NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Protection of Sensitive Data: Creating, Analyzing and Testing Protocols of Differential Privacy

Nikolaos G. Galanis

Supervisor: **Konstantinos Chatzikokolakis,** Associate Professor

ATHENS

JULY 2021

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Προστασία Ευαίσθητων Δεδομένων: Δημιουργία, Ανάλυση και Δοκιμή Πρωτοκόλλων Διαφορικής Ιδιωτικότητας

Νικόλαος Γ. Γαλάνης

Επιβλέπων: **Κωνσταντίνος Χατζηκοκολάκης,** Αναπληρωτής Καθηγητής

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2021

BSc THESIS

Protection of Sensitive Data: Creating, Analyzing and Testing Protocols of Differential Privacy

Nikolaos G. Galanis

S.N.: 1115201700019

SUPERVISOR: Konstantinos Chatzikokolakis, Associate Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Προστασία Ευαίσθητων Δεδομένων: Δημιουργία, Ανάλυση και Δοκιμή Πρωτοκόλλων Διαφορικής Ιδιωτικότητας

Νικόλαος Γ. Γαλάνης
Α.Μ.: 1115201700019

ΕΠΙΒΛΕΠΩΝ: **Κωνσταντίνος Χατζηκοκολάκης,** Αναπληρωτής Καθηγητής

ABSTRACT

The problem of preserving privacy while extracting information during data analysis, has been an everlasting one. Specifically, during the big-data era, user details can be easily compromised by a malicious handler, something considered both as a security, and as a privacy issue.

With that being the case, there is a simple solution of denying the access to user data, thus making the mining of useful information about a plethora of subjects impossible. On the other hand, a successful mechanism would be for the data to be flowing without control, something that would be beneficiary for the advance of sciences (because of the huge amount of information that would be available), but a significant compromisation for the individuals' privacy.

However, none of these two solutions are applicable and helpful for solving our problem. The answer is finding a balance, that would benefit both parties: the users and their privacy, as well as the researchers. The optimal fix to the subject, is Differential Privacy, which is actually a promise, made by the data handler to the user, that they will not be affected, by allowing their data to be used in any analysis, no matter what other studies/databases/info resources are available. Meanwhile, the output data statistics should be accurate enough for any researcher to extract useful information from them.

This is a promise that in the first sight, seems rather hard to be achieved. Despite that, during this thesis, we will look closely into the theory which makes this form of privacy possible, by the addition of random noise to the user data. Differential Privacy is based on probabilistic theories, well known from the 20^{th} century, however, it is a rather new technique, which has yet to be fully implemented in a handy way for all data-miners to use.

The goal of this thesis, is to examine and compare previously created mechanisms for D.P., while also creating our own mechanism, that serves to the purpose of achieving Local D.P., a form of Differential Privacy that is nowadays widely used in machine learning algorithms, aiming to protect the individuals that send their personal data for analysis. We will do so, by creating a library that is easy to use, and applies to all the rules of data privacy, and then extract conclusions from its use.

During this thesis, a lot of testings will be made, in order to convince for the usability and the efficiency of Differential Privacy.

SUBJECT AREA: Data Privacy

KEYWORDS: Differential Privacy, Security, User data, Data Privacy, Noisy Data, Aggregation of Data

ΠΕΡΙΛΗΨΗ

Το πρόβλημα της διατήρησης της ιδιωτικότητας κατά την ανάλυση δεδομένων, υφίσταται για πολύ καιρό. Συγκεκριμένα, στην εποχή των big-data, λεπτομέρειες των χρηστών μπορούν εύκολα να παραβιαστούν από κακόβουλους χειριστές των δεδομένων, γεγονός που θεωρείται ζήτημα τόσο όσον αφορά την ασφάλεια, όσο και την προστασία της ιδιωτικότητας του ατόμου.

Με την υπάρχουσα κατάσταση, υπάρχει η απλή λύση της άρνησης της πρόσβασης σε δεδομένα χρηστών, στον βωμό της προστασίας τους, κάτι που καθιστά την εξαγωγή συμπερασμάτων για ποικίλα θέματα αδύνατη. Από την άλλη, ένας επιτυχημένος μηχανισμός θα ήταν η ελεύθερη διακίνηση των δεδομένων, χωρίς φιλτράρισμά τους, γεγονός που θα ήταν ωφέλιμο για την πρόοδο των επιστημών (λόγω του μεγάλου όγκου δεδομένων που θα ήταν διαθέσιμος), αλλά μία μεγάλη παραβίαση της ιδιωτικότητας των ατόμων.

Ωστόσο, καμία από τις δύο αυτές λύσεις δεν μπορεί να εφαρμοστεί και να μας βοηθήσει στην επίλυση τους προβλήματός μας. Η απάντηση είναι η εύρεση μίας ισορροπίας, η οποία ευνοεί και τα δύο μέρη: τους χρήστες και την ιδιωτικότητά τους, όπως και τους ερευνητές. Η βέλτιστη επίλυση του θέματος, είναι η Διαφορική Ιδιωτικότητα, που στην πραγματικότητα πρόκειται για μία υπόσχεση από τον χειριστή των δεδομένων προς τον χρήστη, πως ο χρήστης δεν θα επηρεαστεί αν επιτρέψει τη χρήση των δεδομένων του σε κάποια ανάλυση, χωρίς περιορισμούς όπως η παράλληλη ύπαρξη άλλων μελετών/βάσεων δεδομένων/πληροφοριών που υπάρχουν για αυτόν. Παράλληλα, τα στατιστικά του αποτελέσματος της ανάλυσης, πρέπει να είναι αρκετά ακριβή, ώστε ο ερευνητής να μπορεί να εξάγει χρήσιμη πληροφορία από αυτά.

Η υπόσχεση αυτή, δείχνει δύσκολα υλοποιήσιμη με την πρώτη ματιά. Παρόλα αυτά, σε αυτήν την πτυχιακή εργασία, θα ερευνήσουμε με λεπτομέρεια τη θεωρία που καθιστά εφικτή αυτή τη μορφή ιδιωτικότητας, με την προσθήκη τυχαίου θορύβου στα δεδομένα. Η Διαφορική Ιδιωτικότητα βασίζεται σε πιθανοτικές κατανομές, γνωστές ήδη από τον 20^o αιώνα, όμως παραμένει μία νέα τεχνική, η οποία δεν έχει πλήρως υλοποιηθεί με τρόπο τέτοιον ώστε να μπορεί να χρησιμοποιηθεί από πολλούς ανθρώπους που είναι υπεύθυνοι για την εξαγωγή δεδομένων.

Σκοπός αυτής της πτυχιακής εργασίας, είναι να μελετήσουμε και να συγκρίνουμε ήδη υλοποιημένους μηχανισμούς πανω στην Δ.Ι., ενώ παράλληλα θα δημιουργήσουμε τον δικό μας μηχανισμό, ο οποίος χρησιμοποιείται για τους σκοπούς της Τοπικής Διαφορικής Ιδιωτικότητας που συναντάται την σήμερον ημέραν σε αλγορίθμους μηχανικής μάθησης, με στόχο να προστατέψει τα δεδομένα που αποστέλλουν για εκμάθηση οι χρήστες. Θα το κατορθώσουμε αυτό δημιουργώντας μία προγραμματιστική βιβλιοθήκη η οποία είναι εύκολη στη χρήση, ικανοποιώνατας παράλληλα τους κανόνες της προστασίας δεδομένων, και τέλος θα εξάγουμε συμπεράσματα από τη χρήση της βιβλιοθήκης αυτής.

Κατά την διάρκεια αυτής της εργασίας, θα πραγματοποιηθούν πολλές μετρήσεις, με στόχο να γίνει πειστική η χρησιμότητα και η αποτελεσματικότητα της Διαφορικής Ιδιωτικότητας.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Προστασία και Ιδιωτικότητα Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Διαφορική Ιδιωτικότητα, Ασφάλεια, Δεδομένα Χρηστών, Προστασία Δεδομένων, Θόρυβος σε Δεδομένα, Συλλογή Δεδομένων

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Need for Privacy

In our days, data is everywhere, including our smartphones, our computers our TVs, even our watches. Every device and nearly every website track down data, in order to provide more personalized services. This, of course, is desired by the users, as they are more likely to see relevant advertisements, and in general, have a more unique experience while they are using their devices.

At the same time, the services that track down the data are also benefited, because of the way that science evolves: Experiments need to be made, thus the more available data in order to conduct them, the better. As an example, we might think the medical community: when someone logs-in to the hospital, it is beneficiary for the doctors to gather his data, in order to study his decease, and his potential recovery, not only for the shake of the patient, but also for the further study of the decease.

While providing data may seem inevitable and yet beneficiary for all parties, there is always a risk that this data will be used in order to compromise the user's privacy. When the information lands in the wrong hands, it can expose some characteristics of the user that he does not want to be shared. In our medical example, let's now consider a patient with a rare decease, who logs-in to a local hospital. He might consent to share his personal data (name, age etc.), but only for the doctors to use it. What will happen when the doctors give the data of the whole hospital for analysis? This patient, considering he is one of the few that has this illness, may be stigmatized, when the data analysts find out his condition. Wouldn't it be better for him, if, let's say, his name was not exposed? We will see later on, why this approach, is found to be successful, but not enough, for extreme cases.

1.2 Definition of the problem of privacy

In general, when we consider the *problem of privacy*, we refer to the protection of the disclosure of sensitive information of individuals, when a collection of data about these individuals (dataset) is made publicly available.

1.2.1 Achieving Privacy via Anonymization

One of the first, and rather successful attempts for preserving privacy, was anonymization, meaning removing all personal identifiers from the dataset. This technique is further developed, using famous algorithms like k-anonymity, l-diversity etc. However, there are several problems with this approach. Firstly, they are very computational heavy, as their complexity rises up to an exponential one, making the anonymization of a large dataset very slow. Also, the anonymization does not guarantee that the user will remain private, if other datasets are not anonymized. Let's once again consider our example. Suppose our patient goes to two separate hospitals for his treatment, and one of them uses the best anonymization techniques, while the other one provides the data without any form of privacy. Our patient is on both of the datasets, thus the techniques adopted by the first hospital are now useless. This expands to the real world, because, no matter how careful you (and the services that you use) are, a single data breach is enough for you to be compromised.

So, right now, things seem a bit pessimistic, supposing that anonymization, no matter how well performed, can not fix our problem. Another successful technique, that is used on many fields, is the addition of noise. During a later section, we are going to examine in which ways it can benefit us while trying to solve our problem.

1.2.2 Achieving Privacy via Randomization

Randomization can be applied to the data of the users in two different forms:

- Apply random noise *directly to the data*. This will result to altered data, which will then be processed, so that the adversary will not be able to individualize the entries in the dataset.
- Apply random noise to *queries asked to the dataset*. In that case, the dataset is not directly available to the analysts. Instead, they are allowed to ask questions to the dataset, and the answers are then being randomized, and returned.

Both the above approaches are utilized, but the second one is widely preferred. During our analysis of data privacy, we are going to dive in both of those techniques, as well as the libraries that they are used in.

As we can see, the randomization method looks good in theory, but we must answer to several questions before implementing it, such as:

- How can we define privacy for noisy queries?
- What type of noise do we need?
- What should we do in the case of extreme amount of noise added?

We are going to answer those questions later on, during our next chapters.

1.3 Goal of this thesis

As discussed in the introduction, that the most effective up-to date method for applying privacy into a dataset, is via randomization. The method used, is called *Differential Privacy*, and is based on injecting noise into the users' data.

The theory behind this method includes many mathematical theorems, however, it can by easily explained. We will proceed by taking a look on those principles, and analysing the theory behind this form on data privacy. Then, we will proceed by examining some existing applications of D.P., especially some libraries that help us to apply this technique in a dataset. Finally, we are going to create our own library in order to apply Local D.P., a form of privacy that we will discuss in the next chapter.

This library will allow a user to fully anonymize a dataset, and afterwards create histogram and counting queries for this dataset. During the implementation of this library, a new protocol will be introduced, which follows the rules of D.P., and produces better results than many already-existing protocols.

2. PRINCIPLES OF DIFFERENTIAL PRIVACY

During this chapter we are going to introduce the term of D.P., and its definition, alongside with the principles that need to be followed while applying it.

2.1 Promise of Differential Privacy

Differential Privacy is actually a promise made by the data handlers, to the participants of a study: ”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/ datasets/ info resources are available”.

The goal is to make the data widely available for analysis, while protecting the users. However, is it possible to learn nothing about an individual, while gathering useful information about a population? This is actually what D.P. is trying to achieve.

2.2 Definition of Differential Privacy

Before defining D.P., we must analyze some of the basic components of its definition.

2.2.1 Randomized Response

Randomized response is one of the earliest privacy mechanisms, that is used to conduct surveys where taboo behaviour is studied. The participants in those surveys are asked to answer truthfully, while they do not want to be stigmatized. There is a micro-world of what we are trying to achieve, thus we are going to give the algorithm of the randomized response in order to answer a binary (yes/no) question.

- Flip a coin.
- If it lands on heads, answer truthfully
- If it lands on tails, flip another one
- If it lands on heads, answer no, else, answer yes

We are going to analyze this algorithm and its success in later chapters, but for now, it is enough to know that there exists a simple mechanism that adds noise, and is rather accurate for large samples.

Before giving the definition of D.P., we must define its components.

- **Probability Simplex**, given a discrete set B, is denoted as $\Delta(B)$ and is defined to be:

$$\Delta(B) = \{x \in R^{|B|} : x_i \geq 0 \ \forall i \text{ and } \sum_{i=1}^{|B|} x_i = 1\}$$

- A **Randomized algorithm** M with domain A and discrete range of results B, is associated with the mapping $M : A \to \Delta(B)$.
- **Distance between Databases:** The l_1 norm of a database x is denoted $||x||_1$ and it is defined to be: $||x||_1 = \sum_{i=1}^{|x|} |x_i|$. Thus, the l_1 distance between 2 databases x and y, is $||x - y||_1$, and the size of a database x os $||x||_1$.

2.2.2 Definition

Differential Privacy is defined as following:

A randomized algorithm M with domain $N^{|x|}$ is (ε,δ)-differentially private, if for all $S \in Range(n)$ and for all $x, y \in N^{|x|}$ s.t. $||x - y||_1 \leq 1$

$$Pr[M(x) \in S] \leq e^{\epsilon} Pr[M(y) \in S] + \delta$$

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

It should be noted that D.P. is rather a definition than a strict algorithm. While relying on the definition of D.P., we can create different algorithms, which will all ensure that the result will be deferentially private. This allows us to create different forms of D.P., that will be analyzed later on this thesis.

The whole point of Differential Privacy, is that the output of a D.P. mechanism, should by *independent* of whether or not an individual is present in the domain N. The "ability" of the adversary to recognise the existence of a column in the dataset, is regularized by epsilon.

2.3 The meaning of epsilon

It is made clear from the above definition, that if we have a computational task, we might find different algorithms for applying D.P., but the result will always be of the same form: each user of the dataset, will get ε-D.P.. But what does the epsilon parameter actually mean?

By reading the mathematical equation, we observe that the higher the value of epsilon, the bigger the difference between the two probabilities (minimum and maximum). Thus, we extract the following statement about the value of epsilon during the application of Differential Privacy:

- The *lower the epsilon* value, the *higher the privacy* guarantees for the users of the dataset.
- The *higher the epsilon* value, the *more accurate the results* produced.

In practice, epsilon values vary in the range $(0, 5]$, as lower values are prohibited, and higher values are considered extreme cases. However, as mentioned in [1], when epsilon is small, failing to be (ε,0)-differentially private is not necessarily alarming, if our algorithm is linearly increasing with ε (ex (2ε,0)-D.P). This happens because of the nature of the epsilon parameter, which guarantees very strict boundaries between databases. However, when ε increases by a lot, users' privacy suffers.

In **Figure 2.1**, we can see in general terms, the function between the epsilon and the accuracy error, as well as the protection guaranteed. We will discuss in later sections the details on how these graphs are created, but now is a good time to get an overall picture of the accuracy error produced when applying D.P.

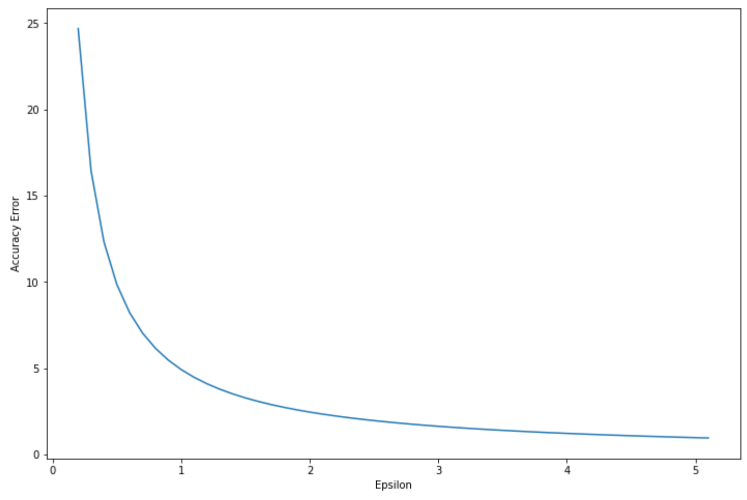


Figure 2.1: Accuracy Error as a function of epsilon

2.4 Different forms of Differential Privacy

As mentioned during the definition, due to the room that is left for its interpretation, there can be many forms of Differential Privacy. There are two major fields recognized, the *Global D.P.* and the *Local D.P.*.

Their major difference is the curator of the data. In the Global model, the curator must be trusted, as he collects the non-private data and has to pass them through a D.P. algorithm.

On the other hand, in the Local model, the curator may as well be untrusted, since the users perturb their data on their own, using a specific protocol. The key differences of the two forms are shown in the *Figure 2.2* below.

An other difference between the two models, is the amount of noise added. With the absence of a trusted curator, the users themselves must add a significant amount of noise into their data, in order to preserve their privacy. This of course results into a need of many users (several thousands), in order for the L.D.P. protocols to function correctly and accurately.

In this thesis, we are going to examine both models, by quoting their definitions, observing already-existing algorithms, and creating our own L.D.P. protocol.

2.5 Existing Problems of D.P.

As every new step in Computer Science, Differential Privacy has some issues that are yet to be solved, and some others not covered by its definition.

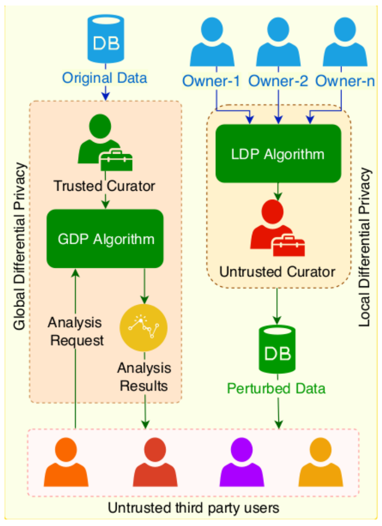


Figure 2.2: Differences between LDP and GDP

One major problem is the behaviour of the protocols *when the number of users is limited*. The definition of D.P. is based on the alteration of the data in order not to reveal sensitive information. Thus, if a small amount of users are involved in those protocols, the accuracy of the results might be way off the standards that we set, in order to satisfy the epsilon requirements of the user.

Another (unsolvable) issue, that mainly lies on the basis of surveys, is *the possibility that conclusions drawn from a survey may reflect statistical information about an individual*.

For example, if a survey about the correlation of smoking and dental problems is conducted, someone that has specific dental problems might be deemed as a smoker, despite keeping his privacy about the fact that he is smoking, during the survey. That is something that D.P. does not promise: unconditional freedom from distinguishing. This is not however a violation of the definition of D.P., as the survey teaches us that specific private attributes correlate with public observable attributes, since this correlation would be observed independent of the presence or absence of the individual in the survey.

There are several more issues as the ones covered above, however we are not going to focus on those, rather on the advantages of D.P.

3. EXAMINATION OF PREVIOUSLY-EXISTING GLOBAL DP LIBRARIES

The first goal of this Thesis is to examine previously existing programming libraries and APIs that provide the application of Differential Privacy to a dataset. This has been achieved by many companies, such as Google and IBM, but also from research programs like ARX that study the benefits of data privacy. We separate those implementations regarding their output. The possible outputs of a mechanism that adds D.P. to a dataset can be:

- An answer to a query, in a private manner.
- An anonymized dataset, that meets the criteria of D.P.

In the first category, we can distinguish libraries such as Google's and IBM's, that have functions which if applied on a dataset, and given a specific query, can return a single answer.

In the second one, we can find libraries such as the ARX tool, that given a dataset and a group of privacy settings (such as the amount of noise to be inserted), produces an anonymized version of a dataset, that has obviously reduced information in comparison to the original one, but is usable by the final user.

In this chapter, we are going to test those libraries by providing different kinds of datasets, in order to determine the advantages and the disadvantages of each category.

We are going to conduct all of our testings using noise generated by the *Laplace Mechanism*, thus we must first define its theoretical behavior.

3.1 The Laplace Mechanism

The Laplace Mechanism is used widely in applications of Differential Privacy regarding *numerical queries*, which are actually functions that match a query to a number, or a vector of numbers, thus answering to it. The mechanism uses noise produced by the Laplace probabilistic distribution, which is proportional to the query's sensitivity.

3.1.1 Query Sensitivity

The l_1 sensitivity of a query f, is defined as following:

$$\Delta f = \max_{\{||x-y||_1=1\}} ||f(x) - f(y)||_1$$

where $x, y \in N^{|X|}$.

This quantity shows the effect by which a single participant's data can change in the worst case during the query f, and thus, the uncertainty that we must insert to to the response in order to protect them.

3.1.2 The Laplace Distribution

The Laplace Distribution with a scale b, is the distribution with probability density function:

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

who's variance is $\sigma^2 = 2b^2$, and is actually a symmetric version of the exponential distribution.

3.1.3 Use of Laplace in D.P.

In order to be of use in our definition, the scale of the noise will be calibrated to the sensitivity of the query f, divided by epsilon. Thus, the noise used will be drawn from

$$Lap(\frac{\Delta f}{\epsilon})$$

Of course, many other probabilistic distributions can be used to ensure differential privacy, but during our testings we prefer to use Laplace.

3.2 Query Answering Libraries

We are going to begin by testing libraries that belong in the first category, and specifically the *IBM's diffprivlib*, which is written in python, and is publicly available here. The library includes a host of mechanisms, the building blocks of differential privacy, alongside a number of applications to machine learning and other data analytics tasks. A full explanation of the library's functionalities is available in [3]. We are going to focus our testings in the simple queries, such as the *mean value*, the *extreme values* and the *histograms* of a numerical dataset. The library consists of three modules:

- **Mechanisms**, as known from the theoretical foundations of D.P.
- **Models**, especially machine learning models, that will not concern us during this thesis
- **Tools** that will allow us to apply D.P. in datasets.

We are going to use the tools available, in order to apply differential privacy in a dataset of our own, guided of course by the mechanisms provided by diffprivlib. First, we are going to take a look at the dataset that we are going to use going forward.

3.2.1 Setup of the mechanism

The first step in order to test the library, is to setup the mechanism by defining its properties and parameters.

3.2.1.1 Bounds' Selection

One of the most important aspects for us if we want to apply DP algorithms, is to define the bounds, i.e. the range that a variable can be in. It would be very convenient in our case to just take the tuple of the smallest and the largest value in the column that we are interested in. However, in the real world, the person who asks the queries is not supposed to know this info about the dataset. Thus, since a solution is not provided by the library, we must define our own bounds by guessing the lowest and the highest values in the fields that we want to examine.

Thus, the user must have somewhat of a previous knowledge regarding the dataset, in order to decide the minimum and maximum value. Those values do not need to be precise, although the more close they are to real ones, the best the protocol will function. At the same time, we must be sure that during our selection we do not leave some of the dataset's values outside of the bounds, as they will be ignored in the final results.

3.2.1.2 Privacy Budget

In this form of D.P., someone trying to breach the users' privacy, could theoretically ask an infinite number of questions, and thus each time gain more and more information about their private data. In the same manner, an untrusted user of the library could ask the same question many times, aiming to determine how much noise is added each time, in order to find out the actual answer to the query, as the way that the noise is drawn is already known.

In order to eliminate this problem, a special parameter call the *privacy budget* is implemented. The library offers the ability to initialize this budget before asking any queries. During the queries, this budget is each time decreased, according to how much data the answer to the question reveals. For example, the answer to the ”mean value of the charges for a surgery”, costs less than the answer to the ”histogram values of heart transplant surgeries in the West coast”.

This parameter is implemented by the library as the budget accountant. This variable tracks the privacy spent, so that our system is not left exposed after lots of ”expensive queries”. The system will allow someone to ask one question that uses the whole privacy budget, or a series of questions whose total impact is less or equal to the initial budget.

3.2.2 Testings' goal

When applying D.P. mechanisms to our data, we provide the privacy settings of our choice (epsilon variable), and obtain an answer to each of our questions. Thus, in our testings, our goal is to *determine the accuracy of the answers*, given a specific ε, or some other settings, and comparing them to the true answer, using some metrics. Those metrics are different for each query type. In this section we are going to focus on two types of queries: statistical, and histograms.

3.2.3 General techniques

In each one of our following testings, we are going to run the query *many times*. As we already know, D.P. relies on probabilistic algorithms that can some times produce extreme results. This may be rare, but we want our testings and conclusions to be accurate. So, we are going to run each query 100 times, and return as a result the mean value of those runs.

3.2.4 Statistical Queries

The first type of queries that we are going to test are those that answer questions like ”What is the mean cost of a surgery?”, or ”What is the largest fee paid by medicare for a transplant?”, known as statistical queries.

3.2.4.1 Metrics used

In the case of statistical queries, their answer is usually a real number, so in order to check their alteration with the true answer, we are going to take into account the *absolute difference between the truth and the query answer*.

3.2.4.2 Bounds Definition

We are considering fees for surgeries in our example, thus a logical lower bound would be 0$ (surgeries could be done pro bono too!), and an upper bound would be 1 million dollars. Either way, we are trying to be extreme with our picks, in order to not find ourselves in the unfortunate situation that a value taken into consideration by the DP query would be out of bounds.

3.2.4.3 The identity of the testing Dataset

The dataset chosen to test the library, is the publicly available ”Surgery Charges Across the U.S.”, that contains many different kinds of surgeries in a plethora of different hospitals. Our goal is to protect each hospital’s data when it comes down to a specific surgery, while helping a patient choose one, depending on the charges that can be found all over the United States. The data provided in this dataset, is going to help a potential patient balance his need of top care, and the need to spend less money. The columns contained in the dataset are:

- Surgery code and definition
- Provider hospital name
- Provider city
- Average total payments
- Average medicare payments

We are going to focus on the last two columns, in order to approximate the charges of a surgery. The above table gives us an image of the containers of the dataset.

The dataset contains a total of 200,000 entries, a more than satisfying number for running D.P. algorithms.

Table 3.1: ”Surgery Charges Across the U.S.” dataset columns

ID	Surgery Type	Hospital Name	Hospital City	Total	Medicare
1	TRANSPLANT	MAYO CLINIC	PHOENIX	$240422.80	$133509.55
2	ECMO	GROSSMONT HOSP	LA MESA	$193617.86	$192003.43
3	CRANIOTOMY	STANFORD HOSP	STANFORD	$32597.87	$29347.12

3.2.4.4 General Dataset Utilities Queries

Our first experiment is just to ask for some of the utilities of the dataset, and specifically its cost column: the *mean value*, the *variance*, the *sum* and the *standard deviation* values of the surgeries' cost.

All of those queries can be executed using the following command (specifically for the mean value query):

```
mean_with_dp = dp.tools.mean(df["Average_Total_Payments"].tolist())
```

where the dataframe column is the one containing the cost of each surgery, and its values should be in a list in order for the library to function.

By running the above mentioned queries, we got the following results:

Table 3.2: General Queries results for Surgeries Dataset

Query	True answer	Private Answer
Mean value	13168.5	13167.3
Sum	262754253.1	262935459.3
Variance	262754253.1	261940796.5
Standard Deviation	18855.1	25825.0

The answers are almost perfect, for example on the mean value, considering that the cost is thousands of dollars, and the error is just 1 dollar. The simplicity of the query just lies to the following instruction:

However, this is just a simple example, executed only once, hence it can not provide us with safe conclusions for the library. In order to do so, we are going to run more complicated examples moving forward.

3.2.4.5 Lowering the Dataset size

As we mentioned above, the entries that the dataset contains are a very large number, and we know as a fact that D.P. functions well when this is the case. How is the library going to respond though if the dataset size is smaller? We are going to run for 4 different values of epsilon, and get the results for an increasing number of entries, starting from 10, and moving to 2000. Thus, the X axis of each plot represents the increasing dataset size, the Y axis the accuracy error, and each plot has a title of the epsilon setting used for the measurements. We can see the results in **Figure 3.1** below.

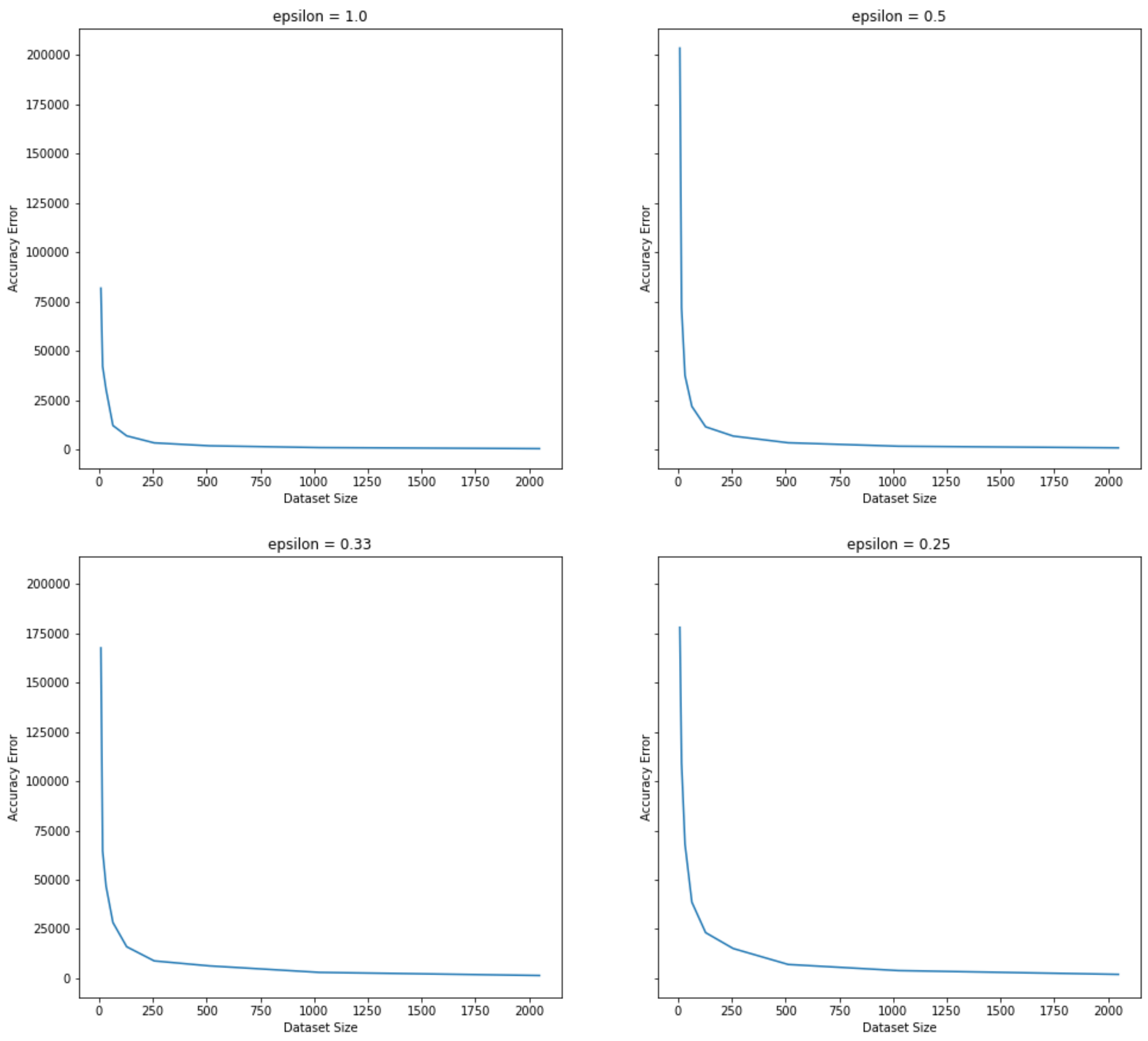


Figure 3.1: Accuracy Error for Increasing Dataset Sizes

By observing the plots, we can make a couple of conclusions:

- *The smaller the epsilon gets, the bigger the accuracy error in the case of small datasets.* This, according to the definition makes sense, because small epsilon indicates higher privacy, thus for small datasets it can mean lower accuracy, due to the high amount of noise added.

- *The accuracy error stabilizes near 0 as the size of the dataset gets over 1000 entries.* Of course, depending to the epsilon value, this point could be earlier in the dataset sizes, as we observe for $\varepsilon = 1$. This again lays in the above mentioned property of the definition.

3.2.4.6 Epsilon measurements

The most important aspect when applying Differential Privacy to a dataset, is the selection of epsilon. This number is held accountable of the trade-off that DP offers: how much accuracy are we going to sacrifice in order to have less privacy loss, and vice versa. Given the library and our surgeries' costs dataset, we are going to measure the accuracy changes with the selection of different epsilon values.

The size of the dataset is too big, and thus the computation of the sensitivity will take too long. We assume that the data are somewhat equally distributed, thus we chose only 1% of the dataset (which is a significant number of members), to take part in our sensitivity calculations.

In order to observe if the library functions well, we are going to compare the results produced with the *theoretical bounds of the Laplace mechanism*.

In theory, the accuracy of an ε-differential private query, when using the Laplace distribution, is equal to the sensitivity of the query divided by epsilon, thus

$$\frac{\Delta f}{\epsilon}$$

where Δf denotes the sensitivity, and ε is our current privacy setting.

Sensitivity is defined as the maximum difference that can be found if we alter a single entry in the dataset. We are interested in global sensitivity in order to conduct our testings, and is defined as:

$$\Delta f = \frac{upper - lower}{length(DB)}$$

where upper and lower are the highest and lowest values of the column we are interested in our dataset, and length is the size of the database.

In order to compute the local sensitivity, we must check the maximum difference in the query result that occurs if we remove a single person from the database. We are going to define a function to do so, so we can check the theoretical bounds during our tests.

The results of those testings are shown in the **Figure 3.2**.

It is clear that as we increase the epsilon value, the privacy loss gets bigger. On the other hand, if epsilon is too small, as we can see in the above plot, extreme errors in accuracy in our queries will emerge.

The optimal value of epsilon varies, there is no general rule for the perfect epsilon. It depends on many different aspects, such as:

- The noise generated by the probabilistic mechanism used
- The implementation of the algorithm
- The size of the dataset
- The query itself.

Moreover, the selection of epsilon depends on the dataset. For example, we might have a dataset that is extremely important to have minimal privacy loss. In that case, we will opt

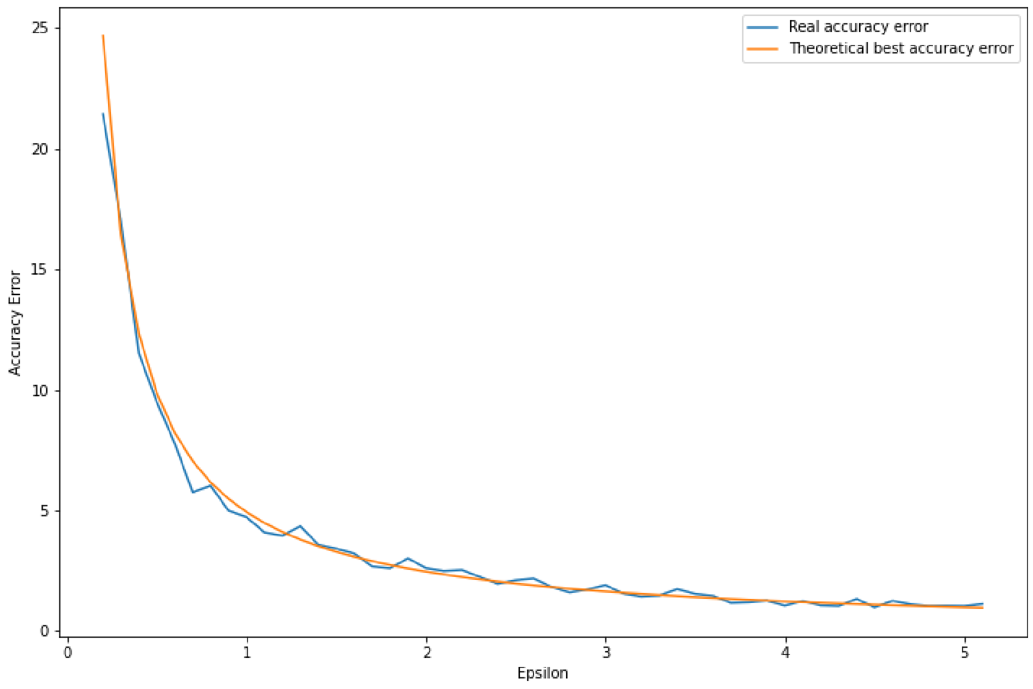


Figure 3.2: Real and Optimal Accuracy Error for Increasing epsilon values

to use a rather small epsilon, thus we do not disclose the sensitive data included. An other dataset could be less sensitive, but the analyst might have a need for extreme accuracy every time, so the epsilon selected should be rather big.

Regarding the comparison with the Laplace bounds, we can see that the library's query performs significantly well throughout the different epsilon values selected. This is due to the fact that IBM uses the same formula to compute the sensitivity, as well as a similar mechanism (Laplace truncated noise), in order to answer to our queries.

3.2.5 Histogram Queries

Histogram graphs are a very handy way to visualize numerical data, compare different values of a specific field, and thus extract conclusions about the dataset. We are going to study the 'diffprivlib''s method of creating an histogram, and its accuracy when changing the epsilon factor.

The IBM DP library offers a differential private way to create histograms. The difference with the simple queries that we tested, is that now, *geometric truncated* noise is added in order to satisfy DP.

3.2.5.1 Metrics Used

The result of a histogram query on a dataset is a vector containing how many entries belong on a specific range. Thus, the comparison between 2 histograms can be held out by comparing those vectors.

There are plenty of metrics used to compare vectors, but we are going to focus on the *Euclidean Metric* and the *Kantorovich metric* (also known as Wasserstein or EMD metric).

The Euclidean metric is one of the most simple metrics, as it takes into account the distance between each pair of the two vectors. Thus, the distance of the vectors is defined by:

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

where n are the total elements of the vectors (must be of equal size), and x and y are the 2 vectors that we are comparing.

The Kantorovich metric is a more complex one, as it examines the cost to move a specific quantity of the one vector to the other, in order for the two to be similar, and thus figuring out their distance. For example, the Kant. distance is larger if we have the vectors $[0001]$ and $[1000]$ than having the vectors $[0001]$ and $[0010]$, as in the first case the first element of the histogram is moved 3 places in order to be similar to the second one. In order to determine the cost, a base metric is used, which in our case will be the euclidean.

The way that the metric works in a naive approach, is explained in **Figure 3.3.**

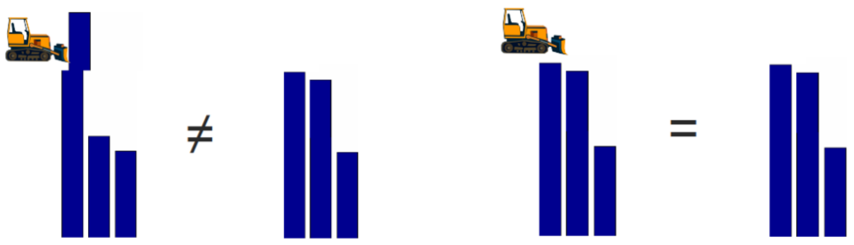


Figure 3.3: Kantorovich Metric Application on 2 histograms

This metric is much more suitable for D.P. as we are not just interested in the alteration of the results, but on the amount of alteration. If, for example, we are examining a histogram that contains the age of a hospital's patients, it is not the same for the private histogram to deem a 10 year old patient as 90 year old, than deeming him as 11.

In order for the Kantorovich metric to be computed, we must solve a Dynamic Programming Problem, and make complex calculations, which are beyond the subject of this thesis. There are many implementations of the metric, and since IBM's library is written in Python, we are going to use the QIF library from [13] which is also available as a Python library.

3.2.5.2 Bounds Definition

In the following testings we are examining salaries, thus we must set the lower and the higher bounds such as to support the most extreme salaries that could possibly be present in the dataset. Therefore, the lower bound is set to 0 dollars, and the higher to 500,000 dollars.

3.2.5.3 The identity of the testing Dataset

During our histogram testings, we are going to use a different dataset, which contains sensitive data regarding employee's salaries in the state of Baltimore, while stating other facts about the members of the dataset.

The columns contained are:

- Name
- Job Title
- Annual earnings
- Gross earnings

We are going to focus on the last 2 columns, containing the employees' salaries. The dataset has 13000 entries, which are more than enough in order for our histograms to be realistic and accurate. The above table gives us an image of the containers of the dataset.

Table 3.3: "Baltimore State Employees Salaries" dataset columns

Name	Job Title	Annual Earnings	Gross
Aaron,Kareem D	Utilities Inst Repair I	32470.0	25743.94
Abadir,Adam O	Police Officer	60200.0	57806.13
Abbeduto,Mack	Council Technician	53640.0	59361.55

3.2.5.4 Simple queries

At first, we are going to run simple instances of histogram queries, so we can take a look at the amount of error that we expect moving forward, and also familiarize with the execution of the commands. To create a private histogram using the library, we must provide the bins. Those should be identical to our bounds, that we have earlier defined. While creating the non-private histograms we are going to use the same bounds, in order for our comparisons to make sense. The creation of the bins and the private histogram are achieved using the following Python instructions:

```
bins = np.linspace(0, 300000, 20)[1:]
result = dp.tools.histogram(input_list, bins = bins,
         epsilon = epsilon, range=bounds_range)[0]
```

where epsilon is the float number selected as the epsilon parameters, and bounds range is the tuple of our dataset bounds.

The result of the execution of both the private and the non-private histogram queries for the annual earnings column are shown below, in the **Figure 3.4**.

The results are quite impressive. This is probably due to the large dataset size: we are not able to locate small changes. In order to do so, we are going to check our error using the accuracy error function that we have defined.

While applying our metrics, the Euclidean error is 14.81, and the Kantorovich error is 91.63. The euclidean distance error determines how many entries were wrongly classified in a

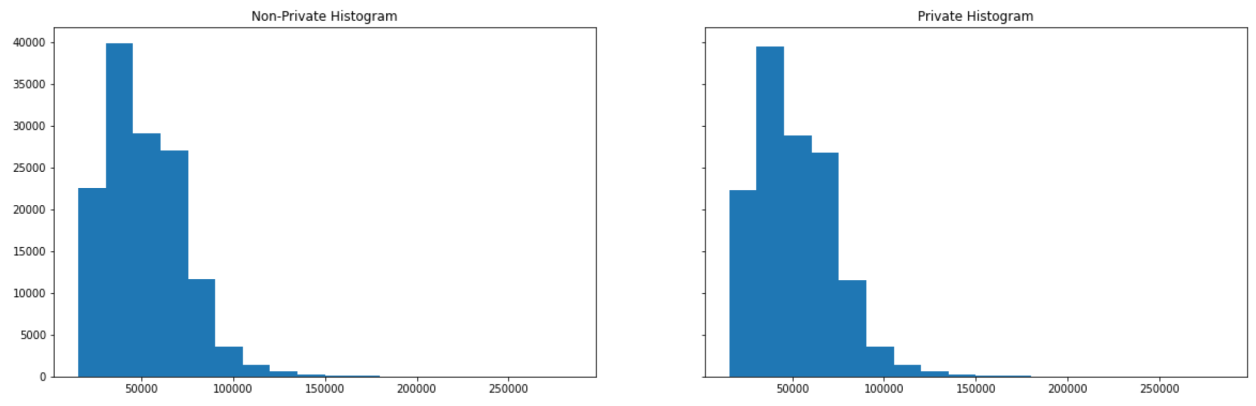


Figure 3.4: Private and Non-private histograms for the Salaries Dataset

bin (in average), when the differentially private query was run. So, less than 20 people out of 13.000 were wrongly classified, whereas their privacy was secured. This is quite a good trade-off!

3.2.5.5 Epsilon Measurements

Once again, we are going to run the histogram queries for different values of epsilon, in order to check their behavior as the parameter increases. The results of the execution for the Euclidean metric are shown in the **Figure 3.4**, and for the Kantorovich metric in the **Figure 3.5**.

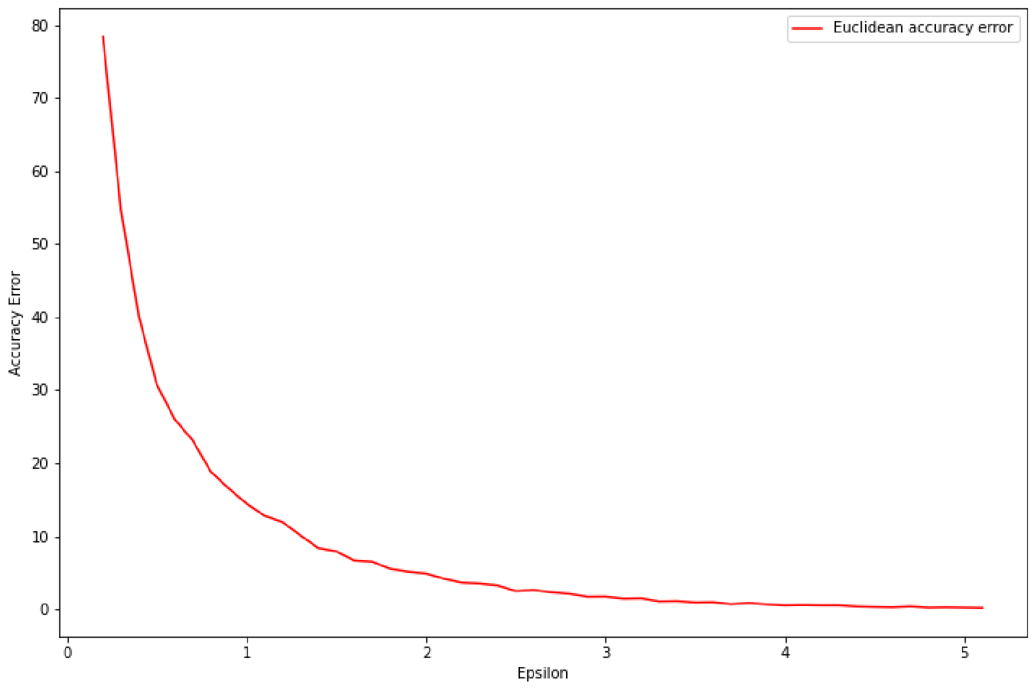


Figure 3.5: Euclidean Metric Error for increasing values of epsilon

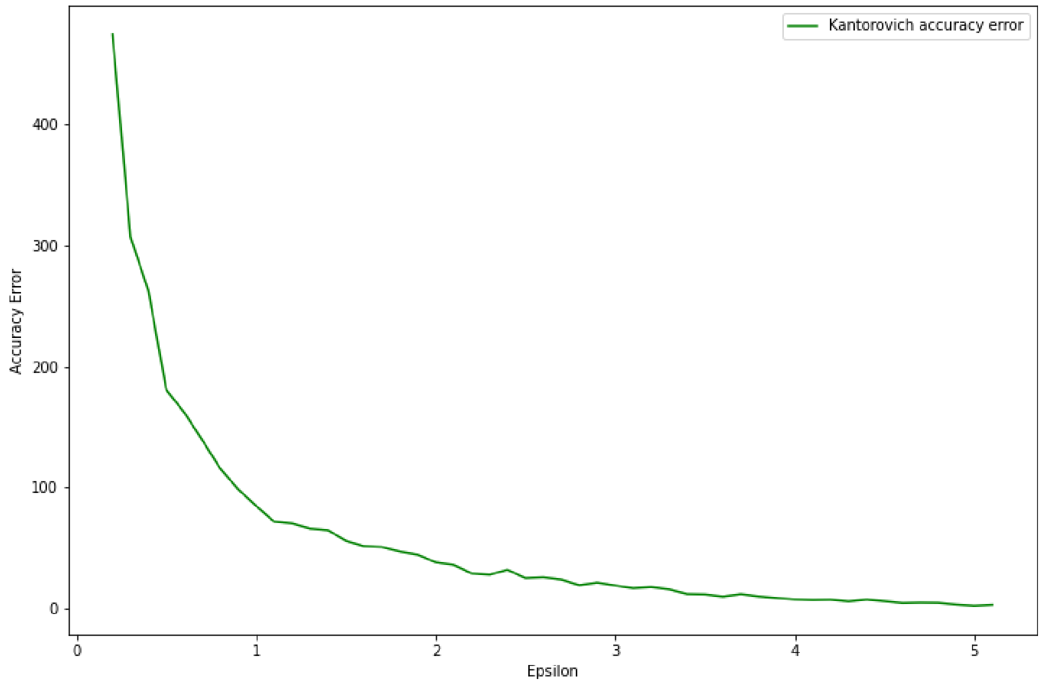


Figure 3.6: Kantorovich Metric Error for increasing values of epsilon

We observe that the error curves follow the same ratio as the previous ones, indicating what we already know, that the error decreases as the epsilon values increase. This is another case where the library produces accurate and definition-aware results.

3.2.5.6 Histogram queries in theory

Based on [1] the histogram queries are very high sensitivity queries, thus a slight change to the bounds could be critical for their result.

The authors suggest that we use noise generated by the Laplace mechanism, but with a slight change. In detail they suggest the following:

"In the special (but common) case in which the queries are structurally disjoint we can do much better — we do not 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 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 $Lap(\frac{1}{\epsilon})$ to the true count in each cell."

3.2.6 Conclusions

After a rather satisfying amount of testing in a large, real world dataset, we suggest that the IBM DP library has quite impressive results when it comes down to the trade-off between privacy and accuracy during both simple counting queries and histogram ones. However, we are cautious about 2 problems that were observed while using the library:

- *Bounds checking*. The user must define himself the bounds, a fact that causes for speculations on the variance of the values in the dataset. It would be convenient to take the lowest and highest value in the field that we are examining, in fact that is how IBM demonstrates those examples, but that violates the rule that prevents the user from having any info of the dataset before the DP processing.
- *Non-DP preprocessing*. If we ask complicated queries (ex Surgeries performed in Stanford), the library does not offer a way to preprocess the data, thus we trust python in doing so, which results in a non-DP way of shrinking the dataset. The result obtained is of course differential private, but what happens if the dataset has only 1 record in it? That, while being a very extreme case, violates the definition of differential privacy.

The bounds' problem can be solved if the users have prior knowledge in the domain of the dataset, in order to safely define the bounds themselves.

The Non-DP preprossessing is a more tricky one, that needs special techniques in order to cover the edge case mentioned. However, we are going to check such techniques in the following section, which is based on returning the whole dataset after the application of D.P.

3.3 Anonymized Dataset Producing Libraries

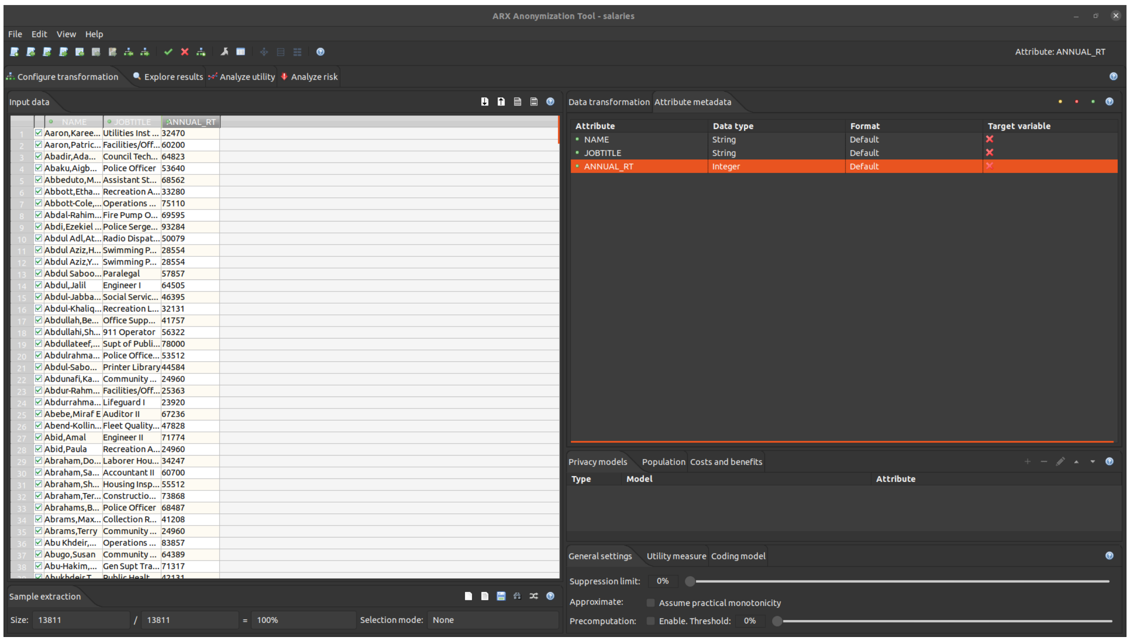
As mentioned earlier, an other possible output of a mechanism that adds D.P. to a dataset can be the dataset itself, after being anonymized using certain algorithms that meet the criteria of D.P.

This technique does not yet have many different implementations, mostly due to the success of the previous model shown, as well as the difficulty, the computer power needed and the poor quality of the result being produced.

Producing an anonymized dataset is the way to go if someone is using earlier forms of data privacy, such as *k-anonymity*, *l-diversity* etc, which we are going to analyze moving forward. However, in order to cover the needs of D.P., several adjustments have to be made. The main idea behind all those libraries, lay behind a theorem, presented in [4], that mixes the use of those previously mentioned techniques with D.P.

In this Thesis, we are going to examine the *ARX tool*, a tool for data anonymization, that supports the method that we are trying to implement. We are going to analyze this tool, and perform similar testings as in IBM.

ARX is a tool for data anonymization, that in general, takes a dataset as an input, applies privacy models, and produces an anonymized version of this dataset, thus offering protection to its members. The Menu of the ARX tool can be seen in the following **Figure 3.7.**



		NAME	JOBTITLE	ANNUAL_RT
1	✓	Aaron,Karee...	Utilities Inst ...	32470
2	✓	Aaron,Patric...	Facilities/Off...	60200
3	✓	Abadir,Ada...	Council Tech...	64823
4	✓	Abaku,Aigb...	Police Officer	53640
5	✓	Abbeduto,M...	Assistant St...	68562
6	✓	Abbott,Etha...	Recreation A...	33280
7	✓	Abbott-Cole,...	Operations ...	75110
8	✓	Abdal-Rahim...	Fire Pump O...	69595
9	✓	Abdi,Ezekiel ...	Police Serge...	93284
10	✓	Abdul Adl,At...	Radio Dispat...	50079
11	✓	Abdul Aziz,H...	Swimming P...	28554
12	✓	Abdul Aziz,Y...	Swimming P...	28554
13	✓	Abdul Saboo...	Paralegal	57857
14	✓	Abdul,Jalil	Engineer I	64505
15	✓	Abdul-Jabba...	Social Servic...	46395
16	✓	Abdul-Khaliq...	Recreation L...	32131
17	✓	Abdullah,Be...	Office Supp...	41757
18	✓	Abdullahi,Sh...	911 Operator	56322
19	✓	Abdullateef,...	Supt of Publi...	78000
20	✓	Abdulrahma...	Police Office...	53512
21	✓	Abdul-Sabo...	Printer Library	44584
22	✓	Abdunafi,Ka...	Community ...	24960
23	✓	Abdur-Rahm...	Facilities/Off...	25363
24	✓	Abdurrahma...	Lifeguard I	23920
25	✓	Abebe,Miraf E	Auditor II	67236
26	✓	Abend-Kollin...	Fleet Quality...	47828
27	✓	Abid,Amal	Engineer II	71774
28	✓	Abid,Paula	Recreation A...	24960
29	✓	Abraham,Do...	Laborer Hou...	34247
30	✓	Abraham,Sa...	Accountant II	60700
31	✓	Abraham,Sh...	Housing Insp...	55512
32	✓	Abraham,Ter...	Constructio...	73868
33	✓	Abrahams,B...	Police Officer	68487
34	✓	Abrams,Max...	Collection R...	41208
35	✓	Abrams,Terry	Community ...	24960
36	✓	Abu Khdeir,...	Operations ...	83857
37	✓	Abugo,Susan	Community ...	64389
38	✓	Abu-Hakim,...	Gen Supt Tra...	71317
39	✓	Abukhdeir,T...	Public Healt...	42131

Size: 13811 / 13811 = 100% Selection mode: None

Attribute	Data type	Format	Target variable
NAME	String	Default	✗
JOBTITLE	String	Default	✗
ANNUAL_RT	Integer	Default	✗

Type	Model	Attribute

Suppression limit: 0%
Approximate: Assume practical monotonicity
Precomputation: Enable. Threshold: 0%

Figure 3.7: The ARX GUI tool

At its core, ARX uses a highly efficient globally-optimal search algorithm for transforming data with full-domain generalization and record suppression. The transformation of attribute values is implemented through domain generalization hierarchies, which represent valid transformations that can be applied to individual-level values.

3.3.1 Classic Privacy Models

The ARX tool offers standard privacy models that are tested in theory and are widely used to ensure anonymity given a plain dataset. Those consist of the implementation of the following protocols:

- **K-anonymity**: A well-known privacy model that aims to protect datasets from re-identification in the prosecutor model. A dataset is k-anonymous if *each record cannot be distinguished from at least $k-1$ other records regarding the quasi-identifiers.* Each group of indistinguishable records forms a so-called equivalence class.
- **l-diversity**: This privacy model can be used to protect data against attribute disclosure by ensuring that each sensitive attribute *has at least l "well represented" values in each equivalence class*. Different variants, which implement different measures of diversity, have been proposed.

Moreover, the tool uses some simple concepts of processing a dataset:

- **Random Sampling**: A method of sampling that utilizes some form of random selection. In order to have a random selection method, we must set up some process or procedure that assures that the different units in the population have equal probabilities of being chosen.
- **Attribute Generalization**: Generalizing a column of the dataset, based on its values. The applications of attribute generalization depend on the type of records (eg. integers, ranges etc).
- **Record Suppression**: Deletion of a specific row on the input dataset.

Those are some techniques that are not going to be analyzed and tested in this thesis, however, if combined with D.P. can produce interesting results. Specifically, according to [4], the following theorem applies:

Random sampling with probability β followed by *attribute generalization* and the *suppression* of every record which appears less than k times *satisfies (ϵ, δ) differential privacy* for every $\epsilon \geq -ln(1-\beta)$ with

$$\delta = \max_{n:n \geq n_m} \sum_{j>\gamma_n}^{n} f(j;n,\beta)$$

where $n_m = \frac{k}{\gamma} - 1$, $\gamma = \frac{e^{\epsilon}-1+\beta}{e^{\epsilon}}$ and $f(j;n,\beta) = \binom{n}{j}\beta^j(1-\beta)^{n-j}$.

In order to achieve attribute generalization, ARX uses the so called *hierarchies*. They are either imported from a csv file, or being hard-coded into the API, and they are used in order to generalize a sensitive field. An example is given in *Table 3.4*. The subject to generalize is the age of a person. Let's see the values as they proceed through generalization.

3.3.2 Conducting D.P. Testings

ARX provides a cross-platform graphical tool, that supports many different ways of anonymizing data, as well as an API that delivers those data anonymization capabilities to

Table 3.4: Generalization of data using hierarchies

1^{st} level	2^{nd} level	3^{rd} level	4^{th} level
1	0-4	0-9	*
3	0-4	0-9	*
5	5-9	0-9	*
10	10-14	10-19	*
18	15-20	10-19	*

Java programs. We are going to use the latter, in order to create our own scripts for testing the tool and its accuracy.

In order to test the accuracy of the models used by ARX, we are going to run simple queries, on the datasets produced by the anonymization process. We want to eliminate the probability of extremely high noise generation, thus we are going to run the anonymization tool multiple times, and the output dataset will be constructed by the mean values of the fields.

As show on the above matrix, ARX hierarchies tend to replace every type of value with an interval. This is not desirable when applying the testings we mentioned. Thus, we had to come up with a better solution of defining hierarchies. The ARX GUI provides a wizard that gives a variety of choices so the user can easily create a hierarchy for many data types.

Another challenge is the number of layers that we are going to use, meaning how far our anonymization will proceed. In each layer, the number of same records increase exponentially, thus we do not want to apply many layers, in order for our results to be accurate, and the output dataset to be readable.

Given the help from Dr. Fabian Prasser, one of ARX's creators, we opted to treat the integer values as numbers, and in each level:

- Group the rows by 2
- Apply a function according to the query we want to ask.

For example, if we want a counting query, the best option would be to apply an *arithmetic mean* function to the group, thus the sum, the mean, the variance etc will be the same. The way that ARX preserves D.P. with those settings, is by record suppression. If that was not the case, the results would be identical to the input dataset. However, now, the output dataset will differ because of its lack of some rows of the input.

Regarding the layers problem, we opted to use 4 layers of anonymization, the last of whom will be the * value, meaning that every record is inseparable. We do not want this to happen early in our anonymization, but we do not want it to never happen either, because then we would have a privacy leak, if the dataset was too small.

The creation of the hierarchies for the salary column, can be shown in **Figure 3.8**, taken from the ARX GUI Hierarchy Creation Wizard.

3.3.3 Metrics Used

We are going to test the applicability of the already given ARX mechanisms on a numerical dataset. Our goal is to run basic queries, such as mean value on the dataset's records.

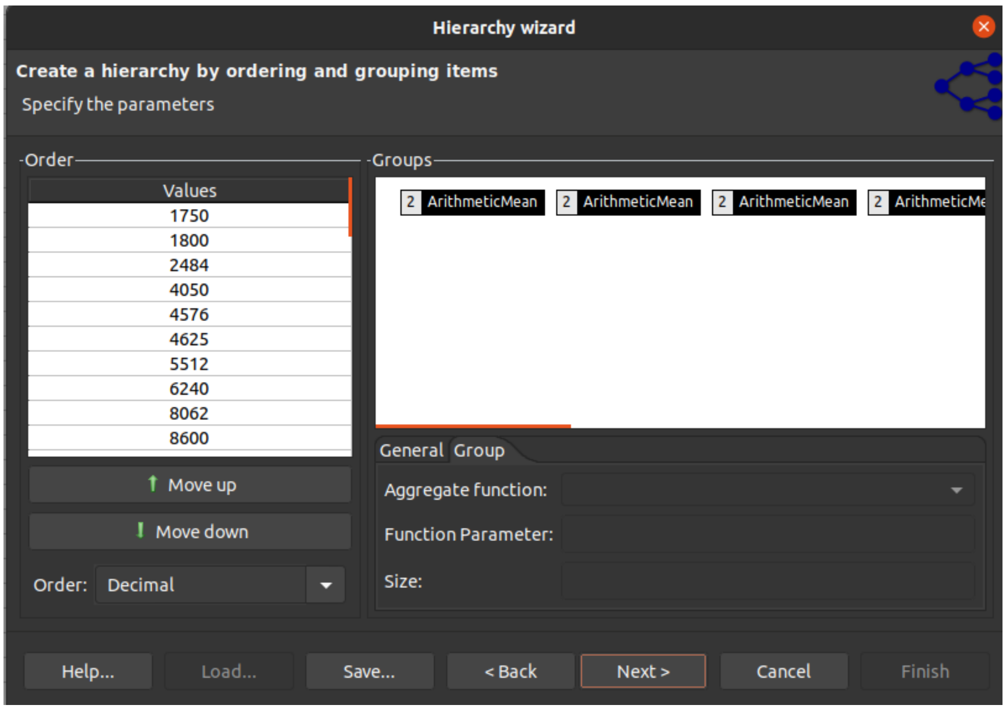


Figure 3.8: Creating an Hierarchy using ARX GUI

We are going to do that first by applying no DP at all, and then by using the API that is presented by ARX, helped by a simple java script that was built for this purpose.

3.3.4 The identity of the testing Dataset

The dataset that we are going to be looking at, contains sensitive data regarding NBA players' *salaries* from the year 1990 until today. It also states other info about them, such as their *age*, their *current team* and their *position*. This particular data is not considered sensitive, as those numbers are widely available, however, when it comes down to certain people's salaries, applying D.P. in order to preserve their privacy is crucial.

3.3.5 Process of running the queries

As we have earlier noted, the application of D.P. in ARX is rather complicated, specifically for the use that we are interested in: We want the output dataset to have numerical values in the earnings' column, in order to apply queries.

For each column of the dataset, we have defined our own hierarchies. For every column except the 'Salaries' one, this hierarchy is semantic, like the ones presented in our intro.

For the salaries column, with it being our goal to analyze, we opt to use the construction mentioned in our solution in the intro. We created 7 layers, in order to give the algorithm the ability to anonymize the dataset without the values being converted to '*'.

A sample of the result of the creation of the Salaries hierarchy is presented in **Table 3.5**, while the whole file is available in the GitHub distribution of the results of this Thesis.

Next up, we are going to setup the use the ARX API, which requires us to specify some variables in order to run Differential Privacy. Those variables are defined in the above Java code, and are those that were use in the actual testings.

Table 3.5: Hierarchy Levels created

1^{st} level	2^{nd} level	3^{rd} level	...	7^{th} level
79.568	291.029	500.776	...	*
502.491	291.029	500.776	...	*
522.738	710.524	500.776	...	*
898.310	710.524	500.776	...	*
1.000.000	1.114.013	1.220.739	...	*
1.228.026	1.114.013	1.220.739	...	*

```
EDDifferentialPrivacy criterion = new EDDifferentialPrivacy(2d, 1d / Rows);

ARXConfiguration config = ARXConfiguration.create();
config.addPrivacyModel(criterion);
config.setSuppressionLimit(1d);
config.setHeuristicSearchStepLimit(100);
ARXResult result = anonymizer.anonymize(data, config);
```

The basic principles that were followed for the above definitions are based on the following instructions and guidance by the ARX Tool documentations:

- The delta value should not be 0, but is suggested to be set lower or equal than the reciprocal of the number of records.
- A suppression limit should be set, preferably to 1.
- In order to improve the quality of the data produced, a heuristic search step limit should be set, in order to tweak the ARX search algorithm that handles data suppression.

Additionally, following the same principles as with the IBM library, we are going to run the D.P. query multiple times before reporting its value. We are going to do so, because the amount of noise generated can be extreme, and because of the low bounds of the heuristic search that we have set. We chose to run each query 1000 times, and then report the mean value of those runs as the result produced by the mechanism.

Because of the structure of the result of the ARX mechanism (a dataset containing numerical values), we can only run queries like *sum* and *mean*. There is no point in running a min or max query: we already know that the result will not be accurate. Thus, we are going to try to run a *mean value* numerical query in the anonymized dataset. The function we are using in order to run this typed of queries is the following:

```
protected static double run_query(ARXResult data, int targetColumn) {
        // iterator that we are going to use to access the data
        final Iterator<String[]> itHandle = data.getOutput().iterator();

        // result of the query
        double result = 0d;
        // length of the dataset
        int totalRecords = 0;

        // ignore the name of the column
        String[] name = itHandle.next();
        if (name.length <= targetColumn) {
                System.out.println("Target␣column␣out␣of␣bounds\n");
                return 0d;
        }

        // iterate through all the values in the dataset
        while(itHandle.hasNext()) {
                String[] next = itHandle.next();
                // check that our target position is legal
                String string = next[targetColumn];
                if (!string.equals("*")) {
                        result += Integer.parseInt(string);
                        totalRecords++;
                }
        }
    // return the __mean__ of the dataset
    return result / totalRecords;
}
```

Finally, before running the queries, we must mention that while using the ARX Tool the dataset size should be significant. In our case, it contains almost 13 contains thousands of rows. The dataset size is a critical parameter when applying D.P., while being even more essential during the use of the ARX tool.

3.3.6 Statistical Queries

As shown by the above Java function, our testings can support every type of statistical queries, such as *mean value*, *sum*, and *average*. However, due to the computer power required and the similarity of the results that those queries produce, we will focus our testings solely to the mean value query.

Given the dataset previously analyzed, the true mean value of the salaries column of the dataset is $\$2.868.981, 32$. This will be the value that we are going to use in order to examine the accuracy of the D.P. results.

3.3.6.1 Running with fixed parameters

We are going to conduct our first test by anonymizing our dataset using the default parameters, as we set ε $= 1$ and δ $= \frac{1}{12377}$. The results are shown in the following table.

We observe that the query results, are somewhat close: We are in the range of millions of dollars, and the ARX mechanism only fails to approach the result by 8 thousand. This is

Table 3.6: Mean value query in ARX with default parameters

Non-DP result	DP Result
$\$2.868.981, 32$	$\$2.860.215, 6$

not of course close to what the IBM library computed, but it is still a reliable result, given all the downsides of this type of anonymization.

3.3.6.2 Running with different epsilon values

Next up, in order to determine if ARX follows the rules of D.P., we should try anonymizing the dataset for different values of epsilon, just like we did with the IBM library.

We observed during our initial runs that if the epsilon value rises above 2, the algorithm faces certain problems, that will be analyzed moving forward. With that being the case, the epsilon values chosen to conduct the measurements are in the range $[0.2, 1.6]$. The results from our testings are shown in the above **Figure 3.8.**

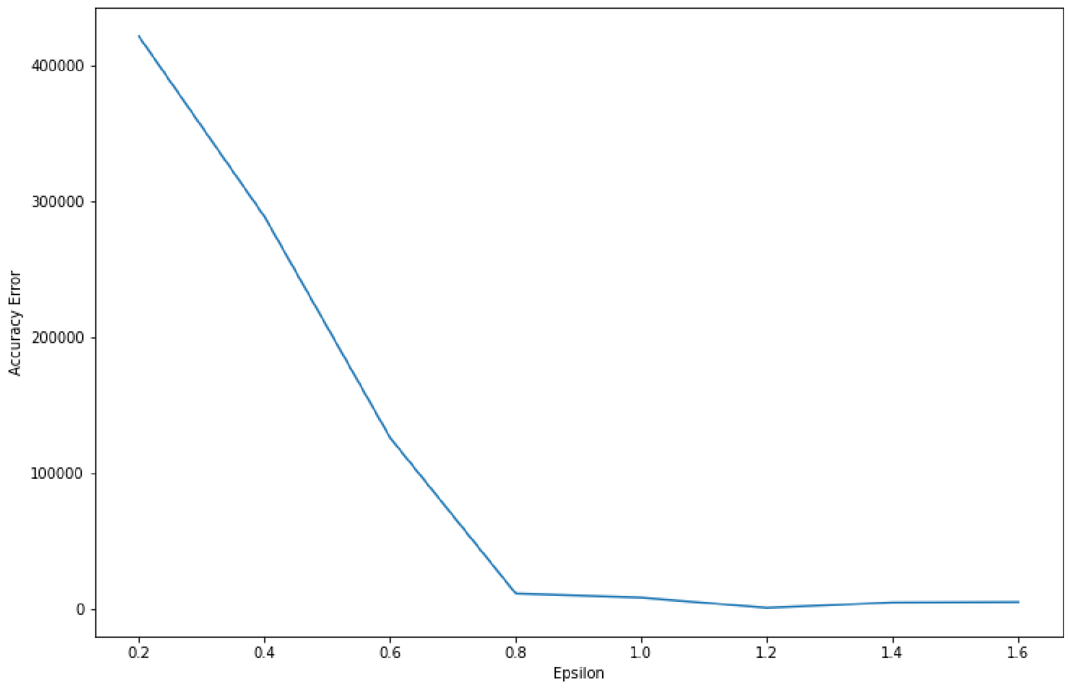


Figure 3.9: Accuracy Error results for increasing values of epsilon

As we can see, the plot produces a linear type curve when the epsilon is below 0.8, and then stabilizes, unlike the Laplace distribution mechanisms, where the error curve is in logarithmic shape. We can not directly compare the results with the Laplace noise distribution, due to the different datasets used.

We generate the answers given by asking the query in the output dataset. Without its records being suppressed, the result dataset would have been perfect, because of the transformation of the data. However, when suppressing many records (nearly 10% each time), the result could be severely altered, and thus the error plot, as we saw, is quite unpredictable.

3.3.7 Observations regarding the Algorithm

During our testings in the dataset using the ARX mechanism, we observe the following regarding its behavior in the DP queries:

- The epsilon variable if raised above 2,5, makes the algorithm *extremely slow*, to the point that it does not respond after minutes of execution. This makes sense, if we take into consideration that when epsilon increases, the accuracy gets better. Thus, the algorithm performs extreme searching techniques in order to find which records to suppress, resulting into slow execution.
- In order for the algorithm not to produce only *(the last level in our hierarchies) in our target column, we set each of the other columns as *non sensitive* in their definition.
- As the epsilon values rise, *the accuracy gets better*, as it is supposed to be, according to the DP principles.
- While the dataset has multiple columns, the algorithm usually fails to present all of them with anonymized values, and just reports * in each row. This could have been a result of the high *Heuristic Search Step Limit*, which was by default set to maximum. Despite us lowering its value, the phenomenon persists.

3.3.8 Conclusion

While researching the ARX mechanism we came to the conclusion that it is for sure a whole different approach in Differential Privacy compared to the other libraries that we studied. With that being the case, it has some advantages and some disadvantages. Its main advantages are the following:

- The result of the mechanism is a *handy dataset* that the user can handle in multiple ways and gain more information than just the result of a query.
- The result *can be iterated*, thus giving the option to the user to run the query in a smaller subset of the rows, while it being differential private.

On the other hand, the main disadvantages are:

- The result can be misleading, because of the *big accuracy error produced*.
- The algorithm *requires a rather big dataset* in order to run properly, while other libraries perform just fine with smaller datasets.
- The algorithm is difficult to implement, as you have to create a self-made function for every query, and moreover tune many parameters if you want to run differential privacy.

4. A LIBRARY FOR LOCAL DIFFERENTIAL PRIVACY

4.1 Introduction in Local DP

As we mentioned in previous chapters, there are two major forms of Differential Privacy. Having analyzed and tested the first one, *Global D.P.*, it is now time to examine *Local D.P.*, by explaining some possible protocols, as well as building our own.

In Local D.P., there is a significant difference compared to Global DP: there is *no trusted curator* between the data and the users, as they just want to send their data, while already being anonymized. Thus, an algorithm must perturb the data before sending it to the untrusted curator, who will then transmit it to the analysts.

In order to achieve that goal, the user must randomize the value before making it public (i.e. sending it to the untrusted curator). Then, the curator which collects the data (we will reference to him as aggregator moving forward), collects the data and tries to retrieve their original values, with a goal of producing the most accurate results possible.

Thus, each LDP algorithm has the following steps:

- Each user encodes, and then perturbs the private value that he wants to make public
- Each user sends out the result of the perturbation process, with that being only the final value, as they keep the intermediate results for themselves
- The untrusted data curator collects each user's value, and implements some kind of aggregation in order to retrieve the stats that he wants from the data given to him.

In comparison with Global D.P., the Local model has advantages, as well as disadvantages. Its main advantages are:

- The user is not forced to trust the data curator, as only the perturbed value is reported
- Simpler implementation of the algorithms, due to the district steps taken by both sides.

while the main disadvantages are the following:

- The noise added should be larger than the Global model, in order to satisfy the definition, thus the number of people in the dataset should be significant for accurate results to be produced.
- Because this is not always possible, many real-world applications use extremely high values of epsilon compared to what we got used to during our testing in the Global models.

During this Thesis, concern was raised for the main disadvantage of L.D.P., and thus *we will present a new protocol aiming to reduce the need for many users, while still covering the definition.* However, the definition for L.D.P. is quite different than the Global model one's.

4.2 Definition of Local DP

Having a general idea in how Local D.P. functions, it is now time to give a strict definition that we are going to depend our work on moving forward.

We can say that an algorithm A satisfies ε-Local Differential Privacy, if and only if for any input v_1, v_2, we have

$$\forall y \in Range(A): \; Pr[A(v_1) = y] \leq e^{\epsilon} * Pr[A(v_2) = y]$$

where $Range(A)$ denotes the set of all possible outputs of the algorithm A.

As mentioned in Chapter 2, this definition can have many interpretations by different algorithms or protocols, but each one must produce a probabilistic space whose elements must satisfy the above equation.

4.3 Simple Application of LDP

The most simple of L.D.P. protocols is already mentioned in this Thesis, and is no other than the *Randomized Response* protocol. This algorithm implements the three steps mentioned in the introduction, as the user chooses a value (Yes or No), perturbs it (by the flipping of the coins), reports the perturbed value, with the sole job of the aggregator being to collect, normalize and report the values provided. It meets the definition of L.D.P., as the fraction of a pair of probabilities in the space of possible outputs (Yes, No) has always the ceiling of a real number.

Our goal is to now find this ceiling, and thus denote the level of privacy that randomized response offers. In order to do this, we are going to select the possibility of the user having chose the answer "Yes". A simple case analysis shows that $Pr[Yes|Truth] = \frac{3}{4}$, and of course $Pr[Yes|False] = \frac{1}{4}$. Thus, by the definition of L.D.P., we have

$$\frac{Pr[Yes|Truth]}{Pr[Yes|False]} = \frac{\frac{3}{4}}{\frac{1}{4}} = 3 = e^{\epsilon} \iff \epsilon = ln(3)$$

Thus, R.R. offers $ln(3)$-differential privacy to its users. This is quite a good setting, but the restriction is that the user can only report 2 values, something not suitable for modern problems and surveys.

In R.R., we care about the total true answers of the users, and not the individual responses. Thus the metric we are going to use is the *absolute difference of the sum of the 2 vectors: the one with the truthful answers, and the one with the reported answers*. We are going to divide this result with the number of the users, in order to get the scale of the error depending on the size of the vector that was reported. The metric is expressed from the following function:

$$\text{Error} = \frac{|\sum \text{true\_values} - \sum \text{reported\_values}|}{\text{number of users}}$$

As always, during the creation of probabilistic distributions, one run is not enough, because of the extreme amount of noise that can occur. Thus, for each number of users we are going to run the R.R. protocol 100 times, and the final accuracy error will be produced by the mean value of those runs.

Having implemented R.R. in Python, we can now display the accuracy error of R.R. as the number of users rises. The results of the testings are shown bellow in **Figure 4.1**.

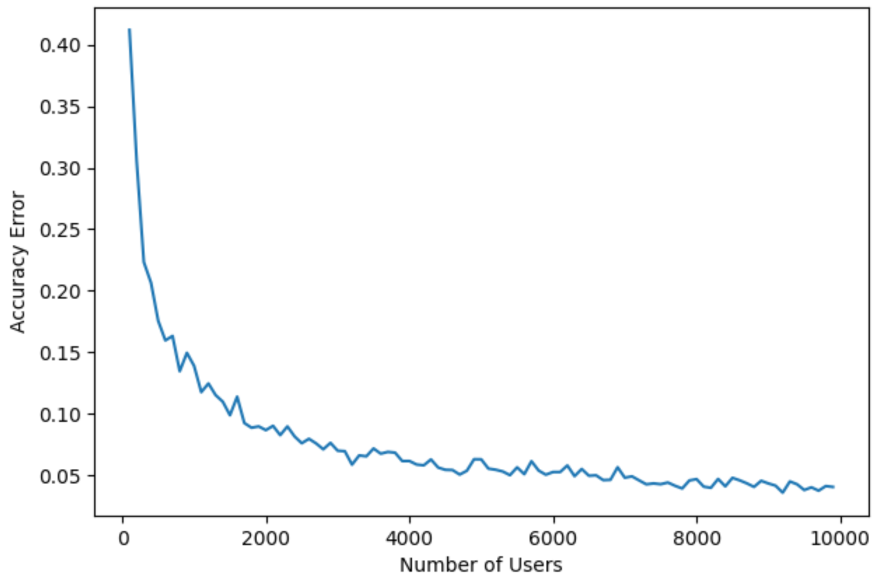


Figure 4.1: Accuracy Error in R.R for increasing values of epsilon

We observe that the plot behaves as expected: the protocol produces a logarithmic curve for the accuracy error, while for a large number of users (over 3000), the error stabilizes bellow 0.1.

4.4 Existing Protocols for Local DP

Apart from R.R., several L.D.P. protocols have been implemented during the years, with many of them being widely used by companies in order to protect users' data. One of the most famous protocols is *RAPPOR*([15]), created by Google, and being currently used in the Chrome browser for the company to provide useful info to its users without compromising their privacy. Also, Apple has created ts own protocol of L.D.P., and utilize it in their products.

However, we are not going to focus on those protocols moving forward, than the ones presented in [10], a paper which introduces many algorithms for L.D.P., each one with different perturbation techniques and suitable for different circumstances.

During this chapter we are going to give a definition of each algorithm, implement it using Python, and compare the accuracy results produced by those protocols, just like during our testings of the G.D.P. models. Each protocol has two parts: the *users* and the *aggregator*. For the users we must each time define the following functions:

- $Encode()$: Encodes the true value that the user wants to report
- $Perturb()$: Perturbs the encoded value, in order to produce the random value that will be reported

For the aggregator we must each time define the $Aggregate()$ function, that collects the reported random values of the users, and produces the results according to the model.

4.4.1 Basic RAPPOR

As mentioned earlier, RAPPOR is a protocol created by Google. Its simpler form, Basic RAPPOR is used in Chrome, where it collects answers to questions such as the user's home page. The protocol's functions are the following:

Encoding: $Encode(v) = A_0$, where A_0 is a d-bit vector, such that: $A_0[v] = 1$ and $A_0[i] = 0$ for every $i \neq v$.

Perturbation: The perturbation consists of 2 steps: the permanent and the instantaneous. The permanent one is carried out only one time, and is the following:

$$Pr[A_1[i] = 1] = \begin{cases} 1 - \frac{1}{2}f & \text{if } A_0[i] = 1 \\ \frac{1}{2}f & \text{otherwise} \end{cases}$$

The instantaneous step is carried out every time a user reports a value, and is defined as:

$$Pr[A_2[i] = 1] = \begin{cases} p & \text{if } A_1[i] = 1 \\ q & \text{otherwise} \end{cases}$$

We observe from the above functions, that the user must define the f, p and q parameters. Google suggests that we set $f = \frac{1}{2}$ or $\frac{1}{4}$, and $p = 0.75$, thus $q = 0.25$. During our testings, those exact parameters were used.

4.4.2 Random Matrix Projection

In [14], a protocol with a random matrix projection is proposed, introducing an additional setup step.

Setup: A random and uniform matrix is generated before any encoding, with it being public and drawn as: $\Phi \in \{-\frac{1}{m}, \frac{1}{m}\}^{m \times d}$, where m and d are user defined. In our testings, we opt to set $m = 5$ and $d = 10$.

Encoding: When it comes down to encoding, the function used is the following: $Encoding = (r, x)$, where r is uniformly randomly selected from the range of m, and x is the v-th element of the r-row of the random matrix.

Perturbation: The perturbation function is defined as following:

$$Perturb(r, x) = (r, b \cdot c \cdot m \cdot x)$$

where

$$b = \begin{cases} 1 & \text{with } p = \frac{e^\epsilon}{e^\epsilon + 1} \\ -1 & \text{with } q = \frac{1}{e^\epsilon + 1} \end{cases}$$

and $c = \frac{e^\epsilon + 1}{e^\epsilon - 1}$

Aggregation: Given all the tuples reported by j users in the form (r, y), the estimation for the i-th value of the dataset, is produced by

$$\sum_j y^j \cdot \Phi[r^j, i]$$

4.4.3 Pure Protocols

The following protocols are presented in [10], and are called ”pure” protocols, because of the way they aggregate the data produced by the user. For each one of them, we should define a $Support()$ function, that indicates for each value of the possible outcomes, the reported values that are supported. Thus, with the notation $\sum_j Support(y^j)$, we mean the sum of all the supported values of the y-th element of the dataset.

Also, for a protocol to be pure, two probabilities must be defined, p^* and q^*, where the first notes the probability that the true value is supported by an element y, and the second one the probability of another value is supported by the element y. The protocol is pure if and only if $p^* > q^*$.

If a protocol is pure, the estimation of the total reported values for an element of the dataset i, is the following:

$$\text{Estimation} = \frac{\sum_j 1_{support(y^j)}(i) - nq*}{p^* - q^*} \tag{4.1}$$

where j denotes the j-th user reporting their value, and n the total size of the vector of the reported values.

4.4.3.1 Direct Encoding

This protocol is the natural method of extending the Randomized Response, without the limitation of 2 possible answers.

Encoding: The protocol does not feature an encoding procedure, thus

$$Encode(v) = v$$

Perturbation: The perturbation is based on the epsilon setting given by the user, and its function is defined as following:

$$Pr[Perturb(x) = i] = \begin{cases} p = \frac{e^{\epsilon}}{e^{\epsilon}+d-1} & \text{if } i = x \\ q = \frac{1}{e^{\epsilon}+d-1} & \text{if } i \neq x \end{cases}$$

where d the size of the dataset of the possible answers, x the true value and i the value selected.

Aggregation: The protocol is pure with $p^* = p$, $q^* = q$ and $Support(i) = i$, thus the predicted results for each of the dataset's values can be calculated from the Equation 4.1.

We observe that this protocol strongly depends on the size of the dataset of the possible answers, thus when the dataset size increases, the protocol becomes less accurate, due to the decreased probability of selecting the truth. Moreover, for the D.E. protocol, all the false values have the same probability to get chosen, a rather disturbing detail for a query such as a person's age. We will return to these thoughts on later sections.

4.4.3.2 Histogram Encoding

An other protocol presented is Histogram Encoding, where an input when having d options is encoded as a d-length vector.

Encoding: The encoding function is for the protocol is

$$Encoding(v) = [0, 0, \ldots, 1, \ldots, 0]$$

where only the v-th element of the vector is equal to 1.

Perturbation: The result of perturbing the encoded vector, is a new vector B', s.t.:

$$B'[i] = B[i] + Lap(\frac{2}{\epsilon})$$

where $Lap()$ denotes the noise drawn from the Laplace distribution, where

$$Pr[Lap(\beta) = x] = \frac{1}{2\beta} e^{\frac{-|x|}{\beta}}$$

Aggregation: Several methods are proposed for aggregating the results created by the H.E. protocol, but as mentioned by the authors, the best one is called *Thresholding with H.E.*, where a threshold value is introduced in order to decide what to keep from the reported values. The support function is altered as following:

$$Support(B) = \{v|B[v] > \theta\}$$

thus, if a noisy output is grater than theta, is set to support the corresponding value. According to the authors, the optimal value for θ is in the range of $(\frac{1}{2}, 1)$. During the testings that are going to be conducted, we are going to use a threshold of $\frac{2}{3}$.

Comparing this protocol to D.E., we observe that is solves the problem of the dependence of the noise drawn by the number of options to choose from. In H.E., no matter how large the domain size is, the noise solely depends on the epsilon value chosen by the user. Thus, when having a large domain size, it is clear that we should prefer the H.E. protocol over D.E.

4.4.4 Unary Encoding

The last protocol that is going to take part in the accuracy testings, is the Unary Encoding method, a further exploration of the Basic RAPPOR. It is a unique protocol, as the user does not set the level of privacy using epsilon, but by giving two probabilities, p and q and the epsilon value is computed using those two parameters.

Encoding: Exactly like in the H.E. method:

$$Encoding(v) = [0, 0, \ldots, 1, \ldots, 0]$$

where only the v-th element of the vector is equal to 1.

Perturbation: This step is different than those that we already saw, and is carried out using the following function:

$$Pr[B'[i] = i] = \begin{cases} p & \text{if } B[i] = 1 \\ q & \text{if } B[i] = 0 \end{cases}$$

The epsilon value is decided given p and q, and is defined as following:

$$\epsilon = ln(\frac{p \cdot (1-q)}{(1-p) \cdot q})$$

Aggregation: The Support function is once again altered, as in the U.E. protocol is defined as following:

$$Support(B) = \{i|B[i] = 1\}$$

and of course, $p^* = p$ and $q^* = q$, in order to make the protocol pure. As for the choice of p and q, we opt to choose $p = \frac{1}{2}$, and $q = \frac{1}{e^{\epsilon}+1}$.

4.5 Testings

4.5.1 Setup

Now that all those protocols where introduced, we are going to compare them in order to decide which is better to use when wanting to apply L.D.P. in a dataset. We are going to use a dataset that was created using random values, but corresponds to the age of a group of people. The distribution of the values of the dataset is shown in the histogram in **Figure 4.2.**

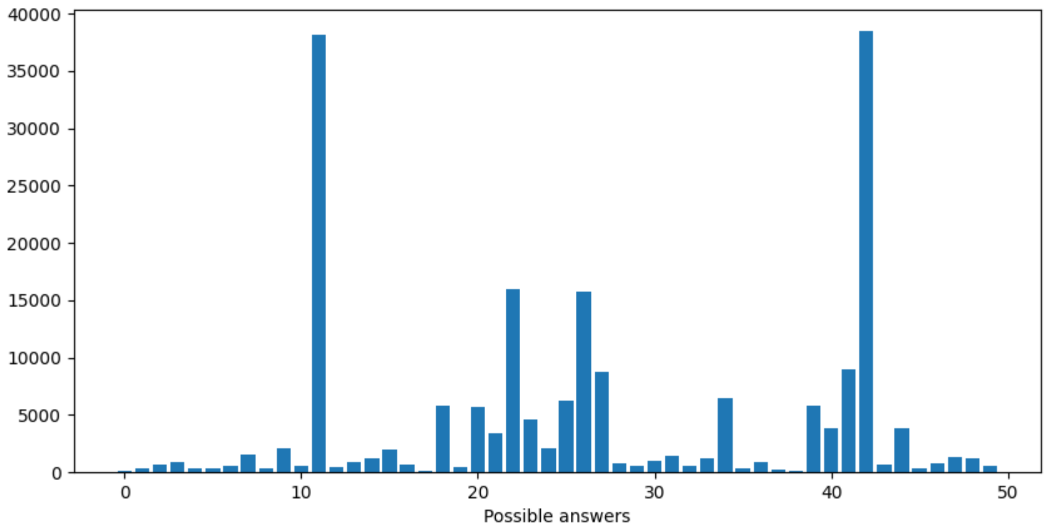


Figure 4.2: True Answers for the Dataset of LDP

Each user will report one of these 50 values, and the aggregator of each protocol will gather the data given, and try to re-create this histogram in the best manner possible.

4.5.2 Goal

We want to decide which protocols behaves better, thus a number of different metrics will be used. The main focus of our testings will be the vectors that the aggregators provide, which we will compare with each other, as well as with the vector containing the true answers. In a similar way as our G.D.P. testings, we are going to run the protocols for different values of epsilon, and different number of users used. The second one is extremely important in L.D.P., as we mentioned earlier that many protocols struggle with a small number of input, as the noise drawn is significant.

With respect to the choice of metrics, we are going to use the *Manhattan Distance*, known as the $l1$-norm, as well as the *Kantorovic Distance*, explained in 3.2.5.1.

4.5.3 Epsilon Measurements

The first comparison between the protocols will be with a changing epsilon value, in order to observe how they behave for lower and higher values of the privacy setting. During these testings, all of the users of the dataset were used (approximately 20 thousand), and each run of each protocol was carried out 10 times, just like in other testings, in order to eliminate the danger of drawing extreme values of noise.

First, we are going to run all the protocols and compare them using the Manhattan Distance. The results are shown in **Figure 4.3.**

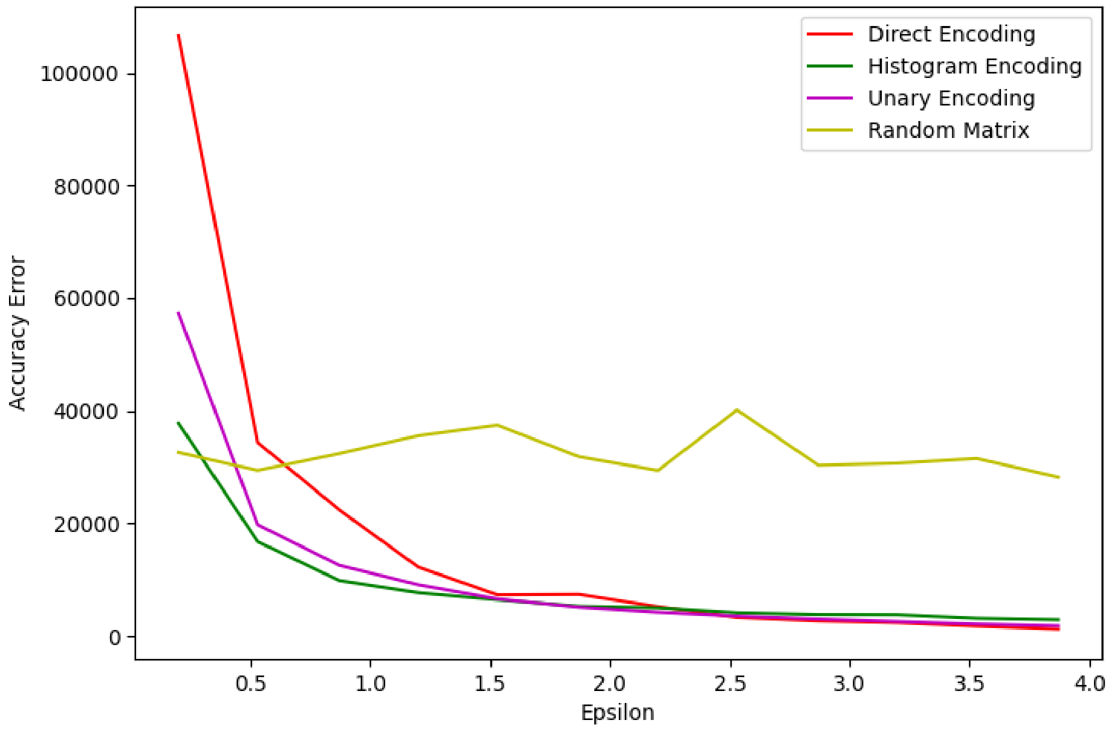


Figure 4.3: Epsilon Measurements compared by Manhattan Distance

We gather many useful observations from the graph:

- The Random Matrix protocol does not function as expected, as *its accuracy does not follow the logarithmic curve we are used to when epsilon increases.* However, for small values of ε, its results are acceptable, and some times even better than the pure protocols.
- *The pure protocols behave in a similar way*, with the error stabilizing when epsilon gets higher than 2.5.
- *The Direct Encoding protocol was the worst behaviour among the pure ones*, with its error being extremely high for epsilon values lower than 1. This is mainly due to the fact that when ε gets too small, the probability of telling the truth gets significantly low, thus creating a big error in accuracy.
- *The optimized U.E. protocol has the best behaviour* in comparison to the other protocols tested.

Next up, we are going to run the same testings, but this time using the Kantorovich metric. We expect the protocols to behave even worse, because of the identity of the metric: the Kant. metric pays attention to the distance of the reported answer from the true one. The

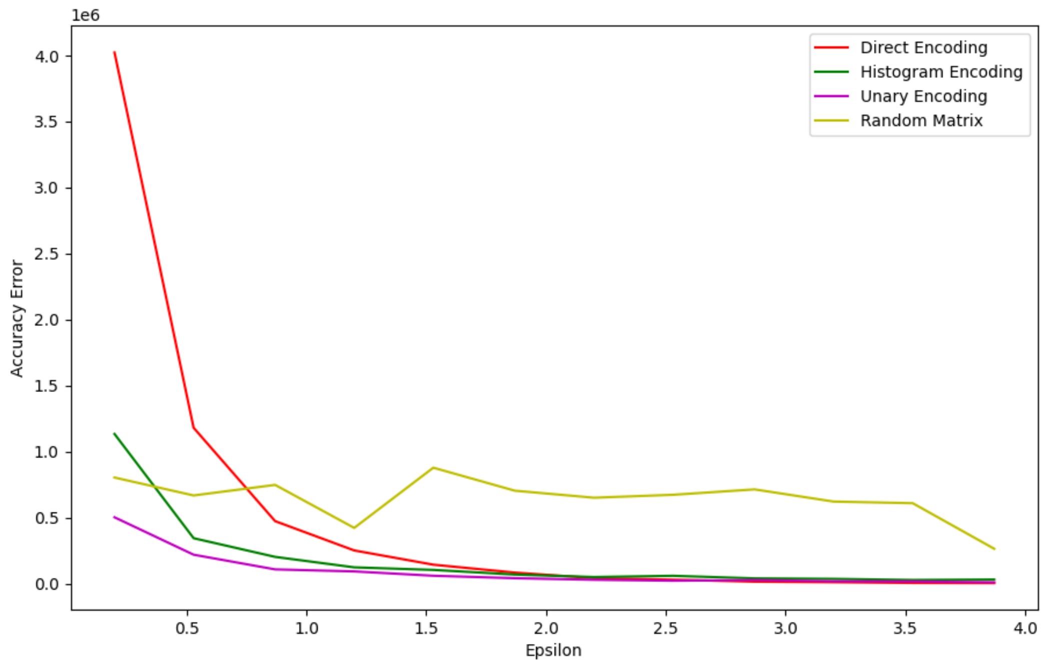


Figure 4.4: Epsilon Measurements compared by Kantorovic Distance

current protocols do not take into account the distance of the two answers, thus the metric will probably report a higher error. The results of the runs are shown in **Figure 4.4.**

As we expected, the protocols produce a higher error, with the D.E. being the worst among them, especially for lower values of epsilon. This is a rather alarming notice, in which we will come back in Section 4.6.

4.5.4 Increasing number of users

The second experiment that we will conduct is the accuracy error depending on the number of users used during the survey covered by the protocol. In the definition of L.D.P. the observation of the need of lots of users was made, and it is now time that we examine it. We are going to use a *fixed epsilon value*, one that our protocols behave similarly for (at least the pure ones, in which we will focus our research moving forward). Our epsilon value that we are going ot used will be fixed and equal to 1.5.

We are going to run the protocols and compare them using the Manhattan Distance. Additionally, we are each time going to divide the result of the metric with the number of users participated, as the simple error is going to increase when the users increase. Hence, this division is going to give us the error depending on the size of our domain. The results are shown in **Figure 4.5.**

The results confirm the allegations made after explaining the definition of L.D.P.: When the number of participants in a survey is low, the error produced is very high. Every protocol has similar behaviour, as we can see that for fewer than 1000 users the relative error is even 6 times larger than for more than 1000 users. Actually, as we can see from the graph, the turning point is around 2000 users: the relative error drops and stabilizes after this number of participants.

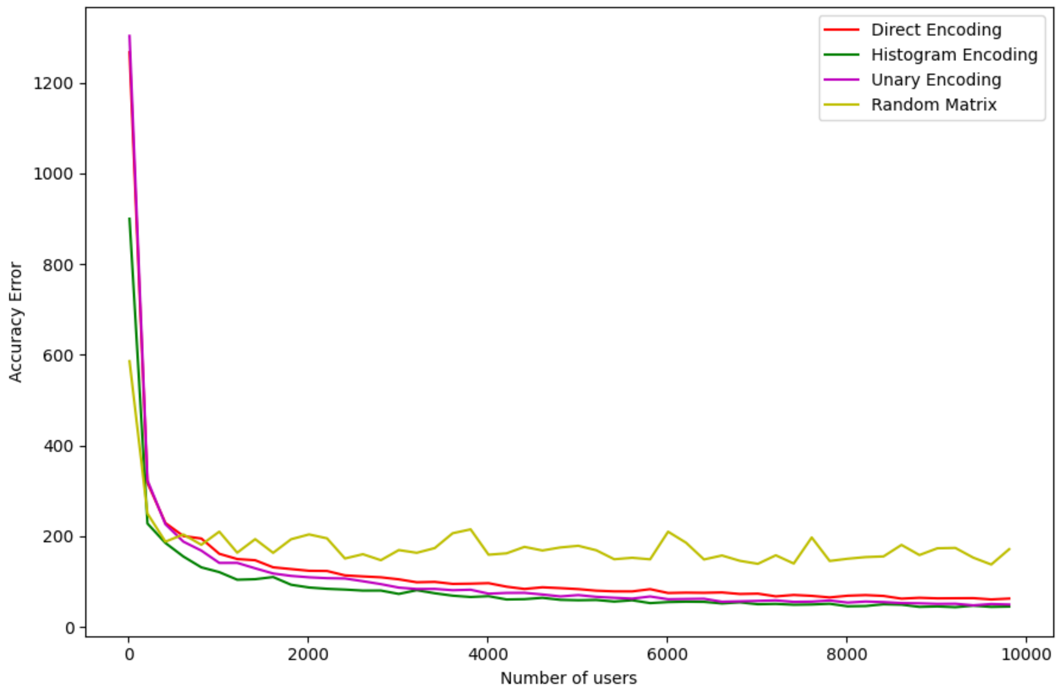


Figure 4.5: Increasing n. of users compared by Manhattan Distance

An other observation made, is that the protocol with the best behaviour during our epsilon measurements (Unary Encoding), is the worst one in the low users testings. This of course gets better as the number of participants rises, and eventually beats the other protocols when it comes down to relative accuracy error.

Next up, we are going to perform the same testings while comparing the protocols with the Kantorovich metric. The results are shown in **Figure 4.6**.

The results for a low number of users differ a lot from the runs with the $l1$ distance as a metric. Here, the D.E. method has obviously the worst behaviour, and the U.E. method the best. We can even claim that U.E. seems to solve our problem of high accuracy error. However, no one of those protocols takes into account the distance of the reported answer from the true one, thus it makes sense that the Kantorovich metric produces a huge amount of accuracy error.

That fact, triggered our thoughts, on what could possibly be done in order to reduce that problem. The thoughts made on this subject are analyzed in the next chapter, by creating a new L.D.P. protocol, sensitive to the distance between the true and the reported values.

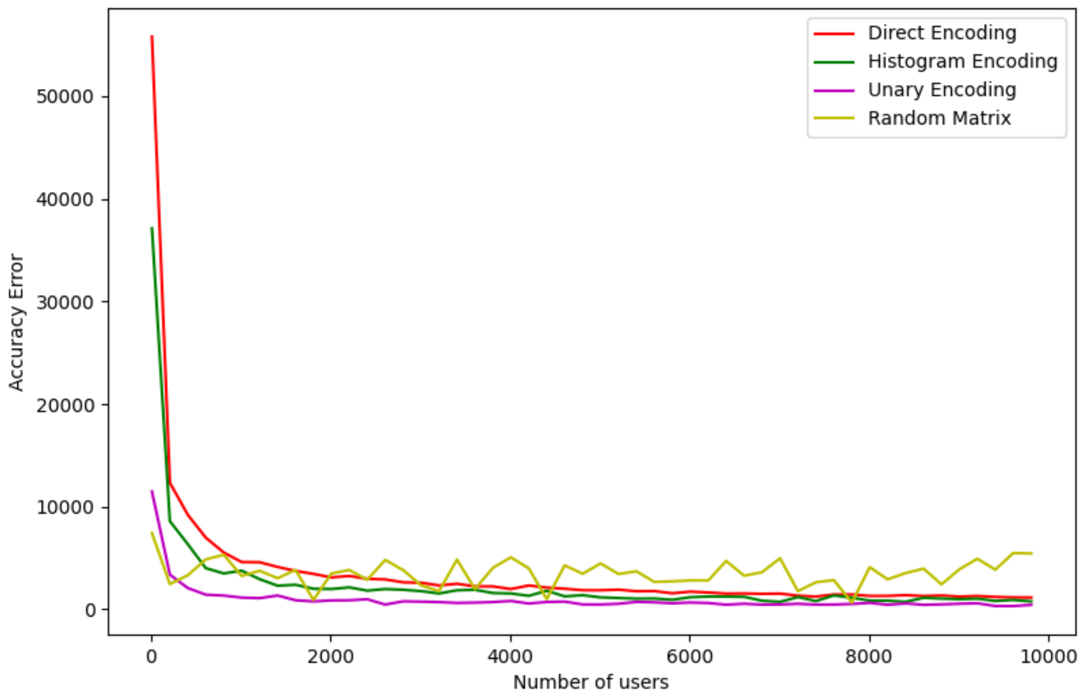


Figure 4.6: Increasing n. of users compared by Kantorovich Distance

4.6 A Distance Sensitive Encoding Protocol

4.6.1 Idea

In the previous section we discussed various LDP Protocols that function extremely well for numerical values (histogram type), with accuracy that is completely acceptable. However, when the number of users is limited, (eg under 1000), we made the observation that the accuracy error is extremely large. This is mainly due to the fact that the probability of an item to be chosen is independent of the distance between the true and the selected value.

Thus, for the needs of this Thesis, a new L.D.P. protocol was constructed. The idea proposed is to *have the probability of choosing an element of the domain to depend on the distance from the true value*. This could prove very helpful for histogram values, but does not make any sense for categorical values. From now on, we are going to focus on histogram values.

Based on our idea, the probabilities' distribution, in comparison with the distribution of the D.E. protocol, will look like the one in the following figures.

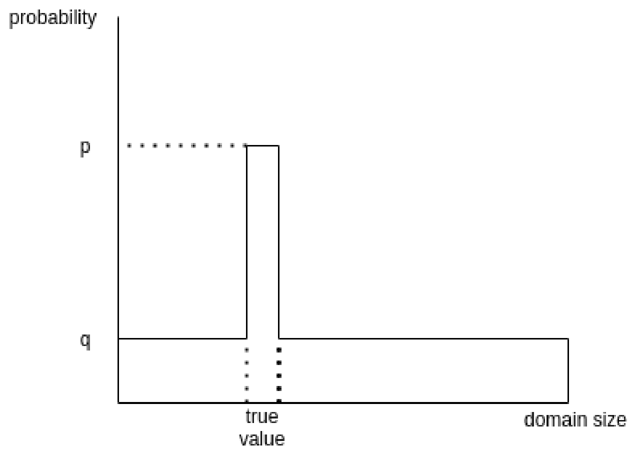


Figure 4.7: D.E. protocol's Probabilistic Distribution

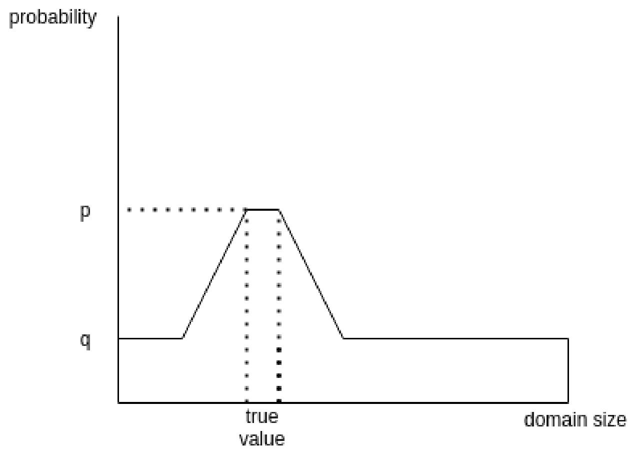


Figure 4.8: Distance Sensitive protocol's Probabilistic Distribution

Thus, there is an area around the true value that has high probability to be selected. The width of this area(from now on θ) is defined by the epsilon setting that the user wants to use. The idea for having a specific area and not decreasing our probabilities as low as it goes when we diverge from the true value, is based on our need to be able to serve low epsilon values, as the first would be a big no for when the users want to use a high privacy setting.

4.6.2 Mathematical Background

In order to achieve our goal, when in the selected area, we should include in the denominator the quantity

$$|x - i|$$

where x is the true value, and the i the false one that we are looking at in order to report it, in order for the probability to depend on the distance of the reported values.

When we are out of this area, the denominator will have a constant value, proportional to the boundaries of the selected area.

Example: Let's suppose that we have a domain size of 100, and our θ value is 4. All the probabilities outside the area, should be in reverse proportion of θ, thus 4.

Like all probabilistic algorithms though, the sum of the probabilities for all the items in the domain size, must be 1. Thus if we chose the probability to be in the shape of $\frac{a}{|x-i|}$, the partial sum of the series will not give as simple results.

It is known that the series in the form of $a_n = \frac{1}{n}$ does not converge, and its partial sums can only be computed using a complex approximation formula. These characteristics make this type of series very hard to use, and so we have to think of a more handy type.

A type of series that is known to have easy to compute partial sums, are the telescopic series, such as $b_n = \frac{1}{n(n+1)}$. It is known that

$$\sum_{n=1}^{n=k} b_n = 1 - \frac{1}{k+1}$$

something that will prove extremely useful moving forward.

So, taking into consideration the quantity $|x - i|$ and the telescopic series b_n, we conclude that the probability of each non-true element in our selected area will be of the shape:

$$\mathsf{q} = \frac{\mathsf{a}}{|\mathsf{x} - \mathsf{i}|(|\mathsf{x} - \mathsf{i}| + 1)} \tag{4.2}$$

and outside of that area:

$$\mathsf{s} = \frac{\mathsf{a}}{\theta \cdot (\theta + 1)} \tag{4.3}$$

The probability p of selecting the true value will have to meet specific criteria that we are going to define later on.

4.6.3 Building the Protocol

Ww are now going to find what the alpha parameter will be, as it is not constant, but clearly it depends on the domain size and the probability p. In order to find it, we must keep in mind that all the probabilities of selecting an item from our domain, must add up to 1.

In order to find out the α value, we must solve the following equation:

$$p + \sum_{i=x-\theta}^{i=x+\theta} q + \sum_{i=1}^{i=x-\theta-1} s + \sum_{i=x+\theta+1}^{i=d} s = 1$$

At this point, we must note that α although not a constant, can be held out of the sums, because it is obviously independent from the i variable, that is the variable parsing through the domain in order to retrieve the false elements' probabilities. Thus, we have:

$$\begin{aligned} p + \sum_{i=x-\theta}^{i=x+\theta} q + \sum_{i=1}^{i=x-\theta-1} s + \sum_{i=x+\theta+1}^{i=d} s = 1 \Longleftrightarrow \\ \sum_{i=x-\theta}^{i=x+\theta} \frac{a}{|x-i|(|x-i|+1)} + \sum_{i=1}^{i=x-\theta-1} \frac{a}{\theta(\theta+1)} + \sum_{i=x+\theta+1}^{i=d} \frac{a}{\theta(\theta+1)} = 1 - p \Longleftrightarrow \\ \cdots \Longleftrightarrow \\ a = \frac{\theta(\theta+1)(1-p)}{2\theta^2 - 2\theta + d - 1} \end{aligned}$$

The proof for the mathematical equations leading to the extraction of the alpha value, can be found in the First Appendix of the Thesis.

4.6.4 Epsilon Requirements

The epsilon value is the most essential in these protocols, as it determines the privacy level that the protocol yields.

Recalling the definition of LDP, we must follow the following rule:

An algorithm A satisfies -LDP *iff* for any input v_1 and v_2, we have

$$\forall y \in Range(A) : \frac{Pr[A(v_1) = y]}{Pr[A(v_2) = y]} \leq e^{\epsilon}$$

Thus, in order to determine the epsilon value for our algorithm, it must satisfy even the worst case of this equation. The fraction gets bigger, if we put on the biggest probability on the numerator, and the smallest probability of all in the denominator.

The numerator must have the probability p, and the denominator s, the probability of all of the elements outside of the θ area.

We want for p to be the biggest probability among all, but not extremely high, in order to be able to restrict the growth of epsilon. Hence, we are going to set it as double of the probabilities of its exact neighbours. The 2 neighbours have $|i - x| = 1$, thus $q = \frac{a}{2}$, so we are going to set p $= 2 \cdot \frac{a}{2} =$ a, where a is the quantity defined above, depending on the domain size, and the θ value.

Now, if we set $p = a$, then the a equation changes, and now our aplha parameter only depends on the domain size and the θ selected. So, we proceed as following:

$$\begin{aligned} a = \frac{\theta(\theta+1)(1-p)}{2\theta^2 - 2\theta + d - 1} &\iff (p = a) \\ a = \frac{\theta(\theta+1)(1-a)}{2\theta^2 - 2\theta + d - 1} &\iff \\ a(2\theta^2 - 2\theta + d - 1) = \theta(\theta+1) - (\theta^2 + \theta)a &\iff \\ a(3\theta^2 - \theta + d - 1) = \theta(\theta+1) &\iff \\ \mathsf{a} = \frac{\theta(\theta+1)}{3\theta^2 - \theta + \mathsf{d} - 1} & \end{aligned}$$

Obviously, we observe the constraint that $\theta > 0$, and this is a special case of our protocol, that can be represented by the direct encoding protocol.

We now have:

$$\begin{aligned} \frac{Pr[A(v_1) = y]}{Pr[A(v_2) = y]} = e \iff qe^\epsilon = p &\iff \\ \frac{a}{\theta(\theta+1)} \cdot e^\epsilon = a &\iff \\ \theta(\theta+1) = e^\epsilon &\implies \\ \theta^2 + \theta - e^\epsilon = 0 & \end{aligned}$$

If we solve the quadratic equation, and reject its one (illegal) solution, we get that:

$$\theta = \lfloor \frac{\sqrt{4e^\epsilon + 1} - 1}{2} \rfloor \tag{4.4}$$

So, to conclude, in order for the protocol to function, the user must provide just the epsilon setting, from which the θ constant is computed, and so the probabilities for each item of the domain will be selected.

4.6.5 Protocol Definition

We are now ready to define our protocol, by determining the 3 basic operations for an LDP protocol, the *encoding*, the *perturbation* and the *aggregation methods*.

We are going to use the following symbols:

- D: The protocol's domain. In this set, we have each i for $1 \leq i \leq |D|$ as each item in the domain D
- a: The quantity that we computed in the previous section
- θ: The constant used in the previous section, which denotes the area around the true value that the probabilities will be higher than others.

Encoding: The encoding procedure is trivial. Just like the Wang paper, we are just going to set:

$$Encode(v) = v$$

for each value v of the domain. The values are going to be randomized during the perturbation step.

Perturbation: Given the previous section, the randomization during the perturbation step is define as following:

$$Pr[Perturb(x) = i] = \begin{cases} p = a & \text{if } i = x \\ q = \frac{a}{|c|(|c|+1)} & c = \min(\theta, |i - x|), \text{ otherwise} \end{cases}$$

where i is the value selected each time, and x our initial selection.

Aggregation: The aggregation step was the most tricky during the building of the protocol. A similar approach to the aggregation of pure protocols was chosen, but with a few changes. After several different tries, the optimal aggregation found, was the following: the protocol supports only the reported values corresponding to the true one, thus $Support(v) = v$. However, the p^* quantity is the sum of all the probabilities inside the area:

$$p^* = \sum_{x \in (-\theta, \theta)} p(x)$$

Finally, the $q*$ quantity is the probability of choosing an element from outside the θ area, thus equal to s. Hence, the estimation generated for a value v of the possible answers in the domain is defined as following:

$$\text{Estimation} = \frac{\sum_j 1_{support(v^j)}(i) - nq^*}{p^* - q^*}$$

4.6.6 Extreme Cases

The downside of a complicated protocol, are of course some extreme cases for the x, θ and i values, all of which we are going to examine in this chapter. The definition of the protocol is going to be altered, and the constraints increased, in order to support those extreme cases.

Extreme theta cases: Of course, we have the constraint that $0 < \theta \leq d$, but what happens when its value is equal to one of the bounds?

- When $\theta \leq 0$, our protocol can not function, as this assignment will result in $a = 0$, something that is prohibited, because the probabilities will not sum to 1. In order to ensure that θ is at least 1, the user must provide at least an $\epsilon = ln(2)$.
- When $\theta = 1$, we can see that the third case in the perturbation step does not exist, thus we have only the first 2 cases. There, $p = a = \frac{2}{d+1}$, and for every other i, $q = \frac{a}{2}$, something that is similar with the Direct Encoding protocol.

- When $\theta = d$, which realistically can only happen when d is extremely small, then our protocol functions as designed, and has its best behavior. However, if the selection of epsilon results in such big a theta, then the user does not have extreme privacy demands.

Extreme x values: Even when the epsilon value and the domain size are normal, in some cases we might face a certain difficulty: if $x - \theta < 0$ or $x + \theta > d$, some of the items in our area are actually outside of our domain boundaries. This results in the sum of the items in the probabilistic distribution to be below 1, something not acceptable.

In order to fix it, we are going to "transfer" those probabilities inside the boundaries of our domain, while not messing with the highest probability, as this would result in problems with the definition of D.P., an thus the value of theta.

The idea is to increase the other selections' probabilities by a bit, in order to fill the gap created, while leaving the maximum as initially created. We are going to boost all of the domain's items, by a portion of $\frac{m}{d-1}$ (as we are altering $d-1$ elements), where m is the sum of the probabilities of the items outside the bounds of our domain.

However, we are not interested in transferring the whole $Pr[Perturb(x) = i]$, but only its difference from the item with the lowest probability, which is $s = \frac{a}{\theta(\theta+1)}$. Thus, the m values are defined as following:

$$m = \sum_{i<0 \bigcup |i-x|<\theta} Pr[Perturb(x) = i] - \frac{a}{\theta(\theta+1)}$$

for the case of $x - \theta < 0$, and as

$$m = \sum_{i\geq d \bigcup |i-x|<\theta} Pr[Perturb(x) = i] - \frac{a}{\theta(\theta+1)}$$

for the second one.
Now, the probabilistic distribution can be altered as following:

$$Pr[Perturb_{DS}(x) = i] = \begin{cases} p = a & \text{if } i = x \\ q = \frac{a}{|c|(|c|+1)} + \frac{m}{d-1} & c = \min(\theta, |i-x|)\text{, otherwise} \end{cases}$$

The definition of D.P. is not altered, because again in the best case we have a probability of $p = a$, and in the worst case $s = \frac{a}{\theta(\theta+1)}$.

4.6.7 Implementation

The most difficult part of the implementation of our protocol consists of creating the probabilistic distribution for each element of the domain, depending on the true value. This can prove to be costly, if we have a large domain or if we are in the case of the extreme x values.

However, we do not need to compute every single probability, as it is clear from the definition that they are independent from the true value: they only depend on a (and on the domain size in case of an extreme x value). The quantity $|i - x|$ can only take values in the range of $[1, \theta]$, thus constant for every possible true value. Moreover, for the domain values outside of the area, the probability is fixed and equal to $\frac{a}{\theta(\theta+1)}$. Hence, the probabilities can be computed in advance, either by each user, or given to the protocol by the aggregator.

The protocol has been implemented using Python, and can be found in the GitHub repository of this Thesis. Moving forward, we are going to use this implementation in order to conduct some testings to ensure the protocol's functionality.

4.6.8 Experiments

First up, we are going to perform the epsilon measurements that we did for the other protocols, this time excluding the Random Matrix approach, and including the D.S. protocol. The results, when running with the Kantorovich metric, are the following:

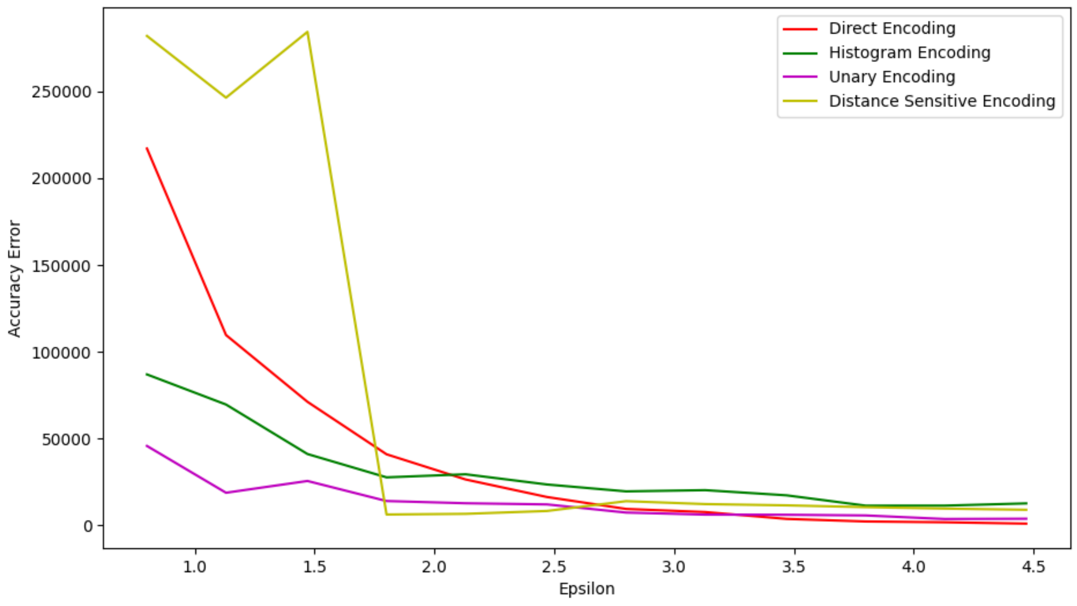


Figure 4.9: Epsilon measurements for D.S. protocol compared by Kantorovich Distance

The first observation is the *strange form of the curve of our protocol*. This can be easily explained: the theta value used to determine the area around the true value, is depended on epsilon, but has a floor function applied to it. Thus, for a specific range of ε, the protocol produces the same results.

An other observation is that *our protocol lacks efficiency for low values of epsilon*, that is natural, since small θ values do not help our idea at all. However, when epsilon gets higher than 1.5 (and thus theta rises above 2), the results are more than satisfying: *our protocol has the best behaviour for epsilon in the range of* $(2, 2.5)$.

The real test though, is how our protocol behaves for an increasing number of users: we must check if it produces better accuracy error than the competitors. This is our next testing, where we are going to set $\epsilon = \ln(20)$, in order for the conditions to be favorable for each one of the protocols. The results are shown in the **Figure 4.10**.

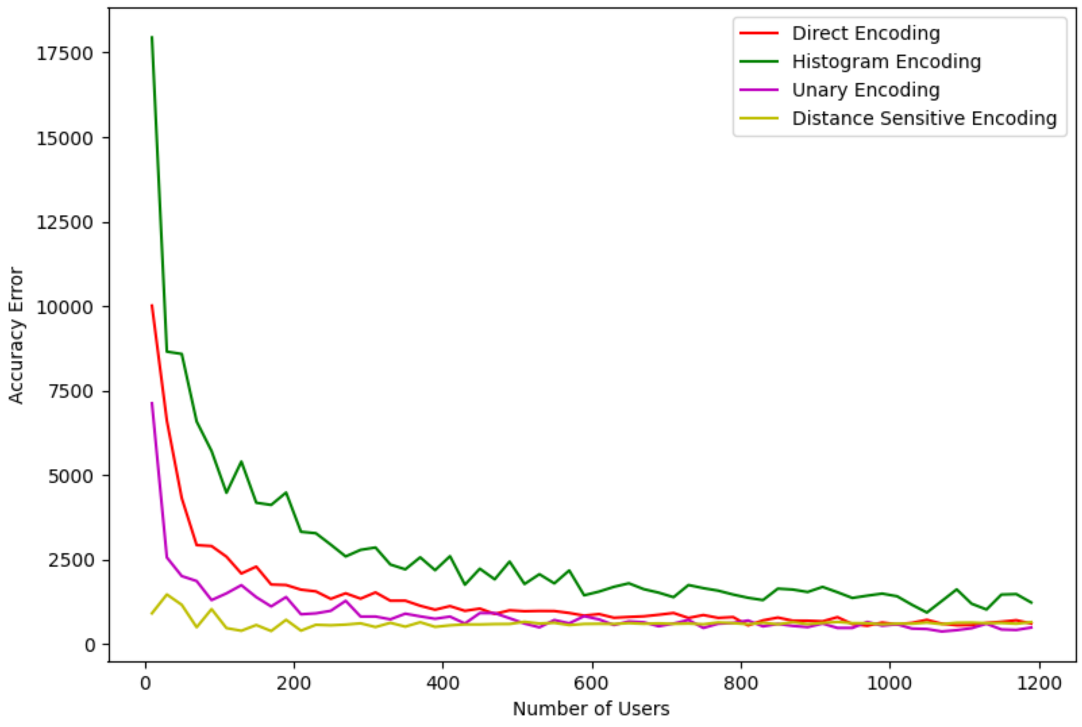


Figure 4.10: Increasing users measurements for D.S. protocol compared by Kantorovich Distance

For the specific epsilon setting, *our protocol produces extremely good accuracy error for a small number of users.*, beating by a lot the U.E. protocol. The comparisons have been made using the Kantorovich metric, the most characteristic of them all, as it takes into account the distance between the real answers and the projections, exactly what our protocol is designed to do. However, we are also going to perform the same testings using the Manhattan metric. The results are shown in the **Figure 4.11**.

The results of the Manhattan-driven tests are similar to the ones made with the Kant. metric. However, we observe that *when the number of users rises, our protocol has worse behaviour in comparison to the other pure ones*. This happens mainly because of the other protocols, that by the law of big numbers, have good accuracy because of the higher probability of choosing the true answer. On the other hand, in our protocol, this probability is reduced and shared with the other elements in the area covered by theta.

4.6.9 Conclusions

In general, *the D.S. protocol succeeds when the number of the participants in a survey is extremely low*, and functions similarly with the other protocols for an increasing number of users. The downside is that it does not always takes full advantage of the epsilon setting, as explained in a previous section. However, the results are more than satisfying. Hence, *this is a fully functioning protocol that can be used for the application of L.D.P., especially in a situation when few people take part in the survey.* The protocol will be further tested in more extreme cases, but this is beyond the scope of this Thesis.

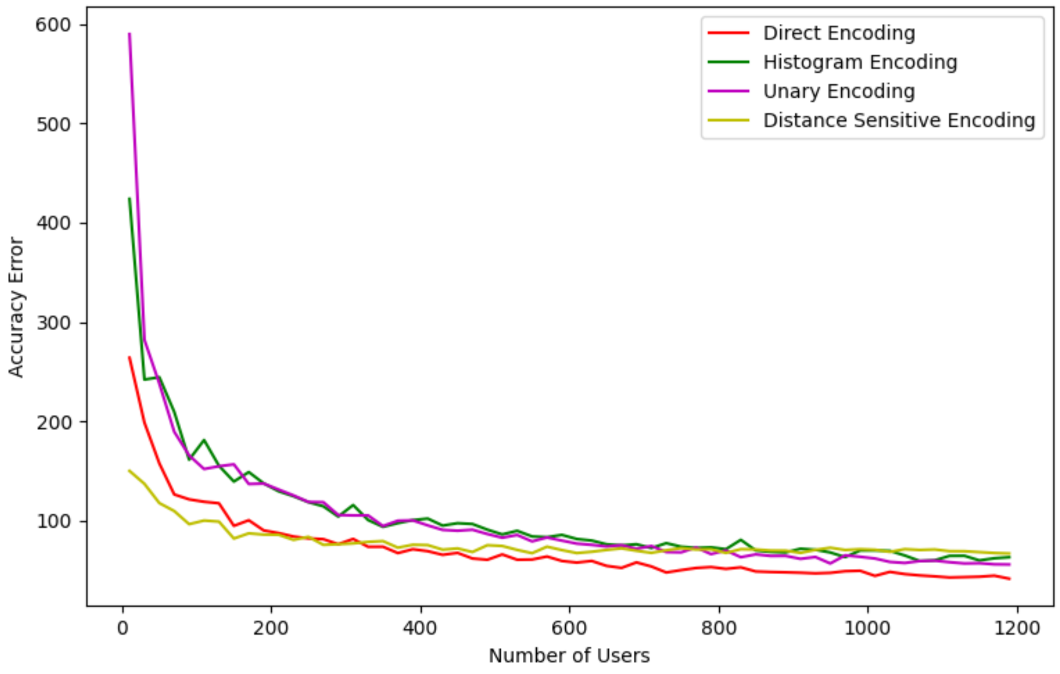


Figure 4.11: Increasing users measurements for D.S. protocol compared by Manhattan Distance

5. CONCLUSIONS AND FUTURE WORK

The goal of this thesis was to analyze the importance of protecting sensitive data, and doing so in an efficient way. After its elaboration, it is clear that Differential Privacy is a secure and efficient way for data anonymization. Having two forms, the Global and the Local, it can cover many different scenarios, including Machine Learning applications.

Differential Privacy is the future of Data Protection and Anonymization, as its results can not be compromised, due to the random noise that the algorithms introduce. Unlike previous methods, such as k-anonymity, there is not yet an attack that can reduce the privacy created by D.P. algorithms, which makes this technique ideal.

Despite the use of random noise the data is still useful, as the mathematical ideas behind the aggregation were built with the mindset of eliminating this noise using data normalization.

Having explored many different applications, algorithms and protocols we can safely say that when it comes down to Global D.P., IBM's diffprivlib is a state of the art library that produces extremely good results. Its use is quite simple as a Python API is provided, thus can be safely added to any numerical dataset.

When someone wants to apply L.D.P. during a survey, the pure protocols analyzed and tested are suitable for high efficiency combined with good protections of the members. With simple algorithms, they do not require a trusted curator in order to perform, hence users can perturb their data, and then safely report it. However, when the number of users is small, *the Distance Sensitive Protocol created for the needs of this Thesis is the best option*, as the other protocols produce extreme noise in order to maintain the privacy levels. On the contrary, the D.S. protocol takes into account the distance between the true value and the one being reported when creating its probabilistic space, thus lowering the error produced.

Our plans for future work are centered around the D.S. protocol. We would like to perfect its aggregation method, as it may produce satisfying results, but with a different approach it can maybe become even better. Moreover, we would like to perform more demanding experiments for extreme cases of dataset sizes, domain sizes and theta values.

Finally, similar testings like the ones introduced in this Thesis can be performed in other D.P. libraries, as the accuracy measurements is a good indicator if someone wants to rank those libraries.

ABBREVIATIONS - ACRONYMS

EMD	Earth Mover’s Distance
QIF	Quantitative Information Flow
DP	Differential Privacy
Kant.	Kantorovich
LDP	Local Differential Privacy
GDP	Global Differential Privacy
CSV	Comma Separated Values
GUI	Graphical User Interface
RR	Randomized Response
DE	Direct Encoding
HE	Histogram Encoding
UE	Unary Encoding
DS	Distance Sensitive

APPENDIX A. MATHEMATICAL PROOF OF THE D.S. PROTOCOL

During this Appendix, a mathematical explanation for the a variable in the D.S. protocol will be given.

In order to find out the α value, we must solve the following equation:

$$p + \sum_{i=x-\theta}^{i=x+\theta} q + \sum_{i=1}^{i=x-\theta-1} s + \sum_{i=x+\theta+1}^{i=d} s = 1$$

At this point, we must note that although not a constant, can be held out of the sums, because it is obviously independent from the i variable, that is the variable parsing through the domain in order to retrieve the false elements' probabilities. Thus, we have:

$$\begin{aligned}
p + \sum_{i=x-\theta}^{i=x+\theta} q + \sum_{i=1}^{i=x-\theta-1} s + \sum_{i=x+\theta+1}^{i=d} s = 1 \iff \\
\sum_{i=x-\theta}^{i=x+\theta} \frac{a}{|x-i|(|x-i|+1)} + \sum_{i=1}^{i=x-\theta-1} \frac{a}{\theta(\theta+1)} + \sum_{i=x+\theta+1}^{i=d} \frac{a}{\theta(\theta+1)} = 1-p \iff \\
\sum_{i=x-\theta}^{i=x-1} \frac{a}{(x-i)(x-i+1)} + \sum_{i=x+1}^{i=x+\theta} \frac{a}{(i-x)(i-x+1)} + \\
+ \frac{a}{\theta(\theta+1)} \cdot (x-\theta-1+d-x-\theta) = 1-p \iff
\end{aligned}$$

For the first sum, we set $u = x - i$ and for the second one $u = i - x$, and we have:

$$\begin{aligned}
\sum_{u=1}^{u=\theta} \frac{a}{u(u+1)} + \sum_{u=1}^{u=\theta} \frac{a}{u(u+1)} + \frac{a}{\theta(\theta+1)} \cdot (d-2\theta-1) = 1-p \iff \\
2 \cdot a(1 - \frac{1}{\theta+1}) + \frac{a}{\theta(\theta+1)} \cdot (d-2\theta-1) = 1-p \iff \\
2 \cdot a\frac{\theta}{\theta+1} + \frac{a}{\theta(\theta+1)} \cdot (d-2\theta-1) = 1-p \iff \\
\frac{a}{\theta+1}(2\theta + \frac{d-2\theta-1}{\theta}) = 1-p \iff \\
\frac{a}{\theta+1}(\frac{2\theta^2-2\theta+d-1}{\theta}) = 1-p \iff \\
\mathsf{a} = \frac{\theta(\theta+1)(1-\mathsf{p})}{2\theta^2-2\theta+\mathsf{d}-1}
\end{aligned}$$

APPENDIX B. REPOSITORY OF THE THESIS

The implementation of all the testings, the libraries and the protocols can be found in the GitHub repository of this thesis, in the link: `https://github.com/nikosgalanis/bsc-thesis`.

In the directory *ibm_lib_work*, all the notebooks with the measurements made for the IBM library are included.

In the directory *ARX_work*, the java code for the measurements in ARX is included, as well as the datasets and the hierarchies used in order to test the protocol.

In the directory *LDP*, the LDP library is implemented, using the already-known protocols from the Wang et. al. paper. Additionally, our own protocol created for the needs of this Thesis is included, alongside with a Python file responsible to create all the testings that were carried out.

Finally, in the directory *papers_used*, all of the papers referenced in this Thesis can be found.

More information for the repository and its contents can be found in the README file included.

BIBLIOGRAPHY

[1] Dwork, C., Roth, A. (2014). The algorithmic foundations of differential privacy. now Publishers Inc.

[2] Dwork, C., McSherry, F., Nissim, K., Smith, A. (2006). Calibrating Noise to Sensitivity in Private Data Analysis. Theory of Cryptography, 265–284.

[3] Holohan, N., Braghin, S., Mac Aonghusa, P., Levacher, K. (2019, July 4). Diffprivlib: The IBM Differential Privacy Library. arXiv.org.

[4] Li, N., Qardaji, W., Su, D. (2012). On sampling, anonymization, and differential privacy or,k-anonymization meets differential privacy. Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security - ASIACCS '12.

[5] Bild, R., Kuhn, K. A., Prasser, F. (2018). SafePub: A Truthful Data Anonymization Algorithm With Strong Privacy Guarantees. Proceedings on Privacy Enhancing Technologies, 2018(1), 67–87.

[6] Christofides, T. C. (2003). A generalized randomized response technique. Metrika, 57(2), 195–200.

[7] Chatzikokolakis, K., Palamidessi, C., Stronati, M. (2015). Location privacy via geo-indistinguishability. ACM SIGLOG News, 2(3), 46–69.

[8] Jain, P., Gyanchandani, M., Khare, N. (2018). Differential privacy: its technological prescriptive using big data. Journal of Big Data, 5(1).

[9] Bebensee, B. (2019, July 27). Local Differential Privacy: a tutorial. arXiv.org.

[10] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. 2017. Locally differentially private protocols for frequency estimation. In Proceedings of the 26th USENIX Conference on Security Symposium (SEC'17). USENIX Association, USA, 729–745.

[11] Chatzikokolakis, K., Andrés, M. E., Bordenabe, N. E., Palamidessi, C. (2013). Broadening the Scope of Differential Privacy Using Metrics. Privacy Enhancing Technologies, 82–102.

[12] Chamikara, M.A.P. Bertok, P. Khalil, Ibrahim Liu, D. Camtepe, Seyit. (2019). Local Differential Privacy for Deep Learning.

[13] Chatzikokolakis, K., Fernandes, N., Palamidessi, C. (2020). Refinement Orders for Quantitative Information Flow and Differential Privacy. Journal of Cybersecurity and Privacy, 1(1), 40–77.

[14] Bassily, R., Smith, A. (2015). Local, Private, Efficient Protocols for Succinct Histograms. Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing.

[15] Erlingsson, Ú., Pihur, V., Korolova, A. (2014). RAPPOR. Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.

[16] ”Surgery Charges Across the U.S.”, https://data.world/dmikebishop/surgery-charges-across-the-u-s.

[17] ”NBA Salaries”, https://data.world/datadavis/nba-salaries