Error Based Algorithm Selection for Differential Privacy

by

Famien A Koko

S.B., Massachusetts Institute of Technology (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author .....
Department of Electrical Engineering and Computer Science
February 19, 2019

Certified by .....
Samuel Madden
Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chairman, Master of Engineering Thesis Committee

Error Based Algorithm Selection for Differential Privacy

by

Famien A Koko

Submitted to the Department of Electrical Engineering and Computer Science
on February 19, 2019, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

Differential privacy has recently emerged as a robust framework for delivering privacy guarantees when performing data analysis. Researchers have developed a wide array of algorithms which can provide these privacy guarantees. These algorithms differ in their performance on different datasets. This thesis explores a system for selecting the best algorithm that achieves a target error, given a database and workload: "error-based algorithm selection". In particular, we explore whether this problem can be solved in a private matter, how accurate it can be, and if it can be done competitively.

The system we propose approaches the problem by building models for differentially private algorithms which use features from the workload and dataset as well as the target error to predict an epsilon value that will achieve the target error. It then selects the algorithm which predicts the best epsilon and uses it to run the workload.

We evaluate the system on 1D and 2D workloads with several different algorithms and datasets. The system predicts the best algorithm a large percentage of the time, usually only being beaten by one algorithm and selects an algorithm that is within the top 2 best algorithms a majority of the time. The individual algorithm models achieve target error within a reasonable margin. The results show this approach is viable for the error-based algorithm selection problem, solving it in a differentially private, algorithm agnostic and competitive manner.

Thesis Supervisor: Samuel Madden
Title: Professor

Acknowledgments

This thesis could not have been done without the support of my advisor Samuel Madden, who has been very patient and helpful to me throughout my program. I am also grateful to my family, friends and God. MIT has given me the chance to grow, opportunities to make great friends, and shown me how to solve problems and make an impact. My sincerest gratitude goes to all who have helped me get this far.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Privacy matters. Even though people desire a certain degree of privacy in their lives, we live in world where this is becoming increasingly difficult. More and more information is being moved to the cloud, and other information such as health records are also being digitized. This wealth of data also provides the opportunity to help people through research such as healthcare analytics. We need ways to be able to analyze data while being able to give privacy guarantees to the individuals to whom the data belongs. [4]

1.1 Motivation

One privacy area of interest is concerned with statistical databases. The goal of these databases is to release statistical information about the underlying data without disclosing information about any particular entry in the database. An example of these may be a dataset containing health information of patients. Researchers would like to analyze the data to look for trends but should not be able to breach an individual's privacy while doing so.

Differential privacy is a widely used formal guarantee which addresses this issue. It promises that answers to any queries performed on the data will not be significantly different whether or not an individual is included in the data.

Since the conception of differential privacy, many differentially private algorithms

have been developed. An early differentially private algorithm, the exponential mechanism [3], produces answers by randomly adding noise to outputs. Doing this with sufficiently large noise satisfies differential privacy, but often adds much more noise than needed. This is the balance of private analysis: analysts want the most accurate results possible, while still preserving the privacy of individuals whose information is in the data. Modern algorithms seek to address this balance, and to provide more and more accurate results while remaining differentially private. This leaves the user to choose which algorithm will be best for their needs. The general case is that a user would want the algorithm which would give them the most accuracy while guaranteeing an acceptable level of privacy.

Differentially private algorithms fall into two broad categories: data-independent and data-dependent. Data-independent algorithms work the same regardless of the database they are operating on. The Laplace Mechanism [3] is a classic example of a data-independent algorithm. It works by randomly drawing from the Laplace distribution, and adding the result as noise to any query on a database. These types of algorithms generally have the same expected amount of error for any database. Data-dependent algorithms seek to use information about a database and often the workload, or set of queries that will be run on the database, to more optimally add noise to minimize the amount of error for the given queries while maintaining privacy.

When it comes to choosing an algorithm to use in a particular situation, there are many factors to consider. Important areas to consider are the datasets and query workloads being used, what amount of privacy is desired and what amount of error is tolerable. Different privacy mechanisms will have different performance characteristics (both error-wise and speed-wise) based on these parameters. This work seeks to address the situation when a user has a target error they are willing to tolerate, and wishes to find the algorithm which gives the best privacy guarantee while staying below the target error.

1.2 Contributions

The goal of this project is to provide a system which can take a database, workload and target error rate and set of differentially private algorithms and select an algorithm which achieves that error rate while maximizing the privacy guarantee. In this work we develop the following:

1.) A method for developing models for differentially private algorithms to determine what privacy budget is needed to achieve an error rate given a database and workload

2.) Evaluation metrics for these models as well as the entire method

1.3 Definition of Differential Privacy

Differential privacy seeks to address the problem of formalizing guarantees for privacy. We live in a world full of information, and new advances computation have allowed us to gain great insights from this information. At the same time, individuals don't want others having access to their own personal data. These two desires are not always incompatible: often only aggregate, or statistical information is needed from data in order to gain insights. For example, you may have a dataset which records smoking activity along with health information of individuals. Researchers may wish to find links from smoking to health issues. This kind of information is statistical in nature, and shouldn't involve revealing any particular information about an individual. An individual in the dataset who is a smoker would likely not want insurance companies to know that they are a smoker and raise their premiums. They would want some sort of privacy guarantee that the answers from queries to the database don't expose their individual information.

Differential privacy lets us tie conclusions from queries to the privacy guaranteed to participants. In the smoking example, we may want to learn about the health risks of smoking. Differential privacy says that reaching a conclusion about the health risk on the population due to smoking does not violate an individual in the study's

privacy because taking out one individual will not affect the overall conclusion about the health risks of smoking. We can generalize this to statistical databases. The guarantee is that a series of outputs to queries on the database are equally likely to occur regardless of whether or not any particular individual is in the dataset [3]. We capture this relationship of probability of outputs to privacy through a parameter: ϵ. A smaller ϵ corresponds to a higher privacy guarantee for an individual in the dataset.

1.3.1 Where other methods fail

A natural question is, why don't other, simpler approaches work in this context? One immediate idea is anonymization. Can't we just strip any personifiable information from a dataset and the publish it? So-called "linkage attacks" on large companies such as Netflix and IMDb and even organizations such as the state of Massachusetts [3] demonstrate that this will not work. In these linkage attacks, attackers cross-reference different data sources in order to de-anonymize the data. In the Netflix example, Netflix had released anonymized viewing histories for a competition, but attackers linked this information with IMDb and were able to uncover identities of individuals in the Netflix database.

Another idea is limiting which queries you can ask the database. If you want to preserve individual's privacy, maybe you can just restrict questions that are about individuals. So you could ask "what percent of people smoke?" but not "does Bob smoke?". This type of approach lays victim to a "differencing attack". In a differencing attack, an attacker asks multiple questions which put together can narrow information down to an individual. For example, if you ask "how many people not named 'Bob Smith' are smokers?", and "how many smokers are in the database?", you can infer whether or not Bob is a smoker.

There are other approaches which fail to give the same guarantees as differential privacy, such as "summary statistics", "query auditing" and releasing "ordinary facts" which are addressed in [3].

1.3.2 Formal Definition

The standard setting for differential privacy analysis involves individuals in a database x, controlled by a curator. We treat databases as collections of records drawn from a universe χ. We represent the database as a vector $\mathbf{x} \in \mathbb{N}^{|\chi|}$, where each entry x_i in $\mathbf{x}$ denotes how many records of type $i \in \chi$ are in the database. $|\chi|$ represents the domain size, or the number of different types of records from which x is drawn [3].

The curator of the database is assumed to be trustworthy. Users of the database will then ask questions in the form of queries which are functions applied to the database. The user may or may not be adversarial, so we want to ensure that statistical information can be released without compromising the privacy of any individual. Users ask queries in either an online or an offline setting. In an offline, or "non-interactive" setting, the curator provides a "synthetic database" which is a perturbed version of the original which provides statistical information useful for analysis. In the interactive setting, users are given an interface which they can use to ask a stream of queries to the database.

The curator will then use a privacy mechanism to give an answer. A privacy mechanism is an algorithm that takes in a database, random seed, and potentially a set of queries and will give an output string [1] which can hopefully answer queries.

Here we define the l_1 norm of a database x, (denoted as $||x||_1$)

$$\big|\big|x\big|\big|_1 = \sum_{i=1}^{|\chi|} |x_i|.$$

The l_1 distance between two databases, x and y is then given as $||x - y||_1$ [1].

In other words, this is a measure of the number of records that differ in two different databases.

We can now define differential privacy as follows:

Definition 1.1. (Differential Privacy [3]). A randomized algorithm A with domain $\mathbb{N}^{|\chi|}$ is ϵ-differentially private if for all $S \subseteq \text{Range}(A)$ and any databases $\mathbf{x}, \mathbf{x}' \in \mathbb{N}^{|\chi|}$

such that $||\mathbf{x} - \mathbf{x}'||_1 \leq 1$:

$$Pr[A(\mathbf{x}) \in S] \leq e^{\epsilon} Pr[A(\mathbf{x}') \in S]$$

Databases

For this work we consider databases where the records drawn from χ have 1 and 2 elements, corresponding to data with only 1 or 2 attributes, but the concepts apply generally to arbitrarily multiple attributes.

Queries

We define a query workload as a set of counting queries on a database $\mathbf{x}$. The workload, $\mathbf{W}$ is a collection of queries where each query $\mathbf{q} \in \{0, 1\}^{|\mathbf{x}|}$ is vector where q_i indicates whether or not to include the count of x_i in the query result.

Examples of workloads include the Identity workload $\mathbf{I}$ which just returns $\mathbf{x}$, as well as the Prefix workload $\mathbf{P}$ which asks the count of all records between x_0 to x_i for i in $|\mathbf{x}|$. Because data curators often would like to answer multiple workloads, for this system we will group workloads together into tasks, for example 1D range queries or 2D range queries.

Target Error

To maintain consistency across different datasets, in this work, target error will refer to the scaled l_1 error between an actual vector $\mathbf{x}$, and its differentially private calculated vector $\tilde{\mathbf{x}}$

1.3.3 Algorithms

The set of algorithms, **A**, is meant to be extensible so that a user can add or remove any algorithms they want (as long as each algorithm satisfies differential privacy). The system treats the algorithms as black boxes, and doesn't need to know the characteristics of the algorithm a-priori.

The algorithms used in the experiments are listed below:

Table 1.1: Algorithms Used

Algorithm Name	**Tasks**	Prior Work
Data Independent		
HB	Range Queries	[11]
Privelet	Range Queries	[12]
Data Dependent		
Identity	General Purpose	[2]
DAWA	Range Queries	[8]
MWEM	General Purpose	[4]
AHP	General Purpose	[15]
AGrid	2D Range Queries	[10]
DPCube	2D Range Queries	[13]

The Hb method is a data-independent hierarchical algorithm which arranges a histogram of data into a tree with branching factor b. Queries come in to a particular level, so the privacy budget can be tuned to accommodate for different ranges where the noise will lead to proportionally more error. [11]

Privelet is a data-independent algorithm that applies a wavelet transform on the input data, adds noise, then maps the transform back to the original domain, allowing results with bounded poly-logarithmic noise. [12]

The Identity method is a simple data-independent algorithm which estimates a dataset by obtaining the count for each bin by adding noise from the Laplace mechanism. [2]

DAWA (*Data-Aware/Workload-Aware*) is a data-dependent mechanism which uses the specifics of the dataset and query workload to add noise in a optimal way to minimize error. It buckets the dataset based on the distribution and then privately estimates bucket counts to fit the query workload. [8]

MWEM (*Multiplicative Weights/Exponential Mechanism*) is a data-dependent algorithm which essentially starts with a distribution over the domain of the dataset, and continually refines the distribution by selecting queries which are poorly answered by the distribution and updating it accordingly. [4]

AHP (*Accurate Histogram Publication*) is another data-dependent algorithm which further pushes the concept of clustering bins of data with similar counts and approximate their counts by the mean to reduce Laplace noise. [15]

AGrid (*Adaptive Grid*) is a method specifically for 2D datasets. It forms a 2 layer hierarchy over the dataset and adaptively finds a good grid size to use for each cell. [11]

DPCube is a data-dependent algorithm for multidimensional data which uses a multi-dimensional partitioning. It releases differentially private "cuboids" as answers to private queries by first creating an equi-width cell histogram made by partitioning based on the domain of the dataset, then estimates the counts of cells by adding Laplacian noise to create a data cube, to which a multi-dimensional partitioning scheme is applied to give the final approximation to the dataset. [13]

1.4 Related Work

This work builds on the use of a variety of differentially-private algorithms. People have developed a vast array of different algorithms all with different use cases and characteristics. In their work on "Pythia" [7], Kotsogiannis et al. introduced the notion of "Algorithm Selection" and developed a system which allowed users to select the algorithm which would give them the best error. They achieved this by implementing a Feature-based Algorithm selector which used decision trees. They would first use a regret based learning method to learn the correlation between features extracted from the data instance and workload with algorithm performance. Once the tree was learned, the system could take a a data instance, privately extract the required features, and input them to the decision tree to determine the best algorithm. The flip side of this problem is when a user wants to achieve a particular error rate,

and wants to select the algorithm which can achieve that error rate while providing the most privacy. The inherent difficulty with this is that given an error rate, it is not straightforward to determine the necessary privacy budget for any given algorithm. This work seeks to find a solution to this issue.

Chapter 2

Approach

In this chapter we formalize the notion of Error-Based Algorithm Selection, describe a system which addresses this problem by learning models of algorithms to predict the privacy budget necessary to achieve a particular error-rate, and finally describe the method used for data generation.

2.1 Error-Based Algorithm Selection

Here we formally define the problem of error based algorithm selection, what we want in solutions, and limitations of other approaches.

Error based algorithm selection is given a set of algorithms, database, workload, and a desired error rate.

To select an algorithm we use a meta-algorithm. The meta-algorithm involves a few different components: a set of differentially private algorithms, a workload of queries, a database, and a desired (scaled) error rate. These terms are all defined below.

$\mathbf{A}$:= Set of differentially private algorithms

$\mathbf{D}$:= Database, a multiset of records, described as a vector $\mathbf{x} \in \mathbb{N}^d$

$\mathbf{W}$:= query workload. A set of queries where each query $\mathbf{q} \in \{0,1\}^{|\mathbf{x}|}$ is vector where q_i indicates whether or not to include the count of x_i in the query result.

$\boldsymbol{\epsilon}$:= epsilon under differential privacy definition

Given vectors $\mathbf{x}$ and $\mathbf{x}'$:

scaled error :=

$$\frac{\sum_i^{|\mathbf{x}|} |x_i - y_i|}{\sum_i^{|\mathbf{x}|} x_i}$$

2.1.1 Criteria

As in Algorithm Selection, we want a method of selecting an algorithm that is differentially private, algorithm agnostic, and competitive. It should be differentially private so that it can be used instead of any particular algorithm while still providing the same privacy guarantees. It should be algorithm agnostic because new algorithms are still being researched and developed and the system should be able to support addition of these new algorithms and still work as expected. The system should be competitive, meaning it provides performance on par with any other particular algorithm.

2.2 Algorithm Selection Method

Given the problem set-up as given above, we need a way to choose an algorithm, given all the parameters. In this section we examine potential methods, and describe the method we will ultimately choose to use.

2.2.1 Potential Methods

Blind Choice

One idea might be to just take a random algorithm, or just use one algorithm for all inputs. There are two major problems with this. The first is that we don't actually know a-priori what epsilon should be used for our selected algorithm to achieve the target error rate. One option is to try to model a particular algorithm's error/epsilon characteristics. We could then plug in our parameters to this model to get a prediction on the epsilon we will need for our target error rate. Even doing this however,

does not address the second major problem, which is finding the optimal algorithm given our parameters. If we simply use one algorithm, or a random one, even if we can predict the epsilon we need to use for that algorithm, a different algorithm may be able to achieve the same target error with a better privacy budget.

As it turns out combining the solutions to these two problems will form the basis of our approach, which we describe after considering a couple other alternatives.

Informed choice

Another idea is to try out different algorithms and epsilons to see which one works best. The issue with this is that if you run the workload on the dataset, you are leaking information, which violates the privacy guarantee [7].

Decision Tree

A learning-based approach could be to learn a decision tree model for picking the best algorithm. You could take representative databases, errors and epsilons, and fit a decision tree model to predict the best algorithm given the features drawn from the dataset and workload. An issue with this method is obtaining the training data. You could generate synthetic data by running the algorithms on datasets with different workloads and epsilons. Ultimately, however, you would want to train on tuples of database and workload parameters, along with a target error and the label for each data sample should the best algorithm for those parameters. The difficulty is finding the actual best algorithm for each data sample, since a large part of the problem is not knowing what epsilon is needed for a target error in the first place. A potential solution to this would be to take a target error, say 10% and try various epsilons for each algorithm, running them on the data until you find a combination that produces the target error. You could then compare the algorithms and use the best one as the label for the data sample. The major problem would be doing this for a wide enough array of target errors.

Learning Epsilon-Error Relation

A similar learning approach to the decision tree method would be to instead learn to predict the epsilon for each algorithm for a given set of parameters. If we know all the epsilons required for each algorithm, we can simply choose the best (smallest) one to use for our workload. Generating data in this case is fairly straightforward. Essentially we run all the algorithms on different combinations of databases, workloads and epsilons. We then observe the resulting error, and then feed the data into a learning model, except we use error as part of the input and epsilon as the output. An advantage of this approach is that we get as a result an approximation of the epsilon needed to obtain a target error rate. This could be useful if a user wishes to know how much privacy it takes to achieve a target error without actually running a workload.

This epsilon-learned approach will be studied in the rest of this thesis. The following sections describe how the system using this method works.

2.2.2 Epsilon-Learned Approach

The approach we take to picking an algorithm is to try to predict what epsilon is needed to achieve the target error rate for each algorithm (given a database and workload), and select the algorithm with the best predicted privacy budget. In order to predict the epsilon, we use an offline regression model to learn what epsilon would be needed to achieve a certain error rate given a set of parameters. We can do this for each algorithm in $\mathbf{A}$. Then, when we are given a database, query workload and a desired error rate, we can input this information into the model for each algorithm, and then select the model with best predicted epsilon.

2.2.3 Feature Based Algorithm Selection

We want a model for each algorithm that will predict the necessary epsilon to achieve a certain error rate given our set of parameters. To build the model, we gather a set of representative data and then use this to train the model. The inputs to the model for each algorithm in **A** will be drawn from the database, query workload, and a desired error rate. The question becomes, how to get the input data? We want a representative selection from the full space of inputs. We discuss how we generate the input data in the next section. Once we have our pairs of inputs and outputs we can train the model.

The particular model used was a Random Forest Regressor. We split the data into training and testing data, and then generated a model for each of the algorithms and evaluated the results. The evaluation methods and results are discussed in Chapter 3.

Once the models are created for all the algorithms, we can now accept workloads from users. Users will input their dataset, workload and target error. We then privately extract features from the dataset. We use the extracted features, workload and target error and the algorithm models to predict the epsilons needed from each algorithm, then select the algorithm which gives the best privacy budget.

2.3 System Architecture

Out goal is to build a system that can go from a database, workload queries and target error to private answers to the workload queries. The high level view is shown in the diagram below.

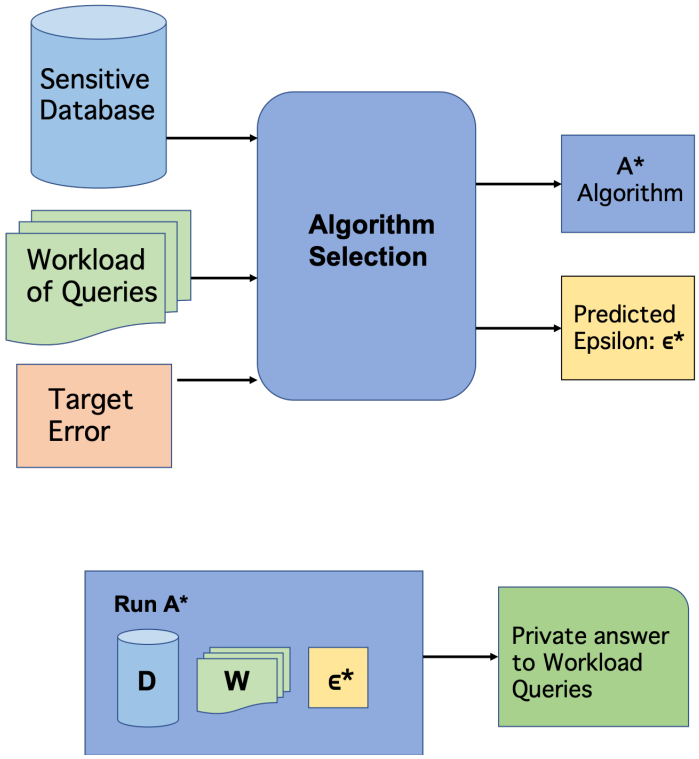


Figure 2-1: Overview of System Architecture

The system will first build models for predicting epsilons for each of the algorithms in **A** offline. It then uses the workload, dataset features and target error to predict epsilons for the algorithms. It selects the algorithm with the lowest predicted epsilon, and uses it to run the workload on the dataset and finally returns the private answers to the user.

Algorithm Models

The model used for experimenting was a Random Forest Regressor. The advantage to this model is that it allows the input of multiple features and is able to decide the weights for different features itself. Other model types such a neural network are possible, but were not used here largely because of training time.

The system builds multiple models for each algorithm, forming a tree structure. At the root of the tree is the workload type. For instance, if the workload is 1D Prefix, that will lead to one set of algorithms, if it is 2D Random Range, that will

lead to another. Within a Workload Class of models, we have two models for each algorithm. One is a "simple" model, which will be used to predict epsilon solely based on target error, and another "feature" model for predicting epsilon using features from the dataset. The models use features from the dataset in order to make better predictions and the process of obtaining these features is described in section 2.3.3.

2.3.1 Defining "Best Algorithm"

Using the models for the algorithms we included, we can give a data set and parameters, and use the meta algorithm to select the "best" algorithm, then evaluate our choice. We define the best choice as follows. For a database and target error, we use the meta-algorithm to choose the differentially-private algorithm with the smallest epsilon, then use that epsilon and plug it back into all the algorithms. The "best" algorithm will be the one with the lowest resulting error. The reasoning for this definition is that the more privacy budget an algorithm has, the smaller the resulting error. Thus, for the other algorithms to achieve the same error as the lowest one, they would need to increase their privacy budget, indicating that the selected algorithm will likely have the best performance for our data.

A limitation of this definition is that it doesn't account for when the "best" algorithm doesn't achieve the target error using the predicted epsilon. For example, the "best" algorithm may actually achieve a much smaller error after being run with the predicted epsilon. In this case, it might be that it could achieve the observed error with the smallest privacy budget, but maybe a different algorithm could have achieved the target error rate using a much smaller privacy budget. If the user would have preferred trading off the error for more privacy (as long as we achieved the target error rate) then this metric fails to capture the "best" algorithm. Similarly, if the algorithm we chose turns out to have an observed error much larger than the target error, it may take much more privacy budget to achieve the target error for that algorithm, while a different algorithm may be able to achieve it with a lower privacy budget. These limitations can be addressed by also measuring the accuracy of the models. We can use the models to predict epsilons, run the algorithms using those

predictions, and measure how close the observed errors are to the target errors. We use this method in the evaluation section in Chapter 3.

The process of algorithm selection is shown below.

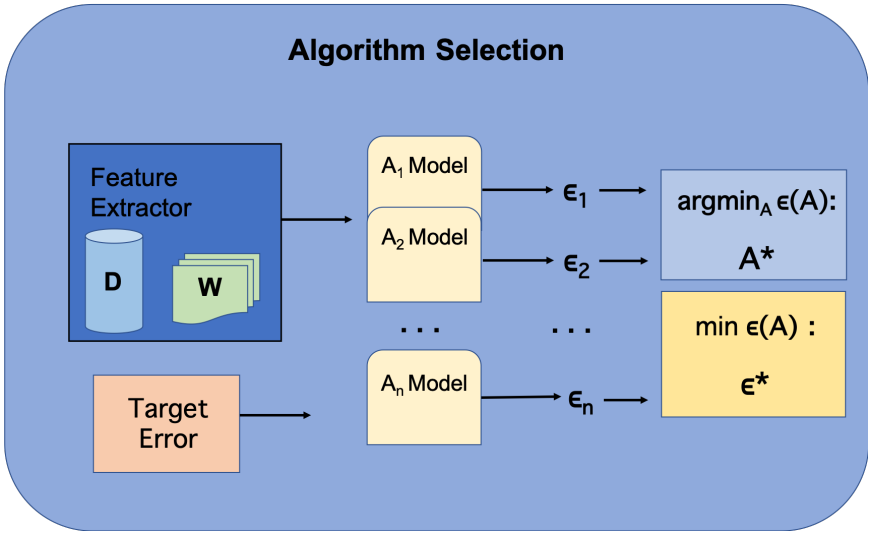


Figure 2-2: Algorithm Selection Mechanism

2.3.2 Differential Privacy Satisfaction

A key criterion for our system is that it satisfies ϵ-differential privacy.

To prove that our system is differentially private, we first give the following theorem:

Theorem 2.1. *(Sequential Composition [9]). Let A_1, ... A_k be algorithms, where each A_i satisfies ϵ_i-differential privacy. Then their sequential execution on the same dataset satisfies $\sum_i \epsilon_i$-differential privacy.*

Sequential Composition lets us combine multiple privacy algorithms and still provide a sound guarantee, and allows us to prove the privacy our system:

Theorem 2.2. *The system satisfies ϵ-differential privacy where $\epsilon = \epsilon_{features} + \epsilon_{best\_predicted}$.*

Our system works in two stages: It first uses $\epsilon_{features}$ of the privacy budget to privately extract features, then uses these features to predict epsilons for all of the algorithms, and picks the one with the lowest predicted epsilon, $\epsilon_{best\_predicted}$. Finally

it runs the selected algorithm on the database and workload using the predicted epsilon, $\epsilon_{best\_predicted}$. Using the Sequential Composition theorem, it directly follows that our system satisfies ϵ-differential privacy.

2.3.3 Training Data

The models for each algorithm take in features extracted from the database and workload, along with the target error, and output an epsilon. In order to deal with different workloads, we build a different sub-model for each task (1D Prefix, 1D Identity, 2D Range etc), and then use the user provided workload to select the proper sub-model to use. In order to build each of these sub-models, we want to have training data tuples of the form (database feature 1, database feature 2..., target error feature) with an associated output of (epsilon). To get these tuples, we will simply run each algorithm with different databases and different epsilons.

For the purpose of limiting total data generation, we did two things: limiting scale and error range. For scale, we only used 2^{15}. This seemed to be large enough to capture many different distributions and show if the models could be trained holding this value constant. Future work could expand the number of scales used and include it as a feature in training. For domains we used 512, 1024, and 2048 for 1D and 32×32, 64×64, 128×128, and 256×256 for 2D. For target errors we used errors in the range (9.5e-9%, 2.1e-6%) for 1D and (5e-6%, 8e-6%) for 2D. To get sufficient data that would be accurate in the target error range, we first ran all the algorithms with a wide range of datasets and epsilons. We then filtered by results where the observed error rate fell within our target range, then generated training data by using the epsilons used in this filtered set to get corresponding training samples.

Databases

As mentioned before, three key qualities of a database are its shape, scale and domain size. Thus, we will want to train our models on databases which vary in all

of these dimensions. In order to get these, we start with a base set of real world datasets as used in [7]. Using these datasets allows us to sample from realistic data. We then further diversify our data by varying each of these datasets by changing it's scale and domain size, resulting in as many databases as we want. The datasets used for different tasks are outlined below.

Table 2.1: Datasets

Dataset Name	**Domain Size**	**Original Scale**	**Prior Work**
Task: 1D Range Queries			
ADULTFRANK	4096	32561	[4],[8]
HEPTH	4096	347414	[8]
INCOME	4096	20787122	[8]
MEDCOST	4096	9415	[8]
NETTRACE	4096	25714	[1],[6],[14],[15]
SEARCHLOGS	4096	335889	[1],[6],[14],[15]
PATENT	4096	27948226	[8]
Task: 2D Range Queries			
ADULT-2D	256x256	32561	[4],[8]
LOAN-2D	256x256	6442863	[5]
MD-SALARY-2D	256x256	70526	[5]
TWITTER	256x256	193563	n\a

Workloads

The workloads used to create training data consisted of 1D and 2D workloads. For 1D we used the Identity and Prefix workloads. For 2D we used Random Range queries. These are very basic workloads and can be combined in many ways. For the purposes of our experiments we only tested with these basic workloads. After generating the training sample, a model was built for each algorithm for each workload. Then, when a user inputs their workload into the system, it is matched with the set of models that most closely matches their workload.

Model Features

As shown in [5], algorithm performance greatly depends on the scale, shape and domain size of the dataset it is operating on. This makes these characteristics good candidate features for the learning model. We need to obtain these features privately however, so we add random noise from the Laplace distribution for reach of our features. We can easily calculate scale and domain size, but we need a way of representing shape. Some options considered were the number of nonzero elements, variation from the uniform distribution (uniform distance), and range. We could pass in all of these features, but because we are using privacy budget to extract them, this will leave less privacy budget for actually running the queries. So we want to balance this trade-off of providing enough information to our model to make accurate predictions while also preserving enough privacy budget for answering queries.

After creating the training data, we calculated all the features for each sample and fed them into the model. The investigated features are described below:

- The *scale* is given as $||\mathbf{x}||_1$, which represents the total number of records in the dataset.

- the *log of scaled error* is the log of scaled error as defined previously.

- The *domain size* is given as $|\mathbf{x}|$, the magnitude of the domain from which the dataset is drawn.

- The *uniform distance*, is the total variation between $\mathbf{x}$ and the uniform distribution, given by $ud(x) = \frac{1}{2}\sum_{i=1}^{|\mathbf{x}|} |x_i - u|$ where $u = ||\mathbf{x}||_1/|\mathbf{x}|$.

- The *standard deviation* is given as $\sigma = \sqrt{\frac{1}{||x||_1}\sum_{i=1}^{||x||_1}(x_i - u)}$ where $u = ||\mathbf{x}||_1/|\mathbf{x}|$ standard deviation

- The *number of non-zeros (NNZ)* is the total number of elements in the dataset not equal to zero, given by $|\{x_i \in \mathbf{x} | x_i \neq 0\}|$

- The *data range* is the difference between the largest and smallest elements in $\mathbf{x}$, given by $max(\mathbf{x}) - min(\mathbf{x})$

The Random Forest Regressor model used gives the importance of features based on performance. We first ran the regression (on 1D data) with all of the above features, and then picked the features that had the highest importance.

Table 2.2: Initial Feature Importances for Algorithm Models

	Log Scaled Error	**Domain Size**	**Uniform Distance**	**Standard Deviation**	**NNZ**	**Data Range**
HB	96%	2%	0	0	1%	0
AHP	75%	2%	7%	4%	9%	2%
Identity	96%	3%	0%	0%	0	0
DAWA	74%	1%	4%	7%	8%	7%
Privelet	100%	0	0	0	0	0
MWEM	81%	1%	6%	3%	5%	5%

Table 2.2 shows the initial feature importances after running the Random Forest Regression model. We used the log of scaled error because when we graphed the relationship between each feature and the corresponding epsilon, error and epsilon seemed to have an exponential relationship in general The other features seemed to do well without transforming them further.

All of the models place the largest significance on the log of scaled error. This makes sense for the data-independent models (HB, Privelet, Identity) because they don't use characteristics of the data when adding noise, so the main determinant for error is the epsilon parameter. For the data-independent algorithms (DAWA,AHP, MWEM), we see the other data-specific parameters (uniform distance and data range) take a greater importance, which makes sense because these mechanisms use aspects of the data to fine tune noise to optimize error. For example, if a dataset is uniform, then a mechanism such as DAWA will likely not provide significant error reduction over a data-independent algorithm, especially since it uses some of the initial privacy budget to try to optimize for noise reduction.

To decide the final features, we looked at which features were most significant on average, which appeared to be log of scaled error, scale, uniform distance and data range. The data we used for training likely affected the importances, and a wider array of data may cause different features to be more important.

Table 2.3: Final Feature Importances for Algorithm Models

	Log Scaled Error	**Uniform Distance**	**Data Range**
HB	97%	1%	2%
AHP	77%	11%	12%
Identity	98%	1%	1%
DAWA	79%	10%	12%
Privelet	100%	0	0
MWEM	82%	10%	8%

Table 2.3 shows the adjusted feature importances selecting the features mentioned previously. Again we see data-independent mechanisms have nearly all of the importance on error and scale, while the data-dependent mechanisms have more weight on the other features.

In terms of optimization there are a few unaddressed areas: how much of the privacy budget to allocate to feature extraction (as discussed previously) and exactly what features to use, and per-mechanism optimization. For our experiments, we use the simple algorithm models to predict the required epsilon for each algorithm in our set of algorithms. Then, for each of the ϵ_{prelim} predicted for each algorithm, we use $\rho * \epsilon_{prelim}$ privacy budget to extract features for each algorithm model by adding noise drawn from the Laplace distribution with parameter $\frac{1}{\rho * \epsilon_{prelim}}$.We then plug the privately extracted features into the final algorithm model to predict the epsilon for each algorithm, $\epsilon_{predicted}$

We used $\rho = 0.1$, but more work needs to be done to find the optimal value of ρ. We used an empirical approach to determine which features to use, but these might not be optimal in all cases. In terms of per-mechanism optimization, as seen in the feature importances, some algorithms use certain features more than others, so privately extracting them to be used equally for all algorithms may be wasting privacy budget. More work needs to be done to see if there are better strategies for picking features and allocating privacy budget for extracting them.

Chapter 3

Evaluation

In this chapter we examine the performance of the system based on several experiments. We evaluate the system as a whole as well as the individual algorithm models and finally discuss challenges and potential areas for future work.

3.1 Evaluation Methods

The main evaluation metric used for evaluating system performance was measuring how often the system picked the "best" algorithm as defined previously (the algorithm with the lowest error for a given epsilon). We also evaluate the system by how accurately each algorithm model predicts epsilons. This is a useful metric because it allows us to see how well we can predict different algorithms, and approximate how the system will do when adding in a new algorithm. It is also a good metric for the use case when a user simply wants to know how much privacy they can get for a database and specific target error. The following sections go into these evaluation methods in detail.

3.2 System Test

The setup of for these experiments is similar to the setup for generating training data. We take databases and workloads and a scaled target error and try to predict which

algorithm will give us the best privacy guarantee while achieving the target error.

The databases are drawn from those given in Table 2.1. The workloads used were 1D Identity, 1D Prefix and 2D Random Range. The target error is given as a desired per-query l_1 error divided by the scale of the database.

For each set of database, workload and target error we input these parameters into the system to get a predicted best algorithm along with it's associated predicted epsilon. Using the predicted epsilon from this predicted best algorithm, we run the predicted best and other algorithms with the same parameters and record the observed l_1 errors. We then rank how well the predicted best algorithm did relative to the other algorithms.

The scaled target errors we used were in the range (9.5e-9%, 2.1e-6%) for 1D and (5e-6%, 8e-6%) for 2D, and were also used when fine-tuning the algorithm models.

After running each error with all of the different databases workloads, we record the predicted algorithm in each case, and run all the algorithms using the predicted best epsilon and evaluate the results.

For comparison, the blind-choice baseline where we would use just one algorithm for all workloads is also shown to compare with the error-based selection method. The graphs below show how often our method chose the best algorithm and how often the other algorithms were actually the best. Graphs are also shown with 25% and 50% slack, representing how often we were within 25% and 50% of the best algorithm respectively.

1D Workloads

In Figure 3-1 we see that most of the time MWEM or DAWA are the best algorithms for a dataset workload and epsilon. This is likely due in part to the non-uniformity of the datasets used in the experiments. The average scaled uniform distance of the datasets used was 0.3, indicating significant variation from the uniform distance. This likely allowed the data-dependent mechanisms to create optimizations unlike the data-independent algorithms.

Figure 3-2 shows how often the algorithm we selected was at least within 25% of the actual best algorithm. The "slack" is only applied to our method, while the

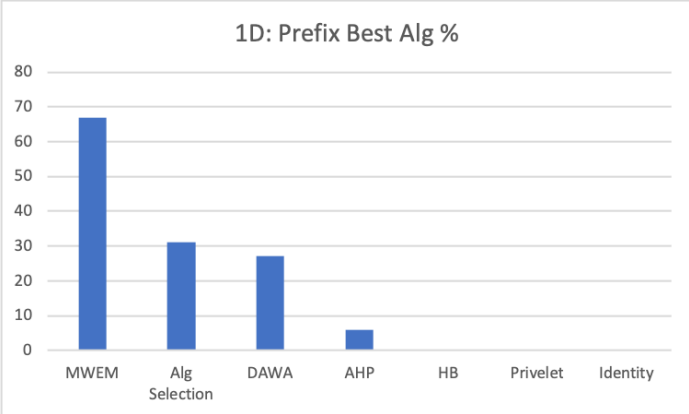


Figure 3-1: Algorithm Selection Performance: 1D Prefix Workload

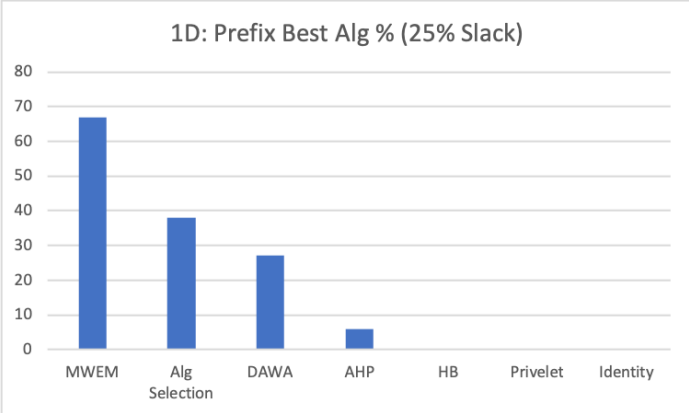


Figure 3-2: Algorithm Selection Performance: 1D Prefix Workload (25% Slack)

performance of the other individual algorithms remains the same. With this margin, our system performances increases from 31% accuracy to 38% accuracy. This indicates that for a section of the data our predicted algorithm is actually close in performance to the best algorithm. We think a majority of this improvement comes from the cases where MWEM and DAWA actually perform similarly, and it becomes challenging to differentiate between them. This is illustrated by the confusion matrix below (Figure 3.1 which shows what our system predicted vs what the actual best algorithm (rows are system predictions, columns are actual best).

68% of the time it picked DAWA, MWEM was actually the best and 36% of the time it picked MWEM, DAWA was actually correct. We believe this shows that there is a lot of overlap to when these two algorithms perform similarly. It is likely that with more training data including more epsilons within a given range will allow the model to

Table 3.1: Algorithm Selection Confusion Matrix: 1D Prefix Workload

	MWEM	DAWA	AHP
MWEM	59%	36%	5%
DAWA	68%	26%	6%
AHP	50%	50%	0

better distinguish between these algorithms and outperform any individual algorithm in the set. Further work should be able to determine how well the algorithms can be distinguished with more training and perhaps different model types. These results show that the system can pick among the best algorithms (in these experiments it only every chose 3 out of the 6 algorithms, which were the best almost 100% of the time), though it can have a hard time distinguishing among the top performers.

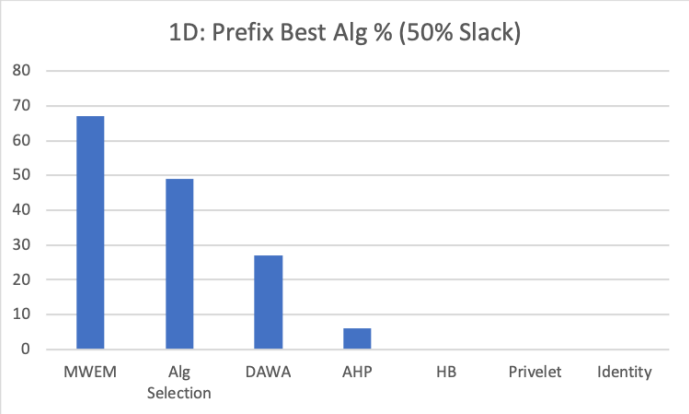
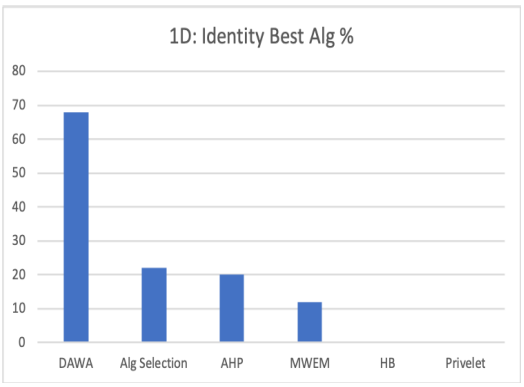
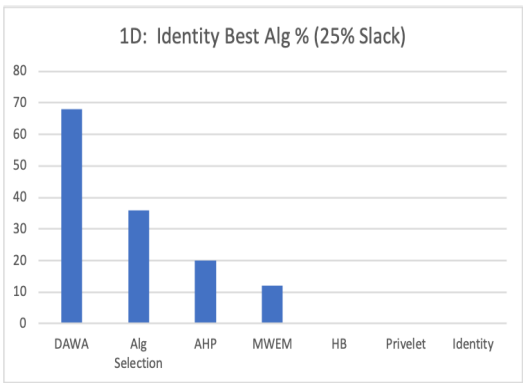


Figure 3-3: Algorithm Selection Performance: 1D Prefix Workload (50% Slack)

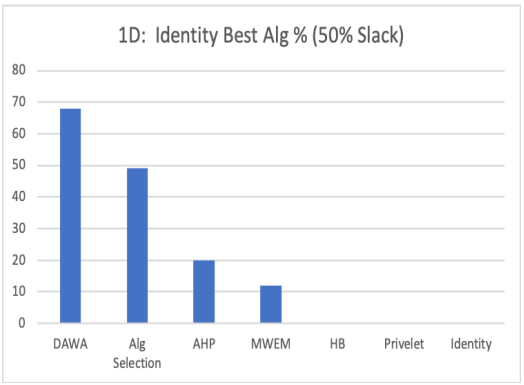
Figure 3-3 shows further increase in accuracy when the slack is 50%. At this point we don't outperform MWEM, but we are picking the best algorithm about half the time. Table 3.3 shows that we are picking at least one of the top 2 algorithms 76% of the time, which also indicates good performance even if it is not optimal.



(a) No Slack



(b) 25% Slack



(c) 50% Slack

Figure 3-4: Algorithm Selection Performance: 1D Identity Workload

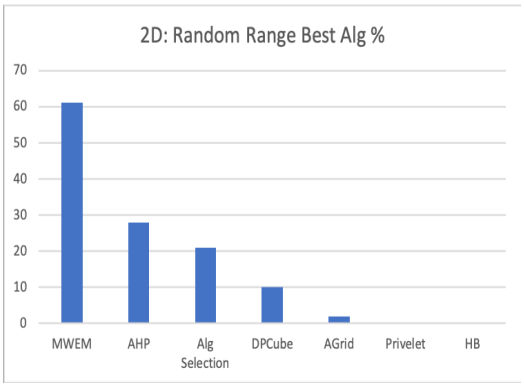
Similar to the Prefix workload, the results in Figure 3-4, show that DAWA performs very well a majority of the time, even more than with Prefix workload. MWEM also performs well, but not nearly as much as for the Prefix workload. This may be because DAWA is taking better advantage of the workload queries. The algorithm selection system does not work as well here as in the Prefix case but still performs better on average than choosing an algorithm at random.

As shown in Table 3.2 The system often chose AHP when in fact DAWA was usually the best in those cases. This might be due to the training data for the models. The AHP model may not have seen enough examples to see how AHP really performs on the parameters that were used for testing, and mistakenly estimated that AHP would do much better than it actually did, or similarly for DAWA, it may just have been estimating that it would perform worse than it did. Either way, more tuning of the model for appropriate data could fix this issue.

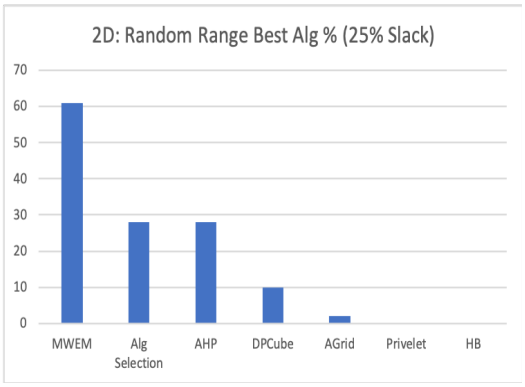
Table 3.2: Algorithm Selection Confusion Matrix: 1D Prefix Workload

	MWEM	DAWA	AHP
MWEM	0%	100%	0
DAWA	1%	85%	13%
AHP	2%	78%	20

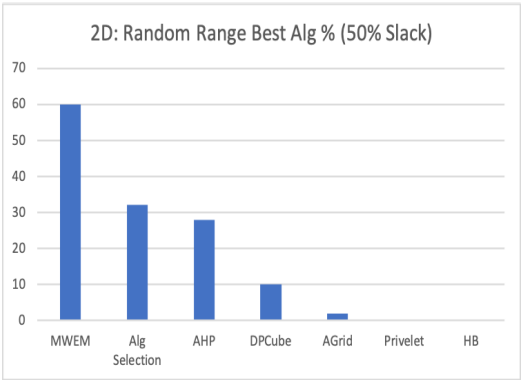
2D Workloads



(a) No Slack



(b) 25% Slack



(c) 50% Slack

Figure 3-5: Algorithm Selection Performance: 2D Random Range Workload

The 2D workloads as shown in Figure 3-5 are also dominated by MWEM, which may be due to similar reasons to the 1D workloads. Interestingly the data-independent methods were never the best. This could be largely due to the nature of the datasets, but may also be partly due to the query types. Only one query size was used for 2D: other query sizes and types may have caused different algorithms to do better. Also the target error range may also have affected these results: it could be that at some error range the data-dependent algorithms lose their effectiveness over the

data-dependent algorithms.

Table 3.3: Algorithm Selection Top 2 Performance

Workload	**Top 2 Best Alg %**
1D Identity	99%
1D Prefix	76%
2D Random Range	54%

Table 3.3 shows how often the system picked an algorithm that was within the top 2 best algorithms. It shows that our system picks a top 2 algorithm a majority of the time across workloads. These results demonstrate the viability of this method as a competitive method for choosing algorithms based on a target error. Though it doesn't always predict the best algorithm, it's choice is often very close to the top algorithm and it usually picks one of the top 2 best algorithms. Further work could investigate how this method does as more differentially private algorithms are added to its set.

3.3 Observed Error vs Target Error

Another metric for evaluating the algorithm models is the relative difference in error. Each model takes in a target error and predicts the privacy budget necessary to achieve that error. The observed error is the error that results from using the predicted epsilon and running the algorithm. The relative error difference is then:

$$RELATIVE\_DIFFERENCE\_ERROR := \frac{target\_error - observed\_error}{observed\_error}$$

When running the system test, we recorded the target error and observed error for each predicted test sample, and then calculated their relative difference.

Table 3.4 shows the average relative difference (ARD) and the 25^{th} and 75^{th} percentiles for the relative differences for the 1D Prefix workload. The ARD for all the algorithms is within an order of magnitude, which we think shows that it is doing a

Table 3.4: 1D Prefix Relative Difference in Errors

Algorithm	Avg % Difference	25^{th} Percentile	75^{th} Percentile
HB	28%	-23%	50%
AHP	2.5%	-23%	68%
DAWA	16%	-10%	53%
Privelet	60%	52%	68%
Identity	49%	30%	72%
MWEM	10%	-23%	50%

reasonable job. It is interesting to note the variation among algorithms, with some predicting more epsilons that lead to higher error than desired (negative relative difference) and some leading to lower error than desired. There doesn't seem to be a trend split along data-dependent vs data-independent algorithms, which seems to indicate it is simply a result of the training data for each of the models. The training data spans different epsilons and target errors, but different algorithms had more data centered around different epsilons and errors due to their difference in performance characteristics. Thus, when we test on a particular range of target errors, these errors may be on the higher or lower side of errors the model had been trained on. This could also be related to not testing on data with a wide variety of scales. A potential solution to address this relates to getting the whole system to be more accurate, which is training on a wider set of data, and also on finer granularity data within that span (wider range of errors/epsilons with more errors/epsilons within that range).

Table 3.5: 1D Prefix Relative Difference in Errors (Simple Model)

Algorithm	Avg % Difference	25^{th} Percentile	75^{th} Percentile
HB	-1426%	-1548%	-1060%
AHP	-1607%	-2358%	-789%
DAWA	-462%	-635%	-252%
Privelet	-1065%	-1277%	-754%
Identity	-1342%	-1899%	-615%
MWEM	-396%	-595%	-132%

Table 3.5 shows relative differences using only the simple model to predict epsilons. We can see that these predictions are very inaccurate, but interestingly they are all negative which means that the observed error was much lower than what we predicted

for. It is not clear why the results are so skewed in this direction, but it likely has to do again with the training data. Because only using the error is a very coarse metric to predict epsilon, it could be that the model just wasn't able to get a very tight relationship and the target errors used for testing exposed this weakness.

Table 3.6: 2D Random Range Relative Difference in errors

Algorithm	Avg % Difference	25^{th} Percentile	75^{th} Percentile
HB	-111%	-153%	28.6%
Privelet	-161%	-230%	-9%
AGrid	60%	56%	86%
AHP	29%	41%	88%
MWEM	66%	78%	94%
DPCube	-125%	82%	97%

Table 3.6 shows the ARD, and the 25^{th} and 75^{th} percentiles for the relative differences for the 2D workload. We see here that our epsilon predictions lead to errors that are further than the target than in the 1D case. This could be largely due to the variance being introduced by adding a 2^{nd} dimension, as well as the variability of the Random Range workload. As before, the key to increasing accuracy is likely increasing training data (as well as tuning it to target error range).

These results show that the individual algorithm models give epsilons which get us close to our target error. As with predicting the best algorithm, we believe creating more test data with finer granularity epsilons will improve our accuracy of observed error vs target error. Another thing to note is that some algorithm models predict epsilons that give us a higher observed error than desired on average, and some give a lower observed error on average. If a user wanted to lean on one side or the other (likely preferring to lean towards lower error than desired) this could be taken into account when building the model, tuning it so that it discourages predictions that result in error that is too high or too low.

3.4 Conclusion

In this thesis we developed and evaluated a system for algorithm selection given a target error in a differentially private context. The system was able to select the "best" algorithm (algorithm with lowest error given an epsilon) better than most other algorithms, and was within a close margin of the best more than any other algorithm. It chose one of the top 2 best algorithms a majority of the time. The system architecture also affords the ability to estimate how much privacy an algorithm can achieve for a target error rate with reasonable accuracy. This approach expands on the system built in [7] , and works towards creating a tool set which will hopefully help researchers in two ways. The first is by helping create a separation of concerns between algorithm development and algorithm use. The system developed helps treat algorithms as black boxes, so that user don't have to worry about how they work. Other researchers can develop algorithms and simply plug them into the system. The second way is by actually using the system to study algorithms. Tools like this system and the system developed in [7] provide ways of better understanding algorithm characteristics such as their epsilon-error relationship, and in what scenarios certain algorithms work better than others.

There were a few limitations to this. One was the scope of data used. Though the datasets were drawn from several real datasets, only a limited set of scales and domains were used. In a real-life setting, the system should be trained on a wider set of parameters. This is also related to the second limitation which is the scope of target errors the system can accurately work with. There was hand-tuning of training data to limit the the data to training samples in the range of the target errors, but ideally this process would either be automated or unnecessary. Hopefully future work will address these issues.

This system shows potential to equip researchers with fluid tools to provide better privacy guarantees when doing data analysis.

Bibliography

[1] Gergely Acs, Claude Castelluccia, and Rui Chen. Differentially private histogram publishing through lossy compression. *2012 IEEE 12th International Conference on Data Mining*, 2012.

[2] Cynthia Dwork, Frank Mcsherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. *Theory of Cryptography Lecture Notes in Computer Science*, pages 265–284, 2006.

[3] Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*. Now Publ., 2014.

[4] Moritz Hardt, Katrina Ligett, and Frank McSherry. A simple and practical algorithm for differentially private data release. *Computing Research Repository - CORR*, 12 2010.

[5] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, and Dan Zhang. Principled evaluation of differentially private algorithms using dpbench. *Proceedings of the 2016 International Conference on Management of Data - SIGMOD 16*, 2016.

[6] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proceedings of the VLDB Endowment*, 3(1-2):1021–1032, 2010.

[7] Ios Kotsogiannis, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. Pythia. *Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD 17*, 2017.

[8] Chao Li, Michael Hay, Gerome Miklau, and Yue Wang. A data- and workload-aware algorithm for range queries under differential privacy. *Proceedings of the VLDB Endowment*, 7(5):341–352, 2014.

[9] Frank D. Mcsherry. Privacy integrated queries. *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD 09*, 2009.

[10] W. Qardaji, Weining Yang, and Ninghui Li. Differentially private grids for geospatial data. *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 2013.

[11] Wahbeh Qardaji, Weining Yang, and Ninghui Li. Understanding hierarchical methods for differentially private histograms. *Proceedings of the VLDB Endowment*, 6(14):1954–1965, 2013.

[12] Xiaokui Xiao, Guozhang Wang, and Johannes Gehrke. Differential privacy via wavelet transforms. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1200–1214, 2011.

[13] Yonghui Xiao, James Gardner, and Li Xiong. Dpcube: Releasing differentially private data cubes for health information. *2012 IEEE 28th International Conference on Data Engineering*, 2012.

[14] Jia Xu, Zhenjie Zhang, Xiaokui Xiao, Yin Yang, Ge Yu, and Marianne Winslett. Differentially private histogram publication. *The VLDB Journal*, 22(6):797–822, 2013.

[15] Xiaojian Zhang, Rui Chen, Jianliang Xu, Xiaofeng Meng, and Yingtao Xie. Towards accurate histogram publication under differential privacy. *Proceedings of the 2014 SIAM International Conference on Data Mining*, 2014.