M.Sc. Thesis
Master of Science in Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

Machine Learning with Respect to Privacy

A distributed machine learning using *differential privacy* and *shared gradient methods*

Ingvar Bjarki Einarsson (s161294)

Kongens Lyngby 2018

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

Distributed privacy preserving machine learning is a growing field which is gaining more relevance by the minute. The aim of distributed privacy preserving machine learning is to enable learning between multiple parties without actually sharing data. This can be very useful as it enables organization to collaborate. However, due to legislation as well as ethical and economic factors, organizations might be inhibited from sharing data. The methods discussed in this thesis might help organizations overcome these issues.

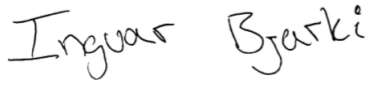
Multiple frameworks exit for achieving privacy preserving machine learning. This thesis examines how privacy and error rate are tied together. Three different algorithms for providing privacy were constructed. A distributed Lasso which only shares gradients between data centers, A regularized logistic regression with differential privacy and a stochastic gradient descent with differentially private updates.

The differentially private algorithms show that the amount of training data used to build the predictive model is important. Both mechanism are able to tie their sensitivity to the amount of training data, resulting in much better error rate for large amount of data. The stochastic gradient decent was able to show that using random projections beforehand can improve the error rate. Finally, the distributed Lasso showed that for a specific amount of data collaboration between parties can result in a strong improvement on the error rate.

Preface

This Master's thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a M.Sc. degree in Mathematical Modelling and Computation.

Kongens Lyngby, July 30, 2018



Ingvar Bjarki Einarsson (s161294)

Acknowledgements

First and foremost I would like to thank to my supervisor Lars Kai Hansen for our weekly discussions and his guidance during the thesis. I would also like to express my gratitude to Ásdís Helga Óskarsdóttir, Björk Óskarsdóttir, Hallgrímur Hrafn Einarsson and Þorsteinn Gunnar Jónsson for proofreading my thesis and for providing support.

I would like to thank Anand D. Sarwate and Shuang Song for replying to my emails regarding their stochastic gradient descent algorithm with differentially private updates. Finally, I would like to thank all of my friends studying with me at DTU for moral support.

Contents

Nomenclature

Abbreviations

DP	Differential privacy
GD	Gradient descent
MLE	Maximum likelihood estimation
MPC	Secure multi-party computation
PCA	Principal component analysis
SGD	Stochastic gradient descent

List of symbols

ϵ, δ	Privacy parameters
η	Learning rate
λ	Weight decay
$\mathcal{M}$	Randomized mechanism
∇E	Gradient of the cost function
Σ	Co-variance matrix
σ	Variance or the logistic function
τ	Maximum iteration count
b	Batch size
d	Dimensions
N	Number of observations
t	Iteration count
w	Weights

y	Targets
E	Cost function
f	Function
R	The random projection matrix
X	Data set
X′	X's neighbouring data set

CHAPTER 1

Introduction

Lately, privacy and big data have gained increasing interest. As the next wave of technology progresses, big data, machine learning and internet of things have gained huge attention. Users are getting conscious about data, and are starting to realize that they generate a trail of data. Furthermore, users have started to recognize that more and more organizations are starting to collect data. This brings up questions of privacy, can those organizations look at your own data and discover something about you that you might not want to/are not willing to share? Can attackers get a hold of sensitive data about you, such as medical history or tax returns? Or does it simply make you uncomfortable that someone might be monitoring your actions on the web? A number of similar questions can be made, thus, the treatment of data is a great concern. Several organizations have made mistakes by making sensitive data public, or have treated data lightly, to their customers´ discontent. To name some who might not take privacy seriously enough, the CEO of Facebook Mark Zuckerberg stated in January 2010 that privacy was no longer a social norm. If the reader is interested in learning more about data mishaps the author suggests [1], where some good illustrations of mismanaged data are discussed.

Although big data has led to privacy concerns, it has also contributed to remarkable progress in many aspects. Machine Learning has helped a range of industries such as health care, transportation, financial services and fraud detection, to name a few. It has for example showed great promise in early detecting of cancer and Alzheimer's diseases [2]–[5], so it is easy to conclude that machine learning is an immensely important field.

As machine learning increases in popularity, an opportunity for different organizations to collaborate emerges. However, due to legislation, as well as, ethical and economic factors many organizations are forbidden to publish or share their data. For instance, if a pharmaceutical company needs data from a health care organization to examine if their own drugs have side effects, due to sensitive data it can be unethical to share. Additionally, some organizations are not willing to share their data, but are however willing to consider collaborating on solving some specific problem, if it is beneficial to them. This further emphasizes the relevance of binding privacy and machine learning together. This point is something that many of the leading companies, such as Google, Apple and Microsoft have recognized. Thus, if we manage to do machine learning while at the same time maintaining privacy we have hit the bull's-eye.

In today's world, some might think that it would be enough to anonymize data to preserve privacy. That could mean removing names, social security number or other similar identifiers. Sadly, this can cause a leak of information. [1] mentioned a famous story about the governor William Weld, who approved release of medical records ranging from state employees to researchers. He noted that the users in the data set were safe, as the obvious revealing identifiers from the data set were eliminated. A few days after the data was released, William got his personal medical records delivered to his office. A MIT student had connected the other data identifiers and identified William in the data set [1].

There might also exist auxiliary public data which the adversary can link together with the published anonymized data set. This will enable the adversary to find the hidden identifiers in the data set. This is often referred to as *linkage attack* [1].

In this thesis we will establish differential privacy mechanisms for distributed machine learning without actually sharing data. Regularized logistic regression and Stochastic gradient descent will be built using differential privacy as well as a distributed private Lasso, were the idea is to share only gradients. The algorithms will be tested on the famous mnist data set. The intention of this thesis will be to analyze different privacy mechanisms, investigate how privacy and accuracy are tied together and to discuss when which algorithm is appropriate.

CHAPTER 2

State of the Art

Many methods of maintaining privacy have been tried. *k-anonymity* is one. The idea is to make pre-determined identifiers homogeneous in k number of rows of the data set. This can be done by either putting an '*' were the identifiers differ, or by changing the identifier so that it has a 'range'. For example, if an identifier is age, the filed could state that a user is born between 1990 and 1999 [6]. *l-diversity* is an extension of the k-anonymity method. Here it is stated that the k-anonymity can leak information because of lack of diversity in the sensitive attribute. Therefore, *l*-diversity constraints the sensitive attribute to have at least l well represented different values [7]. *t-closeness* is then an extension of the *l*-diversity. It adds another constraint on the sensitive attribute, which states that the distribution of the sensitive attribute in any equivalent class is close to the distribution of the overall data set [8]. While all these methods are clever, they can fail to prevent security in extreme circumstances. [9] takes a great example. If a data set hold the income and the city of 4 users, and using 3-anonymity, the average salary of that city is published. Now if the adversary knows that two of them live in Reykjavik and one in Copenhagen, if no data is published the adversary knows that the fourth person does not live in Reykjavik. Moreover, if the adversary knows the salary of the two who live in Reykjavik, it would be easy for him/her to calculate the salary of the fourth user if the data was published.

Secure multi-party computation (MPC) takes another approach to preserve privacy. The idea behind MPC is that a number of participants try to calculate a global function without informing the other parties about their input data. One could consider to bring a trusted outsider to the table. That way everyone could feed their input to the trusted party. He/she would calculate the value of the global function and then reveal the result to the participants. However, here it is expected that the outsider can be trusted. MPC, takes another approach. Thus, MPC builds upon complex protocols to calculate the global function, without sharing the inputs of the parties of the system [10].

Another method is *Federated Learning*. The idea is that a model is distributed over all devices in a system. The system could be composed of smart phones, smart tablet or any device in general. A model is trained locally on each device, then results are summarized and sent to a cloud using encrypted communication. When the cloud has received a given number of updates, it encrypts the information and updates a

global model, which will then be sent to the devices. A model is also able to learn both locally and globally with information summarized by the cloud. This entails that the raw data will never leave each device, making the data private to each party. [11].

Differential privacy (DP) is another method that preserves privacy. The aim of DP is to utilize randomness and noise to build a mechanism to give users in the data set plausible deniability. The objective is to provide privacy for users by perturbing a function realising statistics about the data set. The classic example of a simple differential private mechanism is the coin flip. There, a participant is asked a question, for example if he/she smokes or not. The participant throws a coin, if it ends up being head he/she tells the truth, however if it ends up being tails, he/she throws the coin again, and answers yes if it's head and no if it's tails. This way, DP introduces randomness into the data resulting in plausible deniability for the participant.

Furthermore, DP states that nothing can be learned about an user in the data set. This is due to the fact that the DP mechanism utilizes randomness to guarantee that users in the data set can not be harmed, even if the adversary has auxiliary information. In that note, differential privacy does not, in a sense, protect users from the results of the study. If for example the study is about smoking being unhealthy or not ends up by stating that smoking is unhealthy. DP might hurt some study participants that smoke. However, the study would hurt that participant independently of whether he/she is in the data set. Nonetheless, if it is unknown to an interested party that a participant smokes, he/she can not conclude whether the participant smokes. Hence, DP is a definition/concept that helps to learn about population while protecting the privacy of users [12].

CHAPTER 3

Theory

3.1 Preliminaries

This section provides a a review of mathematical terms used in the Thesis.

Let $f : S \in \mathbb{R}^n$ be a function.

- f is *convex* if for any two points in S, a straight line between them is entirely inside S. formally f i said to be convex if for any two points x and y the following properties are satisfied, $f(\alpha x + (1-\alpha)y \leq f(x) + (1-\alpha)f(y), \quad \text{for all } \alpha \in [0,1]$
- f is *strongly convex* if for any x, y the following properties are satisfied, $\langle \nabla f(x) - \nabla f(y), (x-y) \rangle \geq m\|x-y\|^2, \quad \text{where } m > 0$

Moreover, if f is continuous and twice differentiable function, convex functions can be determind as, $f''(x) \geq 0$, and m-strongly convex as $f''(x) \geq m > 0$.

Let x, y, z be a side of a triangle where $x, y \leq z$. Then *triangle inequality* states that $z \leq x+y$. In euclidean geometry the inequality can be written as $\|x+y\| \leq \|x\| + \|y\|$.

3.2 Differential Privacy

As stated in the introduction, DP is a method, which has the purpose of preserving privacy. The idea is to utilize randomness to give users plausible deniability. This is done by applying a randomised algorithm/mechanism, $\mathcal{M}$, to the data/statistics before handing it to other parties. Furthermore, DP gives the promise that nothing can be learned about an user in the data set, rather it enables learning about the population of the participants [12].

DP can be used in two settings. The first one is referred to as the non-interactive setting. There the DP mechanism is used to publish some statistics where the data is not used more. From there the data and the mechanism can be deleted right after the analysis. The second setting, and the one which will generally be more discussed in this thesis, is referred to as the interactive setting. There the DP mechanism sits between the true data and the user. Now an operator can ask for queries and the mechanism will perturb it and then return it to the operator. In this way, the

data and the mechanism can not be destroyed [13]. Figure 3.1 shows the interactive setting.

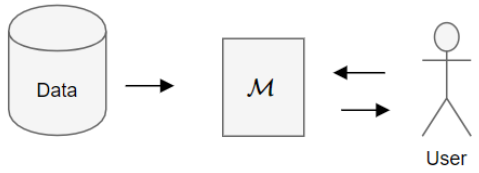


Figure 3.1: The interactive setting for differential privacy, where mechanism $\mathcal{M}$ perturbs the data, before handing it to the user.

Questions arise when building differential private mechanism. What is needed for a mechanism to be differentially private? How much noise or randomness should one perturb the data with? These are valid questions. [12] introduces *Definition 1* to answers these kind of questions.

Definition 1: (Differential Privacy; [12]). A randomized algorithm $\mathcal{M}$ with domain $\mathbb{N}^{|\mathcal{X}|}$ is (ϵ, δ)-differentially private if for all $\mathcal{S} \subseteq$Range ($\mathcal{M}$) and for all X, X$' \in \mathbb{N}^{|\mathcal{X}|}$ such that $\|\mathrm{X} - \mathrm{X}'\|_1 \leq 1$:

$$Pr[\mathcal{M}(X) \in \mathcal{S}] \leq \mathrm{e}^{\epsilon} Pr[\mathcal{M}(X') \in \mathcal{S}] + \delta$$

Here, X and X$'$ are said to be neighbouring data sets if they differ at most by one observation. There exist two different understandings of neighbouring data set in the literature. Some might understand the statement such that one data set has one more observation than the other. While others might understand that the data sets have the same number of observations, but have one sample which is different between the data sets. The first interpretation is stronger [9]. However, in this thesis the latter statement will be used, as inventors of algorithms in the thesis understand neighbouring data sets in that way. In addition, both ϵ and δ are real positive numbers who control the amount of privacy an user in the dataset receives. From the definition it is possible to see that $\mathcal{S}$ denotes all possible events to occur from the randomized algorithm $\mathcal{M}$.

In plain language, the definition simply states that if two data sets differ by one observation, the mechanism, $\mathcal{M}$, should still yield very similar results for both data sets. Thus, the probability of an user being from two different data sets and the probability where said user only belongs to one data set is similar. This, in essence, means that if one has negligible or no influence on the output of the mechanism the data should be save. From the definition it is possible to see that the only slack that is allowed between the output of the mechanism for the two data sets are introduced with ϵ and δ. Thus, if both ϵ and $\delta = 0$, the mechanism should return the exact same value for the two data sets X and X$'$. If a mechanism meets the requirements

of definition 1, it is said to be (ϵ, δ)-differentially private. Further, if δ is not needed to fulfill the requirement, the mechanism is said to be ϵ-private.

One can observe how both ϵ and δ control how much privacy participants receives. The lower the value of ϵ and δ, the more secure the users are in the data set. Thus, if security is of the utmost importance, both ϵ and δ should be chosen as small as possible [12]. Analytically, this can bee seen by taking a closer look at definition 1. Ignoring δ for now, if ϵ is small, then consequently e^{ϵ} is small. This entails that the output from the DP mechanism needs to be very similar for both X and X$'$. Thus, as ϵ decreases the more privacy the participant's in the data set gets. Similar consequences come up for δ, however instead of using multiplication δ uses addition (see definition 1).

However, it is not possible to just select ϵ as some very small number. This is due to the fact that if ϵ is too small the usefulness of the data will be inadequate. [1] takes an excellent illustration of this. Imagine that if $\delta = 0$ and $\epsilon = 0$, then unavoidable $e^{\epsilon} = 1$. Now definition 1 requires the mechanism, $\mathcal{M}$, to be indistinguishable on any two input data sets. This indicates that as ϵ decreases, the more randomness dos the mechanism $\mathcal{M}$ need to account for. On the contrary, if ϵ is too big the participants in the data set are much more vulnerable for data leak/attacks. Thus, there is a trade-off between privacy and usefulness.

The privacy parameters, ϵ and δ, are selected by the person building the mechanism. The choice of their values is a social one [12]. If the values are too great the mechanism might not be private at all. [12] argues that good values for ϵ are e.g. [0.01, 0.1]. Looking at δ, it is typical to choose it as a value smaller than the inverse of the polynomial size of the data set, that is $\delta < \frac{1}{\text{number of observations in the data set}}$. If δ is greater it might risk the security of several of the participants in the data set [12]. However, the choice of the parameter values of course depend on the data, for some sets it is extremely important to remain private while for other less so.

It is important to mention how the *privacy loss* is calculated. Equation 3.1 shows how the loss is calculated by observing the output, ξ. The loss can both be positive or negative depending on which data set is more likely to include a given output, ξ. The privacy loss formulation rises from the fact that the mass of ξ might be much larger for either of the outputs generated by mechanism, $\mathcal{M}$, on each of the data sets [12].

Intuitively the privacy loss shows the loss when $\mathcal{M}$ yields ξ for neighbouring data sets. If the stronger case of differential privacy discussed above is taken for example. Then, if the likelihood that a mechanism yields ξ is equal with and with out an observation, then equation 3.1 will result in $\ln(1) = 0$. However, if the algorithm is more or less likely to result in ξ when it is with or without an observation, then equation 3.1 will result a privacy loss corresponding to the likelihoods seen in equation 3.1.

$$\mathcal{L}^{\xi}_{\mathcal{M}(\mathrm{X})\|\mathcal{M}(\mathrm{X}')} = \ln\left(\frac{Pr[\mathcal{M}(\mathrm{X}) = \xi]}{Pr[\mathcal{M}(\mathrm{X}') = \xi]}\right) \tag{3.1}$$

Finally, [12] states that DP is immune to post processing. This means that it is possible to work on a DP output without affecting the privacy promise previously made by the DP mechanism [1].

3.2.1 Sensitivity

In order to be able to introduce the right amount of randomness to the data a so called *sensitivity* needs to be calculated. *Definition 2* outlines sensitivity in a formal way, where ℓ_1 is the first norm. The sensitivity finds the biggest difference between a function f on two neighbouring data sets. In other words it finds the observation which has the most influence on the function, and finds the difference between having it or not. This gives a bound on how much randomness is needed to hide the users that deviate the most from the data [13].

Definition 2: (ℓ_1-sensitivity; [12]). The ℓ_1-sensitivity of a function $f : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$ is:

$$\Delta f = \max_{\substack{\mathrm{X},\mathrm{X}',\in\mathbb{N}^{|\mathcal{X}|} \\ \|\mathrm{X}-\mathrm{X}'\|_1=1}} \|f(\mathrm{X}) - f(\mathrm{X}')\|_1$$

An example of a function, f, can be the count. More formally, let f answer the question: "How many elements in the data set satisfy the property P?" [12]. This function has the sensitivity of 1. To see this, note that either the element satisfies P or not, hence they contribute either 0 or 1 to the count. This shows that the lower bound of f is 0 and the upper bound is 1. Inserting these results into Definition 2, it is clear that the sensitivity equals 1. This makes sens as the removal of a single element can change the count by at most 1 [12].

Another example of a function, f, could be a regularized logistic regression model. If the reader is not familiar with logistic regression then the author points to section 3.3.5. First, a logistic regression could be trained on the data to obtain the optimal weights, w. From there, the sensitivity can be calculated by how much a single observation can influence the weights in the most extreme case. After calculating the sensitivity, the mechanism could add noise to the weights using the sensitivity in conjunction with the ϵ parameter. Hence, if the only difference between the neighbouring data sets is a participant which is an extreme outlier, the mechanism should be able to protect it with ϵ differential privacy. Section 3.3.6 shows the sensitivity for regularized logistic regression model using differential privacy.

As the sensitivity decreases it is logical to select stricter ϵ and δ. This is because that the mechanism, $\mathcal{M}$, dose not need to generate as much randomization. An intuitive explanation of this is that as the sensitivity decreases, the closer all the data points are. Meaning that $\mathcal{M}$ needs less randomization to make them indistinguishable.

Sometimes the sensitivity is too high to get any meaningful results. This may indicate that the data set is too small. This could also mean that the statistics are

not robust enough, giving the outliers too much influence on the result [1].

3.2.2 Composability and Group Privacy

One thing to have in mind when building DP mechanisms is how they scale with groups of people. A family can for instance have some attributes which relates them, namely same home address, which could harm their security. This could help the adversary gain information on how the privacy mechanism, $\mathcal{M}$, is constructed. Thereupon, it is import when constructing a privacy mechanism, $\mathcal{M}$, to consider how *group privacy* scales. [12] declares that DP declines linearly with each new participant belonging to a group. Mathematically speaking this means that group privacy definition 1 can be rewritten as: $Pr[\mathcal{M}(\mathrm{X}) \in \mathcal{S} \leq \mathrm{e}^{(k\epsilon)} Pr[\mathcal{M}(\mathrm{X}') \in \mathcal{S}]$, where k denotes the group size.

Many mechanism's have been built to ensure DP. Numerous of them utilize some distributions to perturb the original data with noise. Two of which will be discussed here down below, namely *Laplace Distribution* and *Gaussian distribution*. If the adversary would gain knowledge on which distribution and parameters are being used, the participants in the data set might be at risk. The adversary could gain this knowledge by repeating the same queries, or run just multiple queries to find the distribution used to perturb the true data. Therefore, a so called *privacy budget* is needed to ensure the safety of the participants. A privacy budget stands for the total DP of a mechanism, or in other words the total permissible ϵ of a mechanism.

DP maintains composability [1], here two definitions of different compositions will be given, namely *sequential composition* and *parallel composition*. In the sequential composition the DP (ϵ_i) of each query that is run on the data set can be accumulated to find the true privacy loss for all the queries. Therefore the ϵ of a process can be expressed as $\epsilon = \sum_i \epsilon_i$. However, in parallel composition, the queries are applied on disjoint sets of the data. As each query is applied on disjoint data, the ϵ can be calculated as $\epsilon = \max_i \epsilon_i$ [14].

The two types of compositions can be exploited to find the privacy budget. By using them it is possible to gain knowledge of the true security of a mechanism. Hence, if an adversary or a operator tries to run the query multiple times, a privacy budget can be selected to prevent multiple query runs. The number of runs will then depend on the privacy budget [15].

3.2.3 The Laplace Mechanism

The Laplace mechanism uses the Laplace distribution to add noise to the outcome of a function f, which is applied to the true data and by that hiding the true values of f. The Laplace distribution is sometimes called the double exponential distribution as it is a symmetric version of the exponential distribution. Equation 3.2 shows the probability density function of the Laplace distribution. The distribution is centered around 0 with scale b and has the variance $\sigma^2 = 2b^2$ [12].

$$\text{Lap}(x|b) = \frac{1}{2b}\exp(-\frac{|x|}{b}) \tag{3.2}$$

Equation 3.3 shows how the Laplace mechanism is applied. This has been proven to be ϵ-private [12]. Here the noise that is added has zero-mean and the scale $b = \frac{\Delta f}{\epsilon}$. The noise is independently added to all relevant parameters [12]. The Laplace distribution (equation 3.2) shows that by increasing b, the tail of the Laplace distribution will become bigger. This yields the fact that as ϵ decreases the bigger b will become. Thus, as ϵ decreases the larger tail the Laplace distribution holds, resulting in more randomness.

$$\mathcal{M}_L = f(x) + \text{Lap}(\frac{\Delta f}{\epsilon}) \tag{3.3}$$

Figure 3.2 shows the probability density for the Laplace distribution. The density is plotted with $\Delta f = 1$ and different ϵ values. It can be seen that as ϵ decreases the curve of the Laplace distribution flattens, this agrees with [13]. Moreover, the figure emphasizes the above discussion on how ϵ is likened to randomness.

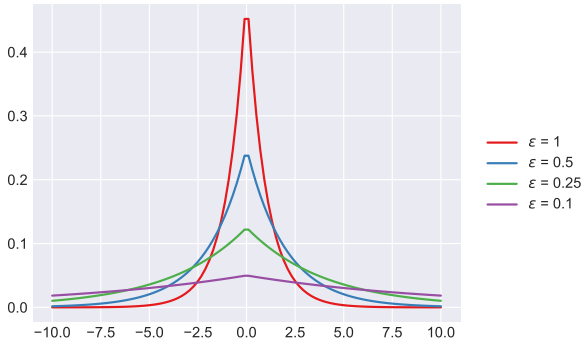


Figure 3.2: The probability density for Laplace distributions having different privacy guaranties.

3.2.4 The Gaussian Mechanism

The Gaussian mechanism uses the Gaussian distribution to add noise to a function f, which is applied on the true data in order to protect the users in the data set. The noise added to each component of the output has the form $\mathcal{N}(0, \sigma^2)$, where $\mathcal{N}$ denotes the normal distribution shown in equation 3.4.

$$f(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.4}$$

The Gaussian mechanism uses ℓ_2 sensitivity instead of the normal ℓ_1 sensitivity. The ℓ_2 sensitivity can be defined in the following way:

Definition 3: (ℓ_2-sensitivity; [12]). The ℓ_2-sensitivity of a function $f : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}^k$ is:

$$\Delta_2 f = \max_{\substack{\mathrm{X},\mathrm{X}',\in\mathbb{N}^{|\mathcal{X}|} \\ \|\mathrm{X}-\mathrm{X}'\|_1=1}} \|f(\mathrm{X}) - f(\mathrm{X}')\|_2$$

Let's consider how the noise of the Gaussian distribution is selected in order for the mechanism to be differentially private. Let $\epsilon \in [0, 1]$, and $\mathrm{c}^2 > 2\ln(1.25/\delta)$. Then a Gaussian mechanism with $\sigma \geq c\Delta_2 f/\epsilon$ is (ϵ, δ)-differentially private [12]. Thus as c increases more privacy is preserved.

The Gaussian mechanism dose not promise as good of a privacy as the Laplace mechanism. The Laplace mechanism is able to promise $(\epsilon, 0)$ differential privacy, while the Gaussian promises (ϵ, δ) differential privacy. As stated in section 3.2, one needs to be careful when selecting δ. If δ is not small compared to the number of participants in the data, a risk of revealing the outliers of the data set arises [12].

One advantage of using the Gaussian mechanism is that the sum of two Gaussian is Gaussian. This can be useful in many settings, as the real world noise/error in data is often considered Gaussian. This might make it easier to understand the effect the mechanism has on the analysis. Another advantage of the mechanism is that it has the same cumulative loss under composition as the Laplace mechanism. Thus, although the Gaussian mechanism can not promise an equal amount of privacy to the Laplacian, the cumulative loss over many computations are comparable [12].

Equation 3.5 shows how the Gaussian mechanism is applied. This has been proven to be differentially private by [12]. Note that the σ is bounded as stated here above.

$$\mathcal{M}_G = f(x) + \mathcal{N}(0, \sigma^2)) \tag{3.5}$$

Figure 3.3 shows the density for the Gaussian, with $\Delta f = 1$ and different combinations of ϵ and δ. Similar to the Laplace density, it can be seen that as ϵ decreases, the curve flattens. Accordingly, more randomness is introduced to the mechanism. The figure

also shows that if δ is very high, then the distribution will be much sharper. This highlights the above discussion on the importance of choosing an appropriate value for δ.

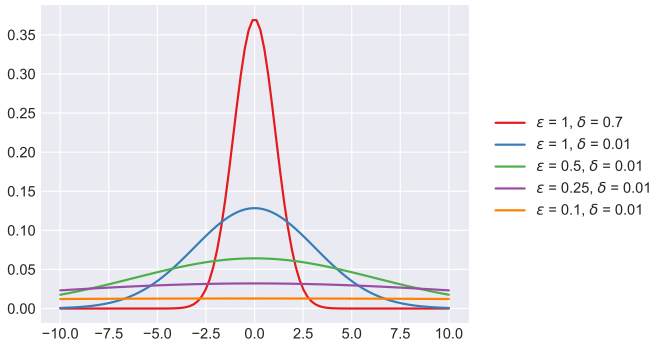


Figure 3.3: The probability density for normal distributions having different privacy guaranties.

3.2.5 Limitations

From the analysis above it is possible to see that if either ϵ or δ are too high, the DP mechanism's will not hide any information. Imagine publishing the mean salary of two groups of people. One with user X_i and one without. If too little noise is added to the statistics, the adversary can gain information of X_i's salary. The adversary will be able to calculate some interval were X_i salary lays, or even if way to small noise is added, the adversary will gain the knowledge of X_i salary.

Another thing which is very important that the data analyst thinks of is the composability of DP. If two many statistics are published, the privacy loss might be to large. This indicates that the data analyst needs to set a strict limits on the privacy budget. Therefore, only few statistics can be run on the data before reaching the privacy budget. This can be troublesome, as the information desired from the data might not be known beforehand [12], [1].

Lastly, some individuals or organizations might not be willing to get noisy results from service they pay for. [1] takes a great example. They point at the simple fact that advertisers in collaboration with Facebook would probably not be happy to pay for counts with added noise of how many times their ad was shown. In addition, these advertiser could be running multiple advertisements on Facebook and also want information of there count very often [1]. Here the comparability of DP appears, destroying the privacy, as Facebook can't simply put a limit on how often they charge there collaborators.

3.3 Machine Learning

3.3.1 Lasso Regression

Lasso regression is a well known regression method which can be used for feature selection. It is similar to the *ordinary least squares*, but also includes a penalization term. Equation 3.6 shows the Lasso regression. There are two reasons for including the penalization term, *prediction accuracy* and *interpretation*. The normal linear regression often has a large variance and a low bias. The penalization term will reduce the complexity of the model which leads to reduced variance and increased bias. This helps avoid overfitting, resulting in increased prediction accuracy. Regarding interpretation, if a data set contains a lot of different attributes, Lasso can be used to find a smaller subset of the most important attributes [16].

$$\hat{w}^{\text{Lasso}} = \arg\min_{w} \left\{ \frac{1}{2} \sum_{i=1}^{N} (y_i - w_0 - \sum_{j=1}^{p} \mathrm{X}_{ij} w_j)^2 + \lambda \sum_{j=1}^{p} |w_j| \right\} \tag{3.6}$$

As can be seen in equation 3.6 the Lasso uses the first norm to penalize. This type of penalization shrinks some constants and sets other to zero. This is because of the geometric shape of the first norm. In figure 3.4 the geometric shape of the Lasso is exhibited. The first norm resembles a diamond, while the contour lines are the quadratic part of the loss function. From there, it is apparent that the loss function seeks towards the corners of the diamond. If the objective is at a corner of the diamond, in some dimension, one variable will become zero [16]. This can thus be used for feature selection. Another attractive thing about the Lasso is that it is convex, which is a very attractive feature for computational purposes [17], that for example entitles that the local minimum is in fact the global minimum.

It can be seen from equation 3.6 that the λ parameter needs to be tuned. It controls the amount of penalization. If λ is high, the w's will be forced towards 0. This entails the fact that fewer attributes will be selected from the feature selection.

Lastly, we note that the Lasso penalty, the first norm, is not differentiable at all points. This fact can make the optimization problem described in equation 3.6 hard to solve. Fortunately, there are many existing algorithms which can deal with such functions [18].

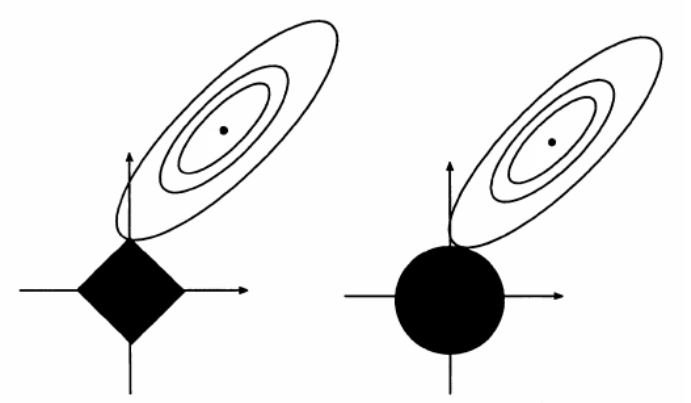


Figure 3.4: The figure shows the geometric properties of both using $l1$ norm (one the left) and the $l2$ norm (on the right) as penalization terms. Both methods select the first points where the constraints surface hits the contour lines [18]. In the case of the $l1$, it can be seen that the w's will most likely be selected in the corner, as the aim is to minimize the quadratic function. However, as the $l2$ norm is a sphere, all points are equally likely to hit the contour surface to begin with. Figure taken from [16].

3.3.2 Distributed Private Lasso; Li, Yang, Zhan, Hibar, Jahanshad, Wang, Ye, Thompson, Wang

[19], [20] proposed a solution to use distributed feature selection while keeping data private. The idea is to use distributed Lasso regression for variable selection over multiple data centers, on the interval $i \in [0, m]$. Each center contains the data X_i, y_i, having N_i number of data points. This data needs to be private. The described setting is illustrated in figure 3.5.

To solve the objective for Lasso (equation 3.6) the Iterative Shrinkage/Thresholding algorithm is applied (equation 3.7). There, t stands for the iteration number, w for the weights, $\nabla\mathrm{E}$ for the gradient of a cost function and η for the learning rate. In addition to that, the Γ function works as follows: $\Gamma(\arg) = sign(\arg) * (|\arg| - \lambda)$ where λ is the regularization parameter from the Lasso regression.

$$w_{t+1} = \Gamma(w_t - \eta_t \nabla \mathrm{E}(w_t; \mathrm{X}, y)) \tag{3.7}$$

It is evident that in order to solve equation 3.7 a gradient needs to be calculated. The purpose of the gradient is to fit a regression line or a plane to the data. Thus, an appropriate cost function must be minimized. The cost function used is the sum of squared errors (see equation 3.9). The gradient of the cost function, can be found by the use of derivatives. Calculating the derivative of the sum of squares gives $\nabla\mathrm{E} = \mathrm{X}^T(\mathrm{X}w - y)$. However, the gradient, $\nabla\mathrm{E}$, can not be calculated like usual, as

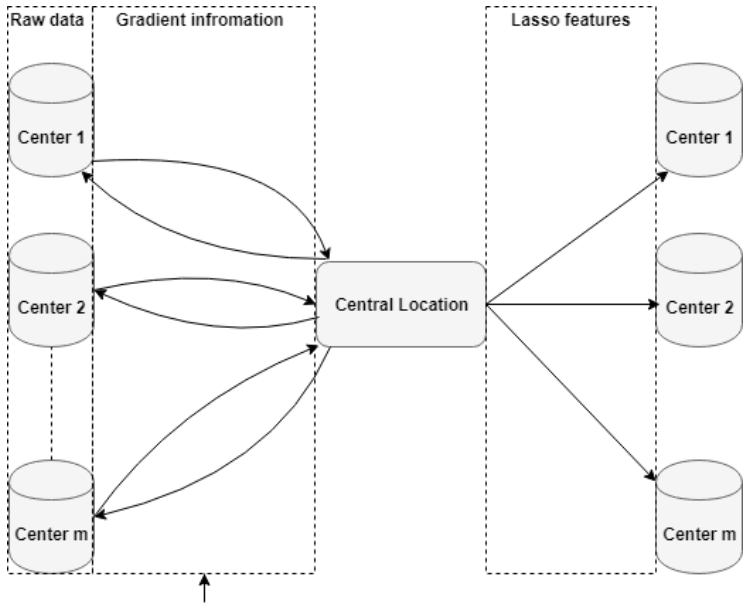


Figure 3.5: The setting for the distributed Lasso. Here there are m data centers which constantly transfer gradient information to a local center, until the Lasso features have been found.

the data is distributed over m data centers. [20] argued that $\nabla \mathrm{E}$ can be calculated by calculating a local gradient $\nabla \mathrm{E}_i = \mathrm{X}_i^T(X_i w - y_i)$ at each data center, and then sum them all together to obtain the global gradient, $\nabla \mathrm{E}$, This shown in equation 3.8.

$$\nabla \mathrm{E} = \mathrm{X}^T(\mathrm{X}w - y) = \sum_{i=1}^{m} \mathrm{X}_i^T(\mathrm{X}_i w - y_i) = \sum_{i=1}^{m} \nabla \mathrm{E}_i \tag{3.8}$$

Looking at equation 3.8 it seems like the model is convex. As stated above, the Lasso algorithm is it self a convex problem. In addition to that, if two convex functions are added together they will result in a convex function (see definition in sec 3.1). Equation 3.8 states that it is equivalent to calculate the gradient of the cost function using the whole data set, and the sum of multiple gradients each containing a subset of the data set. This indicates that the loss function is convex. This can be seen by observing the behaviour of the gradients and convex functions. The gradient is only 0 when it is in it's minimum, and in convex functions the local minimum is the same as the global minimum. Therefore, if the gradient is 0 for the convex function using all the data, it will also be 0 in the distributed setting. Similarly when $\nabla \mathrm{E}$ is not 0, it will not be in a minimum in both cases.

As only the gradients from each private data center are sent to a public central location, only the gradient information will be public. [20] states that this procedure

is done until the total loss is less than a present threshold. The loss can be calculated using equation 3.9.

$$\mathrm{E}(w) = \sum_{j=0}^{N} (\mathrm{X}_j w - y_j)^2 \tag{3.9}$$

Algorithm 1 shows the procidure of the distributed Lasso. The optional step in line 12 in algorithm 1 depends the purpose of the Lasso. If the purpose is to do feature selection, the author recommends doing step 12. However, if the aim is to predict, it is not necessary, as the weights have different importance, depending on the value of each w. Hence without step 12 the weights do the feature selection by themselves in a softer way, which might result in a better prediction.

Algorithm 1 Distributed Private Lasso

1: Given: predictors X, response y, weight decay λ, number of iterations τ
2: Initialize: w as zeros ▷ Initial guess for the weights
3: Let m denote all the data centers
4: **for** $t = 0 \rightarrow \tau$ **do**
5: Calculate the local gradient, $\nabla \mathrm{E}_i$, for each data center, initializing at w
6: **if** $\sum_{j=0}^{n_i} (\mathrm{X}_j w_t - y_j)^2$ in **any** data center $< \psi$ **then** ▷ ψ is some threshold
7: **break**
8: **end if**
9: $\nabla \mathrm{E} = \sum_{\forall i \in m} \nabla \mathrm{E}_i$
10: $w_{t+1} = \Gamma(w_t - \eta_t \nabla \mathrm{E}(w_t; \mathrm{X}, y))$ ▷ η_t, the learning rate (can be tuned)
11: **end for**
12: **remove** all features from **abs**$(w) < r$ ▷ r is some threshold (optional step)
13: **return** w

3.3.3 Principal Component Analysis

Principal component analysis (PCA) is a popular unsupervised algorithm used for dimensionality reduction, visualization, lossy data compression and feature extraction. The idea is to project the data onto a lower dimensional space, such that the variance is maximized. The directions that arise from PCA are orthogonal to each other. They form a new subspace, which is referred to as the *principal subspace*. Each of the dimensions that form the principal subspace are called principal components. The principal components are ordered such that the first principal component explains the most variance in the data, the second principal component explains the second most variance in the data, and so on [21]. From this it can be seen that the principal components are in fact a linear combination of the original data. Note, since the aim is to find the maximum variance, the data is generally standardized before using PCA. This is due to the fact that the scale of the attributes might vary a lot, hence the algorithm might take one attribute to have more variance, while the truth is that it only has greater scale.

The following analysis show how data containing d dimensions can be projected onto the principal components. Starting by projecting the data onto the first principal component, which can then be generalized for the d other dimensions. To do so, a d dimensional vector which explains the direction of the corresponding principal component is defined as u. Hence, the direction of the first principal component is denoted as u_1. Each data point, X_i, will be projected onto the scalar value $\mathrm{u}_1\mathrm{X}_i$, where the sample mean of the projected data is given by

$$\bar{\mathrm{X}} = \frac{1}{N}\sum_{i=1}^{N}\mathrm{X}_i. \tag{3.10}$$

The variance is given as

$$\frac{1}{N}\sum_{i=1}^{N}(\mathrm{u}_1^T\mathrm{X}_i - \mathrm{u}_1^T\bar{\mathrm{X}})^2 = \mathrm{u}_1^T\Sigma\mathrm{u}_1. \tag{3.11}$$

Where Σ is the covariance matrix

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(\mathrm{X}_n - \bar{\mathrm{X}})(\mathrm{X}_i - \bar{\mathrm{X}})^T. \tag{3.12}$$

Now the projected variance is maximized with respect to u. However, in order for $\mathrm{u}_1 \to \infty$ not to happen, a constraint must be set. As we only care about the direction and not the magnitude, we state that the u must be a unit vector. Thus $\mathrm{u}_1\mathrm{u}_1^T = 1$. To optimize this, a Lagrange multiplier a_1 is used. Thus we need to maximize equation 3.14 with respect to u_1.

$$\mathrm{u}_1^T\Sigma\mathrm{u}_1 + \mathrm{a}_1(1 - \mathrm{u}_1^T\Sigma\mathrm{u}_1) \tag{3.13}$$

To maximize equation 3.14, we set the derivative with respect to u_1 equal to 0.

$$\frac{\partial}{\partial u_1}\Big(u_1^T \Sigma u_1 + a_1(1 - u_1^T u_1)\Big) = 0 \tag{3.14}$$

This gives

$$2\Sigma u_1 - 2a_1 u_1 = 0. \tag{3.15}$$

Simplifying gives

$$\Sigma u_1 = a_1 u_1. \tag{3.16}$$

Equation 3.16 shows that a scalar times a vector is equal to a matrix times a vector. Hence, u_1 must be a eigenvector. Now we make use of the fact that $u_1^T u_1 = 1$ and multiply both sides with u_1

$$u_1^T \Sigma u_1 = a_1. \tag{3.17}$$

This shows that the maximum variance is acquired when u_1 is set equal to the eigenvector which is associated with the largest eigenvalue a_1. The next principal components can be found in incremental fashion. Note that each new direction that maximizes the projected variance must be orthogonal to the principal components which have already been chosen. If the objective is to project the data onto the first M principal components. It can be seen that they can be defined by the M largest eigenvalues $a_1, ..., a_M$ and their associated eigenvectors $u_1, ..., u_M$, of the covariance matrix S [21]. To project the data onto the principal subspace, the data is simply projected onto the eigenvectors.

From the discussion above it is possible to see that the eigenvalue describes the variance of each principal component. Utilizing this fact it is possible to calculate the amount of variance our new principal subspace is able to explain. Equation 3.18 shows how much variance the first M principal component is able to explain. This can easily be modified by changing the sum in the numerator to represent the eigenvalues used to from a principal subspace.

$$\text{Explained variance} = \frac{\sum_{j=1}^{M} a_j}{\sum_{j=1}^{d} a_j}. \tag{3.18}$$

3.3.4 Random Projection

Random projections is a method used in dimensionality reduction. The method projects the original data set X, with dimensions, $d\ x\ N$, to much smaller dimensions, $k\ x\ N$. This is done by introducing a new matrix, R, which has dimensions, $k\ x\ d$, where each column in R has unit length. Equation 3.19 shows the random projection.

$$\mathrm{X}_{kxN}^{RP} = \mathrm{R}_{kxd}\mathrm{X}_{dxN} \tag{3.19}$$

They key idea behind the random projection is that if points in a vector space are projected onto a selected subspace with high enough dimension, then the distance between the points is approximately preserved. Thus the method is useful when the distance in the data set X, is meaningful [22].

Lastly, note that the computational complexity of random projection method is very efficient [22].

3.3.5 Logistic Regression

Logistic regression is a classification algorithm although the name might suggest otherwise. It can do both binary and multi-class classification. The underlying mechanism of the logistic regression is the natural logarithm of the odds (equation 3.20). The odds are the ratios of the probability σ, that is the likelihood of y happening divided by the likelihood of y not happening [23]. Equation 3.20 shows the likelihood for the i-th target, y_i, happening. Here X are the predictors, and X_i is the i-th observation. In addition, X_{ij} is a feature of that particular observation, where the feature space ranges from, $\mathrm{j} \in \{0, 1, ..., k\}$.

$$\mathrm{logit}(\mathrm{y}_i) = \ln(\frac{\sigma_i}{1-\sigma_i}) = w_0 + w_1\mathrm{X}_{i1} + ... + w_k\mathrm{X}_{ik} \tag{3.20}$$

As shown in equation 3.20 there is a linear relationship between $\mathrm{logit}(y)$ and X. In addition to that the logit function ranges from $-\infty$ to ∞ [24]. This shows that the relationship between the weights is linear, and by making it range from $-\infty$ to ∞ the weights can take any value. However, in order to classify, a probability of a class is needed. By solving equation 3.20 for σ we obtain:

$$\sigma(X_i) = \frac{e^{z_i}}{1+e^{z_i}} = \frac{1}{1+e^{-z_i}} \tag{3.21}$$

where, $z_i = w_0 + w_1\mathrm{X}_{i1} + ... + w_k\mathrm{X}_{ik}$

The function shown in equation 3.21 is known both as the sigmoid function and the logistic function. The function can be viewed in figure 3.6. It has a nice S curve which asymptotes at both 0 and 1, and crosses the y axis on 0.5. This shows that the function acts like a probability function as it is always positive and is bounded between 0 and 1.

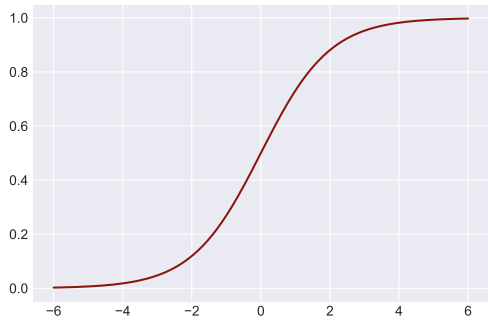


Figure 3.6: Shows the sigmoid function in the interval [-6, 6]

The maximum likelihood estimation (MLE) is used to train the logistic regression model. As the MLE chooses the parameters which maximizes a likelihood function given the observations, a likelihood function is needed. As each y_i can only take the values 0 or 1, the probability of getting 1 equals $\sigma(X_i)$ and thus the probability of getting 0 is 1 - $\sigma(X_i)$. Combining these two equations results in the expression $\sigma(X_i)^y(1-\sigma(X_i))^{1-y}$. If y_i equals 0 then the first part of the equation becomes 1 and similarly if y becomes 1 the second part of the equation becomes 1. This only applies to one observation, however since the observations are assumed to be independent the likelihood function can be expressed as the product

$$l(w) = \prod_{i=1}^{N} \sigma(X_i)^{y_i}(1 - \sigma(1 - X_i))^{1-y_i} \tag{3.22}$$

As equation 3.22 is hard to optimize, the negative log of the equation is taken for mathematical convenience. This results in the famous cross entropy cost function

$$E(w) \equiv l(w) = -\sum_{i=1}^{N} y_i \ln(\sigma(X_i)) + (1 - y_i) \ln(1 - \sigma(X_i))). \tag{3.23}$$

Now the values for the weights, w, can be found by minimizing $E(w)$. That is done by differentiating $E(w)$ and setting it equal to zero. However there exists no closed form solution for $E(w)$ (equation 3.30) [25] as the sigmoid function introduces non

linearity. Yet the equation can be solved using numeric optimization methods such as gradient descent or newtons method, to name some (section 3.3.7 shows some numeric methods). In addition the logistic regression cost function (the cross-entropy) is a convex function [26]. This is convenient since in convex functions the global minimum is the same as the local minimum.

Another thing which is nice about the cross entropy cross function is that it can learn reasonably fast. Inspecting equation 3.23 it can be seen that if the prediction is very wrong the logarithm term will get very large. Thus it penalizes mistakes harshly. This speeds up learning as the slope of the gradient will be steeper. Moreover, [21] shows that the the derivative of equation 3.23 is

$$\nabla \mathrm{E} = \sum_{i=1}^{N} \mathrm{X}_i(\sigma(\mathrm{X}_i) - y_i). \tag{3.24}$$

This shows that as the prediction is better the gradient will be smaller. Therefore with worse predictions the learning will become faster, and similarly the algorithm will learn slower for good predictions.

The method above shows the maximum-liklihood when the targets are either 0 or 1. However, there also exists in the literature the liklihood for when the targets are either -1 or 1. The second notion is used when finding the sensitivity for the regularized logistic regression in section 3.3.6. Note that the two methods are equivalent.

$$\mathrm{P}(y = 1|X) = \sigma(z) = \frac{1}{1 + \mathrm{e}^{-z}} \tag{3.25}$$

$$\mathrm{P}(y = -1|X) = 1 - \sigma(z) = \frac{1}{1 + \mathrm{e}^{z}} \tag{3.26}$$

Taking equation 3.25 and 3.26 together gives

$$\mathrm{P}(y_i = -1|X_i) = \frac{1}{1 + \mathrm{e}^{-y_i z_i}} \tag{3.27}$$

Now as above, the observations are assumed to be independent

$$l(w) = \prod_{i=1}^{N} \frac{1}{1 + \mathrm{e}^{-y_i z_i}} \tag{3.28}$$

Taking the negative log for mathematical convenience, the cost function can be expressed as

$$\mathrm{E}(w) \equiv l(w) = \sum_{i=1}^{N} \ln(1 + e^{-y_i z_i}) \tag{3.29}$$

As above, equation 3.29 introduces also non linearity with the sigmoid function. Hence, needs numerical optimization methods to achieve the optimal weights.

Equation 3.23 and 3.27 can be modified to include regularization term for both of the above mentioned methods, respectively. Equation 3.30 and 3.31 show the likelihood function with added regularization term, where λ is the regularization constant and the norm can by any norm. The regularization sum for both of the equations is over the feature space, P. In this theses a regularized logistic regression with differential privacy is discussed in section 3.3.6. There the $l2$ norm is used to regularize the model. The constraint surface made by $l2$ penalization forms a sphere (see figure 3.4). The optimal solution will then hit the contour of the sphere [18]. As stated in section 3.3.1 the idea behind the regularization is to penalize the model in order to avoid overfitting.

$$\mathrm{E}(w) \equiv l(w) = \sum_{i=1}^{N} y_i \ln(\sigma(X_i)) + (1-y)\ln(1-\sigma(X_i))) + \lambda \sum_{j=1}^{p} \|w_j\| \tag{3.30}$$

$$\mathrm{E}(w) \equiv l(w) = \sum_{i=1}^{N} \ln(1+e^{-y_i z_i}) + \lambda \sum_{j=1}^{p} \|w_j\| \tag{3.31}$$

The logistic regression predicts y using the sigmoid function. Thus the algorithm prediction yields probabilities, not discrete classes. This can be useful, as it makes it possible to distinguishes between predictions. This could be vital information when comes to some tasks, for example classifying something related to health care. Note, that although logistic regression only outputs probabilities it is very easy to use them to classify, by f.x. thresholding at 0.5 for binary classification.

3.3.6 Regularized Logistic Regression with Differential Privacy

In [27] Chaudhuri and Monteleoni reported a mechanism for regularized logistic regression model, using the $l2$ norm, that is able to preserve differential privacy (definition 1). The idea is to add Laplacian noise to the weights of a trained logistic regression model. However, as shown in section 3.2.3 to be able to provide deferentially privacy with the Laplace mechanism, an upper bound for the sensitivity needs to be calculated.

[27] use the fact that the cost function, $\mathrm{E}(w)$, of regularized logistic regression is strongly convex and differentiable at all points. By utilizing this, they are able to prove that an upper bound of the sensitivity can be calculated with equation 3.32. Here, λ is the regularization constant and N is the number of observations in the data set. Note that if this mechanism is to work, each predictor, $\|\mathrm{X}_i\| \in \{\mathrm{X}_1, ..., \mathrm{X}_N\}$ needs to be ≤ 1 [27].

$$\Delta f = \frac{2}{N\lambda} \tag{3.32}$$

To show that the bound of the sensitivity works, it is necessary to show that one observation will never influence the output more than shown in equation 3.32. For convenience the first part of the cost function shown in equation 3.31 will be referred to as E_1. Let's consider how this bound was found.

Let $\mathrm{X} = \{(\mathrm{X}_1, y_1), ..., (\mathrm{X}_N, y_N)\}$ and $\mathrm{X}' = \{(\mathrm{X}'_1, y'_1), .., (\mathrm{X}'_N, y'_N)\}$ differ by only one observation. In addition let $\mathrm{E}(w, \mathrm{X}) = \frac{1}{N}\sum_{i=1}^{N} \mathrm{E}_1(w) + \frac{\lambda}{2}\|w\|_2^2$. Here the $\frac{1}{2}$ in front of the regularization term is for mathematical convenience. This makes it possible to define: $\mathrm{G}(w) = \mathrm{E}(w, \mathrm{X})$ and $\mathrm{g}(w) = \mathrm{E}(w, \mathrm{X}') - \mathrm{E}(w, \mathrm{X})$.

For regularized Logistic regression $G(w)$ and $\mathrm{g}(w)$ are convex continuous differentiable functions. Therefore it is possible to define $w_1 = \mathrm{argmin}_w \mathrm{G}(w)$ and $w_2 = \mathrm{argmin}_w \mathrm{G}(w) + \mathrm{g}(w)$, then from [27], $\|w_1 - w_2\| \leq \frac{g_1}{G_2}$. Where $\mathrm{g}_1 = \max_w \|\nabla \mathrm{g}(w)\|$ and $\mathrm{G}_2 = \min_v \min_w v^T \nabla^2 \mathrm{G}(w) v$ for any unit vector v. This shows that the difference in the weights between the models is bounded by $\frac{\mathrm{g}_1}{\mathrm{G}_2}$.

Lets start by finding g_1. Note that it is possible to write $\mathrm{g}(w) = \frac{1}{N}(\mathrm{E}_1(w_N, X_N, y_N) - \mathrm{E}_1(w'_N, \mathrm{X}'_N, y'_N))$ because the only difference in the sum are N and N'. Moreover, both the norms are the same as well as all the other observations, hence they cancel each other out.

$$\begin{aligned} \nabla \mathrm{g} &= \frac{\partial}{\partial w}(\frac{1}{N}(\mathrm{E}_1(w_N, \mathrm{X}_N, y_N) - \mathrm{E}_1(w_{N'}, \mathrm{X}_{N'}, y_{N'}) \\ &= \frac{1}{N}(\mathrm{E}'_1(w_N, \mathrm{X}_N, y_N) y_N X_N - \mathrm{E}'_1(w_{N'}, \mathrm{X}_{N'}, y_{N'}) y_{N'} \mathrm{X}_{N'}), \end{aligned} \tag{3.33}$$

where

$$\mathrm{E}'_1(z) = \frac{\partial}{\partial z}(\ln(1 + \mathrm{e}^{-z})) = \frac{-1}{\mathrm{e}^z + 1}. \tag{3.34}$$

In equation 3.33 $y \in [-1, 1]$ and, as stated above, $\|\mathrm{X}_i\| \leq 1$. Looking at equation 3.34 it can be seen that $\|\mathrm{E}'_1(w)\| \leq 1$. This can be seen because e^z is always positive for any real z and $\lim_{z \to -\infty} e^z = 0$, hence $\|E_1(w)\| \leq 1$. Now it can be seen that for any w in $\nabla \mathrm{g}(w)$ the following applies $\|\nabla \mathrm{g}(w)\| \leq \frac{1}{N}\|\mathrm{X}_N - \mathrm{X}'_N\| \leq \frac{1}{N}(\|\mathrm{X}_N\| + \|\mathrm{X}'_N\|) \leq \frac{2}{N}$.

Let's consider G_2. Start by deriving the second derivative of $v^T \nabla^2 \mathrm{G}(w) v$, where v is a unit vector.

$$\frac{\partial^2}{\partial w}(\sum_{i=1}^{N} \ln(1 + \mathrm{e}^{-y_i w_i \mathrm{X}_i}) + \frac{1}{2}\lambda \|w\|_2^2) = \sum_{i=1}^{N} \frac{\mathrm{X}_i^2 y_i^2}{(1 + \mathrm{e}^{-y_i w_i \mathrm{X}_i})(1 + \mathrm{e}^{y_i w_i \mathrm{X}_i})} + v^T \lambda v \tag{3.35}$$

Let's look at each term in equation 3.35 separately. Starting with the term on the left. It's minimum value is equal to the maximum value of it's the denominator.

To find the maximum consider the limit: $\lim_{z\to\infty}(1 + \mathrm{e}^{-z})(1 + \mathrm{e}^{z}) = \infty$. Thus, $\lim_{z\to\infty}(\frac{\mathrm{X}_i^2 y_i^2}{(1+\mathrm{e}^{-z})(1+\mathrm{e}^{z})}) = 0$. Looking at the second term in equation 3.35. It can be seen that it results in λ because v is a unit vector, so $v^T v = 1$ applies. Note that this makes the function strongly convex on λ.

Inserting g_1 and g_2 into $\|w_1 - w_2\| \leq \frac{\mathrm{g}_1}{\mathrm{G}_2}$ results in $\frac{2}{N\lambda}$. This bounds the sensitivity of the regularized logistic regression [26], [27]. This bound only applies if the regularization term is included. That is due to the fact that the regularization term makes the cost function strongly convex. If it were not for the regularization term then $\mathrm{G}_2 = 0$ and therefore $\lim_{w\to\infty}(\frac{\mathrm{g}_1}{\mathrm{G}_2}) = \infty$.

Algorithm 2 shows the procedure to embed differential privacy to the regularized logistic regression model. There it is possible to see that the noise which perturbs the weights is drawn from the $\mathrm{Lap}(\frac{2}{N\lambda\epsilon})$ distribution. This noise provides ϵ differential privacy. Note that the privacy guarantee can change if there is risk that the mechanism is used multiple times due to the composability of differential privacy.

Algorithm 2 Regularized Logistic Regression with differential privacy

1: Compute the weights $w_1, ..., w_j$ for the regularized logistic regression.
2: Add independent Lapacian noise with $\mu = 0$ and $\mathrm{b} = \frac{2}{N\lambda\epsilon}$, to each weight.
3: Updated weights.
4: Predict on test data using the new weights and the sigmoid function.

3.3.7 Gradient Descent and Variants

3.3.7.1 Gradient Descent

Gradient Descent (GD), sometimes also known as steepest descent, is one of the most popular optimization algorithm for solving nonlinear optimization. The aim of gradient descent is to minimize a so called cost function, by selecting the optimal weights. To start off, the algorithm takes some random weights, and then iteratively changes the weights to improve the value of the cost function. To improve, the algorithm finds the derivate of the cost function, which tells the algorithm the slope of the cost function. The slope points in the opposite direction of the minimum. This indicates that the negative slope points towards the minimum [28]. However, it can be tricky to know how 'big step' one should take towards the minimum. By taking to big step opens up for the possibility of overshooting, while taking to small step might take long time to converge. This can be controlled by the *learning rate*. Equation 3.36 describes the update process, where w stands for the weights, η stands for the learning rate, t stands for the iteration count, and E stands for the cost function.

$$w_{t+1} = w_t - \eta \nabla \mathrm{E}(w_t) \tag{3.36}$$

GD dose not guarantee a global optima, as it depends on the starting point of the algorithm and the cost function. If the cost function is very complex, the algorithm might start in a valley which has a saddle point which is only a local optima, and from there converge to the local optima.

Looking at equation 3.36 one can see that although the learning rate, η, is constant, the change of the weights will differ depending on where they lie on the cost function. If for example the slope of the cost function is very steep, the gradient of the cost function, $\nabla \mathrm{E}(w_t)$, is high. At the same time, if the slope is low, the value of $\nabla \mathrm{E}_n(w_t)$ will be small. This points to the fact that if GD is far away from a minima, GD will take bigger step, simultaneously when GD is close to minima it will zig zagg slowly towards the minimum. Figure 3.7 shows how the algorithm on an one-dimensional cost function.

Although GD is relatively fast, as it only requires to calculate the first derivative, it is often two slow for machine learning applications. That is because the cost function in GD requires all the data points in the training set (see equation 3.37). Thus in each step of the algorithm a run trough all the training points is needed [21]. Algorithm 3 shows the procedure of the gradient descent. Note that the function, $\nabla \mathrm{E}(w_t)$, uses both the training data X, and y to calculate the error.

$$\mathrm{E}(w_t) = \sum_{n=1}^{N} \mathrm{E}_n(w_t) \tag{3.37}$$

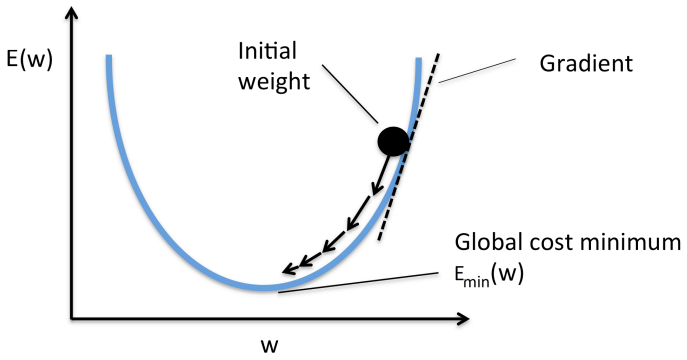


Figure 3.7: An example of how the gradient descent works. From the arrows in the figure it can be seen that if the algorithm is close to the optima the slope will produce small jumps. On the other hand if it is far away from the optima it will produce much larger jumps. Figure taken from [29].

Number of stopping criterion's exist for gradient methods. It can be seen that when the gradient is 0 the algorithm has converged. Hence, the model has stopped learning. Another thing which is sometimes done with gradient descent is to check if the cost difference is below a certain threshold. As the gradient can be very slow when it is close to a local minima, it can be convenient to stop before, as the training can take very long time. However, this can be cumbersome with stochastic gradient descent (discussed here below) as it's cost can fluctuate a lot between iterations. Finally, a maximum number of iterations are sometimes selected. Then the algorithm is terminated after it has done number of iterations.

Algorithm 3 Gradient Descent

1: Given random weights w, learning rate η.
2: **while** stopping criteria **is not** meet **do**
3: $\quad w_{t+1} = w_t - \eta \nabla \mathrm{E}(w_t)$ ▷ Update weights
4: **end while**
5: **return** w

3.3.7.2 Stochastic gradient descent

Stochastic gradient descent (SGD) is a variant of GD. Rather than using all the data for each learning step, SGD takes small subset of the data and learns from it. This subset is referred to as a mini batch. This way, an estimate of the gradient vector can be found using only subset of the data instead of using all the data, like the GD dos. This, indeed, speeds up the learning. The most important property of SGD is that the computation time per update dos not grow with the number of training samples [30]. This can be very useful as many data sets used in machine learning are very big.

The procedure of the SGD is the following: it starts by taking all the training data and split it up into mini batches. It then takes a random mini batch and learns from it. The algorithm then repeatedly takes another random mini batch until they are finished. This is known as one epoch of training. After one epoch of training is finished, the process is repeated. This is done until the corresponding stopping criteria is meet. Algorithm 4 shows the procedure in more formal way.

Algorithm 4 Stochastic Gradient Descent

1: Given random weights w and learning rate η.
2: **while** stopping criteria **is not** met **do**
3: Randomly split the data into m equally sized mini batches
4: **for** batch b in batches **do** ▷ one epoch of training
5: $w_{t+1} = w_t - \eta \nabla \mathrm{E}(w_t, b)$ ▷ Update weights
6: **end for**
7: **end while**
8: **return** w

The learning rate is crucial in SGD. [30] states that in practice it is necessary to gradually decrease the learning rate over time. This is due to the fact that SGD introduces noise in each iteration. The noise is because of both the size and randomness of the mini batch, which is used to calculate the estimated gradient. The noise can still occur if SGD is at a minimum. However, at a minimum, the gradient using all the data points will result in 0, and therefore will not move.

[30] goes on to state that the choices for the learning rate is more art than science, however when using linear schedule the final learning rate is usually set as 1% of the starting learning rate. Thus, the main problem will be to select the initial value for the learning rate.

It can be hard to select the right batch size. Larger batch sizes will provide a more accurate estimate of the gradient, but they take longer time to calculate. This can be seen in figure 3.8, where the true gradient is compared to a stochastic gradient. As the batches get bigger, the algorithm starts to resemble the true gradient and the fluctuations decrease. However, small batches can offer some regularization effect as

they introduce noise into the learning process. As SGD is often used on large data sets it can be convenient to make it highly parallelized. It is usually better to have the batch sizes in the power of 2 in order to utilize the hardware as much as possible. Note that the memory usually scales with batch size, making the memory a limiting factor for many architectures. Hence, the architectures are usually motivated by small batch sizes [30].

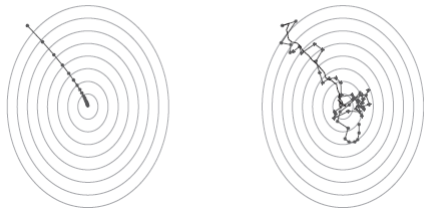


Figure 3.8: The left figure shows the convergence of a typical gradient descent method while the left figure shows the convergence of a typical stochastic gradient descent one. Taken from [31].

3.3.7.3 Gradient clipping

As stated above, the slope of the gradient is not constant. It depends on the solution space from where the gradient is found. Occasionally, this can be a problem. If, for example, the landscape around the gradient is very steep (like a cliff) where the minimum is right at the bottom of the cliff and then the surroundings changes quickly. If the gradient is on the cliff, the magnitude of the gradient will be very large. This will make the gradient overshoot the minimum and jump way too far. This could destroy all the prior work the algorithm has done.

A classic solution to deal with this is to use gradient clipping. There exist many approaches which deal differently with this problem. One is to clip the norm of the gradient just before the parameters are updated. Here a threshold for the gradient norm is set and if the norm of the gradient is bigger than the threshold then the gradient will be normalized using equation 3.38. By normalizing the gradient with a single scaling factor it is guaranteed to be in the same gradient direction. This bounds the gradient, therefore avoids taking too big jumps.

When using gradient clipping with SGD, a bias is introduced to the gradient which is known to be useful. In other words, although the direction of each mini-batch will not change direction, the true gradient will not be in the same direction as the average of all the mini-batches. Which is the case when no clipping is done [30].

$$\nabla E \leftarrow \frac{\nabla E_{\text{threshold}}}{\|\nabla E\|}. \tag{3.38}$$

3.3.8 Stochastic Gradient Descent with Differential Privacy

[32] implemented differential privacy (DP) into the stochastic gradient descent algorithm (SGD) , discussed in section 3.3.7. The SGD uses the weaker interpretation of DP, where two data sets are said to be neighbouring data sets if they differ in the n-th observation. For more discussion see section 3.2.

As mentioned in section 3.3.7, the SGD is a great algorithm for large data set which can be a very useful trait when doing machine learning. Many learning algorithms can use SGD for training, such as neural networks, support vector machines and logistic regression to name few. This shows that by making SGD differentally private, many other algorithms can simultaneously gain differential privacy.

The idea is to make the gradient updates differentially private. This can be done by adding noise to each step of the gradient. Equation 3.39 shows the procedure for SGD with batch size equal to 1, where Z_t is the noise added to the gradient in iteration t. By adding the $l2$ regularization term λ, the cost function becomes λ-strongly convex. This is an useful trait as strongly convex functions have some favorable theoretical guarantees [32].

$$w_{t+1} = w_t - \eta(\lambda w_t + \nabla E(w_t, \mathrm{X}_t, y_t) + Z_t) \tag{3.39}$$

[32] claims that equation 3.39 can be rewritten to take the batch size, b, into account. Note that the noise is divided by b as the pair of neighboring data sets only differ in one observation. Therefore, the gradient in each iteration will differ at most by the gradient of one observation [33]. This means that most of the time the data points which form the gradients are the same between the neighboring data sets, as only one observation in both data set differ. These gradients will then be averaged over the batch to take a gradient step. This reveals that in a batch the gradient between the data set will differ by at most two observations. Therefore, the noise only needs to account for that maximum difference.

$$w_{t+1} = w_t - \eta(\lambda w_t + \frac{1}{b} \sum_{X_i, y_i \in B_t} \nabla E(w_t, \mathrm{X}_t, y_t) + \frac{1}{b} Z_t) \tag{3.40}$$

Looking at equation 3.40 one can observe that if many observations are added to a batch less noise is added to each observation. However, it also makes fewer iterations in each epoch. One can see that if more observations are in a batch, then less randomness is needed to hide the participants in the data set as the average will drag the observations closer together. Similarly equation 3.40 claims that if fewer observations are in a batch more noise is needed. Note that for small batches both more iterations will be taken in each epoch and more randomness will be added in each iteration. On the contrary, if the batch is too big, then the algorithm might take too few steps. Therefore, there is a clear trade-off in terms of the batch size.

The noise added to each update is drawn from the density shown in equation 3.41. [32] states that by generating a sample from this density, the algorithm can be proven differentially private. However, to be able to sample a random number from

the density one needs to use spherical coordinates and the gamma distribution. The procedure is shown in algorithm 5. For further analysis of the density see section A.1 in the appendix.

$$p(z) \propto \mathrm{e}^{-(\epsilon/2)\|z\|} \tag{3.41}$$

For the statement above to be true some constraints need to be made. First, note that the mechanism assumes that for all predictors X_i applies $\|\mathrm{X}_i\| \leq 1$. Second, the gradients must be bounded as they can take on any value. [32] states that the gradients of the cost function can not exceed 1, hence $\|\nabla \mathrm{E}(w)\| \leq 1$. The proof made by [34] can be seen in section A.1 in the appendix.

The global sensitivity of each update can now be found as $\frac{2\eta_t}{b}$ [32]. This can be seen by looking at equation 3.40 in conjunction with the fact that the gradient is bounded. As stated above, the maximum difference which can occur in a batch is if one observation is changed for another one. Therefore the difference in the gradient values between neighbouring data sets can be at maximum 2. This is because if the $\|\nabla \mathrm{E}(w)\| \leq 1$ and the two observation are going in the opposite direction, then the difference in their gradients will be 2. Looking at b it can be seen that it comes from the fact that there are many observations in a batch. As more observations come into play, they help hiding the difference between the observation as the average will drag them closer together. Finally, as every update is multiplied by η_t, the global sensitivity will be $\frac{2\eta_t}{b}$.

Algorithm 5 Noise generation [35]

1: Sample uniform vector in the unit ball, v
2: Sample l from the Gamma distribution, $\Gamma(d, 2/\epsilon)$
3: **return** k = l*v

As each batch in the SGD is disjoint, the algorithm follows the parallel composition introduced in section 3.2.2. Therefore, each epoch is $\epsilon = \max_i \epsilon_i$ differentaly private. However, if more than one epoch are taken, then each data point will be used more than once. Thus, the sequential composability applies. From this it is possible to see that if the batch size is 1, then the $\epsilon = \max \sum_j \epsilon_j$ in the worst case. This also applies if the batch sizes is bigger. This can be seen by imagining that the adversary has all but the N-th data point, and knows how the batches are randomly sampled, he/she can replicate that. Hence, the sequential composability also applies for batches having more than one observation.

As the method provides privacy for each batch it is said to be locally differentially private. This differs from the generally differential privacy as the data scientists will never touch the raw data but instead gain some perturbed version of it. Therefore, the local differential privacy proposes a stronger form of differential privacy than the ordinary one [36] [34].

Since the method provides local differential privacy it can be leveraged into a distributed framework. Now that each noisy gradient is differentially private, it is

possible to send it to different locations without privacy concerns. This of course demands a sensible ϵ to be chosen. If ϵ is too high it can leak information as discussed in section 3.2. If the reader is interested in a distributed framework for the SGD with differential privacy, the author points to [34]. There the same density is used for noise generation, but the locations have different privacy budget in each location.

3.3.9 Model Evaluation

3.3.9.1 Hold out

In the hold out method, the data set is split into training and test sets. The training set is used to train a machine learning model and the test set is used to evaluate how good the model is. This method is not good if our data set is small. This is because if the data set is too small, the training data may not represent the underlying distribution of our data [25].

3.3.9.2 K-Fold Cross validation

Cross validation is another method used to evaluate a machine learning model. It works by randomly splitting up your data into k-folds, where one fold will represent the test error while the other folds the training data. All folds will be both testing and training samples. Thus, k number of models will be built with k different test sets. Each model predicts on its assigned test set and stores the result. The model is then evaluated by taking the average of the k number of results [18].

It differs with what k is selected. A special case is when K = n, then the folds are equal to the number of observations in the dataset. This special case is refereed to as *leave one out cross validation*. Using leave one out, all but one data points are used to predict for the one point left out. This kind of cross validation is as unbiased as it gets, as all data points but one are used to train the model. However, the model can catch a lot of variance, as many of the training samples are very similar to each other. In addition to that, the leave one out method is much more computationally demanding, as n number of models need to be trained. Selecting k = 10 or 5 can also be cumbersome. Here the model has less variance but more bias, as fewer number of points are used to train the model [18].

To select the correct value for k, one needs to look at the size of the data set. If the data set is very small, then k = 5 will have much bias on the model. However, if the data set is large, the 5 fold cross validation will not introduce much bias since it has enough data to generalize well [18]. Thus, in general one needs to think about the size of the data set as well as the signal to noise ratio when applying cross validation. With that in mind, [18] recommends using 5 or 10 fold cross validation in practice.

3.3.9.3 Grid Search

Many machine learning models use hyperparameters which need to be adjusted so the model can perform optimally. A Grid search is a technique to tune the hyperparameters of a model. If a model only needs to tune one hyperparameter, the grid search simply goes through all values to be tried for that particular parameter, trains a model using each value of the parameter and evaluates the model using *cross validation* or something alike. The search selects the parameter which result in the model having the lowest validation error [37].

If there are more than one hyperparameters to tune, a grid of parameters will be formed; hence, the name grid search. This means that the search goes through all combinations of parameter values and selects the best pair of parameters.

Note that the grid search becomes computationally heavy as the number of hyperparameters to tune increases. This is due to the fact that the number of combinations to try grows exponentially, or number of trails $= \prod_{k=1}^{K} |\text{Loss Function}^{(k)}|$. This product over the hyperparameters indicates that grid search suffers from the *curse of dimensionality*. However, it is reliable in low dimensional space [37].

CHAPTER 4

Methods

4.1 Tools

4.1.1 Python

Python is a potent programming language with large echo system around it. The language is open sourced and has a lot of packages. Furthermore, it is gaining popularity within the machine learning community and is starting to be one of the most popular languages for developing machine learning [38]. Python has a very clean syntax which is vastly convenient when doing complex machine learning. The package used to develop the python applications in this thesis are:

- **Numpy** is a powerful library for scientific computing with Python. A lot of the library is implemented in FOR-TRAN or C, two fast programming languages. The core of the library is the ndarray, which is both fast and suitable for matrix operations [39].
- **Scipy** is a another important library for scientific computing. It is built on Numpy and has a lot of modules for optimization, linear algebra, statistics and other general functions [40].
- **Scikit-learn** is a great machine learning library. It can resolve both unsupervised and supervised problems as well as all sorts of feature selection and extraction [41].
- **Pandas** is a very useful library for data manipulation. Its main object is the DataFrame, which simulates the data frame from the programming language R [42].
- **Matplotlib** is one of the fundamental libraries to make plots in Python [43].
- **Seaborn** is a visualization library which builds up on matplotlib. It makes the plots much more attractive as well as generates an interface to do some high level plotting [44].
- **Collections** is a module which contains a lot of nice data structures alternative from the Python general data structures [45].

- **Multiprocessing** is a package which allows running multiple Python processes simultaneously. It escapes the global interpreter lock in Python, enabling the program to take advantage of multiple processors [46].
- **NUMBA/Cython** Both of these libraries transform Python code to C. Numba does it on the fly, while Cython compiles the code. In Cython, variables can also have static type to make the program run faster. Both of these libraries were tried and not used. That is because Numpy is extremely fast, and using Numba or Cython did not result in any notable speed up. However, it is worth noting that both of these libraries are in general suitable for speeding up a Python code [47] [48].
- **Json** is a format that stores data as a text. It is organized as a JavaScript object, which makes it easy to transfer the data between modules [49].

4.1.2 Bash

Bash is the shell or the command line tool in GNU. It has a lot of functions and is a quick way to investigate the contents of files or outputs. Many of these commands can form a recipe in a shell script. Typically, the shell script is used for investigating and/or chaining files, executing programs and printing text [50].

4.1.3 High Performance Cluster

The High performance cluster (HPC) in DTU was used to run the code. The reason for running it on the HPC is that the implemented methods were computationally demanding. That is because the data set was quite big, many models were built to generate an error rate and tuning multiple parameters is expensive. Not only is the cluster faster in general, it also gives the opportunity to parallelize the code heavily. Therefore, more runs can be done in each experiment approximately at the same time. To run a code on the HPC, one needs to submit a job file to the machines. In the job file, one needs to specify the amount of time the job is running, memory limit, number of cores and finally how many machines the code should run on. The stats of the HPC cluster for the new LSF HPC-setup, which the code was run on, can be seen here below.

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    1
Core(s) per socket:    12
Socket(s):             2
```

```
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
Stepping:              1
CPU MHz:               1298.515
BogoMIPS:              4400.68
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0-11
NUMA node1 CPU(s):     12-23
```

4.2 Data

4.2.1 Mnist

The Mnist data set is a classical data set where many well known benchmarks from different machine learning algorithms exists. The data set was made by Yann LeCun, Corinna Cortes, Christopher J.C. Burges and received from their website [51]. It contains images of handwritten digits as well as their labels. The data set contains 60.000 training samples and 10.000 test samples. Each image is $28 * 28$, resulting in 784 features. The targets span from 0 to 9. Each feature represents a pixel in an image. The value of the pixels represent the color of that pixel, 0 for black, 255 for white, and everything between a mix of black and white (e.g gray = 127.5). After downloading the images, a code used from [52] was run to get the data into CSV format. Figure 4.1 shows the images for each target.

As each pixel represents a color, it is interesting to examine how the color of the pixels distribute over all the images. Figure 4.2 shows this distribution. It shows that most of the data points are black, which makes sense when looking at the example figure above (figure 4.1). At the end of figure 4.2, there is a spike where the white colors come into play. In addition to that it is possible to see from the figure that there is not a lot of gray in the images, as there are only spikes at the extremes on the x axis.

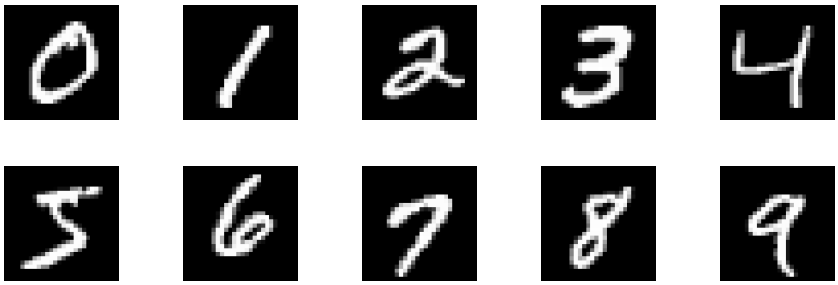


Figure 4.1: Shows an example of the images to predict for. The label of each of the images is in numerical order.

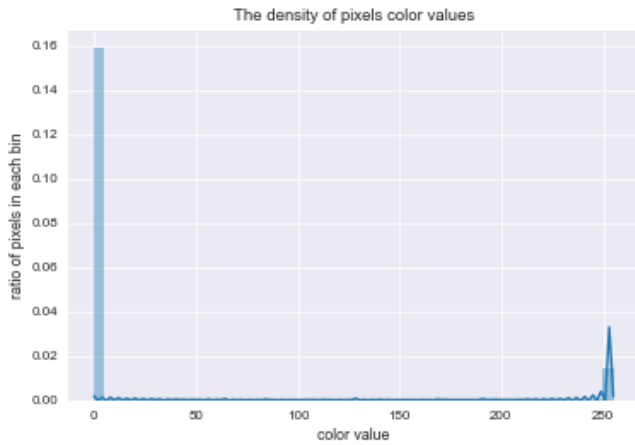


Figure 4.2: Shows the density of the color values of each pixel, it is visible that most of the pixels are black.

Next, an exploration of how the images differ from the typical image was made. The typical image was calculated by taking the mean of every feature for each target, resulting in 784 means, one for each dimension. To see how each image differs from the typical one, the euclidean distance from each image towards the typical image was calculated. The results are shown in the violinplot in figure 4.3. The highest points are the images that are furthest away from the typical image, in each category. It can be seen from the figure that the 1 is the number which is drawn most alike. On the other hand, it seems like 0 and 2 have the most variations in how they are drawn, as the mass of these categories in the violinplot appears the highest. Lastly, it appears as 8 has the single worst drawing since it has the highest spike in the figure.

The idea behind the methods generating figures 4.2 and 4.3 are taken from [53].

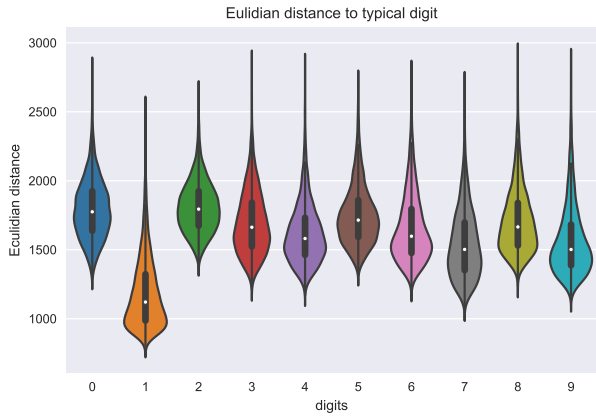


Figure 4.3: Shows the euclidean distance for every image to its typical image in each category.

Figure 4.4 shows the 5 images having the greatest euclidean distance from the typical image in each category. It is clear when viewing the images that some of the numbers are drawn very poorly. Furthermore, some of the numbers in the images do not look anything like the number, for example the first 7 and 3.

A simple count of the targets was made, too see if there was any class imbalance in the data set. Table 4.1 shows that there is not.

Table 4.1: The count of all target data points in each category.

	0	1	2	3	4	5	6	7	8	9
Training samples	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949
Test samples	980	1135	1032	1010	982	892	958	1028	974	1009

The data was standardized since the machine learning algorithms react differently to unstandardized data. For example, the *Distributed private Lasso* uses gradients which are sensitive to standardization. By standardizing the gradient, the step size will differ much less, as the magnitude of the step size will not be as influenced by the data. This way, the gradient only decides the optimal direction, while the learning rate, η, decides the magnitude of the step size.

As the data is high dimensional (784 dimensions) it is hard to visualize. A *principal component analysis* (PCA) was therefore applied to visualize the data. Note that the data has already been standardized, but that is important for the PCA to work

correctly. Otherwise features with greater scale will be said to explain more variance, than they do in reality. Section 3.3.3 offers a PCA discussion in greater detail.

Figure 4.5 shows all the training examples projected onto the first two principal components. From the figure it is hard to see a difference between variables as the plot only explains 9.7% of the variance of the original data. However, it is possible to see how similar numbers group together. For example one can see how 7 and 9 are similar, as well as 0 and 7 are dissimilar.

However, as the aim is to do a binary classification in both section 4.3 and 4.4, it is worth visualizing the two interesting classification problems, using PCA. Note, that two different data sets have been sampled to make the PCA, one which only contained 4 and 9 and the other only contained 0 and 1. Both of the data sets where projected onto their first two principal components. Figure 4.6 shows the results. These two classification problems were chosen because the other believes that classifying between 4 and 9 is one of the hardest binary classification problems generated from the mnist data set, while classifying between 0 and 1 one of the easiest.

The left figure shows the PCA results for the projection of 4 and 9 onto their principal components. Here light turquoise points are of class 9 and purple of class 4. The points are located in very similar spots, if there was no prior knowledge of which class the point belongs to it would be hard determine its class. The results here only explain 12.7% of the variance.

The right figure shows the results for the projection of 0 and 1 onto their principal components. Here the data spreads a lot smoother. It would be possible to label the classes with no prior knowledge. Here 16.2% of the original variance is explained in the first two principal components. It can be seen that although both 0 and 1 spread well on both of the components, the 0 has much more variance. That is logical, as the images of 0 have much more white in them, (that is the 0 is more scattered over the image). It can also be seen that the first principal component distinguishes the classes quite well, as most of the 0 have bigger values than the 1.

Thus, looking at the figures we see that it is easier to discriminate between 0 and 1 than 4 and 9, which match prior believes. To further support that claim, figure 4.7 shows how the explained variance increases with principal components. Seemingly, the 0 and 1 need much fewer components to describe the data.

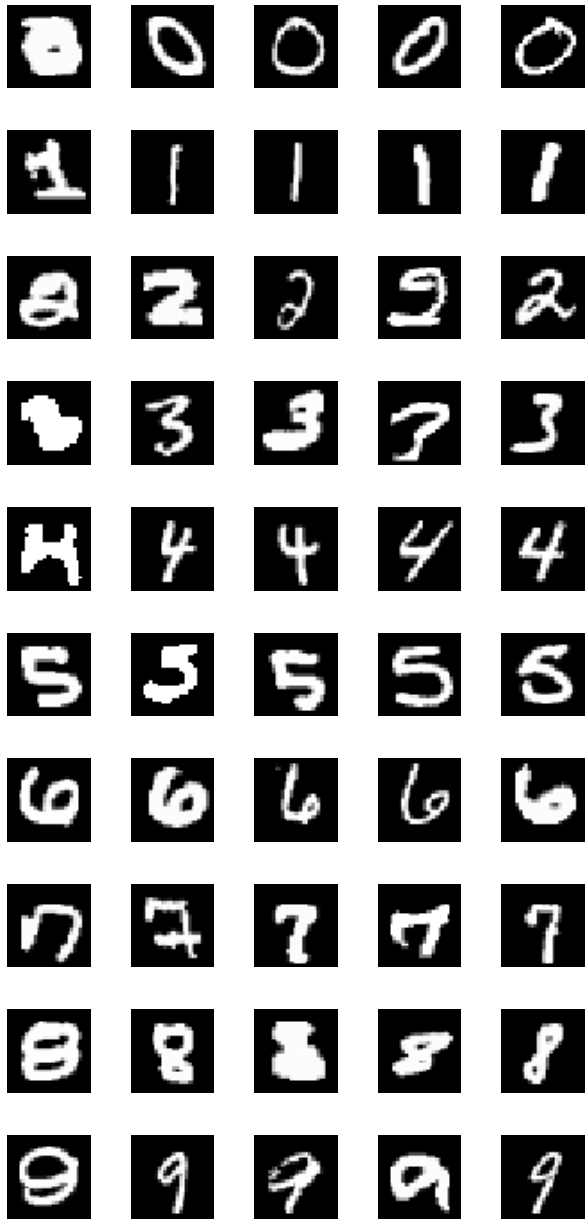


Figure 4.4: Shows the images which are furthest away from the typical image. The distance is based on the euclidean distance.

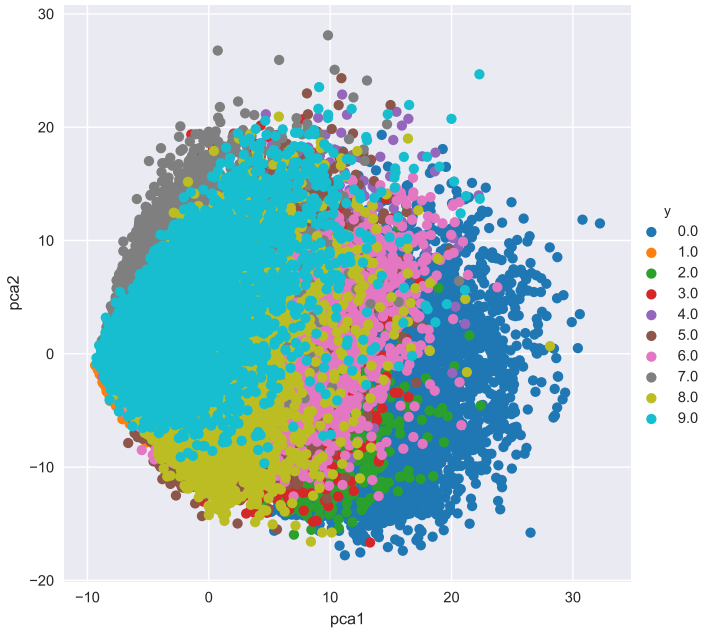
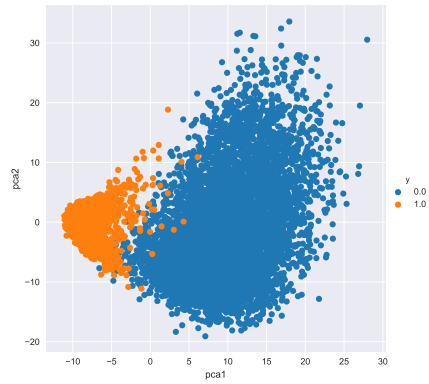


Figure 4.5: PCA of all target variables.

4 and 9 scattered onto their first two principal componenets.



0 and 1 scattered onto their first two principal componenets.

Figure 4.6: A PCA of two datasets, one including only the labels 4 and 9, the other only including labels 0 and 1.

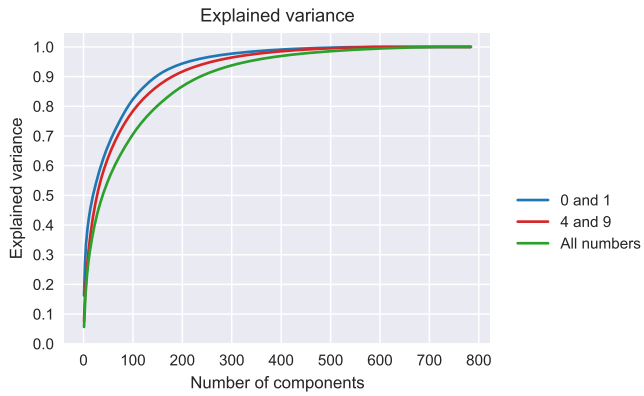


Figure 4.7: Shows the increase in explained variance by accumulating over the principal components. Note that each result, has different data, where the blue one only contains data with 0 and 1, the red one only 4 and 9, and finally the green one contains all of the targets.

4.3 Distributed Private Lasso

A simulation of the distributed private Lasso (discussed in section 3.3.2) was built. The model generated was used for binary classification problems. It was decided to classify between 4 and 9, which the author believes is the hardest classification problem generated from the mnist data set. In order to classify, the targets were mapped to -1 and 1 to represent 4 and 9 respectively. This makes it possible to use the $\texttt{sgn}(\hat{y})$ function to discriminate between classes. If the result is negative the data point is labeled as 4 and similarly, as a 9 if it's positive.

As stated in section 3.3.2 the idea is to send gradients to a central location, add all off them together, and then apply equation 3.7. After that a summarized gradient is sent to the data centers, which acts as a starting point from where the next gradient step will be taken from. A more detailed description of the theory behind the model can be found in section 3.3.2.

To better understand how well the distributed private Lasso is doing, two other models were trained. The first one simulates if only one data center decides to do the analysis on its own, having only half amount of the data, and the second if all the data was located at one data center. These two models use the exact same algorithm, but do not send the gradients to a central location. Instead, they perform those calculations privately. As many models will be built in our analysis, we will refer to these models as methods, and then each of these methods can have multiple models, which use the same procedure.

The prior idea is that when limited amount of data is available the distributed algorithm is better than using only one data center with half of the available data, but worse than having all of the data at one data center. Then as the data increases the models should converge with the same error rate.

4.3.1 Procedure

Each model has a number of parameters which need to be adjusted. First for the stopping criteria, when should the model stop? It would be optimal to stop when the model has converged to the lowest possible saddle point. However, this can be hard, as convergence can take a very long time. Thus, it was decided to use three different stopping criteria. A upper limit of the number of times the data centers can communicate (maximum iterations), a minimum tolerance for the cost difference between iterations, and a threshold for the total loss. The tolerance between cost functions was set to be 10^{-20}. The max iterations were set to be 3000 and the threshold for the total loss was set as 10^{-4}. The hope is that the model will be able to stop on the threshold for the total loss. The reason being, that the founders of the algorithm [20], only claimed to be using such stopping criteria.

A grid search was used to tune all of the models. The two parameters which were tuned where the weight decay λ and the learning rate η. The grid search used

5−fold cross validation to evaluate the models. Note that a test set was put aside, and the training data was only used in the training phase. This process was simulated multiple times and each time the data was shuffled to introduce randomness. This way, a better estimate of how well the models are doing on average is generated. Each simulation returns the model with the lowest validation error as well as the λ and the η used to generate that error. After receiving the results from all the simulations, the best parameters were selected as those most frequent for each model. However, in case of ties, the one with the smallest validation error was selected.

After the models have been trained and optimal parameters selected, the models are trained once again on all of the training data using the optimal parameters selected from the training process. The models then predict for the test set which was held out from the training process. The results are simulated multiple times to gain better estimate of the generalization error.

In the following sections two results are presented, one were principal component analysis is applied to the data before it is split between the data centers and another which only standardizes the data and then splits it.

4.3.2 Distributed Lasso with Principal Component Analysis

A distributed Lasso using principal component analysis was built. In this section the data is projected onto the principal components before splitting it between the data centers. There are two reasons for projecting the data onto the principal components. Firstly, the gradient can be very sensitive in high dimensional space. Secondly, it makes the training much faster. To do so, the data was first standardized so that the principal component analysis would work as expected (see section 3.3.3). The data was projected onto the first 100 principal components, which explain around 80% of the variance form the data. Note that principal component directions were formed by the training data, but both the train and test data were projected onto those directions.

4.3.2.1 Training

For the models generated after the data had been projected onto the first 100 principal components, two grid searches were performed. The first search was a broad search to investigate the solution space. The parameters used there can be seen in table 4.2. The result for all models indicated that most of the λ were very small, or 10^{-20} to be exact, however they ranged up to 0.01. All of the η were in the range $\langle 0.1, 0.01]$. It was therefore decided to run a sharper grid around the smaller λ as well as a sharper and larger grid for η. The grid can be seen in table 4.3. All of the parameters which were selected in the second grid were between the boundary points of the search except $\lambda = 10^{-20}$. However, as $\lambda = 10^{-20}$ is very close to 0 it was decided to overlook that result; hence, it was not necessary to perform another search.
This process was simulated 48 times. That was done because each time a simulation is run, different data points are used to calculate the models. Therefore to get the

Table 4.2: The parameters for the first grid search for the PCA models.

λ	10	1	0.1	0.01	10^{-5}	10^{-10}	10^{-15}	10^{-20}
η	1.3	1.1	0.9	0.7	0.5	0.3	0.1	0.01

Table 4.3: The parameters for the second grid search for the PCA models. Only 5 values of λ were tested. X represent not a value.

λ	0.1	0.01	10^{-5}	10^{-10}	10^{-15}	10^{-20}	X	X
η	0.1	0.05	0.01	0.005	0.0001	0.00005	10^{-5}	$5 * 10^{-6}$

real underlying optimal parameters of the models the process was done multiple times. The parameters which occurred most often were selected as optimal for the corresponding model.

In addition to that it was noticed that the model converged very fast. Hence, it was decided to split the training data into 105 bins, where each method had a single model for every bin. The first 100 bins were made by accumulating equally over the first 1000 observations. The next 5 points were only done to show that the model had already converged. Note that in each process the observations are shuffled in the beginning.

4.3.2.2 Experiment Results

After the optimal values for λ and η were discovered, the models were trained on the whole training data set, and tested on the test set. Interestingly, all of the models stopped on the maximum iterations stopping criteria. That indicates that the threshold for the cost and convergence were to small. However as all the runs use the maximum iteration stopping criteria they are convenient to compare.

Figure 4.8 shows the error rate for all the models. From there it is possible to see that the models converge very fast. In addition, it can be seen that the model containing the full amount of available data and the model having only half amount of available data share some errors. That is because as the data grows, the data center containing only half amount of data gets the same amount of data as the method containing the full amount of available data had at some given period.

Comparing the distributed algorithm to the other two it can be seen that it is doing very well. Most of the time it is as good as having all the data in one central data center. This exceeds our prior beliefs. Looking at the right figure it is apparent that the distributed algorithm is at times even doing better than having all the data at one place. The reason for this might be that by adding the gradients, the model will regularize a bit, which can result in a better model. That is due to the fact that by splitting up the data between the data centers the model will have less variance and more bias, as the models are trained on fewer data points.

A close up of the first 20 data points.

All results on log axis.

Figure 4.8: Shows the error rates for the PCA models. It can be seen that the methods converge very fast with increased data. The results are based on 48 simulation. Both figures are plotted with 95% confidence interval. Note that the data center using only the half amount of the data, has in fact the half amount of what is represented on the axis. e.g. if the amount of data is 20, then the corresponding x-axis value represents that the data center containing all of the data has 20 data points, in the distributed framework, the two data centers split the 20 between each other, resulting in 10 data points for each, and finally the data center having only half amount has only 10 data points available.

From the analysis above it is possible to see that the distributed algorithm is doing very well in terms of classification. However, in terms of feature selection no feature is excluded. By looking at the absolute values for all of the weights, it was clear that no feature was lower than $9 * 10^{-9}$. A possible reason for this is the PCA dimensionality reduction. As the data is projected down to fewer dimension some information from the full data set will be lost. In addition to that, the new dimensions will be made of linear combinations of the old variables, and as the first principal components generate the most variance, they might as well be the dimensions which describe the problem best. This indicates that the PCA might in fact already have done some feature selection on the data, as it has excluded the features which have the least variance.

In addition, if the reason for using Lasso is to gain knowledge on what attributes are important, PCA might not be an ideal starting algorithm. That is due to the fact that when all the dimensions are linear combinations of the old dimensions it will be hard to read what attributes are meaningful. Although it might potently tell if some linear combination of variables are useful.

4.3.3 Distributed Lasso without Principal Component Analysis

The Lasso with the principal component analysis did not do well with regard to feature selection. Therefore it was decided to run the distributed Lasso without the principal component analysis. However, in this case tuning is much harder. The reason for that, is due to the fact that the computational complexity will be greater, as instead of calculating the gradients for 100 dimensions, the gradient will be composed of 784 dimensions. Additionally, as the feature space is much bigger, it is harder to find the optima, as the gradient travels in more dimensions.

4.3.3.1 Training

Two grid searches were performed. As previously, first a broad grid search was performed (see table 4.4). The results from the grid included both of the extremes in λ. When taking a closer look at the selection it was clear that in the beginning more regularization was preferred. In other words, all models which had more than 360 data points, selected the two smallest λ. The other models which had less than 360 data points fluctuated around the other λ. Therefore it was decided to run a larger grid search on the first 360 data points, (table 4.5), and only search between the smallest λ for the models which had more than 360 data points. In the case of η all of the models chose 0.01 as the optimal learning rate. Therefore it was decided to enlarge the grid for the learning rate and make it sharper (table 4.5). All parameters which were chosen in the final models were inside the boundary points of the grid except $\lambda = 10^{-20}$, but as that is very close to 0 it was decided that there was no need for another grid search.

Table 4.4: The parameters for the first grid search.

λ	5	1	0.01	10^{-5}	10^{-20}
η	1.5	1.0	0.5	0.1	0.01

Table 4.5: The parameters for the second grid. Note that for models which contain more than 360 data points, only the three smallest λ were searched.

λ	15	10	5	1	0.01	10^{-5}	10^{-10}	10^{-20}
η	0.1	0.05	0.01	0.005	0.0001	0.00005	10^{-5}	$5 * 10^{-6}$

As these models were much slower than the PCA models, this process was only simulated 24 times. There were also fewer models made in every simulation. Instead of dividing the data into 105 bins it was decided to split it up into 53 bins. The first 50 points accumulated equally over the first 1000 observations. The final 3 points were only done to verify that the model had already converged.

4.3.3.2 Experiment Results

As in the above research all of the models stopped on the max iterations stopping criteria. This again, indicates that the threshold for the cost and convergence could be smaller.

Figure 4.9 and 4.10 shows the error rate for the models. It can be seen that the distributed algorithm is doing very well. It even outperforms the data center containing all the data. The reason for this might be the same as for the PCA models. That is, by splitting the data between the data centers, a bias is introduced. This regularizes the model by a very small scale. Therefore, the regularizing effect of splitting the data between the centers results in a little better generalization error. However, the error rate of the models is similar.

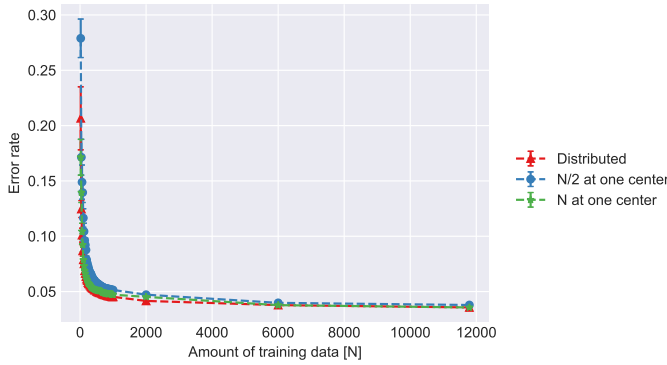
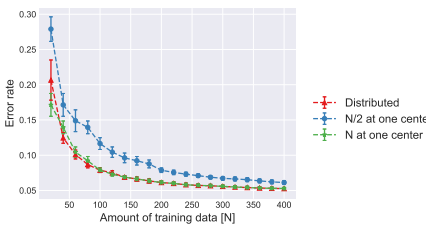
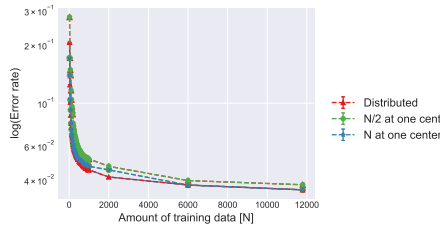


Figure 4.9: Shows the error rate for the distributed Lasso without PCA.

It can also be seen that right in the start, the distributed algorithm is doing worse than the data center containing all of the data. This could potentially be because if there is a big difference between the gradients, they will drag each other in their direction. If one of those gradients is big, it will influence the other gradient to become big as well. This might make the gradient go far from the optima and therefore it could take many iterations to get to a good neighbourhood.



The error rate for the first 20 models.



The error rate for all models on a log scale.

Figure 4.10: Shows the error rate for the normal models. The results are based on 24 simulations, and are shown with a 95% confidence interval.

Figure 4.11 shows the feature selection for the models. It was decided to discard all the features which had absolute value less than 10^{-100}. From the figure it can be seen that 166 features are discarded. Here the feature selection works much better than for the PCA models, where all the features were somewhat important. Intuitively this makes sense as the PCA only uses the linear combinations of the variables that form 80% of the variance. Most of the variables in the PCA are important to describe the image. However, without using the PCA the feature space is much bigger. Naturally some of them do not help in any way separating the classes.

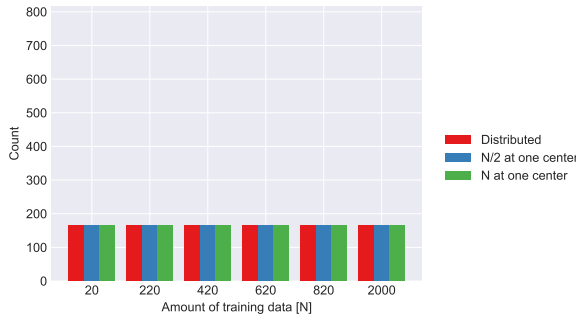


Figure 4.11: Number of weights having lower absolute value than $= 10^{-100}$.

It is intresting to see that no matter how many data points the methods have, they all exclude the same amount of features. To look at this in clearer light, figure 4.12 was made. The figure shows the features which were excluded from the distributed Lasso when the maximum amount of data is used to train the model. The red color in the figure stands for the pixels which have weights less than 10^{-100} and cyan are

the weights with the highest absolute values. Looking at the figure it is clear that the pixels which are colored red have no predictive power. This can be seen as most of the time nothing is written in these places. Hence, it is black in every or almost every picture. This also describes why the Lasso algorithm was not able to do any feature selection when the PCA analysis were performed beforehand. It is clear that these pixels have little or no variance.

It is also interesting to look at the pixels having cyan color. These are the pixels which the model considers most important for discriminating between 4 and 9. Looking at them it is clear that they appear where the difference between 4 and 9 is.

shows a image of 4

shows a image of 9

Figure 4.12: Shows the feature selection for the distributed Lasso, containing the maximum amount of data. The red color shows the features which are excluded, while the cyan shows the features with the greatest predictive power. Note that the 4 shows the best 30 features, while the 9 shows the best 15 features.

4.3.4 The Program

The code was run on the HPC data center in DTU. Six scripts were constructed: `main.py`, `tuning.py`, `computer.py`, `optimization_algorithms.py`, `plot_results.py` and `get_count_stop_crit.sh`. Figure 4.13 shows how these scripts are connected.

The `computer.py` is a class which represents a data center. Two instances of this class are made in both `main.py` and `tuning.py`, respectively. By using these classes, the data can be divided between them. Furthermore, each class can contain information which a real data center could hold onto. Those could be previous cost, if the cost is small enough and how many data points are in the data center. As each data center can hold onto previous cost, it is easy for them to find out if the cost has converged by comparing the previous cost and the current cost. The `optimization_algorithms.py` is used to find the gradients of the cost function.

The `main.py` and `tuning.py` are almost identical. The biggest difference is that the main script runs the parameters generated by a grid search in the `tuning.py`, and uses both the training and test data. In addition, the main file returns the results

as well as other interesting statistics to plot. Both of these scripts are considered as global. Meaning that there, the gradients can be summed and then transferred to the `computer.py` data clusters. To get a better estimate of the results, both the `main.py` and `tuning.py` are run multiple times. To save time this is run in parallel.

A print statement is put into every stopping criteria. All these prints are fed to a regular text file. Next, `get_count_stop_crit.sh` is run on the text file to count how many times each stopping criteria was meet.

Lastly, all of the results generated by each individual program are transferred using JSON files. The code can be found in section A.3

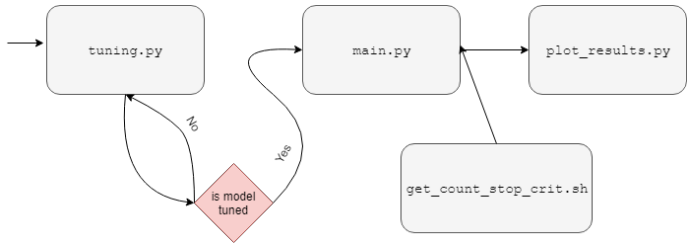


Figure 4.13: Shows a flow chart of the procedure. Here it is possible to see what faces the model goes through.

4.3.5 Privacy

Although [19] and [20] indicate that the algorithm is privacy preserving it is hard to state how much privacy it provides. As the gradient information is made public, it is a question of whether it can be reverse engineered, revealing some data from the data set. If the adversary had all the data but one observation in a data set, he/she could potentially find out some information about that observation by looking at the public gradient. Assuming that the adversary knows the cost function of the distributed private Lasso, he/she could calculate the gradient with all the observations except the missing one. That way, the two gradients could be compered to figure out the difference between the gradients, which would reveal information about a participant in the dataset. Of course if the adversary did not know the cost function which was used to calculate the gradient, it would be harder for him/her to gain information about a participant.

4.4 Regularized Logistic Regression with Differential Privacy

A regularized binary logistic regression with differential privacy was built using the mnist data set. As previously mentioned, the data set includes 10 numbers, from 0 to 9. In order to make binary classification, the author decided to use only the targets 4 and 9, which he believes is the hardest binary task available from the mninst data set.

The idea behind differentially private logistic regression is to add noise to the weights learned by the model. That is, first a logistic regression model is trained privately to find the optimal weights. From there the sigmoid function is used to predict the class label of each sample. To add privacy to the model, Laplacian noise is added to the weights before the sigmoid function is used. This perturbs the weights so their value change around the Laplacian noise. However, there is always the question of how much noise should be added to the weights. As stated in section 3.2 the amount of noise can be determined by ϵ and the sensitivity of the function used to publish information about the data. The ϵ is a hyperparameter, however the sensitivity can be calculated. In the case of binary logistic regression, [27] have shown that the upper bound of the sensitivity can be calculated with equation 4.1, where N is the total number of observations and λ is the weight decay.

It is interesting to see that the sensitivity is a function of N. If N increases, then the sensitivity decreases. This makes sense because if there are few users in the data set, then a user's data will have greater influence on the results. Take for example the mean of a group. If there are many participants, a user's result will have less impact than if there were few participants. In addition to that, if there is a lot of data, the influence of an outlier is much smaller. Thus, it should be easier to cover up the data with noise as the data increases.

It can also be seen that as the λ increases, the sensitivity decreases. This harmonizes well with N. This is because if the model is very regularized, then the individual data samples do not have much difference on the outcome of the model. Regularization forces the weights to change and if it is large, the model ends up only predicting the average which would result in predicting just one class in the case of binary logistic regression. This hides the participants in the data set in some way if the adversary does not know the regularization method and λ. Thus, if a model is very regularized, it needs less noise to hide its participants.

$$\Delta f = \frac{2}{N\lambda} \tag{4.1}$$

4.4.1 Training

Multiple logistic regression models were tuned to examine the relation between differential privacy and the amount of training data used to train the models. It was decided to train 50 models, where the first model was trained on $\frac{1}{50}$ of the training data, the second model on $\frac{2}{50}$ of the training data and so on.

The models were regularized with the $l2$ norm. This means that the weight decay, λ, for all the models needed to be tuned. Each model was tuned with a 5-fold cross validation and a grid search of λ. First, a large grid was used to get a sense of where the optimal λ is located in the solution space. The grid for λ spanned from $\frac{1}{128}$ to 128, where the interval between the values doubles between values, e.g. $\frac{1}{128}, \frac{1}{64}, \frac{1}{32} \cdots$ After gaining the results from the first grid search it was clear that the optimal λ for all the models were located around three possibles values of λ, namely $\frac{1}{4}$, $\frac{1}{16}$, and $\frac{1}{32}$. Thus, to further examine the solution space, another layer of grid search was preformed. This time it was observed that all the selected values from the first grid search span between $\frac{1}{64}$ and $\frac{1}{2}$. It was therefore decided to make a huge grid search which spanned from $\frac{1}{64}$ to $\frac{1}{2}$ with evenly spaced interval on a log_2 scale. At the end of the process, the best validation error and λ are returned.

The process described here above was simulated 48 times, for each grid search round. Each time the targets and the predictors were shuffled in order to achieve randomness when evaluating the intervals not containing all the data. The parameters which where most common were selected as the optimal parameters; however, if there was a tie, then the one with the lowest validation error was selected.

4.4.2 Experiment Results

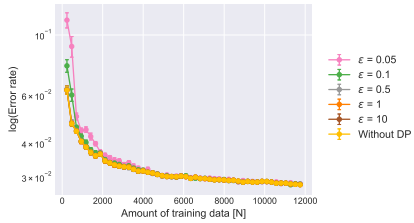
After the model has been trained, it is run to find the optimal weights for each N. This is simulated 48 times to get more robust results. After gathering the weights in each simulation two things occur. First, the model predicts on the test data using the weights from the logistic regression model. Second, a noise is generated from the Laplacian distribution with mean equal to 0 and scale equal to $\frac{\Delta f}{\epsilon}$. This noise is then added to the weights which are used to predict. If the scale of the noise is great, larger values will be added to the weights, resulting in more privacy and worse accuracy. Thus, there is a trade-off between accuracy and privacy. Note that a independent sample from the Laplacian distribution is drawn for each weight.

As the sensitivity can be calculated analytically, one needs to be very careful when selecting ϵ. Apple uses differential privacy in some of their applications. They claim that their privacy budget ranges between $\epsilon \in [2, 8]$ [54], although [55] disagrees and states that it ranges from $[1, 10]$. On the other hand, Dwork suggests that a good budget value is around $\epsilon \in [0.01, 0.1]$ [12], which is much safer. However, one must note that Apple uses more methods to gain privacy [54].

Figure 4.14: Shows the error rate of logistic regression model, both with and without differently privacy. The data is split up into 50 equally sized bins which are then accumulated to train multiple logistic regression models. The results are based on 48 simulations.

Figure 4.14 shows the error rate of the logistic regression model for different amount of training data. The sensitivity was calculated using equation 4.1. It can be seen that as the ϵ parameter is increased the error rate decreases. This makes sense since, as ϵ increases, less amount of noise is added to the weights. Thus, it has little or no impact at all on the error rate. The worst error rate is for $\epsilon = 0.0005$, where the error rate fluctuates around 0.5, equal to flipping a coin. Thus, it is indeed not performing adequately. The best results are achieved with $\epsilon = 1$ and $\epsilon = 10$. There, the error rate is very close to the one with the optimal weights for all the data intervals.

Another thing which can be seen from figure 4.14 is that as the logistic regression gets more data two things occur. The noise in the prediction decreases and the error rate drops. The noise decrease is due to the fact that as N increases, less noise is added to the weights, as the scale of the noise is calculated by $\frac{\Delta f}{\epsilon} = \frac{2}{\epsilon N \lambda}$. The error rate drop comes from the fact that as the learning model gets more data it improves. More data also means that less noise is added to the weights reducing the error rate further.



The best 6 accuracy's.

The best 8 accuracy's.

Figure 4.15: A close up of the error rates in figure 4.14 on a log axis.

Figure 4.15 shows a close up of figure 4.14. The figure on the right shows clearly how the noise generated by the sensitivity diminishes with increased data. In both figures it is clear that models with ϵ ranging from 0.5 and upwards are able to converge to the same error rate as the logistic regression with pure weights. This indicates that the amount of noise which is added to the weights is very small. The models with $\epsilon = 0.1$ and $\epsilon = 0.05$ also converge very quickly to the pure weighted results. However, as the magnitude of noise is not enough to influence the error rate, one might wonder whether this is indeed a private model.

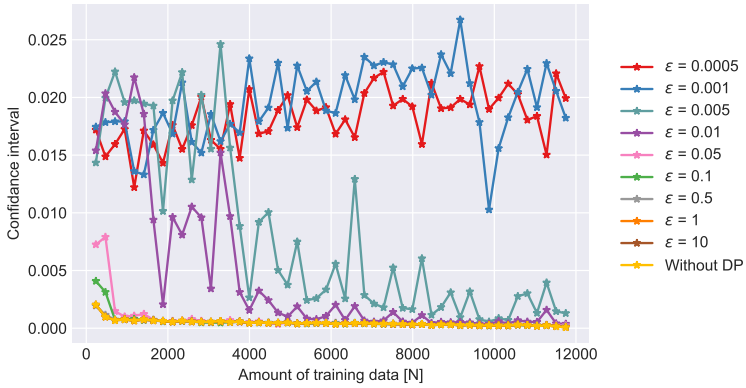


Figure 4.16: Shows the the confidence interval from the error rate.

Figure 4.16 shows the magnitude of the confidence interval from the error rate results. The red and blue lines, representing models with $\epsilon = 0.0005$ and $\epsilon = 0.0001$, show that their uncertainty increases with more data. The reason for this might be that in the beginning too much noise is added to the weights. This makes them confidently

predict some observations wrongly. However, as the amount of data increases less noise is added to the weights. This drives the weights to a point were they are unsure about some observations. Hence, their uncertainty will increase. Similar arguments applie to the other learning curves. Although they start by being unsure, as less noise is added to the weights, they become more and more certain.

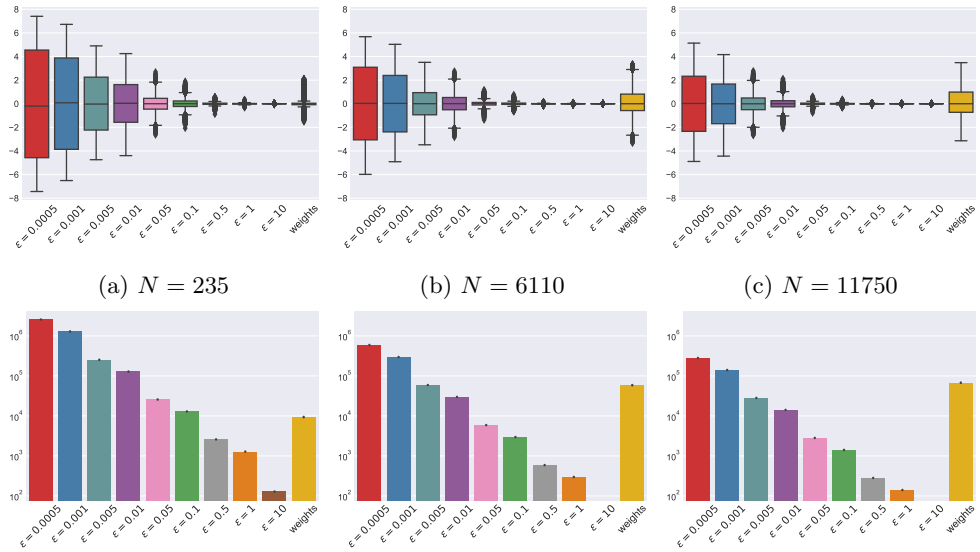
To analyze the behavior of the added noise in greater detail, figure 4.17 was created. The first row shows the distribution of the weights and the noises transformed onto $\sinh^{-1}$. The second row shows the absolute magnitude of the noises and the weights on a log axis. The first column shows the round where N was the smallest, the second column shows the middle N and the last column shows the final N. These three N where selected from the 50 different N as an example of how the noises change in comparison to the weights.

From the box plot it is possible to see how both the noises and the weights spread. It is apparent that the smallest ϵ has the biggest range. Thus, perturbing the logistic regression the most. Then as ϵ increases, both the magnitude and the variance shrink. This can be seen from both the bar plots and the box plots, respectively.

It is interesting to compare the magnitude in figure 4.17 to the error rate found in figure 4.14. If the magnitude for the first N in figure 4.17 is taken, for example, it can be seen that all the bars which are lower than the weights are able to converge right from the start in figure 4.14. If the next N is taken, then it can be seen that both the pink and the green bars have decreased and are now smaller than the magnitude of the weights. In the middle of figure 4.14 these ϵ have converged. The same applies for the final N.

The figures also show that different values of N result in different amounts of noise. This goes in hand with equation 4.1 as mentioned above, the larger the N is the smaller the sensitivity is.

Lastly, table A.1 in the appendix shows the variance of all the noises for each pair of N and ϵ. It also shows both the mean and the variance of the weights. By looking at these values it is possible to verify the discussion above.



(a) $N = 235$ (b) $N = 6110$ (c) $N = 11750$

(d) $N = 235$ (e) $N = 6110$ (f) $N = 11750$

Figure 4.17: Comparison of the weights and the noises for models with different number of training samples. The first row contains the distribution of the noises and weights transformed onto $\sinh^{-1}$. The second row shows the absolute magnitude of the noises and the weights on a log scale.

4.4.3 Program

The program for the differential private logistic regression contained three scripts: `plot_results.py`, `main.py` and `tuning.py`. Figure 4.18 shows how the program is ll. Both the `main.py` and `tuning.py` were simulated 48 times. To speed up the process the simulations were run in parallel on the HPC. The code can be found in section A.3.

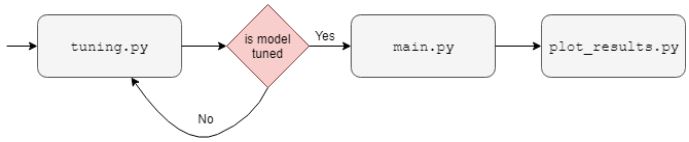


Figure 4.18: The procedure of the logistic regression program.

4.4.4 Summary

In conclusion, as there is a trade-off between privacy and accuracy it is hard to tell which ϵ is optimal. This, of course, depends on the data and as pointed out in section 3.2, the choice of ϵ is in fact a social one. Sometimes the accuracy is of the utmost importance and other times privacy is of the utmost importance.

However, there are some measures that can be taken. From figure 4.14 there is no visible difference between a model with $\epsilon = 10$ and one with $\epsilon = 0.1$ for most of the data intervals. Both models add small amount of noise to the weights. However, the promise of differential privacy between $e^{0.1}$ and e^{10} varies greatly, as the exponential function increases very rapidly. This makes the model with $\epsilon = 0.1$ a clear winner in this case as the error rate is the same but a model with $\epsilon = 0.1$ promises much more privacy. Additionally, if the logistic regression under the microscope has a lot of data, it would even be possible to provide $\epsilon^{0.01}$ differential privacy, as that model promises even less privacy loss and is able to converge to the same error rate as the model without differential privacy.

4.5 Stochastic Gradient Descent with Differential Privacy

A stochastic gradient descent (SGD) introduced in [32] was implemented. As in the previous chapters, the algorithm was run on the mnist data set. The problem at hand was decided to be the classification task between 4 and 9. The idea here is to use the SGD with differential privacy to train a logistic regression model. The SGD will iterate on the cross entropy cost function (equation 3.23) to learn the optimal weights and use the sigmoid function (equation 3.21) to discriminate between the classes.

Before running the differentially private SGD some preprocessing was done. As stated in section 3.3.8, the predictors were normalized such that $\|X_i\| \leq 1$ and the targets were mapped to -1 and 1 for each class. Next all observations (rows) were projected onto the unit ball. Then for faster and more secure computation, random projections projected the data onto 50 dimensions. Furthermore, it can be seen that by having fewer dimensions the privacy mechanism adds less total noise to the gradients in each iteration [35].

Generally speaking, one would cross validate to find the best number of projections to project onto, where some trade-off between dimensionality and accuracy would occur. However, [35] implemented the SGD with differential privacy and used random projections to reduce the dimensions of the mnist data set. In their case, the classification task was between all of the digits. They observed that projecting the data onto 50 dimensions resulted in a very small loss in accuracy. It was therefore decided to project the data onto 50 dimensions in this thesis. Using the argument that the binary classification task is easier than the multiclass classification task, the loss should be similar to or less than the multiclass classification.

It was decided to use the conventional random projections rather than differentially private ones. This was done as the aim here is to look at the stochastic gradient descent algorithm with differentially private updates. However, in some cases random projections can themselves be differentially private. This was not examined in this thesis as it is out of it's scope. However, for the interested reader [56] shows that if the singular values of a data set are large, the random projections are them self's differentially private.

Finally, as stated in section 3.3.8, the gradient needs to be bounded in order to gain differential privacy. Moreover, the gradient needs to be less than or equal to 1, $\|\nabla \mathrm{E}\| \leq 1$. This constraint was implemented using gradient clipping. The gradient clipping used the $l2$ norm to clip the gradient. Equation 4.2 shows the procedure.

$$\nabla \mathrm{E}_{\text{clipped}} = \frac{\nabla \mathrm{E}}{\max(1, \|\nabla \mathrm{E}\|_2)} \tag{4.2}$$

4.5.1 Training

Multiple stochastic gradient descent models were trained to explore how the stochastic gradient descent with differentially private updates evolves with the amount of training data. 50 models were built where the first model used $\frac{1}{50}$ of the training data, the second model used $\frac{2}{50}$ and so on. Note that in each step the model under construction uses the data from the previous models and then adds additional data to it.

As stated in section 3.3.8 the idea of the algorithm is to add noise to each update of the weights. This can make the training hard as when a grid search is performed on the hyperparameters a ϵ differentially private noise needs to be added. This can influence the training a lot, since the noise is sampled randomly from the density described in equation 3.41. Additionally, the SGD with differential privacy needs to tune 3 hyperparameters. Those are the learning rate η, batch size b and the weight decay λ. It was decided to make an adaptive learning rate as that is often good in practice (see section 3.3.7). Note that for each of the hyperparameters which need tuning, the complexity grows exponentially. It was therefore decided to fix the learning rate scheme as $\frac{1}{\sqrt{t}}$. This is the same scheme as was used in [32]. Note that t stands for the iteration count.

One thing is important to note before tuning the model. As the aim of local differential privacy is that no one is trusted, not even the data scientist, it is important to consider the privacy budget of the tuning. If one epoch is ϵ private, then for each epoch used for tuning the algorithm the ϵ add up. Therefore, tuning with a $5-$fold cross validation and large grid is in fact not that private. There exit special training methods for tuning hyperparameters under differential privacy. One such way is to tune the hyperparameters on public data. Another way is to do a special form of grid search were each pair of parameters are tuned on disjoint subsets of the data [26]. That way it only costs ϵ to train the model due to the parallel composition (see section 3.2.2).

However, in order to get a better understanding of both how the model works and how good it can be a $5-$fold cross validation was preformed. In addition to that, 24 instances of training process were run in parallel. In each process the hyperparameters which gave the best validation errors were selected as optimal for that process. If there was a tie, then the hyperparameters which had bigger batch size were selected. That is due to the fact that when the batch is bigger it is closer to the true gradient; hence, it does not introduce as much variance as the one with fewer observations when calculating the gradient. After all processes have finished their run, the hyperparameters which appeared most often were selected as the optimal for each model. In the case of a tie, the ones with the lower average validation error were selected.

A grid search was used to find the optimal values for the weight decay, λ, and batch size, b. Table 4.6 shows the parameters for the first grid search. Unfortunately, for both hyperparameters the grid was not big enough as all of the end values were used.

Therefore another grid was constructed. Table 4.7 shows the second grid.

From the second grid search it was apparent that the b also hit the extreme for that grid. This might be due to the fact that when more observations are in the batch, less noise is added in each iteration. Therefore a larger space was made between the grid values. The third grid search can be seen in table 4.8

The result from the third grid showed again that the algorithm tends to pick the bigger b when noise is added to the algorithm. However, the model which did not add noise picked only non boundary points in the grid. Figure 4.19 shows which points were chosen by the grid search. From the left figure it is possible to see that when lesser noise is added to the algorithm at training, the batches get smaller. The reason for this is how the algorithm adds noise. When too much noise is added to the algorithm, it tries to reduce it by having bigger batches. Looking at the right figure it seems that for the smaller models more regularization is needed, while the regularization for the bigger models is inside the grid. Note that λ is inside the grid for all N in the regular SGD.

Figure 4.20 shows how λ and b work together. The figure shows clearly that the models which have little or no noise tend to select both small b and λ, while the models which add more noise tend to select large b, while the λ ranges through the whole grid, although most often it is very small.

Due to time constraints it was decided to run with theses parameters, although all of the models are not completely tuned. However, as the SGD without differential privacy is tuned, a reasonable benchmark can be made. Moreover, as most of the parameters which are not tuned, are in the noisier models, it is hard to state if they are not the optimal values. That is because in each iteration a new fresh noise is added to the model. This noise can be very different, and as very few parameter values are in the extremes, it is possible that they are there just because of noise. This demonstrates the complexity of training with noise.

Table 4.6: The parameters for the first gird search of the SGD with differential privacy. Here X represent not a value.

λ	0.1	0.01	10^{-5}	10^{-10}	10^{-20}	X	X	X	X	X
b	1	2	5	10	50	75	100	150	200	250

Table 4.7: The parameters for the second gird search of the SGD with differential privacy. Here X represent not a value.

λ	1	0.5	0.1	0.01	10^{-5}	10^{-20}
b	1	2	5	10	50	75
	100	150	200	300	400	X

Table 4.8: The parameters for the second gird search of the SGD with differential privacy. here X represents not a value.

λ	3.0	2.0	1.5	1	0.5	0.1	0.01	10^{-5}	10^{-20}
b	1	2	5	10	50	75	100	150	200
	250	300	400	500	1000	2000	X	X	X

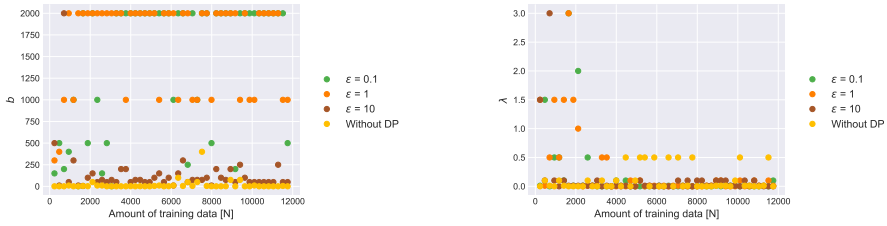


Figure 4.19: A scatter of the optimal parameters from the third grid search.

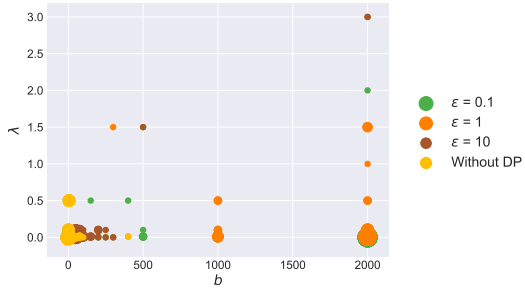


Figure 4.20: A scatter plot between the λ and η. The size of the circles count for how many parameters have the corresponding values.

4.5.2 Experiment Results

Figure 4.21 shows the error rate for the logistic regression built using SGD with differential privacy. It can be seen that the algorithm is not doing so well. Even when there is no noise added to the iterations the algorithm is doing poorly. This strongly indicates that one epoch is not enough for solving this classification task.

From the figure it is also visible that although the ϵ is selected as 10, the algorithm is not able to converge to the SGD without differential privacy. However, it comes close. It can be seen that when ϵ is 1, the algorithm is able to converge to similar

values as $\epsilon = 10$ when a lot of data is used. Lastly it can be seen that when $\epsilon = 0.1$ the algorithm is doing very poorly. Although the algorithm is able to learn more with more data, it still has around 35% error rate in the best case.

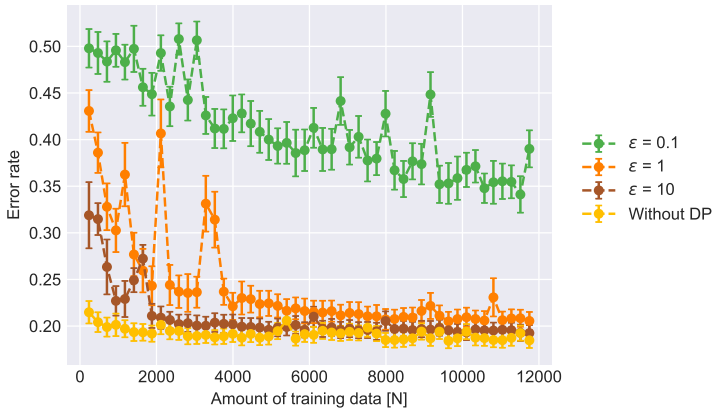


Figure 4.21: The error rate of the SGD with differential privacy.

As the figure above indicated that more epochs were needed to gain better error rate, it was decided to run the SGD without differential privacy for 10 epochs. With additional epochs the optimal parameters of the model could change, therefore it was decided to run a grid search on the model again. The model was run on the previously selected grid shown in table 4.8.

Looking at figure 4.22 it can be seen that the algorithm is not doing any better. Although it seems to be more stable. The reason for this might be the fact that the algorithm has adaptive learning rate. This indicates that it learns very fast in the beginning, and then decreases with every iteration. Therefore, the algorithm might start by going fairly fast towards the minimum, and then slowly reduce the speed in every iteration. This could either make the algorithm go towards the minimum where it can circle around with small steps, or the algorithm might not be able to reach the minimum before the learning rate got too small. This implies that it would be interesting to examine another learning rate scheme, which either decreases slower, or starts with a greater value.

Another thing which could potentially cause the error rate, are the random projections. That is because as the data is projected onto lower dimensions some information will be lost [21].

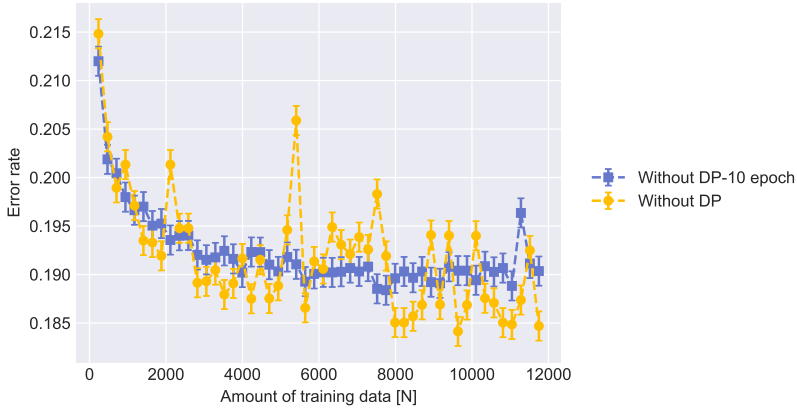


Figure 4.22: The error rate of the SGD without differential privacy shown with different amount of epochs.

4.5.2.1 Dimensional Analysis

The number of dimensions play a important role when building differentially private algorithms. Therefore it was decided to examine how the model performs with dimensions. This is interesting because with more dimensions, more noise is added to each iteration. This noise could disturb the algorithm, making it take different directions compared to the true gradient of the function. Therefore, as dimensions grow it is harder for the algorithm to go towards the true minimum. It is interesting to see how dimensionality reduction can influence this. However, it is a known fact that by doing dimensionality reduction, some information from the data may vanish. Therefore, there is a possibility for a trade-off between the noise generation and dimensionality.

It was decided to fix $\lambda = 0.0001$, $b = 5$ and use the same learning rate scheme for η. The dimensions which were investigated were $d \in \{15, 50, 100, 200, 784\}$, where the original data set is 784 dimensions. Although we are not examining the best models, it is ideal to see that by fixing the aforementioned hyperparameters, the effect of dimensionality versus noise level gives a clearer picture. The other parameters should not in any way influence the error rate.

Figure 4.23 shows the results. The figures on the left hand side show the error rate, while the figures on the right show the corresponding confidence interval. From the figure showing $\epsilon = 10$ it is clear how dimensonality and differential privacy can go hand in hand. There, it is possible to see that the model including all of the features is actually doing worst. Moreover, it shows that with more dimensions the model will be more delicate to noise. However, it can also be seen that if the data is projected onto to few dimensions, information will be lost. This is clear for $\epsilon = 10$, $d = 15$, where

the model clearly converges before other models with higher dimensonality. Therefore the figure shows a clear trade-off between dimensions and differential privacy.

Looking at the figures showing $\epsilon = 1$, similar things happen. There, ϵ with the lowest dimensonality is doing best. However, the results for all of the models both with $\epsilon = 1$ and $\epsilon = 0.1$ are unacceptable. Nonetheless, as seen in figure 4.21, if the hyperparameters were tuned properly the model can result in a better error rate with indistinguishable noise level.

Compering the models having no noise and the ones having $\epsilon = 10$ noise it can be seen that when $d = 15$ the error rate is very similar with and without differential privacy. Now, looking at the figure having $\epsilon = 1$ differential privacy, the model having $d = 15$ is doing a little worse but is still decreasing. It would be interesting to see if it would be able to converge to the same error rate as the other two, if more data was available. Looking back at the figure having $\epsilon = 10$ differentially privacy, it is visible that three dimensions, namely $d \in \{50, 100, 200\}$ are doing better than the non differentially private model with $d = 15$. Moreover, when $\epsilon = 10$ and $d = 50$ the model is almost as good as the non differentially private model for the corresponding dimension.

However, in the case of the non differentially private models, it can be seen that with increasing dimension the error rate drops. This makes sense as keeping all the dimensions, no information is lost from the data.

By looking at the figures with $\epsilon = 1$ & 10 it can be seen that there is a trend between dimensionality, noise, and error rate. The figures show that as noise level drops, more dimensions are suitable. In the figure with $\epsilon = 1$ noise, the optimal d is 15. However in the figure with $\epsilon = 10$ noise, the optimal d is 50. This might be do to the fact that as ϵ aims toward less noise, the noisy gradient will aim towards the true gradient, therefore, the dimensionality loss will have greater impact on the results. However, note that as ϵ increases, privacy decreases. Therefore one can not simply increase both ϵ and d without facing consequences.

These result imply that there is a clear trade-off between dimension, noise and error rate. Moreover, the SGD with differential privacy seems to be more suitable for smaller dimensional problems.

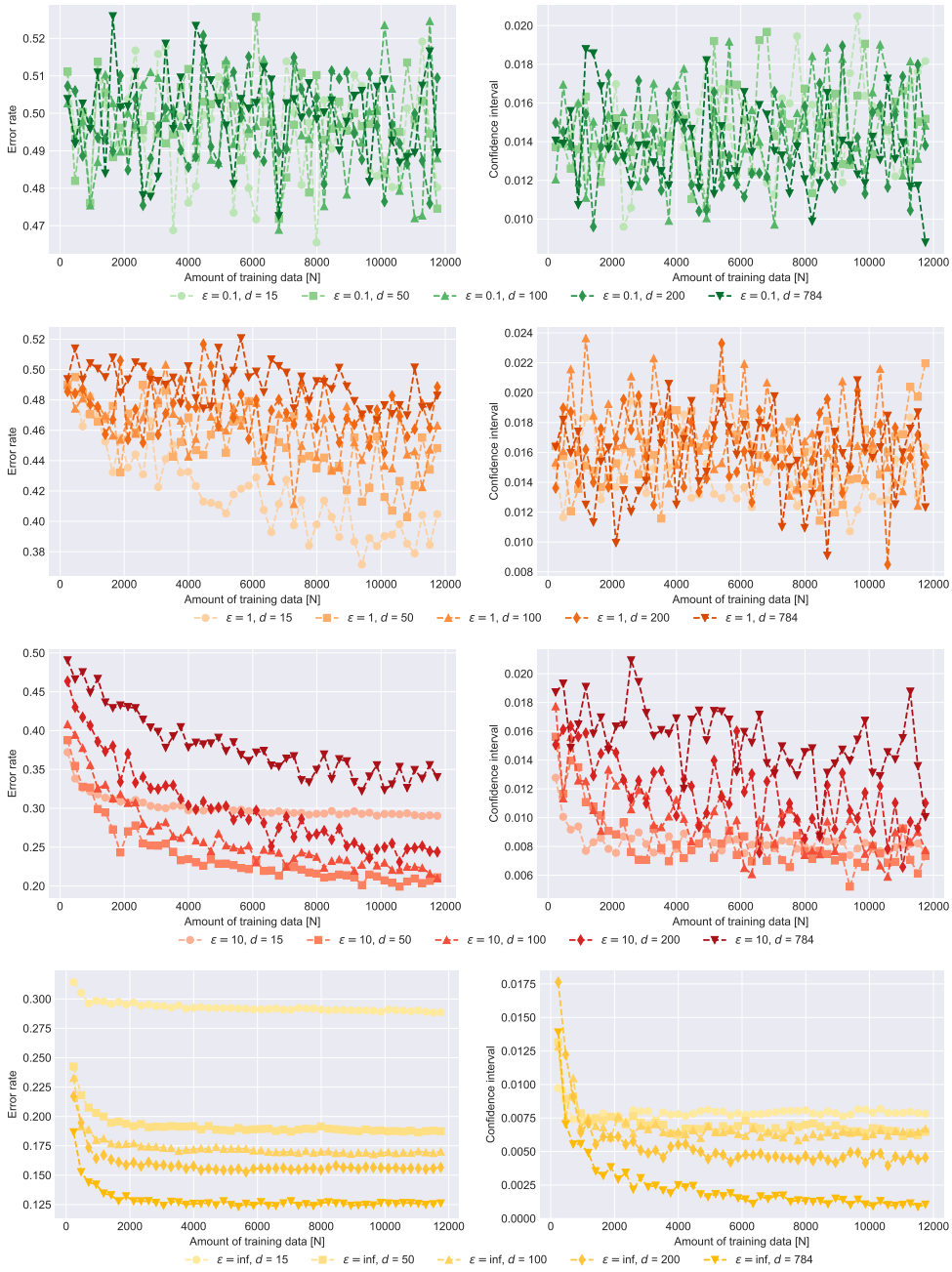


Figure 4.23: Shows how the error rate changes with ϵ noise and dimension d. The right figures shows the error rate and the left figures shows the corresponding confidence interval.

4.5.3 Program

Six scripts were written to perform the SGD with differentially private updates. They were: `plot_results.py`, `investegate_parameters.py`, `main_dim.py`, `main.py`, `tuning.py` `utils.py`. Both main and tuning were run parallel on the HPC cluster to speed up the running. Figure 4.24 shows how the programs were run. Both `main.py` and `tuning.py` generate a JSON file which is fed to the other programs. The code can be found in section A.3.

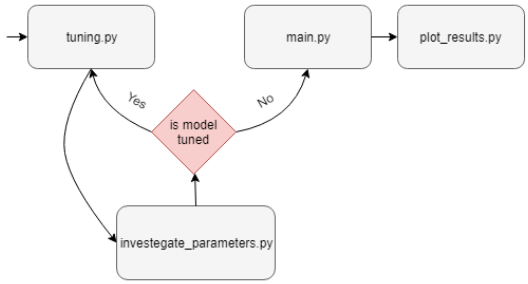


Figure 4.24: Shows of the procedure of the program.

4.5.4 Privacy

As for the other algorithms it is clear that there is a trade-off between accuracy and privacy. Furthermore, it is interesting to see the relationship between batch size, privacy and accuracy. If the batch size is large much less noise will be added to the iteration, than if the batch size is small. However, this means that fewer iterations can be made, and as the gradient has been clipped, the algorithm can not take very big steps. Therefore the algorithm might not be able to converge. Furthermore, if more than one epoch will be taken, the composition theorem from differential privacy states that the ϵ between iterations add up. This means that if for example the algorithm takes two epochs, and each iteration is ϵ differentially private, the privacy budget for the algorithm will be 2ϵ.

When looking at figure 4.22 it would not be advisable to run the algorithm for more epochs, as even the algorithm without differential privacy is not gaining significant amount in accuracy. Therefore, it is unconvincing that by adding noise to the iterations, the accuracy will improve. However, the promise for privacy will decrease, as the privacy budget goes from ϵ differential privacy to 10ϵ differential privacy. Moreover, the difference between these privacy budgets is huge. As the promise is a function of e^{ϵ}, hence the difference between ϵ and 10ϵ is exponential.

4.6 Discussion

All of the models that have been considered have ensured privacy in one way or another. However, they do not promise the same level of privacy. Judging from the discussion above, it is clear that the distributed Lasso promises the least privacy, then the logistic regression and finally the Stochastic gradient descent, where even the data scientist is not trusted with the data. Of course the amount of privacy generated by the differentially private algorithms is hugely dependent on ϵ. Nonetheless, as long as the ϵ leads to at least some noise, the algorithms will be more private than the Lasso.

Although only the Lasso was simulated as a distributed machine learning problem all of the other models could be expanded to a distributed framework. Moreover, the stochastic gradient descent with differentially private updates could be thought of as a distributed algorithm as each update is differentially private and could therefore be published. The logistic regression with differential privacy could send the perturbed objective values to another location, where the next step of the algorithm could be taken from.

One advantage of having distributed algorithm is that the data is scattered around multiple places. This means that if a hacker wants to get a hold of all the information of a particular analysis, he/she would need to hack multiple places to gain all of the data. This entails that if organizations want to work together, and one organization is hacked, the other will not suffer. Moreover, this also entails that the data will never leave each organization.

Another scenario where it would be an advantage of making the algorithms distributed is when data is very big. If the data is to big to fit in one place, the data could be scattered around many machines, and still remain private. This could potentially be a huge advantage as internet of things take off. That is because these methods can able machines with limited memory to work privately together.

All of these methods take into assumption that the organizations using them are willing to share their features. In practice, this might be difficult as feature engineering is an extremely important field in machine learning. The features which are used might be something which gives companies competitive advantage, and therefore they might not be willing to share their features. However, this difficulty disappears in research, education and healthcare, where everyone is working together towards the same goal. Furthermore, this difficulty also disappears if companies want to use these methods between their products.

Generally speaking, it is clear from all of the algorithms that privacy and accuracy are tightly linked together. As the algorithms get more private, their accuracy tends to suffer. The differentially private algorithms show this very clearly, as when ϵ increases, the algorithms are in fact less private, but simultaneously their accuracy improves. This makes the task for selecting appropriate ϵ even harder, as if ϵ is too high the algorithm might leak information, and if it's too low, the accuracy will

suffer. However, in spite of the general trend being with more privacy resulting in less accuracy, there are some bright spots in between. For example, the distributed private Lasso is able to perform better than the other general Lasso models in this case.

It is also interesting to observe how the amount of data comes into play. The learning curve of all the algorithms show that more data results in better accuracy. Surely, machine learning in general models tend to become better with more data. However, when looking at the differentally private algorithms, it can be seen that the learning curve is much steeper than for the corresponding models which do not have differential privacy. This is because of how noise is integrated into both of the algorithms. Equations 4.1 and 3.40 show this clearly, as for both of them the output depends on the amount of observations. This could be a very valuable trait, as society is driving towards a more data driven world.

4.6.1 More on Lasso

The distributed private Lasso had very good accuracy. However it simultaneously provided the least amount of privacy. That is because differential privacy promises that auxiliary information can not be used to reveal users in the data set [12]. However, this has not been proven for the distributed Lasso. Imagine if the adversary could use auxiliary information to gain information on all participants in a data set except for one. If the gradient of this data set is then made public, the adversary might be able to reverse engineer the gradient and gain knowledge of the user he/she had no prior information on. Moreover, the adversary could calculate the gradient with all of the users he/she has under control and then compare it to the public gradient. Note that this not only demands the auxiliary to have information of almost all of the participants, but also he/she has to know the cost function used to calculate the gradient.

However, in the case of differential privacy the adversary could not gain knowledge from auxiliary information and hence could not gain any information about users in the dataset.

Although the private distributed Lasso is not as strict as the other algorithms examined in this thesis, it can still be very useful. To be able to gain information about the data the adversary needs to have a lot of information beforehand, which is not necessarily possible. The algorithm is indeed more private than simply making all the records public, and can achieve good predictions and results concerning the data. However, if the data is very sensitive, the user should think very carefully if this algorithm is the right choice.

Two different Lasso algorithms were built in this thesis. One where PCA was used to project all the data onto the first 100 principal components before the Lasso was run, and another where no PCA was made. The accuracy of both methods were similar. However, the feature selection worked better when no PCA was used. This makes

sense as the dimensionality reduction made by the PCA removed all of the features which had limited variance.

It took considerable time to tune the algorithm. With 5−fold cross validation and a grid search over two hyperparameters. The tuning of the PCA models was much faster. That is because the innermost loop calculates the gradient of the cost function. To calculate the gradient, a dot product of all the features with the corresponding weights is needed. Consequently, reducing the dimensions from 784 to 100 results in a much smaller dot product.

Interestingly, both with and without the PCA the distributed algorithm was able to outperform both of the centralized algorithms. The reason for this is most likely that by separating the data between the centers the algorithm reduces variance, and therefore is able to regularize the model. However, note that the $l1$ norm is very small. Thus it would be interesting to see how the algorithms compare when more regularization is appropriate. Furthermore, it would be exciting to see how the distributed algorithm works if more data centers are used, and from that if there is an optimal number of data centers.

4.6.2 The Logistic Regression Models

Both the Stochastic gradient descent (SGD) with differentially private updates and the logistic regression with differential privacy, performed logistic regression. From the analysis it is clear that the objective perturbation is a winning strategy in our case. Although the SGD algorithm is known to be a fast algorithm it was surprisingly slow, considering that it only ran one epoch. [35] also implemented the SGD with differentially private updates and had similar conclusions, blaming the noise added in each iteration. This could very well be the case, as generating the noise demands a number of additional operations. The noise is then added in the inner most iteration, making the algorithm perform these operations multiple times.

The logistic regression with differential privacy was considerably fast, as it only needed to add noise after the logistic regression model was trained. The model was trained using the coordinate descent optimization algorithm as that was the default optimization algorithm in the python module (scikitlearn). Note that the algorithm could have been trained with stochastic gradient descent which should also result in a faster algorithm than using the SGD with differential privacy. Additionally, as the cost function is strongly convex for both methods, the iterative algorithm should find the same minimum.

For each epoch the SGD with differential privacy uses the privacy decreases. This is a shortcoming of the algorithm as it makes it hard to converge and be private at the same time. In contrast the logistic regression model using the objective perturbation can take as many epochs as it needs to converge, and still remain ϵ−differentially private. However, in the case of the logistic regression with differential privacy the users

which are in the data set need to trust the data scientist, while in the case of the SGD with differential privacy they do not as it preserves local differential privacy. This means that the whole training process can be published for the SGD with differential privacy, and the algorithm would still not leak any information.

The tuning process in the SGD with differentially private updates evokes some questions. If the algorithm is to be tuned with a grid search, multiple iterations through the data will be made. In each iteration, $\epsilon-$differentially private noise will be added to the algorithm. From the above discussion it can be seen that with this the level of privacy decreases. Therefore, the grid search sacrifices the local differential privacy.

A common approach for tuning differentially private algorithm is to use public data. If the data has the same distribution as the private data, the hyperparameters can be learned on the public data, as they will not reveal anything about the users in the private data set. If no such data is available a grid search associated with the private tuning algorithm introduced in [26], can be used to tune the algorithm.

However, note that in this thesis, a normal grid search was used to obtain the optimal parameters for the SGD with differential privacy. Although the grid search gave up on local privacy, it derived some interesting results. It showed how batch size and noise level interact. If very small amount of noise is added to the algorithm, the algorithm tends to select a small batch size, however, if a bit more noise is added to the algorithm, the selected batch size will increase a lot. This can be seen in figures 4.19 and 4.20.

Despite the fact that the SGD with differentially privacy promises more privacy for users, it comes at a cost. The logistic regression model with the objective perturbation results in much better error rate. Therefore, there is a clear trade-off between the algorithms, as one of them preserves more privacy, while the other gives better accuracy. Comparing figures 4.14 and 4.21 it can be seen that even $\epsilon = 0.005$ for the logistic regression is doing better than $\epsilon = 10$ in the SGD model, for large amount of data. The difference in privacy here is huge, as the privacy promise is exponential.

However, this comparison is not fair, as the SGD with differential privacy uses random projections before running the SGD with differential privacy. Moreover, as examined in section 4.5.2.1 it is clear that the dimensionality of the random projections have a great impact on the final results. That is because as noise is added to the dimensions, the search direction taken by the algorithm will be influenced by the noise.

Although the optimal dimension d was 50 in the case of $\epsilon = 10$, it can not be said that this is the best dimension to project the data onto, as only 5 dimensions were considered. Furthermore, the hyperparameters which were selected in the experiment were not found as optimal parameters for all the models. In spite of that, there was a clear trend with dimensionality and noise (see figure 4.23). With decreased noise more dimensions are favorable. The reason for this is that the dimensionality reduction entails information loss, while adding noise to many dimensions can mislead the algorithm. Therefore, as less noise is added, the algorithm starts to resemble an SGD

without differential privacy more closely. However, this is a clear trade-off between privacy and error rate. Considering these facts, it is possible to see that if the SGD with differential private updates is to perform well, it needs dimensionality reduction.

From the analysis above it can be seen that the logistic regression using objective perturbation is more favorable if the aim is to get good error rate. Moreover, the logistic regression using objective perturbation does a better job than the SGD with differential privacy right from the start. Where it is able to archive only 10% error rate, for all ϵ which were tried in the SGD with differential privacy. The SGD with differential privacy never had a chance because of the information loss introduced by the random projections. Figure 4.22 shows that even using SGD without differential privacy for 10 epochs, the algorithm is not able to achieve the same result as the logistic regression with objective perturbation.

Even though objective perturbation gives better error rate in our case, that does not mean that it is always better to use objective perturbation. [30] also created a differentially private stochastic gradient descent algorithm which adds noise into the iteration process. They have managed to get a better bound on how much noise is added in each iteration by using their moments accountant and the Gaussian mechanism introduced in section 3.2.4. They don't base their algorithm on dis-joint batches, but rather that an observation is picked randomly into a batch. In addition to that they observe that the error rate can be decreased even further by handling the privacy parameters in a smart way, by fixing ϵ and making the δ vary. This allows them to do more epochs, while preserving $\epsilon-$differential privacy.

CHAPTER 5

Conclusion

In this thesis we investigated how privacy and machine learning are linked together. Three methods were built. A distributed Lasso where the data never leaves a data center, Logistic regression with differential privacy, and Stochastic gradient descent with differentially private updates. All of the algorithms provided different sight on how to provide and protect privacy. The Lasso algorithm provided the least privacy, then the logistic regression, and finally the Stochastic gradient descent provided the most privacy. All of the algorithms showed that there is a tight link between privacy, error rate and training data.

In the distributed Lasso we showed how sharing only gradient between data center can lead to a good error rate. Furthermore, we demonstrate that two data centers are better off working together if not enough data is available. When enough data is available the algorithm converges to a similar error rate.

The logistic regression with differential privacy shows how the bound of the sensitivity can decrease with more data. Section 3.3.6 describes how the bound is calculated, and section 4.4 shows the results using the bound with the Laplace mechanism. In addition, we show how different ϵ affect the logistic regression model. The amount of noise generated by ϵ is then compared with the weights of the regression model.

In the Stochastic gradient descent with differential private updates, another type of differential privacy is considered, namely local differential privacy. The algorithm results in the worst error rate of all the methods. Number of things which could affect the error rate were examined, namely, how batch size and noise go hand in hand, how many epochs are suitable for the algorithm, and finally the relationship between iterative differential private algorithms and dimension is considered.

Privacy preserving machine learning is a growing field which is getting more relevant by the minute. As organizations are gathering more and more data for doing machine learning, the need for privacy increases. Therefore the work done in this thesis is of great importance. It is positive to observe that with more data, more privacy can be obtained with the same amount of accuracy. As a result, the methods discussed in this thesis fit well, with the future trend of data collection.

5.1 Future Work

The algorithms considered in this thesis offer a lot of attractive adjustments. First of, it would be interesting to try all of the algorithms on another data set, as well as try to run them on a multiclass-classification task. This could help examine how they generalize on real world data.

The analysis done in this thesis do not show all the potential variance in the model. This is due to how the models are trained. As for each model, the most common parameter from the grid search is used on the test data. Therefore, it would be interesting to analyze the behaviour of the variance in the models. This could be done by using every optimal parameter from the grid search for each model instead of selecting the most common parameters. However, this might require a much larger grid, which could result extreme computational time.

5.1.1 Distributed Lasso

As the main shortage of the distributed Lasso compared to the other algorithm is privacy, it would be appropriate to add differential privacy into the algorithm. This could be done by bounding the gradient and adding corresponding noise to the algorithm. This could involve two advantages. First, by bounding the gradient the larger gradients will not influence the smaller gradients as greatly. This could help with tuning, as it will be less likely for the gradients to make each other spiral up towards infinity. Second, as the distributed algorithm was able to regularize by adding gradients from multiple data centers, it would be intriguing to see if the noise could improve the generalization error, as the noise will introduce regularization effect.

The research could also be further extended by examining if there is an optimal number of data centers. As the distributed algorithm manages to improve, it would be intriguing to see if there is an optimal number of data centers.

The distributed Lasso took a considerable amount of time to run. An idea of dealing with this might be to introduce batches. Instead of calculating the gradient of the whole data set, one could calculate the gradient of random subset of the data set. This would speed up computation time, but not be as accurate as the true gradient. This also introduces more privacy, because it will be harder to reverse engineer the gradient if it is build upon stochastic batches. However, this is not as strict as the differentially private algorithms, but adds complexity if compared to the original method.

5.1.2 Logistic Regression with Differential Privacy

The logistic regression model with differential privacy showed how the amount of training data can influence test accuracy. To extend this work even further it would be interesting to examine how this method works in a distributed framework. That can be done by communicating the perturbed objective between data centers. This

poses some questions about privacy, and when it actually is sensible to distribute the algorithm.

5.1.3 Stochastic Gradient Descent with Differential Privacy

In this thesis, the stochastic gradient descent with differential privacy was used with one type of learning scheme. As noise is added to each iteration, the learning scheme is of great importance. It could be interesting to examine whether another learning scheme could be valuable. To name some, one learning scheme which could be advantageous to research is $\frac{1}{\lambda t}$. This learning scheme is said to give fast convergence rate for a strongly convex problems [57].

While building this algorithm, another idea was born that might be worth a further exploration. As the amount of noise added to each iteration is influenced by the batch size, and as only one iteration is run to preserve ϵ differential privacy, one might use an adaptive batch size. By starting on a larger batch that also has a larger learning rate, less noise could be added to the iteration. This entitles that the algorithm starts to take good steps and as it gets closer to the minimum both the batches and the learning rate decrease. Hence, this would force the algorithm to start off by taking large steps towards the minimum in the beginning, and then when it is close to the minimum it will walk randomly around the solution space.

In the corresponding section an analysis on dimensionality and privacy was formed. From there it was shown that there was a trade-off between iterative differential privacy algorithms and random projections. It would be interesting to explore this further, and that could be done both by implementing other dimensonality reduction techniques, as well as a feature selection, and try it on another data set.

It might also be intriguing to see whether the algorithm is able to converge. That could be done by looking at the objective function and the gradient. The research could take on whether the algorithm is able to converge without noise, and if so, how many iterations are needed with different types of ϵ noise. This entails that the privacy budget for the algorithm would need further examination.

Lastly, it might be interesting to compare the logistic loss with the hinge loss. As the hinge loss is maximum-margin classifier, the sign function could be used to predict the targets. This could be interesting as hinge loss is strongly linked to support vector machines, which are known to be a great machine learning algorithm.

APPENDIX A

An Appendix

A.1 Proving Differential Privacy for Section 3.3.8

In order for the prove to hold few constraints need to be meet. First both the predictors for $X \in \{x_1, x_2, ..., x_n\}$ and $X' \in \{x_1, x_2, ..., x'_n\}$ differing only in the n-th element need to be set so that for any x, x', $\|x_i\| \leq 1$ holds. In addition, y and $y' \in 1, -1$ and the gradient is bounded as follows $\|\nabla \mathrm{E(w)}\| \leq 1$. Here G is a Oracle which feeds data points into the algorithm. Suppose that the Oracle draws in random order of permutation π, for any t in the data set, we have

$$
\begin{aligned}
\frac{p(G(w_t) = g | (x_{\pi(t)}, y_{\pi(t)}) = (x, y))}{p(G(w_t) = g | (x_{\pi(t)}, y_{\pi(t)}) = (x', y'))} &= \frac{p(Z_t = g - \lambda w - \nabla \mathrm{E}(w, x, y))}{p(Z_t = g - \lambda w - \nabla \mathrm{E}(w, x', y'))} \\
&= \frac{\mathrm{e}^{-(\epsilon/2)(\|g - \lambda w - \nabla \mathrm{E}(w, x, y)\|)}}{\mathrm{e}^{-(\epsilon/2)(\|g - \lambda w - \nabla \mathrm{E}(w, x', y')\|)}} \\
&= \exp((\epsilon/2)(-\|g - \lambda w - \nabla \mathrm{E}(w, x, y)\| \\
&\qquad + \|g - \lambda w - \nabla \mathrm{E}(w, x', y')\|)) \\
&\leq \exp((\epsilon/2)\|\nabla \mathrm{E}(w, x, y) - \nabla \mathrm{E}(w, x', y')\|) \\
&\leq \exp((\epsilon/2)\|\nabla \mathrm{E}(w, x, y)\| + \|\nabla \mathrm{E}(w, x', y')\|) \\
&\leq \exp(\epsilon).
\end{aligned}
$$

The first inequality follows from the triangle inequality and the last step comes from the fact that $\|\nabla \mathrm{E(w)}\| \leq 1$. Therefore the mechanism is differential private [34].

A.2 Sampling from the Density in 3.41

A.2.1 $l1$ Norm

A.2.1.1 1 dimension

$$p(x) = Ce^{-\lambda|x|} \tag{A.1}$$

Lets find what the constant C is. The probability density function of p(x) can be written as:

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{A.2}$$

This leads to

$$\int_{-\infty}^{\infty} e^{-\lambda|x|}dx = C^{-1} \tag{A.3}$$

Now the integral can be calculated, note that the middle equation arises due to symmetry.

$$\int_{-\infty}^{\infty} e^{-\lambda|x|}dx = 2\int_{0}^{\infty} e^{-\lambda|x|}dx = 2[-\frac{1}{\lambda}e^{-\lambda x}]_0^{\infty} \tag{A.4}$$

Therefore

$$C^{-1} = 2[-\frac{1}{\lambda}e^{-\lambda x}]_0^{\infty} = 2[0 - (-\frac{1}{\lambda})] \tag{A.5}$$

Which gives

$$C^{-1} = \frac{2}{\lambda} \tag{A.6}$$

Thus the equation A.1 can be written as

$$p(x) = \frac{\lambda}{2}e^{-\lambda|x|} \tag{A.7}$$

Note that this is the Laplace distribution, thus if the l1 norm is used in density described in equation 3.41, the noise will be drawn from the Laplace distribution.

A.2.1.2 D dimensions

Next for the multivariate case

$$p(x) = C_D e^{-\lambda\|x\|_1} \tag{A.8}$$

Note that for the one norm applies

$$\|x\|_1 = |x_1| + |x_2| + ... + |x_D| \tag{A.9}$$

Therefore equation A.8 can be rewritten as:

$$p(x) = C_D e^{-\lambda(|x_1|+|x_2|+...+|x_D|)} = C_D \prod_{d=1}^{D} e^{-\lambda|x_d|} \tag{A.10}$$

the equation can now be multiplied by $\frac{2}{\lambda}$ and $\frac{\lambda}{2}$ as that is equal to 1. That gives:

$$C_D(\frac{2}{\lambda})^D \prod_{d=1}^{D} \frac{\lambda}{2} e^{-\lambda|x_d|} = C_D \frac{2}{\lambda} \prod_{d=1}^{D} p_d(x_d) \tag{A.11}$$

Then for d integrals

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}...\int_{-\infty}^{\infty} p(\bar{x})dx_1dx_2..dx_D = 1 \tag{A.12}$$

Therefore

$$C_D(\frac{2}{\lambda})^D \prod_{d=1}^{D} \int_{-\infty}^{\infty} p_d(x_d) = C_D(\frac{2}{\lambda})^D * 1 = 1 \tag{A.13}$$

Thus

$$C_D = (\frac{2}{\lambda})^{-D} \tag{A.14}$$

Inserting C_D into equation A.10 results in

$$(\frac{\lambda}{2})^D e^{-\lambda\|x\|_1} \tag{A.15}$$

Which is the multivariate Laplace distribution. Hence, the density described in equation A.8 results in the Laplace distribution, for the $l1$ norm in d dimensions.

A.2.2 l2 Norm

Consider that the l2 norm is calculated as:

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_D^2} \tag{A.16}$$

A.2.2.1 1 Dimensions

$$p(x) = Ce^{-\lambda\|x\|_2} = Ce^{-\lambda\sqrt{x^2}} = Ce^{-\lambda|x|} \tag{A.17}$$

Which is the same thing as for the 1 dimensional case for the l1 norm, which gives

$$C = \frac{\lambda}{2} \tag{A.18}$$

therefore using the $l2$ sensitivity for 1 dimensional case, the noise from the density in equation 3.41 will be drawn from the Laplace distribution.

A.2.2.2 D Dimensions

$$p(x) = Ce^{-\lambda\|x\|_2} = Ce^{-\lambda\sqrt{x_1^2+x_2^2+...+x_D^2}} \tag{A.19}$$

This function is hard to integrate. However, one can recognize that $\sqrt{x_1^2 + x_2^2 + ... + x_D^2}$ is the same as r in spherical coordinates. Hence we change the coordinate system.

It should be taken into consideration that the transformation to spherical coordinates gives

$$dx = r^{D-1}dr\Omega_D \tag{A.20}$$

Where Ω stands for all the directions in the space.

Changing to spherical coordinates thus gives:

$$p(x) \propto \int_{-\infty}^{\infty} Ce^{-\lambda r}r^{D-1}dr\Omega_D = 1 \tag{A.21}$$

This looks like the gamma distribution. In addition it can be seen that the function is independent of the direction Ω. Hence, to draw from this distribution, first a uniform vector, v, can be drawn from the unit ball. Then a random vector, l, can be drawn from the gamma distribution. Thus the noise from the density in equation 3.41 for the D dimensional case can be drawn by multiplying l and v, which are drawn as described above.

A.3 Code

All of the code built in this thesis can be found on `https://github.com/IngvarBjarki/master_thesis`

A.4 Regularized Logistic Regression with Differential Privacy

Table A.1: The mean and variance of the weights and the variance of the noises for all data runs.

interval	$\epsilon = 0.0005$	$\epsilon = 0.001$	$\epsilon = 0.005$	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1$	$\epsilon = 10$	E(weights)	var(weights)
235	9 155.229 5	2 353.795 4	89.686 3	23.121 8	0.931 3	0.234 8	0.009 5	0.002 3	$2.330\,3 \cdot 10^{-05}$	0.018 9	0.245 1
470	20 757.913 5	5 263.089 5	211.927 4	51.118 9	2.114 4	0.533 8	0.021 0	0.005 2	$5.183\,4 \cdot 10^{-05}$	0.058 8	0.928 2
705	2 348.749 9	575.872 3	23.069 2	5.763 6	0.231 6	0.058 1	0.002 4	0.000 6	$5.775\,2 \cdot 10^{-06}$	0.049 5	0.733 9
940	2 291.911 0	581.373 7	23.112 0	5.868 6	0.227 1	0.059 0	0.002 3	0.000 6	$5.921\,2 \cdot 10^{-06}$	0.067 2	1.076 5
1175	4 645.812 7	1 117.112 4	45.483 1	11.383 9	0.449 7	0.112 3	0.004 5	0.001 1	$1.138\,3 \cdot 10^{-05}$	0.097 6	1.816 0
1410	4 098.721 1	1 042.382 5	41.129 8	10.357 9	0.411 7	0.101 8	0.004 0	0.001 1	$1.046\,0 \cdot 10^{-05}$	0.116 2	2.211 0
1645	1 702.763 2	430.285 5	16.844 4	4.289 7	0.170 0	0.043 1	0.001 7	0.000 4	$4.248\,3 \cdot 10^{-06}$	0.110 6	1.964 5
1880	329.915 9	79.911 3	3.193 7	0.813 5	0.033 0	0.008 1	0.000 3	$8.330\,7 \cdot 10^{-05}$	$8.357\,2 \cdot 10^{-07}$	0.084 9	1.318 5
2115	1 047.480 9	262.143 2	10.406 5	2.600 1	0.105 2	0.025 8	0.001 0	0.000 3	$2.583\,6 \cdot 10^{-06}$	0.127 0	2.255 1
2350	1 842.612 5	470.634 9	18.841 8	4.730 1	0.183 8	0.046 6	0.001 9	0.000 5	$4.786\,3 \cdot 10^{-06}$	0.164 8	3.180 6
2585	1 875.168 7	472.013 6	18.881 2	4.743 0	0.191 3	0.047 3	0.001 9	0.000 5	$4.801\,6 \cdot 10^{-06}$	0.185 6	3.610 8
2820	1 922.560 8	488.488 8	19.305 3	4.816 9	0.194 3	0.048 4	0.001 9	0.000 5	$4.763\,5 \cdot 10^{-06}$	0.203 8	4.062 1
3055	876.693 3	222.347 2	8.876 1	2.191 2	0.089 1	0.021 7	0.000 9	0.000 2	$2.181\,0 \cdot 10^{-06}$	0.182 8	3.362 5
3290	3 061.327 5	764.254 4	30.747 9	7.597 8	0.296 3	0.073 7	0.003 0	0.000 8	$7.661\,9 \cdot 10^{-06}$	0.262 8	5.856 8
3525	1 272.329 4	310.832 4	12.667 2	3.116 5	0.124 8	0.031 0	0.001 2	0.000 3	$3.086\,6 \cdot 10^{-06}$	0.230 9	4.567 3
3760	736.226 4	184.065 5	7.327 6	1.847 5	0.073 4	0.018 0	0.000 7	0.000 2	$1.857\,9 \cdot 10^{-06}$	0.217 4	4.076 9
3995	197.859 1	49.461 2	1.978 3	0.505 6	0.019 9	0.005 1	0.000 2	$5.031\,4 \cdot 10^{-05}$	$4.977\,8 \cdot 10^{-07}$	0.168 6	2.789 6
4230	865.010 3	216.826 4	8.648 6	2.164 8	0.085 7	0.021 7	0.000 9	0.000 2	$2.171\,3 \cdot 10^{-06}$	0.251 6	5.019 4
4465	521.373 6	131.158 1	5.100 2	1.281 2	0.051 8	0.013 0	0.000 5	0.000 1	$1.300\,2 \cdot 10^{-06}$	0.237 7	4.458 9
4700	284.749 9	70.257 7	2.844 0	0.710 3	0.028 6	0.007 1	0.000 3	$7.129\,5 \cdot 10^{-05}$	$7.085\,6 \cdot 10^{-07}$	0.217 7	3.833 8
4935	256.421 4	63.440 0	2.530 3	0.637 2	0.025 4	0.006 4	0.000 3	$6.386\,2 \cdot 10^{-05}$	$6.406\,7 \cdot 10^{-07}$	0.222 7	3.937 2
5170	400.733 3	98.575 1	3.942 8	0.987 8	0.039 2	0.009 8	0.000 4	$9.669\,1 \cdot 10^{-05}$	$9.567\,7 \cdot 10^{-07}$	0.254 4	4.814 9
5405	215.406 0	53.533 1	2.157 2	0.529 2	0.021 6	0.005 4	0.000 2	$5.423\,7 \cdot 10^{-05}$	$5.355\,0 \cdot 10^{-07}$	0.230 5	4.125 7
5640	197.611 4	48.740 7	1.956 8	0.494 0	0.019 7	0.005 0	0.000 2	$5.004\,1 \cdot 10^{-05}$	$4.956\,4 \cdot 10^{-07}$	0.234 5	4.215 6
5875	295.933 2	73.643 3	3.034 9	0.758 7	0.030 1	0.007 6	0.000 3	$7.556\,4 \cdot 10^{-05}$	$7.474\,5 \cdot 10^{-07}$	0.268 7	5.133 1
6110	499.578 4	122.987 8	4.942 7	1.252 8	0.049 1	0.012 4	0.000 5	0.000 1	$1.218\,3 \cdot 10^{-06}$	0.312 3	6.466 4
6345	157.733 0	38.474 7	1.547 7	0.385 5	0.015 7	0.003 9	0.000 2	$3.993\,6 \cdot 10^{-05}$	$3.870\,1 \cdot 10^{-07}$	0.249 8	4.477 4
6580	953.751 6	238.494 0	9.700 3	2.361 3	0.096 8	0.023 7	0.000 9	0.000 2	$2.370\,8 \cdot 10^{-06}$	0.388 0	9.087 8
6815	224.609 7	54.481 6	2.229 3	0.575 7	0.022 9	0.005 6	0.000 2	$5.583\,9 \cdot 10^{-05}$	$5.510\,7 \cdot 10^{-07}$	0.291 7	5.524 7
7050	125.188 5	31.175 4	1.268 4	0.318 0	0.012 3	0.003 2	0.000 1	$3.129\,1 \cdot 10^{-05}$	$3.126\,2 \cdot 10^{-07}$	0.264 0	4.710 0
7285	240.988 9	60.915 0	2.390 0	0.602 3	0.024 0	0.005 9	0.000 2	$6.102\,6 \cdot 10^{-05}$	$5.978\,2 \cdot 10^{-07}$	0.316 6	6.149 6
7520	329.821 7	82.946 2	3.269 2	0.807 2	0.032 5	0.008 1	0.000 3	$8.105\,8 \cdot 10^{-05}$	$8.143\,0 \cdot 10^{-07}$	0.347 8	7.123 9
7755	136.191 3	34.491 0	1.361 8	0.341 0	0.013 6	0.003 4	0.000 1	$3.375\,2 \cdot 10^{-05}$	$3.370\,9 \cdot 10^{-07}$	0.292 9	5.420 1
7990	128.778 8	32.210 1	1.279 5	0.318 6	0.012 8	0.003 2	0.000 1	$3.196\,4 \cdot 10^{-05}$	$3.256\,0 \cdot 10^{-07}$	0.296 9	5.506 1
8225	421.301 8	104.454 5	4.265 7	1.074 5	0.043 1	0.010 4	0.000 4	0.000 1	$1.076\,2 \cdot 10^{-06}$	0.398 3	8.799 6
8460	112.160 6	28.464 8	1.147 7	0.286 5	0.011 6	0.002 8	0.000 1	$2.899\,4 \cdot 10^{-05}$	$2.895\,4 \cdot 10^{-07}$	0.304 9	5.672 5
8695	207.093 8	52.013 5	2.025 0	0.511 2	0.020 9	0.005 2	0.000 2	$5.130\,2 \cdot 10^{-05}$	$5.083\,6 \cdot 10^{-07}$	0.356 5	7.201 9
8930	275.101 2	68.298 1	2.776 7	0.679 0	0.027 2	0.006 8	0.000 3	$6.743\,8 \cdot 10^{-05}$	$6.692\,4 \cdot 10^{-07}$	0.387 5	8.237 7
9165	97.352 5	24.231 7	0.972 6	0.247 2	0.009 8	0.002 5	$9.816\,6 \cdot 10^{-05}$	$2.397\,5 \cdot 10^{-05}$	$2.443\,8 \cdot 10^{-07}$	0.315 9	5.903 8
9400	117.987 3	29.345 3	1.168 8	0.292 4	0.011 7	0.002 9	0.000 1	$2.987\,4 \cdot 10^{-05}$	$2.994\,8 \cdot 10^{-07}$	0.337 2	6.487 6
9635	86.744 1	22.285 7	0.882 6	0.218 4	0.009 0	0.002 2	$8.631\,3 \cdot 10^{-05}$	$2.178\,5 \cdot 10^{-05}$	$2.173\,1 \cdot 10^{-07}$	0.323 1	6.057 6
9870	47.634 6	11.751 6	0.472 1	0.117 3	0.004 8	0.001 2	$4.690\,6 \cdot 10^{-05}$	$1.192\,0 \cdot 10^{-05}$	$1.161\,5 \cdot 10^{-07}$	0.287 8	5.038 2
10105	79.640 6	20.236 1	0.819 8	0.202 9	0.008 1	0.002 0	$8.087\,8 \cdot 10^{-05}$	$2.023\,1 \cdot 10^{-05}$	$2.007\,1 \cdot 10^{-07}$	0.333 7	6.207 4
10340	77.089 5	18.999 6	0.769 8	0.191 7	0.007 6	0.001 9	$7.604\,4 \cdot 10^{-05}$	$1.953\,0 \cdot 10^{-05}$	$1.928\,2 \cdot 10^{-07}$	0.337 1	6.277 5
10575	224.955 0	55.320 0	2.218 9	0.560 3	0.022 8	0.005 7	0.000 2	$5.571\,8 \cdot 10^{-05}$	$5.675\,2 \cdot 10^{-07}$	0.432 6	9.445 8
10810	187.150 5	46.620 2	1.820 1	0.470 1	0.018 7	0.004 6	0.000 2	$4.683\,5 \cdot 10^{-05}$	$4.574\,3 \cdot 10^{-07}$	0.422 9	9.052 6
11045	150.441 1	37.277 2	1.529 4	0.372 4	0.015 2	0.003 7	0.000 1	$3.812\,8 \cdot 10^{-05}$	$3.717\,5 \cdot 10^{-07}$	0.410 5	8.629 7
11280	732.507 7	179.781 4	7.359 0	1.843 7	0.073 8	0.018 4	0.000 7	0.000 2	$1.833\,7 \cdot 10^{-06}$	0.565 6	15.811 6
11515	137.023 6	34.341 8	1.410 0	0.352 7	0.014 1	0.003 5	0.000 1	$3.464\,7 \cdot 10^{-05}$	$3.429\,6 \cdot 10^{-07}$	0.418 7	8.791 6
11750	110.955 8	27.946 2	1.120 5	0.281 3	0.011 1	0.002 8	0.000 1	$2.780\,7 \cdot 10^{-05}$	$2.859\,0 \cdot 10^{-07}$	0.407 5	8.356 9

Bibliography

[1] O. Heffetz and K. Ligett, "Privacy and data-based research," *Journal of Economic Perspectives*, volume 28, number 2, pages 75–98, 2014.

[2] N. Amoroso, M. La Rocca, S. Bruno, T. Maggipinto, A. Monaco, R. Bellotti, and S. Tangaro, "Brain structural connectivity atrophy in alzheimer's disease," *arXiv preprint arXiv:1709.02369*, 2017.

[3] A. Singanamalli, H. Wang, and A. Madabhushi, "Cascaded Multi-view Canonical Correlation (CaMCCo) for Early Diagnosis of Alzheimer's Disease via Fusion of Clinical, Imaging and Omic Features," *Scientific Reports*, volume 7, number 1, page 8137, December 2017, ISSN: 2045-2322. DOI: 10.1038/s41598-017-03925-0. [Online]. Available: http://www.nature.com/articles/s41598-017-03925-0.

[4] M. Bahl, R. Barzilay, A. B. Yedidia, N. J. Locascio, L. Yu, and C. D. Lehman, "High-Risk Breast Lesions: A Machine Learning Model to Predict Pathologic Upgrade and Reduce Unnecessary Surgical Excision," *Radiology*, page 170 549, October 2017, ISSN: 0033-8419. DOI: 10.1148/radiol.2017170549. [Online]. Available: http://pubs.rsna.org/doi/10.1148/radiol.2017170549.

[5] *How a computer can help your doctor better diagnose cancer | MIT News*. [Online]. Available: http://news.mit.edu/2015/how-computer-can-help-your-doctor-better-diagnose-cancer-0423.

[6] L. Sweeney, "K-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, volume 10, number 05, pages 557–570, 2002.

[7] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "L-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, volume 1, number 1, page 3, 2007.

[8] N. Li, T. Li, and S. Venkatasubramanian, "T-closeness: Privacy beyond k-anonymity and l-diversity," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, IEEE, 2007, pages 106–115.

[9] Z. Ji, Z. C. Lipton, and C. Elkan, "Differential privacy and machine learning: A survey and review," *arXiv preprint arXiv:1412.7584*, 2014.

[10] Y. Lindell and B. Pinkas, "Secure Multiparty Computation for Privacy-Preserving Data Mining," *The Journal of Privacy and Confidentiality*, volume 1, number 1, pages 59–98, 2009. [Online]. Available: `http://repository.cmu.edu/cgi/viewcontent.cgi?article=1004&context=jpc`.

[11] B. McMahan and D. Ramage, *Research Blog: Federated Learning: Collaborative Machine Learning without Centralized Training Data*, 2017. [Online]. Available: `https://research.googleblog.com/2017/04/federated-learning-collaborative.html`.

[12] C. Dwork, A. Roth, *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, volume 9, number 3–4, pages 211–407, 2014.

[13] C. Dwork, "Differential privacy: A survey of results," in *International Conference on Theory and Applications of Models of Computation*, Springer, 2008, pages 1–19.

[14] F. D. McSherry, "Privacy integrated queries: An extensible platform for privacy-preserving data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, ACM, 2009, pages 19–30.

[15] A. Friedman and A. Schuster, "Data mining with differential privacy," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2010, pages 493–502.

[16] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[17] ——, "Regression shrinkage and selection via the lasso: A retrospective," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, volume 73, number 3, pages 273–282, 2011.

[18] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, 2nd edition. Springer, 2009. [Online]. Available: `http://www-stat.stanford.edu/~tibs/ElemStatLearn/`.

[19] Q. Li, T. Yang, L. Zhan, D. P. Hibar, N. Jahanshad, Y. Wang, J. Ye, P. M. Thompson, and J. Wang, "Large-scale collaborative imaging genetics studies of risk genetic factors for alzheimer's disease across multiple institutions," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2016, pages 335–343.

[20] D. Zhu, Q. Li, B. C. Riedel, N. Jahanshad, D. P. Hibar, I. M. Veer, H. Walter, L. Schmaal, D. J. Veltman, D. Grotegerd, *et al.*, "Large-scale classification of major depressive disorder via distributed lasso," in *12th International Symposium on Medical Information Processing and Analysis*, International Society for Optics and Photonics, volume 10160, 2017, 101600Y.

[21] C. Bishop, *Pattern Recognition and Machine Learning*, series Information Science and Statistics. Springer, 2006, ISBN: 9780387310732. [Online]. Available: `https://books.google.dk/books?id=kTNoQgAACAAJ`.

[22] E. Bingham and H. Mannila, "Random projection in dimensionality reduction: Applications to image and text data," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2001, pages 245–250.

[23] C.-Y. J. Peng, K. L. Lee, and G. M. Ingersoll, "An introduction to logistic regression analysis and reporting," *The journal of educational research*, volume 96, number 1, pages 3–14, 2002.

[24] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*. John Wiley & Sons, 2013, volume 398.

[25] T. Herlau, M. N. Schmidt, and M. Mørup, *Introduction to Machine Learning and Data Mining*. Technical University of Denmark, 2017.

[26] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *Journal of Machine Learning Research*, volume 12, number Mar, pages 1069–1109, 2011.

[27] K. Chaudhuri and C. Monteleoni, "Privacy-preserving logistic regression," in *Advances in Neural Information Processing Systems*, 2009, pages 289–296.

[28] S. Wright and J. Nocedal, "Numerical optimization," *Springer Science*, volume 35, number 67-68, page 7, 1999.

[29] S. Raschka, *Fitting a model via closed-form equations vs. Gradient Descent vs Stochastic Gradient Descent vs Mini-Batch Learning. What is the difference?* [Online]. Available: `https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html`.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[31] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[32] S. Song, K. Chaudhuri, and A. D. Sarwate, "Stochastic gradient descent with differentially private updates," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, IEEE, 2013, pages 245–248.

[33] S. Song, private communication, 2018.

[34] S. Song, K. Chaudhuri, and A. Sarwate, "Learning from data with heterogeneous noise using sgd," in *Artificial Intelligence and Statistics*, 2015, pages 894–902.

[35] X. Wu, F. Li, A. Kumar, K. Chaudhuri, S. Jha, and J. Naughton, "Bolt-on differential privacy for scalable stochastic gradient descent-based analytics," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ACM, 2017, pages 1307–1322.

[36] M. J. Wainwright, M. I. Jordan, and J. C. Duchi, "Privacy aware learning," in *Advances in Neural Information Processing Systems*, 2012, pages 1430–1438.

[37] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, volume 13, number Feb, pages 281–305, 2012.

[38] David Robinson, *Why is Python Growing So Quickly? - Stack Overflow Blog*. [Online]. Available: `https://stackoverflow.blog/2017/09/14/python-growing-quickly/`.

[39] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing USA, 2006, volume 1.

[40] E. Jones, T. Oliphant, P. Peterson, *et al.*, *SciPy: Open source scientific tools for Python*, [Online; accessed <today>], 2001–. [Online]. Available: `http://www.scipy.org/`.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, volume 12, pages 2825–2830, 2011.

[42] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, Austin, TX, volume 445, 2010, pages 51–56.

[43] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, volume 9, number 3, pages 90–95, 2007.

[44] Michael Waskom., *seaborn: statistical data visualization — seaborn 0.8.1 documentation*. [Online]. Available: `https://seaborn.pydata.org/index.html`.

[45] *8.3. collections — Container datatypes — Python 3.6.5 documentation*. [Online]. Available: `https://docs.python.org/3/library/collections.html`.

[46] *16.6. multiprocessing — Process-based "threading" interface — Python 2.7.15 documentation*. [Online]. Available: `https://docs.python.org/2/library/multiprocessing.html`.

[47] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, ACM, 2015, page 7.

[48] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science Engineering*, volume 13, number 2, pages 31–39, 2011, ISSN: 1521-9615. DOI: `10.1109/MCSE.2010.118`.

[49] *19.2. json — JSON encoder and decoder — Python 3.6.5 documentation*. [Online]. Available: `https://docs.python.org/3/library/json.html`.

[50] C. Ramey and B. Fox, *Bash reference manual*. Network Theory Limited, 2003.

[51] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: `http://yann.lecun.com/exdb/mnist/`.

[52] Joseph Redmon, *MNIST in CSV*. [Online]. Available: `https://pjreddie.com/projects/mnist-in-csv/`.

[53] David Robinson, *Exploring handwritten digit classification: a tidy analysis of the MNIST dataset – Variance Explained*. [Online]. Available: `http://varianceexplained.org/r/digit-eda/`.

[54] Team Apple, "Learning with Privacy at Scale," [Online]. Available: `https://machinelearning.apple.com/docs/learning-with-privacy-at-scale/appledifferentialprivacysystem.pdf`.

[55] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang, "Privacy loss in apple's implementation of differential privacy on macos 10.12," *arXiv preprint arXiv:1709.02753*, 2017.

[56] J. Blocki, A. Blum, A. Datta, and O. Sheffet, "The johnson-lindenstrauss transform itself preserves differential privacy," in *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, IEEE, 2012, pages 410–419.

[57] A. Rakhlin, O. Shamir, K. Sridharan, *et al.*, "Making gradient descent optimal for strongly convex stochastic optimization.," in *ICML*, Citeseer, 2012.

{2.7182818284}ϑφερτυθιοπσδφγηξκλ