Siri) or recommending the best videos to watch to hundreds of millions of users every day (e.g. YouTube).

The artificial neural networks, that will concern us, are powerful non linear models for classification and regression that use a different strategy to overcome the perceptron's limitations. Artificial neural networks are described by three components. The first is the model's architecture, or topology, which describes the layers of neurons and structure of the connections between them. The second component is the activation function used by the artificial neurons. The third component is the learning algorithm that finds the optimal values of the weights.

There are two main types of artificial neural networks:

- Feed-forward neural networks are the most common type of neural net, and are defined by their directed acyclic graphs. Signals only travel in one direction—towards the output layer—in feed-forward neural networks. Feed-forward neural networks are commonly used to learn a function to map an input to an output.
- Feed-back neural networks, or recurrent neural networks, do contain cycles. The feed-back cycles can represent an internal state for the network that can cause the network's behavior to change over time based on its input. The temporal behavior of feed-back neural networks makes them suitable for processing sequences of inputs.

The multilayer perceptron (MLP) is the one of the most commonly used artificial neural networks. The name is a slight misnomer; a multilayer perceptron is not a single perceptron with multiple layers, but rather multiple layers of artificial neurons that can be perceptrons. The layers of the MLP form a directed, acyclic graph. Generally, each layer is fully connected to the subsequent layer; the output of each artificial neuron

in a layer is an input to every artificial neuron in the next layer towards the output. MLPs have three or more layers of artificial neurons.The input layer consists of simple input neurons. The input neurons are connected to at least one hidden layer of artificial neurons. The hidden layer represents latent variables; the input and output of this layer cannot be observed in the training data. Finally, the last hidden layer is connected to an output layer.

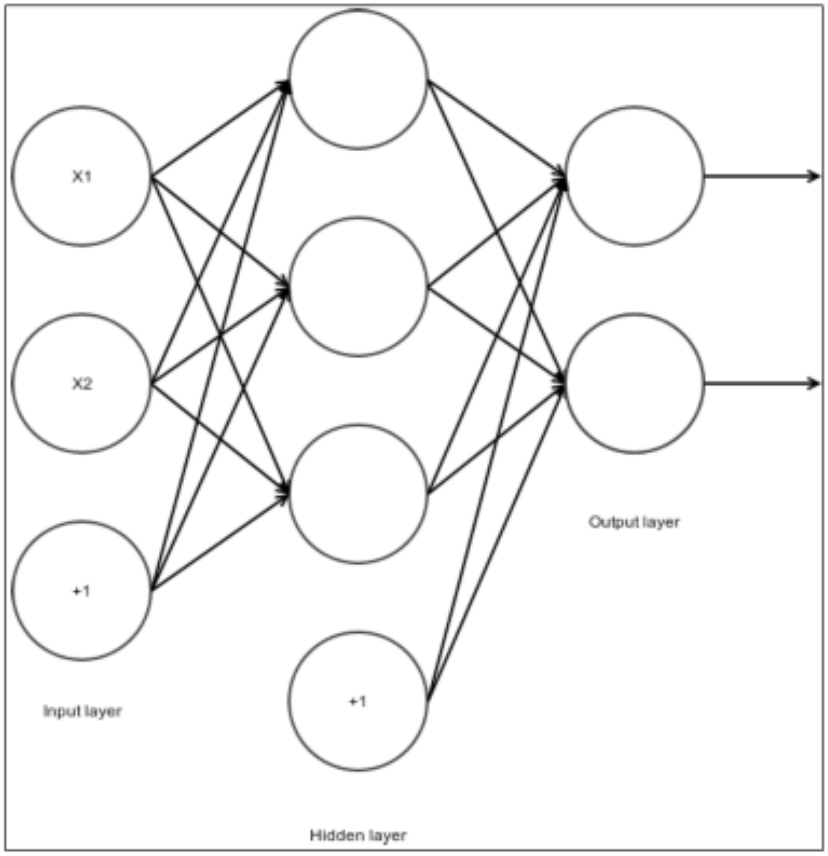


Figure 50: A simple neural network with two inputs, two outputs and one hidden layer with three neurons

The artificial neurons, or **units**, in the hidden layer commonly use non linear activation functions such as the hyperbolic tangent function and the logistic function, which are given by the following equations respectively:

$$f(x) = \tanh(x)$$

$$f(x) = \frac{1}{1+e^{-x}}$$

Figure 51: The most common activation functions

As with other supervised models, our goal is to find the values of the weights that minimize the value of a cost function. The mean squared error cost function is commonly used with multilayer perceptrons. It is given by the following equation, where m is the number of training instances:

$$MSE = \frac{1}{m}\sum_{i=1} m\left(y_i - f(x_i)\right)^2$$

Figure 52: The most common loss function

Obviously, there are many other activation and loss functions, each suited to specific applications. The reason we wish to minimize the aforementioned loss function is because we want the difference between the label y_i (or target) of the sample and the response $f(x_i)$ of the network (or prediction) about the sample to be a small as possible. In other words, we want for y_i to be equal to $f(x_i)$, so that the network classifies the samples correctly at (almost) all times. In the next section, we will discuss the process with which we accomplish this.

4.4.1 Deep Learning

Deep neural networks, which are remarkably effective for many machine learning tasks, define parameterized functions from inputs to outputs as compositions of many layers of basic building blocks, such as affine transformations and simple non linear functions. Commonly used examples of the latter are sigmoids and rectified linear units (ReLUs). By varying parameters of these blocks, we can "train" such a parameterized function with the goal of fitting any given finite set of input/output examples.

Deep learning aims to extract complex features from high-dimensional data and use them to build a model that relates inputs to outputs (e.g., classes). Deep learning architectures are usually constructed as multi-layer networks so that more abstract features are computed as non linear functions of lower-level features. We mainly focus on supervised learning, where the training inputs are labeled with correct classes, but in principle our approach can also be used for unsupervised, privacy-preserving learning,

too. Multi-layer neural networks (2 hidden layers or more) are the most common form of deep learning architectures.

In general, the values computed in higher layers represent more abstract features of the data. The first layer is composed of the raw features extracted from the data, e.g., the intensity of colors in each pixel in an image or the frequency of each word in a document. The outputs of the last layer correspond to the abstract answers produced by the model. If the neural network is used for classification, these abstract features also represent the relation between input and output. The non linear function f and the weight matrices determine the features that are extracted at each layer. The main challenge in deep learning is to automatically learn from training data the values of the parameters (weight matrices) that maximize the objective of the neural network (e.g., classification accuracy).

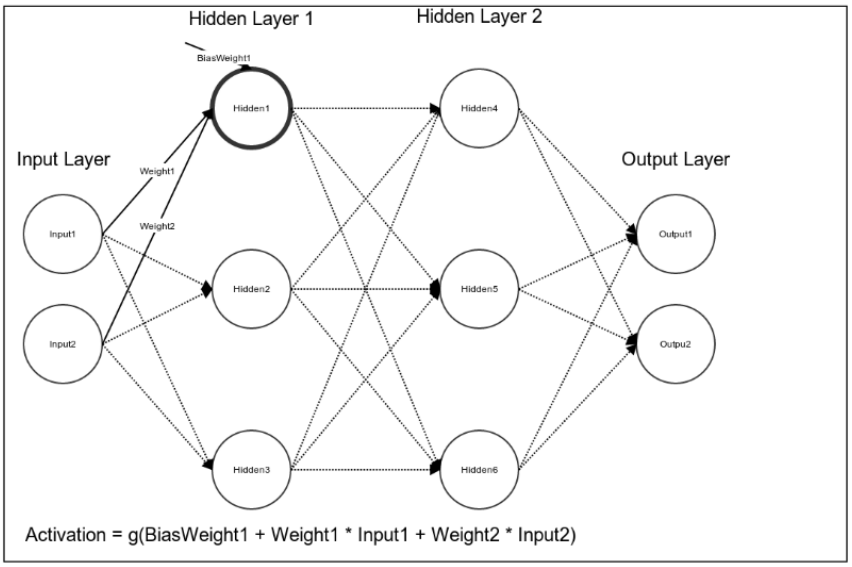
4.5 Learning Process - Minimizing the cost function

Learning the parameters of a neural network is a non linear optimization problem. In supervised learning, the objective function is the output of the neural network. The algorithms that are used to solve this problem are typically variants of *gradient descent*. Simply put, gradient descent starts at a random point (set of parameters for the neural network), then, at each step, computes the gradient of the non linear function being optimized and updates the parameters so as to decrease the gradient. This process continues until the algorithm converges to a local optimum.

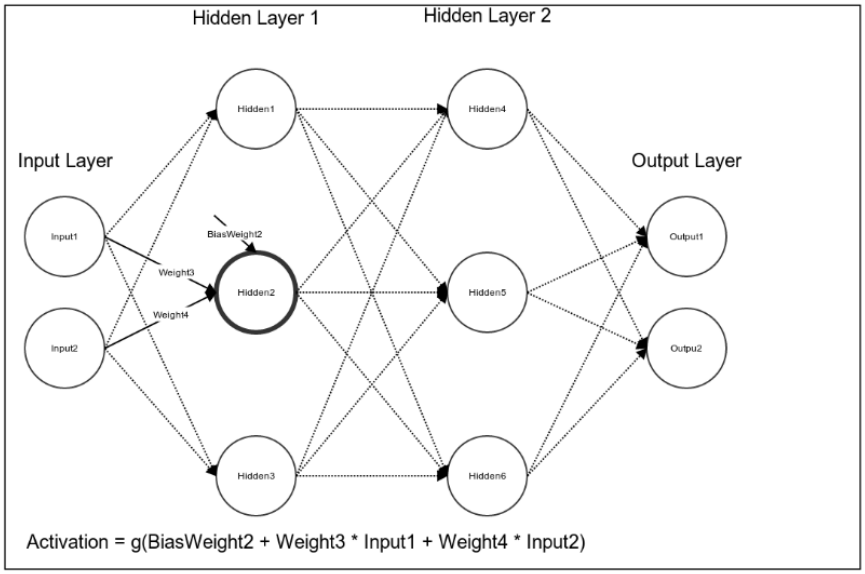
In a neural network, the gradient of each weight parameter is computed through feed-forward and back-propagation procedures. Feed-forward sequentially computes the output of the network given the input data and then calculates the error, i.e., the difference between this output and the true value of the function. Back-propagation propagates this error back through the network and computes the contribution of each neuron to the total error. The gradients of individual parameters are computed from the neurons' activation values and their contribution to the error.

4.5.1 Forward propagation

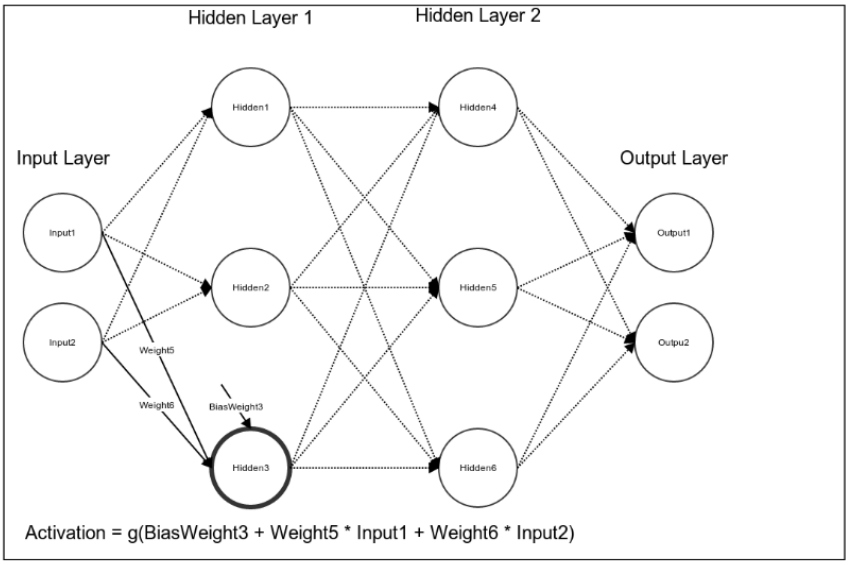
During the forward propagation stage, the features are input to the network and fed through the subsequent layers to produce the output activations. First, we compute the activation for the unit Hidden1. We find the weighted sum of input to Hidden1, and then process the sum with the activation function. Note that Hidden1 receives a constant input from a bias unit that is not depicted in the diagram in addition to the inputs from the input units. In the following diagram, g(x) is the activation function:



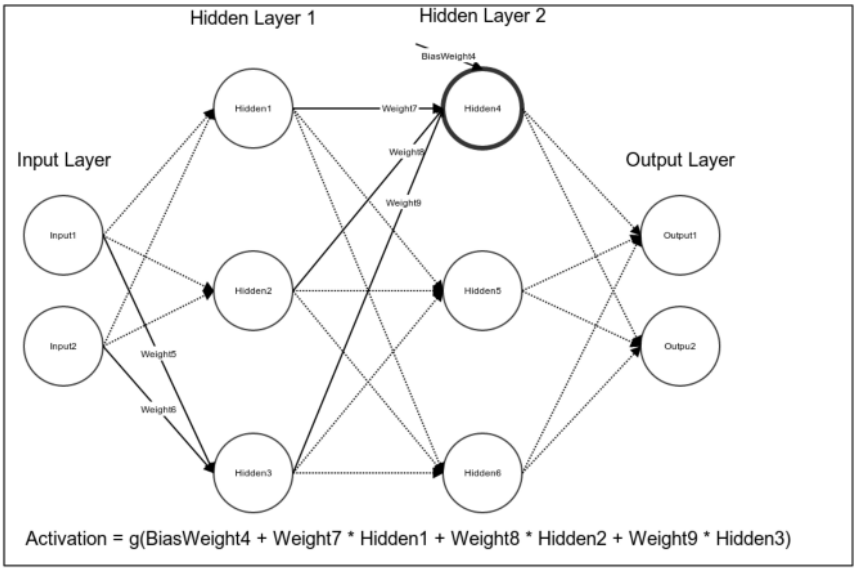
Next, we compute the activation for the second hidden unit. Like the first hidden unit, it receives weighted inputs from both of the input units and a constant input from a bias unit. We then process the weighted sum of the inputs, or preactivation, with the activation function as shown in the following figure:



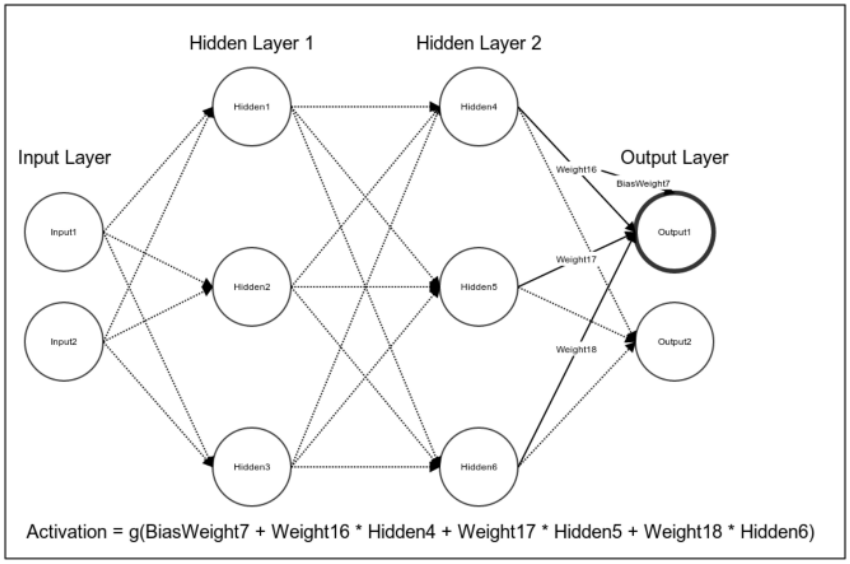
We then compute the activation for Hidden3 in the same manner:



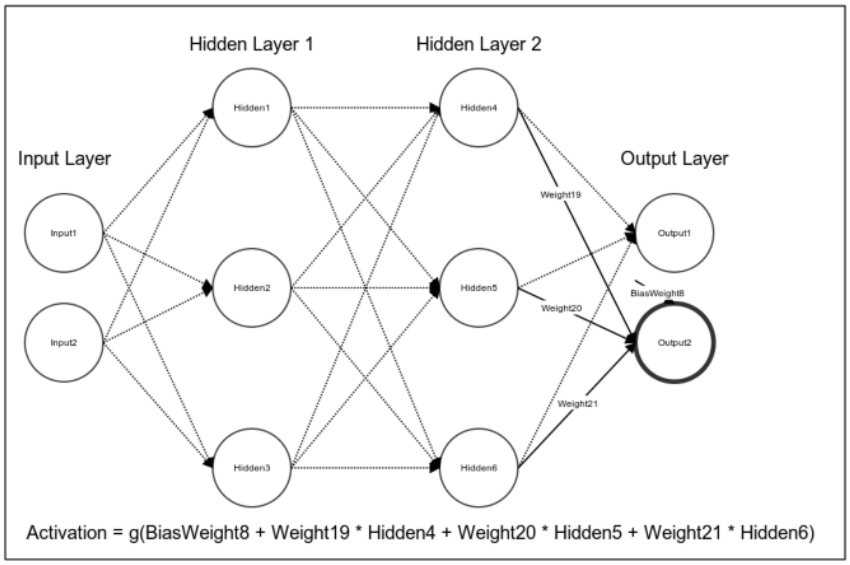
Having computed the activations of all of the hidden units in the first layer, we proceed to the second hidden layer. In this network, the first hidden layer is fully connected to the second hidden layer. Similar to the units in the first hidden layer, the units in the second hidden layer receive a constant input from bias units that are not depicted in the diagram. We proceed to compute the activation of Hidden4:



We next compute the activations of Hidden5 and Hidden6. Having computed the activations of all of the hidden units in the second hidden layer, we proceed to the output layer in the following figure. The activation of Output1 is the weighted sum of the second hidden layer's activations processed through an activation function. Similar to the hidden units, the output units both receive a constant input from a bias unit:



We calculate the activation of Output2 in the same manner:



We computed the activations of all of the units in the network, and we have now completed forward propagation. The network is not likely to approximate the true function well using the initial random values of the weights. We must now update the values of the weights so that the network can better approximate our function. We will do so using the algorithm of backpropagation.

4.5.2 Gradient Descent

In this section, we will discuss a method to efficiently estimate the optimal values of the model's parameters called gradient descent. Note that our definition of a good fit has not changed; we will still use gradient descent to estimate the values of the model's parameters that minimize the value of the cost function.

Gradient descent is sometimes described by the analogy of a blindfolded man who is trying to find his way from somewhere on a mountainside to the lowest point of the valley. He cannot see the topography, so he takes a step in the direction with the steepest decline. He then takes another step, again in the direction with the steepest decline. The sizes of his steps are proportional to the steepness of the terrain at his current position. He takes big steps when the terrain is steep, as he is confident that he is still near the peak and that he will not overshoot the valley's lowest point. The man takes smaller steps as the terrain becomes less steep. If he were to continue taking large steps, he may accidentally step over the valley's lowest point. He would then need to change direction and step toward the lowest point of the valley again. By taking decreasingly large steps, he can avoid stepping back and forth over the valley's lowest point. The blindfolded man continues to walk until he cannot take a step that will decrease his altitude; at this point, he has found the bottom of the valley.

Formally, gradient descent is an optimization algorithm that can be used to estimate the local minimum of a function. We can use gradient descent to find the values of the model's parameters that minimize the value of the cost function. Gradient descent iteratively updates the values of the model's parameters by calculating the partial derivative of the cost function at each step. It is important to note that gradient descent estimates the *local* minimum of a function. A three-dimensional plot of the values of a convex cost function for all possible values of the parameters looks like a bowl. The bottom of the bowl is the sole local minimum. Non-convex cost functions can have many local minima, that is, the plots of the values of their cost functions can have many peaks and valleys. Gradient descent is only guaranteed to find the local minimum; it will find a valley, but will not necessarily find the lowest valley. Fortunately, the residual sum of the squares cost function is convex.

An important hyperparameter of gradient descent is the *learning rate*, which controls the size of the blindfolded man's steps. If the learning rate is small enough, the cost function will decrease with each iteration until gradient descent has converged on the optimal parameters. As the learning rate decreases, however, the time required for gradient descent to converge increases; the blindfolded man will take longer to reach the valley if he takes small steps than if he takes large steps. If the learning rate is too large, the man may repeatedly overstep the bottom of the valley, that is, gradient descent could oscillate around the optimal values of the parameters.

There are two varieties of gradient descent that are distinguished by the number of training instances that are used to update the model parameters in each training iteration. The gradients of the parameters can be averaged over all available data. This

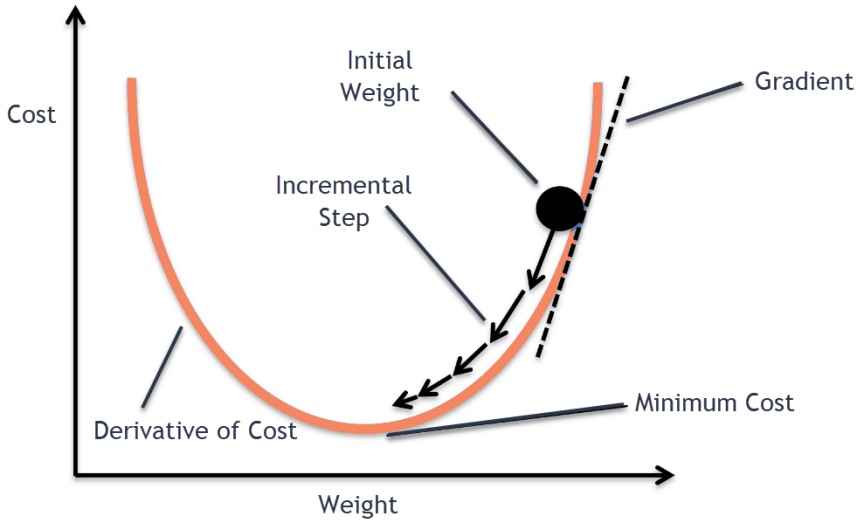


Figure 53: Finding the local optimum of an one-dimensional convex function using GD

algorithm, known as batch gradient descent, is not efficient, especially if learning on a large dataset. Stochastic gradient descent (SGD) is a drastic simplification which computes the gradient over an extremely small subset (mini-batch) of the whole dataset. In the simplest case, corresponding to maximum stochasticity, one data sample is selected at random in each optimization step.

Let w be the attened vector of all parameters in a neural network, composed of W_k for every k. Let E be the error function, i.e., the difference between the true value of the objective function and the computed output of the network. The back-propagation algorithm computes the partial derivative of E with respect to each parameter in w and updates the parameter so as to reduce its gradient. The update rule of stochastic gradient descent for a parameter w_j is:

$$w_j := w_j - \alpha \frac{\partial E_i}{\partial w_j}$$

where α is the learning rate and E_i is computed over the mini-batch i. We refer to one full iteration over all available input data as an epoch. Note that each parameter in vector w is updated independently from other parameters. Some techniques set the learning rate adaptively, but still preserve this independence.

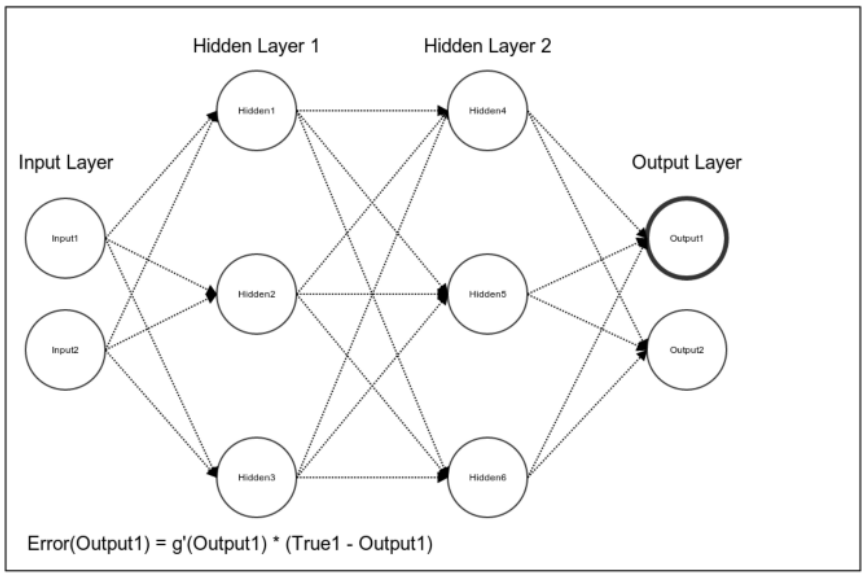
4.5.3 Backpropagation

The backpropagation algorithm is commonly used in conjunction with an optimization algorithm such as gradient descent to minimize the value of the cost function. The algorithm takes its name from a portmanteau of backward propagation, and refers to

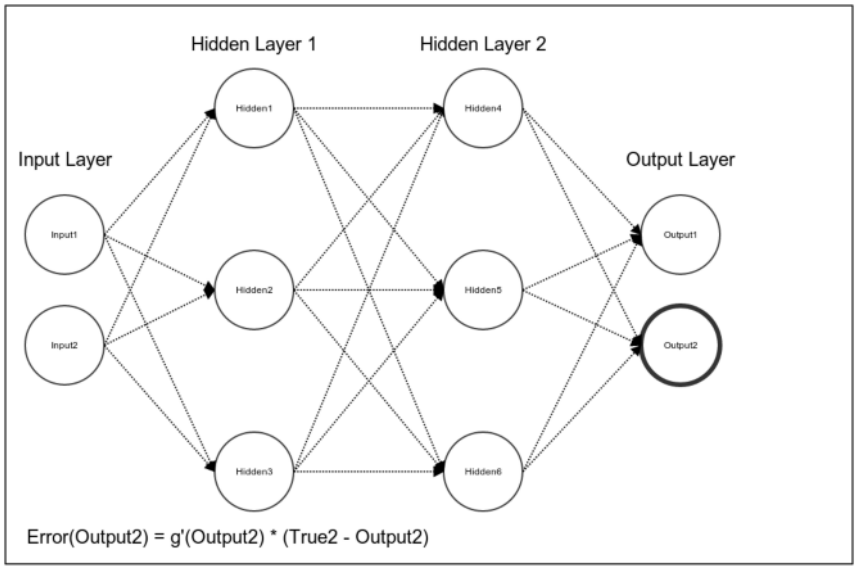
the direction in which errors flow through the layers of the network. Backpropagation can theoretically be used to train a feed-forward network with any number of hidden units arranged in any number of layers, though computational power constrains this capability.

Backpropagation is similar to gradient descent in that it uses the gradient of the cost function to update the values of the model parameters. Unlike the linear models we have previously seen, neural nets contain hidden units that represent latent variables; we can't tell what the hidden units should do from the training data. If we do not know what the hidden units should do, we cannot calculate their errors and we cannot calculate the gradient of cost function with respect to their weights. A naive solution to overcome this is to randomly perturb the weights for the hidden units. If a random change to one of the weights decreases the value of the cost function, we save the change and randomly change the value of another weight. An obvious problem with this solution is its prohibitive computational cost. Backpropagation provides a more efficient solution.

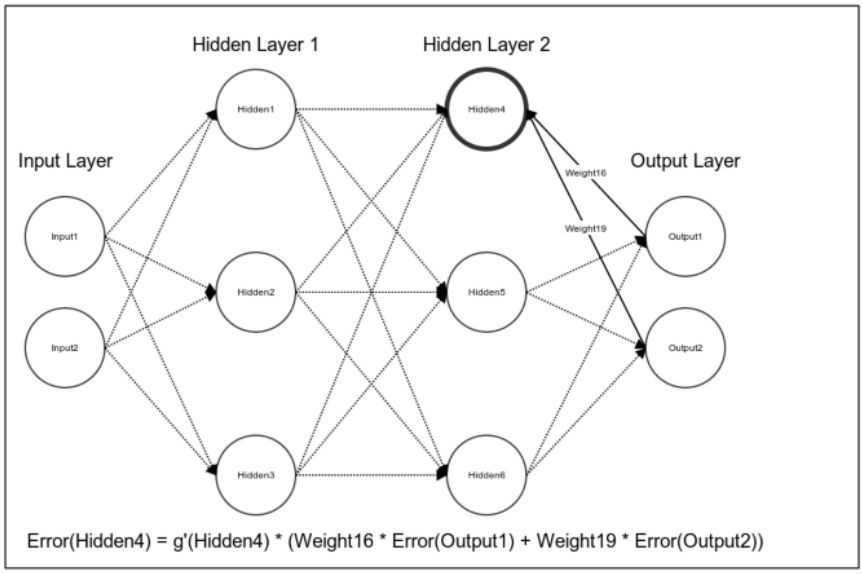
We can calculate the error of the network only at the output units. The hidden units represent latent variables; we cannot observe their true values in the training data and thus, we have nothing to compute their error against. In order to update their weights, we must propagate the network's errors backwards through its layers. We will begin with Output1. Its error is equal to the difference between the true and predicted outputs, multiplied by the partial derivative of the unit's activation:



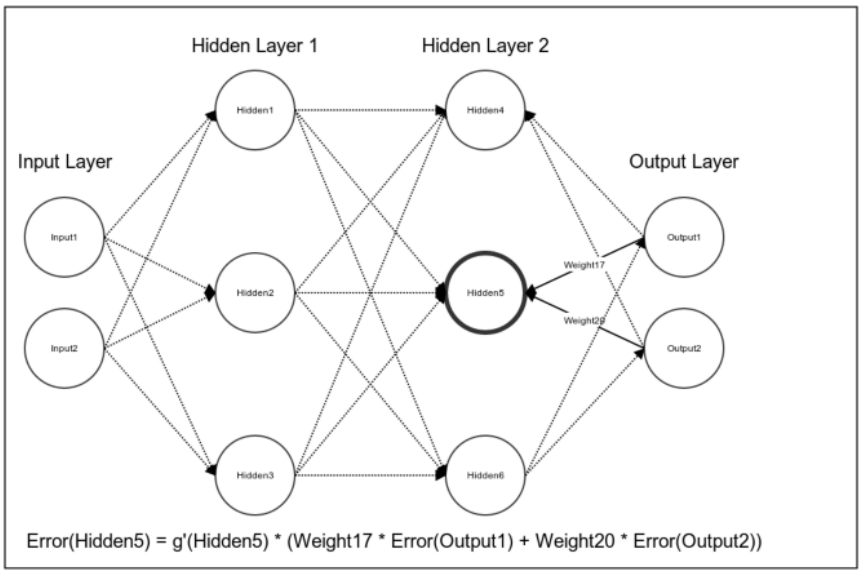
We then calculate the error of the second output unit:



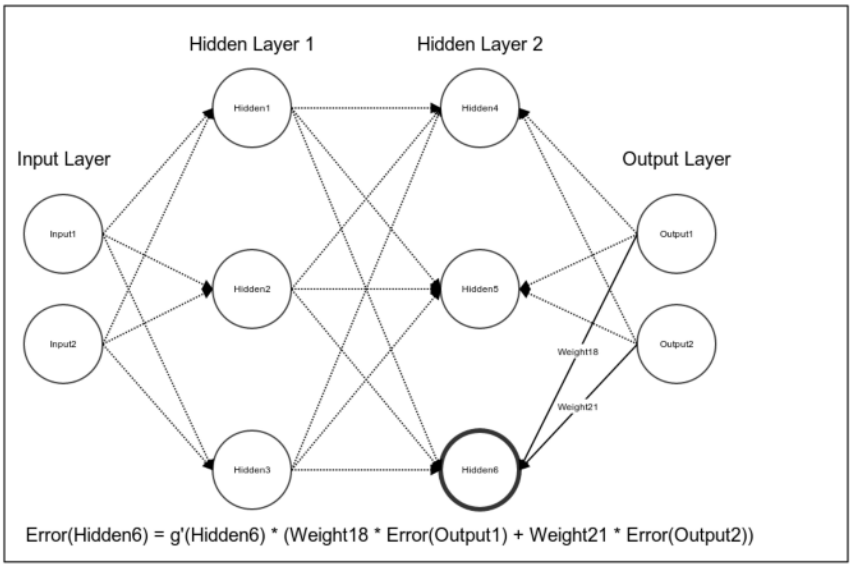
We computed the errors of the output layer. We can now propagate these errors backwards to the second hidden layer. First, we will compute the error of hidden unit Hidden4. We multiply the error of Output1 by the value of the weight connecting Hidden4 and Output1. We similarly weigh the error of Output2. We then add these errors and calculate the product of their sum and the partial derivative of Hidden4:



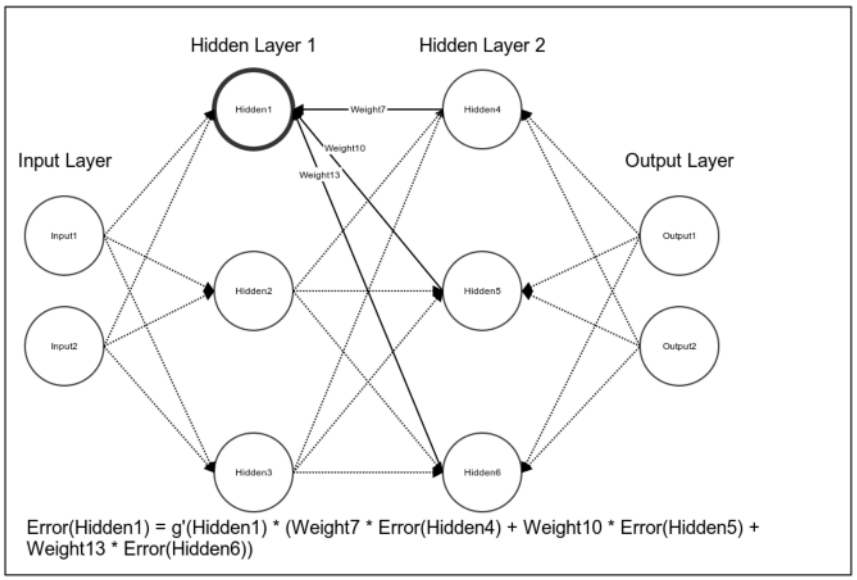
We similarly compute the errors of Hidden5:



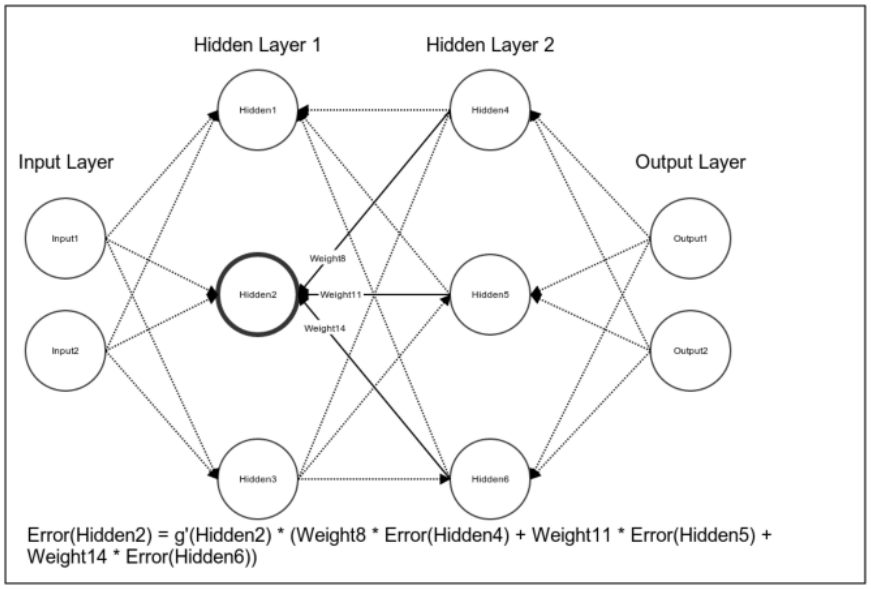
We then compute the Hidden6 error in the following figure:



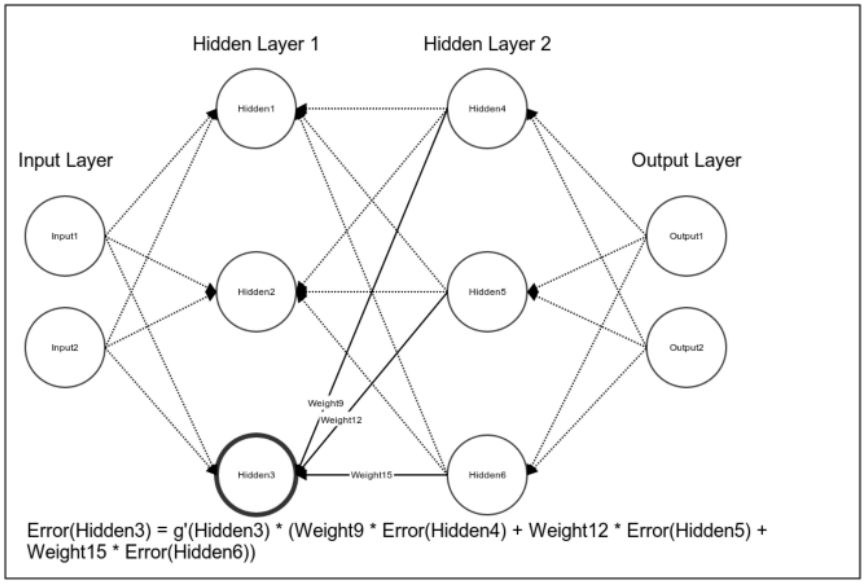
We calculated the error of the second hidden layer with respect to the output layer. Next, we will continue to propagate the errors backwards towards the input layer. The error of the hidden unit Hidden1 is the product of its partial derivative and the weighted sums of the errors in the second hidden layer:



We similarly compute the error for hidden unit Hidden2:

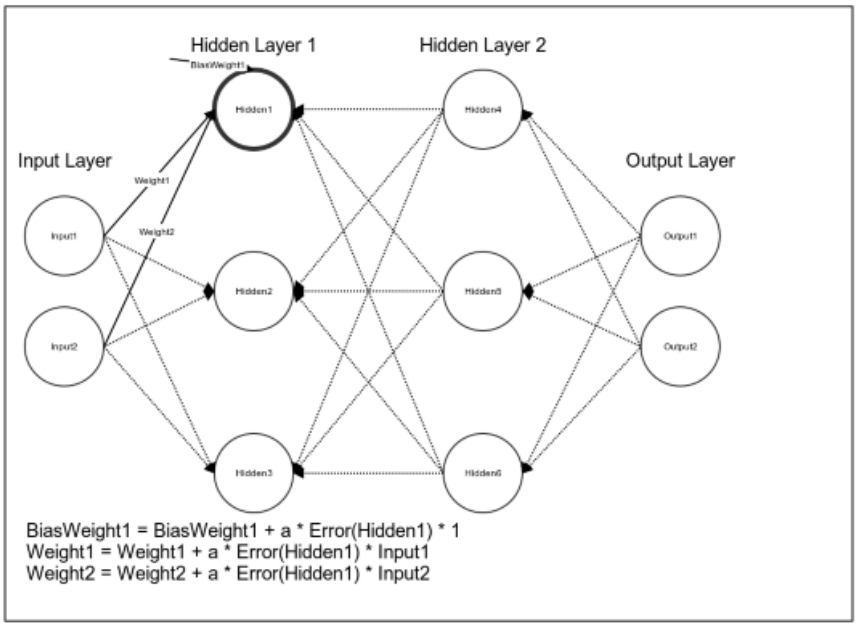


We similarly compute the error for Hidden3:

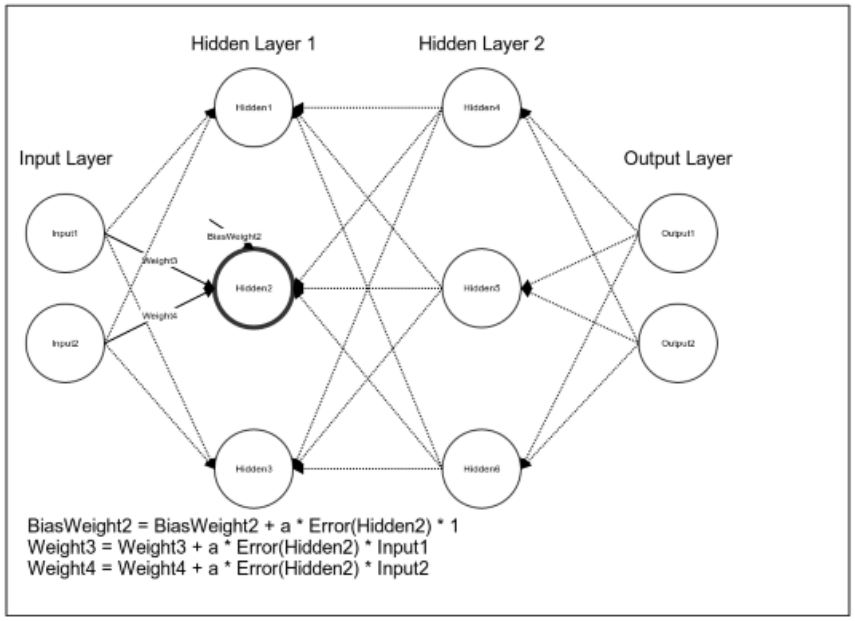


We computed the errors of the first hidden layer. We can now use these errors to update the values of the weights. We will first update the weights for the edges connecting the input units to Hidden1 as well as the weight for the edge connecting the bias unit to Hidden1. We will increment the value of the weight connecting Input1 and Hidden1 by the product of the learning rate, error of Hidden1, and the value of Input1.

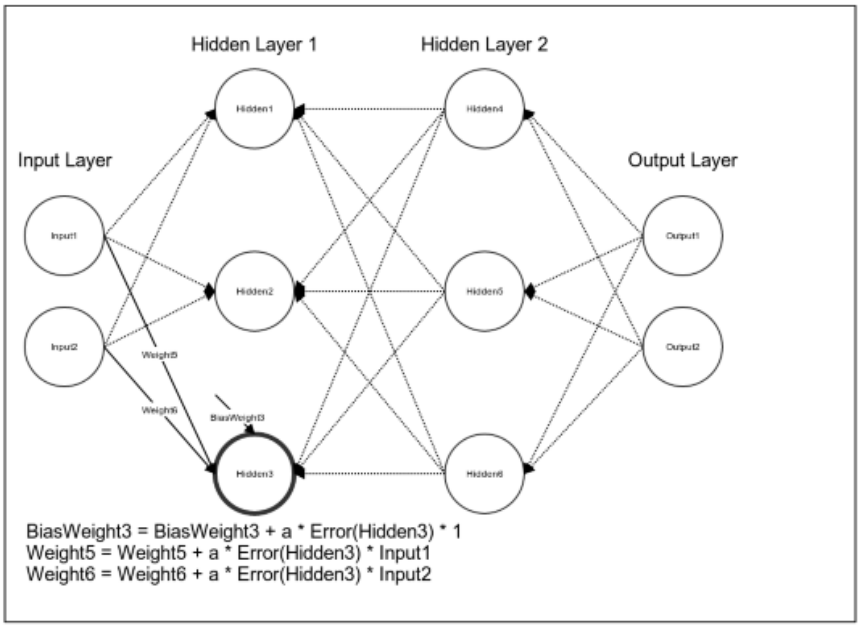
We will similarly increment the value of Weight2 by the product of the learning rate, error of Hidden1, and the value of Input2. Finally, we will increment the value of the weight connecting the bias unit to Hidden1 by the product of the learning rate, error of Hidden1, and one.



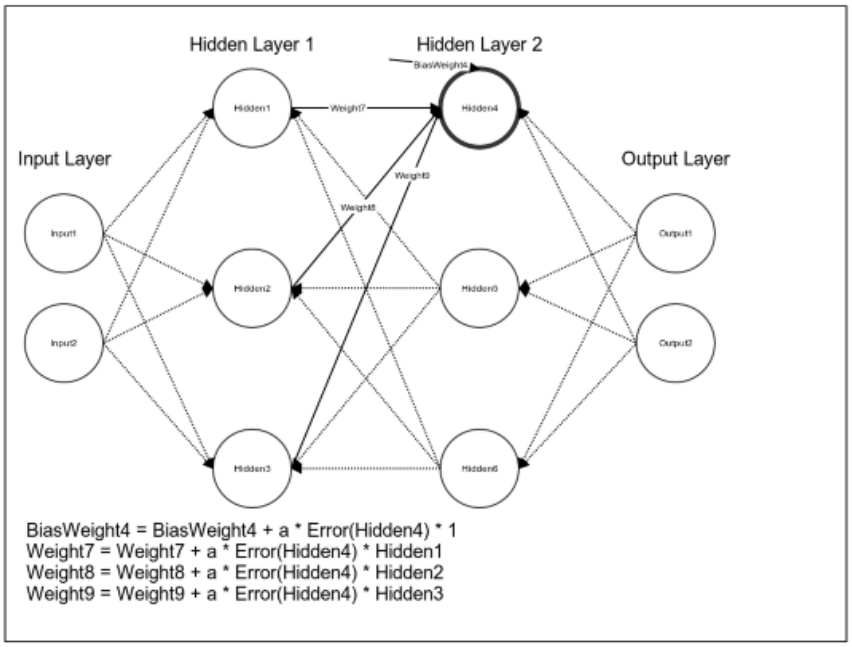
We will then update the values of the weights connecting hidden unit Hidden2 to the input units and the bias unit using the same method:



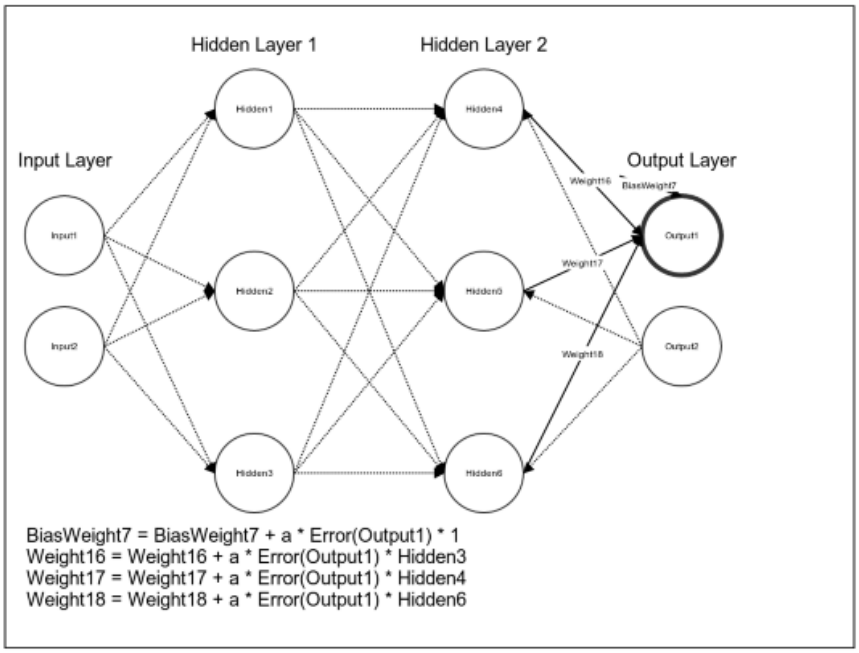
Next, we will update the values of the weights connecting the input layer to Hidden3:

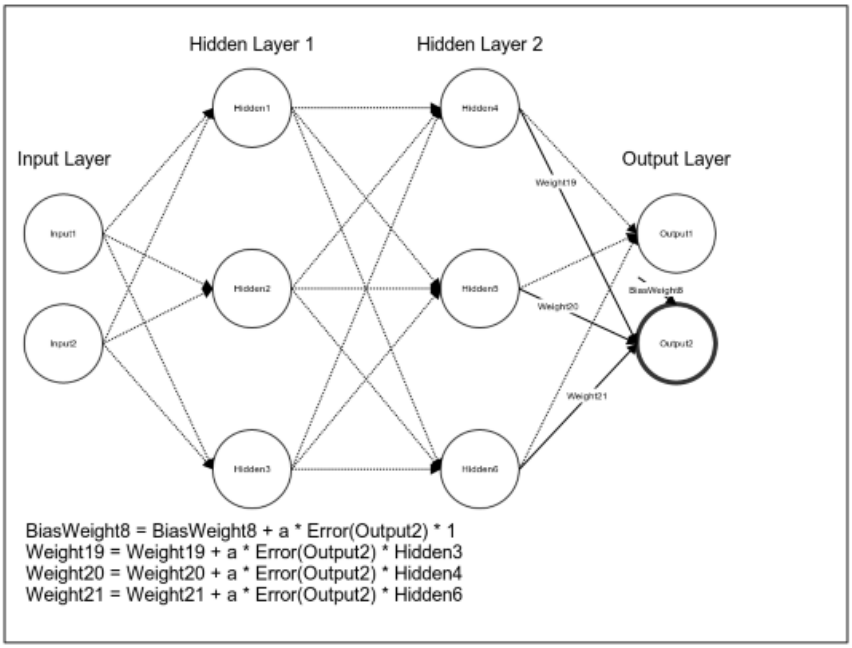


Since the values of the weights connecting the input layer to the first hidden layer is updated, we can continue to the weights connecting the first hidden layer to the second hidden layer. We will increment the value of Weight7 by the product of the learning rate, error of Hidden4, and the output of Hidden1. We continue to similarly update the values of weights Weight8 to Weight15:



The weights for Hidden5 and Hidden6 are updated in the same way. We updated the values of the weights connecting the two hidden layers. We can now update the values of the weights connecting the second hidden layer and the output layer. We increment the values of weights W16 through W21 using the same method that we used for the weights in the previous layers:





After incrementing the value of Weight21 by the product of the learning rate, error of Output2, and the activation of Hidden6, we have finished updating the values of the weights for the network. We can now perform another forward pass using the new values of the weights; the value of the cost function produced using the updated weights should be smaller. We will repeat this process until the model converges or another stopping criterion is satisfied. Unlike the linear models we have discussed, backpropagation does not optimize a convex function. It is possible that backpropagation will converge on parameter values that specify a local, rather than global, minimum. In practice, local optima are frequently adequate for many applications.

4.6 Generative Adversarial Networks

4.6.1 Introduction

In recent years, more and more data in different application domains are becoming readily available for the rapid development of both computer hardware and software technologies. Many data mining methodologies have been developed for analyzing those big data sets. One representative example is deep learning, which typically needs a huge amount of training samples to achieve promising performance. However, there exists domains where it is impossible to get as much data as we want. Medicine and Health Informatics are such fields. On individual patient level analysis, each patient is treated as a sample in model training process. However, considering the complexity of many diseases, the number of all patients from the whole world is still very small and far from enough. Moreover, we can never get the medical data from all patients for privacy and sensitivity reasons. Further, the expensive and time-consuming data collection process also limits the amount of data. Thus, the problem of building high-quality medical

analytics models remains very challenging at present.

Generative models have provided us a promising direction to alleviate the data scarcity issue. By sketching the data distribution from a small set of training data, we are able to sample from the distribution and generate much more samples for our study. By combining the complexity of deep neural networks and game theory, the Generative Adversarial Network (GAN) and its variants have demonstrated impressive performance in modeling the underlying data distribution, generating high quality "fake" samples that are hard to be differentiated from real ones. Ideally, with the high quality generative distribution in hand, we can protect the privacy of raw data by releasing only the distribution instead of the raw data to the public or constrained individuals, and can even sample datasets to fit our needs and conduct further analysis.

Generative Adversarial Networks (GANs) have the potential to build next-generation models, as they can mimic any distribution of data. Major research and development work is being undertaken in this field because it is one of the most rapidly growing areas of machine learning (ML).

4.6.2 Definition

GANs are a type of generative model in which two neural networks, commonly known as the Generator (G_y) and Discriminator (D_w), are trained against each other in a zero-sum game [1]. These neural networks are parameterized by their edge weights —y and w for G_y and D_w, respectively—which specify the function computed by each network.

The Generator takes as input a random vector drawn from a known distribution, and produces a new data point that (hopefully) has a similar distribution to the true data distribution. If we are given a finite-size database, then the true data distribution can be interpreted as the empirical distribution that would arise from sampling entries of the database with replacement. The Discriminator then tries to detect whether this new data point is from the Generator or from the true data distribution. If the Discriminator is too successful in distinguishing between the Generator's outputs and the true data, then this feedback is used to improve the Generator's data generation process.

We want to train D_w to maximize the probability of assigning right labels, whereas

[1] In game theory and economic theory, a zero-sum game is a mathematical representation of a situation in which each participant's gain or loss of utility is exactly balanced by the losses or gains of the utility of the other participants. If the total gains of the participants are added up and the total losses are subtracted, they will sum to zero. Thus, cutting a cake, where taking a larger piece reduces the amount of cake available for others, is a zero-sum game if all participants value each unit of cake equally.

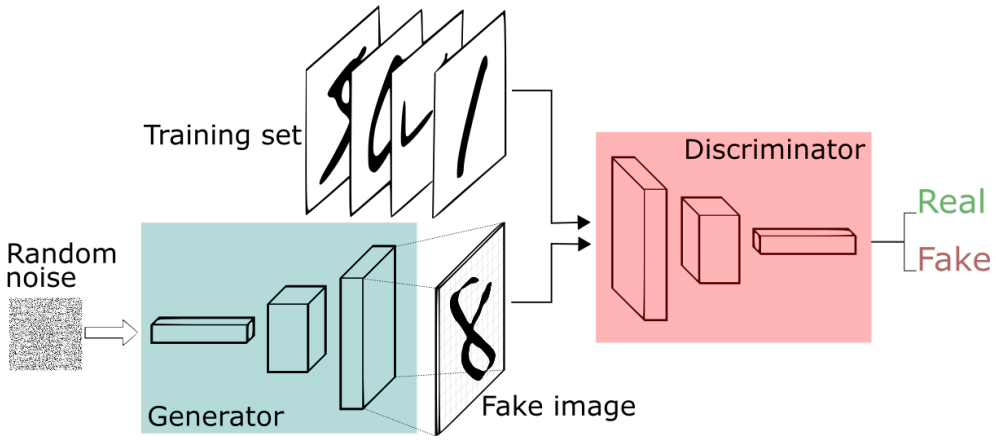


Figure 54: A block diagram of a GAN

G_y should minimize the difference between its output distribution and true data distribution. The value of this two player zero-sum game between G_y and D_w can be written as following min-max optimization problem:

$$\min_G \max_D L(D,G) = \underbrace{E_{x\sim p_r(x)}[\log \mathrm{D}(x)]}_{\text{Recognize real images better}} + \underbrace{E_{z\sim p_z(z)}[\log(1\text{-}\mathrm{D}(\mathrm{G}(z)))]}_{\text{Recognize generated (fake) images better}}$$

where p_{data} is true data distribution and p_z is a known noise distribution. In the min-max form of the game, D_w chooses w to maximize O(y,w) and G_y chooses y to minimize O(y,w). Their equilibrium strategies will achieve objective value min $max_y max_w$ O(y,w). However, since O (y,w) is a non-convex non-concave objective, these optimal strategies are typically not efficiently computable. Instead, we use gradient descent/ascent schemes to allow D_w and G_y to iteratively learn their optimal strategies.

We estimate the function and its gradients by sampling random elements from p_{data} and p_z. Let $[z_1, ..., z_m]$ and $[x_1, ..., x_m]$ be random samples from p_z and from p_{data} respectively. We write O_i(y,w) := log($D_w(x_i)$) + log(1 - $D_w(G_y(z_i))$) as i-th sampled function, and take the average value over the m samples to get estimate of O:

$$O_{\text{sample}}(y,w) = \frac{1}{m}\sum_{i=1}^{m} O_i(y,w)$$

Next we take the gradient with respect y and w: $g_y := \nabla_y O_{sample}(y,w)$ and $g_w := \nabla_w O_{sample}(y,w)$. Since the input data were randomly sampled, these are stochastic gradients. Finally, we do the gradient update step, with gradient ascent for D, $w \leftarrow w + n_w * g_w$ and gradient descent for G, $y \leftarrow y + n_y * g_y$ for step sizes n_w and n_y. This update process repeats either until the parameters converge or until a pre-specified number of update steps have occurred.

4.6.3 Nash Equilibrium

The Nash equilibrium describes a particular state in game theory. This state can be achieved in a non-cooperative game in which each player tries to pick the best possible strategy to gain the best possible outcome for themselves, based on what they expect the other players to do. Eventually, all the players reach a point at which they have all picked the best possible strategy for themselves based on the decisions made by the other players. At this point in the game, they would gain no benefit from changing their strategy. This state is the Nash equilibrium.

A famous example of how the Nash equilibrium can be reached is with the Prisoner's Dilemma. In this example, two criminals (A and B) have been arrested for committing a crime. Both have been placed in separate cells with no way of communicating with each other. The prosecutor only has enough evidence to convict them for a smaller offense and not the principal crime, which would see them go to jail for a long time. To get a conviction, the prosecutor gives them an offer:

- If A and B both implicate each other in the principal crime, they both serve 2 years in jail.
- If A implicates B but B remains silent, A will be set free and B will serve 3 years in jail (and vice versa).
- If A and B both keep quiet, they both serve only 1 year in jail on the lesser charge.

From these three scenarios, it is obvious that the best possible outcome for A and B is to keep quiet and serve 1 year in jail. However, the risk of keeping quiet is 3 years as neither A nor B have any way of knowing that the other will also keep quiet. Thus, they would reach a state where their actual optimum strategy would be to confess as it is the choice that provides the highest reward and lowest penalty. When this state has been reached, neither criminal would gain any advantage by changing their strategy; thus, they would have reached a Nash equilibrium.

4.6.4 Practical Applications

GANs have some fairly useful practical applications, which include the following:

- **Image generation**: Generative networks can be used to generate realistic images after being trained on sample images. For example, if we want to generate new images of dogs, we can train a GAN on thousands of samples of images of dogs. Once the training has finished, the generator network will be able to generate new

images that are different from the images in the training set. Image generation is used in marketing, logo generation, entertainment, social media, and so on.

- **Text-to-image synthesis**: Generating images from text descriptions is an interesting use case of GANs. This can be helpful in the film industry, as a GAN is capable of generating new data based on some text that you have made up. In the comic industry, it is possible to automatically generate sequences of a story.

- **Face aging**: This can be very useful for both the entertainment and surveillance industries. It is particularly useful for face verification because it means that a company doesn't need to change their security systems as people get older. An age-cGAN network can generate images at different ages, which can then be used to train a robust model for face verification.

- **Image-to-image translation**: Image-to-image translation can be used to convert images taken in the day to images taken at night, to convert sketches to paintings, to style images to look like Picasso or Van Gogh paintings, to convert aerial images to satellite images automatically, and to convert images of horses to images of zebras. These use cases are ground-breaking because they can save us time.

- **Video synthesis**: GANs can also be used to generate videos. They can generate content in less time than if we were to create content manually. They can enhance the productivity of movie creators and also empower hobbyists who want to make creative videos in their free time.

- **High-resolution image generation**: If you have pictures taken from a low-resolution camera, GANs can help you generate high-resolution images without losing any essential details. This can be useful on websites.

- **Completing missing parts of images**: If you have an image that has some missing parts, GANs can help you to recover these sections.

4.6.5 Challenges of GAN models

- **Setting up failure and bad initialization**: the generator and discriminator reach a state where they cannot improve any further given the other is kept unchanged. Now the setup of gradient descent is to take a step in a direction that reduces the loss measure defined on the problem—but we are by no means enforcing the networks to reach Nash equilibrium in GAN, which have non-convex objective with continuous high dimensional parameters. The networks try to take successive steps to minimize a non-convex objective and end up in an oscillating

Figure 55: Face generation using generative modeling has improved significantly in the last four years (15, 5).

process rather than decreasing the underlying true objective. In most cases, when your discriminator attains a loss very close to zero, then right away you can figure out something is wrong with your model. But the biggest difficulty is figuring out what is wrong. Another practical thing done during the training of GAN is to purposefully make one of the networks stall or learn slower, so that the other network can catch up. And in most scenarios, it's the generator that lags behind so we usually let the discriminator wait. This might be fine to some extent, but remember that for the generator to get better, it requires a good discriminator and vice versa. Ideally the system would want both the networks to learn at a rate where both get better over time. The ideal minimum loss for the discriminator is close to 0.5— this is where the generated images are indistinguishable from the real images from the perspective of the discriminator (16, 6).

- **Mode collapse**: One of the main failure modes with training a generative adversarial network is called mode collapse or sometimes the helvetica scenario. The basic idea is that the generator can accidentally start to produce several copies of exactly the same image, so the reason is related to the game theory setup. We can think of the way that we train generative adversarial networks as first maximizing with respect to the discriminator and then minimizing with respect to the generator. If we fully maximize with respect to the discriminator before we start to minimize with respect to the generator, everything works out just fine. But if we go the other way around and we minimize with respect to the generator and then maximize with respect to the discriminator, everything will actually break and the reason is that if we hold the discriminator constant, it will describe a single region in space as being the point that is most likely to be real rather than fake and then the generator will choose to map all noise input values to that same most likely to be real point.

- **Problems with counting**: GANs can sometimes be far-sighted and fail to differentiate the number of particular objects that should occur at a location.

- **Problems with perspective**: GANs sometimes are not capable of differentiating between front and back view and hence fail to adapt well with 3D objects while generating 2D representations from it.
- **Problems with global structures**: GANs do not understand holistic structures, similar to problems with perspective. For example, in the bottom left image, it generates an image of a quadruple cow, that is, a cow standing on its hind legs and simultaneously on all four legs. That is definitely unrealistic and not possible in real life!

5 Implementing DP with GANs

5.1 Introduction

While the utility of deep learning is undeniable, the same training data that has made it so successful also presents serious privacy issues. building on large amounts of contextually rich information. Centralized collection of photos, speech, and video from millions of individuals is filled with privacy risks. The most important and commonplace are:

1. Users' data kept by companies is subject to subpoenas and warrants, as well as spying by national-security and intelligence outfits. Furthermore, the Internet giants' monopoly on "big data" collected from millions of users leads to their monopoly on the AI models learned from this data.

2. Images and voice recordings often contain accidentally captured sensitive items, license plates, computer screens, the sound of other people speaking and ambient noise , etc.

3. Companies gathering this data keep it *forever*; users from whom the data was collected can neither delete it, nor control how it will be used, nor influence what will be learned from it.

4. Users may benefit from new services, such as powerful image search, voice-activated personal assistants, and machine translation of websites in foreign languages, but the underlying models constructed from their collective data remain proprietary to the companies that created them.

5. In many domains, most notably those related to medicine, the sharing of data about individuals is not permitted by law or regulation. Consequently, biomedical and clinical researchers can only perform deep learning on the datasets belonging to their own institutions. It is well-known that neural-network models become better as the training datasets grow bigger and more diverse. Due to not being able to use the data from other institutions when training their models, researchers may end up with worse models.

The GANs were introduced as a solution to these problems. However, even the GAN itself can implicitly disclose privacy information of the training samples. The adversarial training procedure and the high model complexity of deep neural networks, jointly encourage a distribution that is concentrated around training samples. By repeated sampling from the distribution, there is a considerable chance of recovering the training samples. Due to possible privacy violations of the individuals whose data is used to train these models, publishing or sharing generative models is not always viable.

5.2 Differentially Private Synthetic Data Generation via GANs

For the remainder of this study, we will explain two algorithms that are utilized in order to solve the privacy problems of the GAN and we will perform some experiments. Unlike our experiments with the Bayesian Networks, the algorithms that we will utilize in this chapter employ the centralized model.

The first algorithm suggested by Liyang Xie et al. (4,) implements a Differentially Private Generative Adversarial Network (DPGAN), which provides proven privacy control for the training data using the concept of Differential Privacy. The difference from the traditional GAN is that the proposed model applies a combination of carefully designed noise and gradient clipping and uses the Wasserstein distance (31, 1) as an approximation of the distance between probability distributions.

The second algorithm suggested by (32, 2) generates differentially private data by adding a layer that adds appropriate Gaussian Noise to the outputs of one of the convolutional layers of the Discriminator.

5.3 Experimental Evaluation

To implement the aforementioned algorithms, we used the Tensorflow (Python) package to build a DCGAN (Deep Convolutional Generative Adversarial Network). The DCGAN that we utilized has the following structure:

- **Discriminator:**

    1. A 2D Convolutional layer with a kernel of size 5 x 5, strides 2 x 2 and 64 filters. This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. The reason we use a Convolutional Neural Network is because it is able to successfully capture the spatial and temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. Another role of the Convolutional Neural Network is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.
    2. A leaky ReLU layer. This layer performs a threshold operation, where any input value less than zero is multiplied by a fixed scalar. Values greater than zero are unaffected.

3. A Dropout layer with rate = 0.3. Dropout randomly sets a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting. All Dropout layers are disabled when we use the second algorithm.
4. A 2D Convolutional layer with a kernel of size 5 x 5, strides 2 x 2 and 128 filters.
5. A leaky ReLU layer.
6. A Gaussian noise layer that simply adds noise to its inputs, the outputs of the previous layer. This layer is disable when we use the first algorithm.
7. A Dropout layer with rate = 0.3.
8. A Flatten layer. This layer converts the 2-dimensional image into an one-dimensional vector. This process is known as vectorization.
9. A Dense layer with 1 unit. A Dense layer is regular densely-connected NN layer.

- **Generator:**

1. A Dense layer with 12544 units.
2. A BatchNormalization layer. This layer normalizes the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. It is a technique for improving the speed, performance and stability of artificial neural networks.
3. A leaky ReLU layer.
4. A Reshape layer. It reshapes an output to a certain shape.
5. A Conv2DTranspose layer with a kernel of size 5 x 5, strides 1 x 1 and 128 filters. A transposed convolutional layer carries out a regular convolution but reverts its spatial transformation. It is followed by a BatchNormalization and a leaky ReLU layer.
6. A Conv2DTranspose layer with a kernel of size 5 x 5, strides 2 x 2 and 64 filters. It is followed by a BatchNormalization and a leaky ReLU layer.
7. A Conv2DTranspose layer with a kernel of size 5 x 5, strides 2 x 2 and 1 filter.

For the first algorithm, we added Gaussian noise to the discriminator loss. For the second algorithm, we placed a Gaussian noise layer after the second convolutional layer. Then we tested the two algorithms for various values of ϵ on the MNIST database. The

MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The MNIST database contains 60000 training images. Our purpose is that the GAN learns to generate its own handwritten digits after training with the original dataset. Then we will observe how the ϵ value that we choose affects the quality of the generated images. Since a smaller ϵ means a greater amount of noise, we expect the images to get more blurred as we decrease the ϵ value.

For the first algorithm, we have:

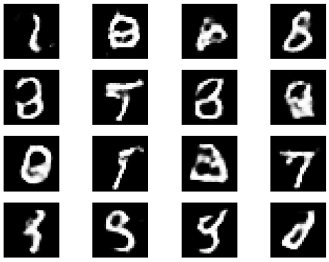


Figure 56: Generated digits with $\epsilon = 0.5$

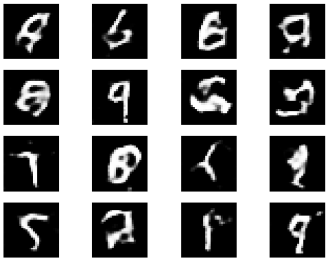


Figure 57: Generated digits with $\epsilon = 1$

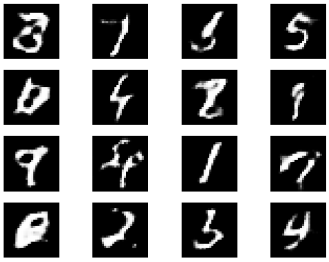


Figure 58: Generated digits with $\epsilon = 5$

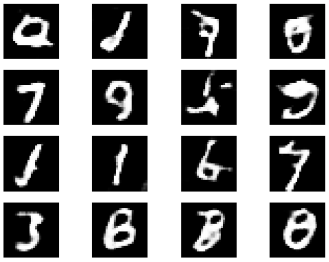


Figure 59: Generated digits with $\epsilon = 10$

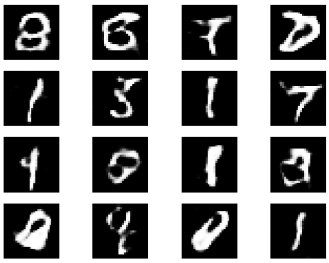


Figure 60: Generated digits with $\epsilon = 20$

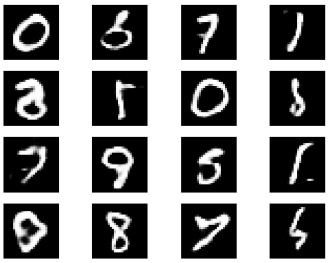


Figure 61: The generated digits without any noise

For the second algorithm, we have:

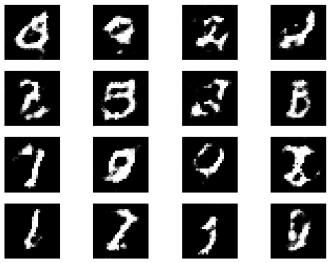


Figure 62: Generated digits with $\epsilon = 0.5$

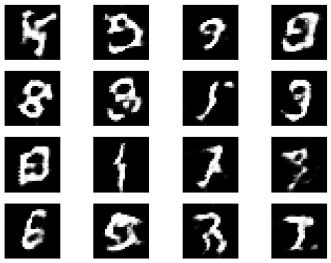


Figure 63: Generated digits with $\epsilon = 1$

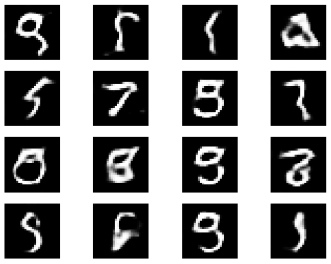


Figure 64: Generated digits with $\epsilon = 5$

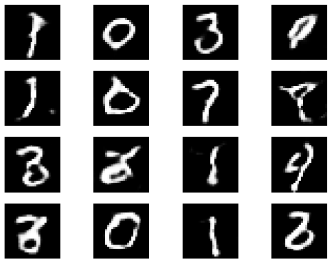


Figure 65: Generated digits with $\epsilon = 10$

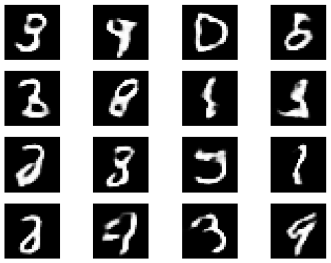


Figure 66: Generated digits with $\epsilon = 20$

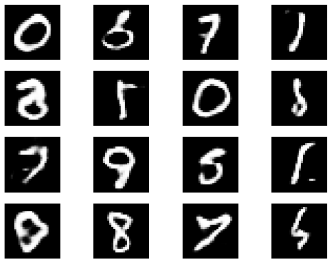


Figure 67: The generated digits without any noise

We notice that with both algorithms, the image quality deteriorates as we decrease the ϵ value and some of the generated digits are unrecognizable even to the human eye.

Next, we will attempt to judge the utility of the generated images using the classification accuracy as a metric. We have generated 60000 images for each algorithm and ϵ value. Now we will use a separate convolutional neural network to label them. This network is known as a *student* in the Machine Learning literature. Real data will

be used as the training set and the generated data will be used as the testing set. Afterwards, we will train the same network using both real and generated data. In both cases, we will use a testing set from the MNIST database. Finally we will compare the classification accuracy that is produced when we train the network using real data, with the accuracy produced when we use generated data. The student network that we utilized has the following structure:

1. A 2D Convolutional layer with a kernel of size 3 x 3.
2. A 2D Max Pooling layer with a pool of size 2 x 2. Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the convolved feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. Max Pooling returns the maximum value from the portion of the image covered by the Kernel.
3. A Flatten layer.
4. A Dense layer with 128 units and ReLU as an activation function.
5. A Dropout layer with rate = 0.2.
6. A Dense layer with 10 units (since the MNIST database has 10 classes) and SoftMax as an activation function.

We have trained and tested the network for each algorithm and ϵ value and received the classification accuracy. Both algorithms perform excellently, since the classification accuracy is quite high even for small ϵ values. The first algorithm seems to perform slightly better. The results are presented in the following table:

Algorithm	ϵ	Classification Accuracy
Real Data	**-**	**98.58 %**
Generated Data	**-**	**96.67 %**
First Algorithm	20	96.32 %
First Algorithm	10	95.71 %
First Algorithm	1	90.35 %
First Algorithm	0.5	86.85 %
Second Algorithm	20	96.61 %
Second Algorithm	10	94.89 %
Second Algorithm	1	91.67 %
Second Algorithm	0.5	84.81 %

5.4 Conclusion & Future Work

In this and the previous chapter, we introduced Neural Networks and more specifically the Generative Adversarial Networks. We also discussed some methods on how to utilize them to generate differentially private synthetic data from real data. In contrast to when we used Bayesian Networks, we now used image data due to the fact that most of the literature concerning GANs does so. We implemented two of the suggested algorithms and executed some experiments.

Using GANs to generate differentially private synthetic data is a very new idea and it is still at its infancy. However the studies and the results that are published so far are quite promising. It is quite possible that, in the near future, GANs will be the state-of-the-art method to generate synthetic data and as such they are worthy of further study. To someone that is truly interested, we would suggest:

1. One could test the algorithms with many different datasets, especially datasets that are comprised of non-binary images.

2. The performance of the neural networks heavily depends on the choice of their hyperparameters such as learning rate, batch size, epochs e.t.c. We chose values based on the papers we studied, but it is quite possible that another set of values can be perform better. One could also experiment with the number and type of layers included in the neural networks.

3. Besides the two papers that we referenced, there are many others that show promise. One that is quite interesting is published by Digvijay Boob et al (13, 3). In this study, an algorithm is suggested that uses GANs on non-image data such as categorical and continuous data. However, the author has not provided an implementation or an evaluation for the algorithm, which makes it interesting for researching purposes.

4. It would be interesting to implement and experiment with algorithms that utilize GANs, but operate on *distributed data*, like we did in Chapter 3 using Bayesian Networks.

References

[1] The Algorithmic Foundations of Differential Privacy by Cynthia Dwork Microsoft Research, USA and Aaron Roth, University of Pennsylvania, USA

[2] Context-Aware Generative Adversarial Privacy by Chong Huang, Peter Kairouz, Lalitha Sankar and Ram Rajagopal

[3] On the Protection of Private Information in Machine Learning Systems: Two Recent Approaches Martın Abadi, Ulfar Erlingsson, Ian Goodfellow, H. Brendan McMahan,Ilya Mironov, Nicolas Papernot, Kunal Talwar, and Li Zhang – Google

[4] Differentially Private Generative Adversarial Network Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, Jiayu Zhou - Computer Science and Engineering, Michigan State University, Department of Computer Science, Rutgers University Department of Healthcare Policy and Research, Weill Cornell Medical School

[5] Privacy-Preserving Deep Learning - Reza Shokri,Vitaly Shmatikov - The University of Texas at Austin, Cornell Tech

[6] Differential Privacy: An Economic Method for Choosing Epsilon - Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C. Pierce, Aaron Roth

[7] Explaining Differential Privacy in 3 Levels of Difficulty by Nicolas Sartor - https://aircloak.com/explaining-differential-privacy/

[8] Wolfram Mathworld - http://mathworld.wolfram.com/

[9] GameTheory.net - http://www.gametheory.net/dictionary/Utility.html

[10] Pattern Recognition and Machine Learning - Chapter 5 - C.M.Bishop

[11] Mastering Machine Learning With Scikit-Learn - Chapter 1 and 10 - Gavin Hackeling

[12] Deep Learning with Differential Privacy - Martín Abadi, Andy Chu, Ian Goodfellow. H. Brendan McMahan, Ilya Mironov, Kunal Talwar, Li Zhang

[13] Differentially Private Synthetic Data Generation via GANs - Digvijay Boob, Rachel Cummings, Dhamma Kimpara, Uthaipon (Tao) Tantipongpipat, Chris Waites, Kyle Zimmerman,

[14] Generative Adversarial Networks Projects - Kailash Ahirwar

[15] Generative Deep Learning - David Foster

[16] Learning Generative Adversarial Networks - Kuntal Ganguly

[17] Introduction to Probability - Dimitri P. Bertsekas, John N. Tsitsiklis

[18] Learning Bayesian Networks - Richard E. Neapolitan

[19] "What can we learn privately?" - Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, Adam Smith

[20] Probabilistic Inference and Differential Privacy - Oliver Williams (Microsoft Research), Frank McSherry (Microsoft Research)

[21] PrivBayes: private data release via bayesian networks - Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, Xiaokui Xiao

[22] Robust and private Bayesian inference - Christos Dimitrakakis, Blaine Nelson, Aikaterini Mitrokotsa, Benjamin I. P. Rubinstein

[23] Approximating Discrete Probability Distributions with Dependence Trees - C.K. Chow, Senior Member of IEEE and C. N. Liu ,Member of IEEE

[24] Introduction to Algorithms - Thomas H. Cormen

[25] The Algorithm Design Manual - Steven S Skiena - Chapter 15.2

[26] Uberveillance and the social implications of microchip implants : emerging technologies. Michael, M. G., Michael, Katina, 1976-. Hershey, PA. ISBN 978-1466645820. OCLC 843857020

[27] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. Journal of the American Statistical Association, 60(309):63–69, 1965.

[28] A. V. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In ACM SIGKDD, pages 217–228, 2002.

[29] Data Analytics with Differential Privacy - Diploma Thesis - Technical University of Crete - Vassilis V. Digalakis, Jr.

[30] DataSynthesizer: Privacy-Preserving Synthetic Datasets, Haoyue Ping - Drexel University, USA, Julia Stoyanovich - Drexel University, USA, Bill Howe - University of Washington, USA

[31] Wasserstein GAN by Martin Arjovsky, Soumith Chintala and Léon Bottou

[32] Generating Differentially Private Datasets using GANs by Anonymous Authors