Applying Differential Privacy with Sparse Vector Technique

by

Yan Chen

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ashwin Machanavajjhala, Supervisor

_____
Ronald Parr

_____
Jun Yang

_____
Jerome P Reiter

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2018

Abstract

Applying Differential Privacy with Sparse Vector Technique

by

Yan Chen

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ashwin Machanavajjhala, Supervisor

_____
Ronald Parr

_____
Jun Yang

_____
Jerome P Reiter

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2018

Abstract

In today's fast-paced developing digital world, a wide range of services such as web services, social networks, and mobile devices collect a large amount of personal data from their users. Although sharing and mining large-scale personal data can help improve the functionality of these services, it also raises privacy concerns for the individuals who contribute to the data.

Differential privacy has emerged as a de facto standard for analyzing sensitive data with strong provable privacy guarantees for individuals. There is a rich literature that has led to the development of differentially private algorithms for numerous data analysis tasks. The privacy proof of these algorithms are mainly based on (a) the privacy guarantees of a small number of primitives, and (b) a set of composition theorems that help reason about the privacy guarantee of algorithms based on the used private primitives. In this dissertation, we focus on the usage of one popular differentially private primitive, Sparse Vector Technique, which can support multiple queries with limited privacy cost.

First, we revisit the original Sparse Vector Technique and its variants, proving that many of its variants violate the definition of differential privacy. Furthermore, we design an attack algorithm demonstrating that an adversary can reconstruct the true database with high accuracy having access to these "broken" variants.

Next, we utilize the original Sparse Vector Technique primitive to design new solutions for practical problems. We propose the first algorithms to publish regression

diagnostics under differential privacy for evaluating regression models. Specifically, we create differentially private versions of residual plots for linear regression as well as receiver operating characteristic (ROC) curves and binned residual plot for logistic regression. Comprehensive empirical studies show these algorithms are effective and enable users to evaluate the correctness of their model assumptions.

We then make use of Sparse Vector Technique as a key primitive to design a novel algorithm for differentially private stream processing, supporting queries on streaming data. This novel algorithm is data adaptive and can simultaneously support multiple queries, such as unit counts, sliding windows and event monitoring, over a single or multiple stream resolutions. Through extensive evaluations, we show that this new technique outperforms the state-of-the-art algorithms, which are specialized to particular query types.

Contents

List of Tables

List of Figures

Acknowledgements

I am most grateful to my advisor Ashwin Machanavajjhala. Completing the thesis would not happen without the mentorship from him. Ashwin has been a generous and inspiring advisor. He always had many positive suggestions to my research and gave me a lot of instructions helping me get over kinds of obstacles when I was doing my research. Since I am an international student coming from a non-English speaking country, I faced lots of trouble having a good English writing. Ashwin was very patient and helped me do a lot of practice. Ashwin's ability, friendliness and generosity in sharing his wisdom and experience make me enjoy working with him.

I have a great pleasure of collaborating with many brilliant scholars over the years. I would like to thank all my co-authors for their support and everything I have learned from them. Special thanks to Michael Hay, Gerome Miklau, Jerry Reiter, Andres F. Barrientos, Jean-Francois Paiment, Dan Zhang and Sidney Feygin. Thanks also to my entire thesis committee: Ashwin Machanavajjhala, Jun Yang, Ron Parr and Jerry Reiter.

I will also show many thanks to all the previous and current members, Xi He, Nisarg Raval, Ios Kotsogiannis, Sam Haney, Ben Stoddard and Maryam Fanaeepour, from our privacy team. It was very interesting and helpful discussing kinds of projects with these smart people. All the discussions inspire my own research. I will thank all the other members from the Duke database group. The weekly database group meetings enrich my research insight and I learned a lot of different domain knowledge

from it.

Many thanks to all my friends for making my time so much meaningful and enjoyful in the graduate school. Special thanks to my friend Qi Guan, my tennis partner. I played much better tennis now.

Finally, I will show my highest appreciation to my parents for their endless love and support. My parents teach me to be an independent person, and most importantly, they teach me how to be a good man. Thank you for always believing me, always supporting me, and for all the sacrifices you have made because of me. I love you so much.

This work was supported in part by the National Science Foundation under grants 1253327, 1408982, 1443014; and by DARPA and SPAWAR under contract N66001-15-C-4067.

1

Introduction

The increasing digitization of personal information in the form of medical records, administrative and financial records, social networks, location trajectories, and so on, creates new avenues for data analytics research. Sharing and mining of this large scale personal data help improve the quality of people's life. However, sharing and analyzing the data, or even deriving aggregates over the data raise concerns over the confidentiality of individual participants in the dataset. As a result, privacy-preserving data analysis has been recently given significant attention.

A wealth of literature shows that anonymization techniques that redact identifiers, coarsen attributes, or even systems that allow users to query the database indiscriminately, do not prevent determined adversaries from being able to learn sensitive properties of individuals [19]. In an earlier work, some experiments have been conducted on the 1990 U.S. Census summary data to determine how many individuals within geographically situated populations had infrequent combinations of demographic values [56]. One important finding is that even after removing the unique identifiers like social security number, still approximately 87% of the population in the United States can be uniquely identified just based on the values of

Table 1.1: Example dataset of people with different ages

Age	...	20	21	22	23	24	25	26	...
# of people	...	10	15	70	20	15	45	12	...

gender, full date of birth, and the 5-digit zip code.

Differential privacy [21] was first proposed over a decade ago and has now become a de facto standard for privacy protection due to its provable guarantee and nice composition properties. Informally, a (randomized) algorithm ensures differential privacy if its output distributions are approximately the same when executed on two input databases that only differ in a single individual's record. This requirement prevents any attacker with access to the output of differentially private algorithms from learning anything substantial about the presence or absence of any single individual. The privacy level under differential privacy is represented by a parameter ϵ, which is usually called the *privacy budget*, with smaller values corresponding to stronger privacy guarantees. In order to achieve differential privacy, the basic idea is to introduce noise to the output. *Laplace Mechanism* [21], which will be formally defined in Section 2.1.3, is a standard method for achieving differential privacy. Basically, *Laplace Mechanism* injects Laplace noise to perturb the output for ensuring differential privacy.

Example 1. *Given a dataset about the age information of people, we derive a histogram (shown in Table 1.1), reporting the number of people associated with different ages. Suppose we want to ask one query "How many people whose ages are within 20 and 25" and we apply* Laplace Mechanism *with total privacy budget ϵ. The returned private answer should be equal to something like $175 + \xi$, where $\xi \sim Lap(\frac{1}{\epsilon})$ is a random variable drawn from the Laplace distribution with mean 0 and the scale parameter $\beta = \frac{1}{\epsilon}$.*

There is a rich literature [19] that has led to the development of differentially pri-

vate algorithms for numerous data analysis tasks. There are typically two approaches to designing algorithms that satisfy differential privacy. In the first approach, researchers directly design the algorithm for a target problem and then try to prove that the proposed algorithm satisfies the differential privacy guarantee. However, this strategy is not popular since the privacy analysis of a complex algorithm is usually hard. Thus, an alternative popular strategy to design differentially private algorithms is to make use of differentially private primitives. These primitives are a small set of simpler algorithms like the *Laplace Mechanism* [21] that have been well proved to ensure differential privacy. Complex algorithms are constructed ensuring that the private data are only accessed via these primitives. The privacy guaranteed by such algorithms are proven using composition theorems. For example, the sequential composition is one such composition theorem, which states that if an algorithm accesses a dataset multiple times via differentially private primitives $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ each satisfying differential privacy with parameters $\epsilon_1, \epsilon_2, \ldots, \epsilon_k$, then the overall algorithm also ensures differential privacy with a total privacy loss of $\epsilon = \epsilon_1 + \epsilon_2 + \cdots + \epsilon_k$. The composition properties of differential privacy will be formally defined later in Section 2.1.2. By taking advantages of composition properties, we are able to design more complex differentially private algorithms by just composing simpler differentially private primitives.

However, in practice, we usually want to support multiple queries. For instance, if we want to compute the CDF of the people by the age in terms of the dataset in Table 1.1. We may need to answer a set of queries like q_i for the total number of people whose ages are not above i. If the i is from 1 to 100, there is 100 queries in total. We can also apply *Laplace Mechanism* to each of the query and the output to each query q_i should be like $\tilde{q}_i(D) = q_i(D) + \xi_i$, where $\xi_i \sim Lap(\frac{1}{\epsilon})$. Based on the sequential composition of differential privacy, this strategy ensures $100 \times \epsilon$- differential privacy for answering all 100 queries. Thus, the total privacy cost increases linearly

in terms of the number of answered queries, which is not good in practice when there is a large number of queries to be answered with a limited privacy budget. We may either quickly use up of all the privacy budget or we can only spend a small portion of the privacy budget to each query (introducing too much noise).

Prior work has shown that when we merely care about the identity of the queries that lie above (or below) a given threshold instead of requiring the numeric answers to the queries, it is possible to save the total privacy cost. As a result, by getting rid of the numeric value answers, we can reduce the total privacy cost to be only based on the number of queries that are above (or below) the given threshold rather than the total number of queries [22]. Therefore, if there is a large number of queries to be tested but only very few of them are actually above a given threshold, there is a possible huge saving of privacy cost. The basic idea of algorithm, called the *Sparse Vector Technique*, is to introduce noise to both threshold value and the query answers. Then it outputs the identity of each tested query by comparing the noisy query answer to the noisy threshold value. The algorithm will stop when it finds the pre-defined number of queries that are above the given threshold.

Example 2. *We still use the same dataset from Table 1.1. Suppose we have a sequence of queries* q_i = *"How many people whose ages are within 20 and 20 + i", where* $i = 1, 2, \ldots$. *Assume that we are applying* Sparse Vector Technique *with the threshold* $\theta = 110$ *and the privacy budget* $\epsilon = \infty$ *(the injected noise goes to zero), and we only need to find one query that is above the given threshold. The* Sparse Vector Technique *works as follows: (1) It tests the first query* $q_1 = 25 < \theta$ *and outputs a label showing* q_1 *is below the threshold. (2) It tests the second query* $q_2 = 105 < \theta$ *and outputs a label showing* q_2 *is below the threshold. (3) It tests the third query* $q_3 = 125 > \theta$, *outputs a label showing* q_3 *is above the threshold and then stops the algorithm.*

C. Dwork et al. [22] even proved that the noise scale under *Sparse Vector Technique* for testing k queries is $O(\log(k))$, while the noise scale will become to be $O(k)$ if applying *Laplace Mechanism* with sequential composition. *Sparse Vector Technique* is one differentially private primitive that helps reduce privacy cost as well as the output error, and as such we would like to use it for designing novel algorithms to solve practical problems under differential privacy.

Sparse Vector Technique has become a popular differentially private primitive for building differentially private algorithms for a number of tasks [28, 51]. Recent work [14, 37, 55] has further explored the possibility of extending *Sparse Vector Technique* to eliminate the dependence of privacy cost even on the number of queries above (or below) the threshold. The proposed *Sparse Vector Technique* variants try to add less noise to each test query and allow more queries to be tested under the same privacy level.

In this dissertation, we critically analyze the incorrect extensions of *Sparse Vector Technique* in terms of its privacy guarantee. We found out that many of these variants indeed violate differential privacy. Adversaries can even reconstruct the input data based on the output of mechanisms designed under those variants. We further make use of the correct version of *Sparse Vector Technique* as a key differentially private primitive to propose novel algorithms for solving multiple practical problems with differential privacy guarantee. We summarize the contributions of this dissertation as follows.

1. **Understanding Privacy Properties of Variants on Sparse Vector Technique:** In this study, we critically analyze the privacy properties of multiple variants of *Sparse Vector Technique* from the previous works, showing that these variants do actually not satisfy differential privacy. We show specific examples of neighboring datasets, queries and outputs that violate the require-

ment of differential privacy. We display an attack and demonstrate that these variants make it possible for adversaries to reconstruct the real frequency of the input data with high probability. We also propose a correct generalized version of *Sparse Vector Technique*.

2. **Differentially Private Regression Diagnostics:** In this research, we focus on the problem of privacy-preserving diagnosing regression models. We develop several new algorithms for doing differentially private regression diagnostics. In particular, we take advantage of the idea from the *Sparse Vector Technique*, designing the first algorithm *PriRP* for computing differentially private residual plots , which is a popular diagnostic tool for evaluating the model fit for linear regression. We also propose the new algorithms *PriBRP* for computing binned residual plots and *PriROC* for computing ROC curves under differential privacy, which are diagnostic tools for measuring the model fit as well as the predictive power for logistic regression. Comprehensive experiments show that the proposed private diagnostic algorithms provide trustful diagnostic information.

3. **Differentially Private Stream Processing:** In this work, we design PeGaSus, a novel algorithm for answering a large class of continuous queries over real-time data streams under differential privacy. The PeGaSus uses a combination of a *Perturber*, a data-adaptive *Grouper* (by using the idea of *Sparse Vector Technique*) and a query specific *Smoother* to simultaneously support a range of query workloads over multiple resolutions over the stream. A thorough empirical evaluation, on real-world data streams, shows that by using different query specific *Smoother* methods, PeGaSus outperforms the state-of-the-art algorithms specialized to given workloads.

The rest of this dissertation is organized as follows. We provide a detailed back-

ground on *Differential Privacy* and *Sparse Vector Technique* in Chapter 2. In Chapter 3, we analyze the variants of *Sparse Vector Technique*, and propose attacks for deriving sensitive information from the incorrect variants. We then present new solutions to solve the problem of doing regression diagnostics under differential privacy in Chapter 4. In Chapter 5, we discuss differentially private stream processing. Existing related works are presented in Chapter 6, and finally the conclusion of this dissertation is discussed in Chapter 7.

2

Differential Privacy and Sparse Vector Technique

2.1 Differential Privacy

In this section, we explore the notion of differential privacy from multiple perspective.

2.1.1 ϵ-differential privacy

Differential privacy, first introduced by [17, 21], aims to protect the private information of any single individual by limiting the privacy risk raised by the data of the individual is used by certain analysis, compared with the result when the data is not used. In other words, differential privacy guarantees that the information which adversaries get from the outputs of any analysis with or without the presence of any single individual is approximately the same, so that it will help hide the presence or absence of any single individual from the adversaries who access to the outputs.

The formal definition of differential privacy depends on the notion of neighboring datasets. We call D and D' are neighboring datasets, if D' differs from D by only adding or removing one single record ($|D \oplus D'| = 1$). The record can be defined differently based on different applications and different privacy object. For example, one record can contain all the location information associate with one single individ-

ual, or one record can represent a single connection of one user to one WiFI sensor within a 5 minutes time interval. Then we can formally define differential privacy as follows:

Definition 1 (ϵ-differential privacy)**.** *Let $\mathcal{A}$ be a randomized algorithm that takes as input the form of dataset D and outputs an element from a set of possible outputs $\mathcal{O}$. Then $\mathcal{A}$ satisfies ϵ-differential privacy if for any pair of neighboring datasets D and D', and $\forall O \subset \mathcal{O}$,*

$$Pr[\mathcal{A}(D) \in O] \leqslant e^{\epsilon} \times Pr[\mathcal{A}(D') \in O]. \tag{2.1}$$

The value of ϵ, called *privacy budget*, controls the privacy risk and limits how much an adversary can distinguish one dataset from its neighboring datasets given the output. Smaller ϵ's correspond to stronger privacy protection. Intuitively, differentially private algorithms guarantee that the output distributions from any two neighboring datasets are close enough such that the adversaries cannot tell which one of the neighboring datasets is the input by just assessing to the output.

2.1.2 Composition properties of differential privacy

The following composition properties hold for differentially private algorithms [19]. Suppose $\mathcal{A}_1(\cdot)$ and $\mathcal{A}_2(\cdot)$ be two algorithms ensuring ϵ_1- and ϵ_2-differential privacy, respectively.

- **Sequential Composition:** Computing both $\mathcal{A}_1(D)$ and $\mathcal{A}_2(D)$ on the same dataset D satisfies $\epsilon_1 + \epsilon_2$-differential privacy.
- **Parallel Composition:** Let A and B be disjoint subsets of the domain, where $A \cap B = \varnothing$. Computing $\mathcal{A}_1(D \cap A)$ and $\mathcal{A}_1(D \cap B)$ ensures ϵ_1-differential privacy.

- **Postprocessing:** For any algorithm $\mathcal{A}_3$, releasing $\mathcal{A}_3(\mathcal{A}_1(D))$ still satisfies ϵ_1-differential privacy for any D. That is, post-processing an output of a differentially private algorithm does not incur any additional loss of privacy.

The composition properties of differential privacy allow people to execute multiple differentially private computations and reason about the cumulative privacy risk. Thus, complex differentially private algorithms can be built by composing simpler private building blocks. In each application, we can bound the total privacy risk so we impose a total ϵ "privacy budget" and allocate a portion of the budget to each private computation.

2.1.3 Laplace Mechanism

An arbitrary numerical function f can be made differentially private by injecting noise to its output. The amount of noise depends on the sensitivity of the function.

Definition 2 (Sensitivity)**.** *Let f be a function that maps datasets to $\mathbb{R}^n$. The* sensitivity *denoted as $\Delta(f)$, is defined to be the maximum L_1 distance between the function outputs from any pair of neighboring datasets D_1 and D_2,*

$$\Delta(f) = \max_{D_1, D_2 : |D_1 \oplus D_2| = 1} ||f(D_1) - f(D_2)||_1. \tag{2.2}$$

The most widely used differentially private building block, which is used for publishing data or answering queries in a differentially private manner, is called *Laplace Mechanism* [21]. *Laplace Mechanism* achieves differential privacy by injecting noise from Laplace distribution calibrated to the sensitivity.

Definition 3 (Laplace Mechanism)**.** *Given a function f that maps datasets to $\mathbb{R}^n$, the* Laplace Mechanism *outputs $f(D) + \eta$, where η is a vector of independent random variables drawn from a Laplace distribution with the probability density function $p(x \mid \lambda) = \frac{1}{2\lambda} e^{-|x|/\lambda}$, where $\lambda = \Delta(f)/\epsilon$.*

Theorem 1. Laplace Mechanism *satisfies ϵ-differential privacy.*

2.2 Sparse Vector Technique

In this section, we explore the preliminaries about *Sparse Vector Technique*.

2.2.1 Shortage before Sparse Vector Technique

The composition properties of differential privacy allow people to analyze the total privacy risk raised by multiple private computations. For example, if we have a algorithm $\mathcal{A}$, say *Laplace Mechanism*, that satisfies ϵ-differential privacy. We apply $\mathcal{A}$ for answering multiple queries $Q = q_1, q_2, q_3, \ldots, q_k$. If we spend each ϵ privacy budget on every computation of q_i, the total privacy budget used will be accumulated as $k \times \epsilon$ under the sequential composition property of differential privacy. Note that, the more privacy budget consumed, the higher privacy risk encountered. Thus, the privacy risk will be proportional to the number of queries (or computations) we make. If there is too many queries, we are either hard to yield reasonable privacy guarantees when spending fixed privacy budget on each query, or generate poor outputs with fixed total privacy budget (by spending too little budget on each query).

2.2.2 Original Sparse Vector Technique

There are some situations that people only care about the identity of the queries that lie above (or below) a certain threshold, instead of the actual numerical values of the queries. For instance, when people are doing feature selections in order to reduce the domain of a regression model, they may only care about the features whose scores are above some threshold instead of their real score values. Another example is that when doing frequent item mining task, we only want to indicate the items with frequency above a given threshold and do not care about their real frequency counts.

Fortunately, C. Dwork et al. found that, by discarding the numeric answers to the queries and merely reporting the identity of queries that lie above or below a certain threshold, we may save a lot of privacy budget reducing the privacy risk. In fact, the total privacy budget (or the total privacy risk) will only increase with the number of queries which actually lie above (or below) the threshold, rather than with the total number of queries. If we know that the set of queries that lie above (or below) the given threshold is much smaller than the total number of queries (which indicates a sparse answer vector), there will be a huge saving in terms of the privacy budget cost.

Algorithm 1 Sparse Vector Technique

Input: Dataset D, a stream of queries $q_1, q_2, \dots$ with bounded sensitivity Δ, threshold θ, a cutoff point c and privacy budget ϵ
Output: a stream of answers

$\tilde{\theta} \leftarrow \theta + Lap(2\Delta/\epsilon)$, $count \leftarrow 0$
for each query i **do**
    $v_i \leftarrow Lap(4c\Delta/\epsilon)$
    if $q_i(D) + v_i \geqslant \tilde{\theta}$ **then**
        Output $\top$, $count \leftarrow count + 1$
    else
        Output $\perp$
    end if
    if $count == c$ **then**
        Abort
    end if
end for

The first version of the *Sparse Vector Technique* was proposed by [22]. Algorithm 1 presents the details of it. The inputs of *Sparse Vector Technique* include a stream of queries $Q = \{q_1, q_2, \dots\}$, where each $q \in Q$ has sensitivity bounded by Δ, a threshold θ separating the identity of queries, a cutoff point c limiting the number of queries with identity above the threshold to output at most. For every targeting query, *Sparse Vector Technique* outputs either $\top$, showing the query is above the threshold, or $\perp$, indicating the query is below the threshold. It works as the following two steps:

1. Perturb the threshold θ by injecting noise drawn from the Laplace distribution with scale $\frac{2\Delta}{\epsilon}$, getting $\tilde{\theta}$.

2. Perturb each query q_i by adding Laplace noise with scale $\frac{4c\Delta}{\epsilon}$, getting $\tilde{q}_i$. Then it outputs $\top$ when $\tilde{q}_i \geqslant \tilde{\theta}$, and $\perp$ otherwise.

Sparse Vector Technique will stop when it detects c $\top$ queries.

Theorem 2. Sparse Vector Technique *satisfies ϵ-differential privacy.*

Proof. Suppose D and D' are any two neighboring input datasets. Let any output $O = O_\perp \cap O_\top$, where $O_\perp$ and $O_\top$ contains the set of queries labeled as $\perp$ and $\top$, respectively. The cutoff point c indicates that $|O_\top| = c$. Then we have

$$Pr[\mathbf{A}(D) = O] = \int_{-\infty}^{\infty} Pr[\tilde{\theta} = t](\prod_{q \in O_\perp} Pr[\tilde{q}(D) < t])(\prod_{q \in O_\top} Pr[\tilde{q}(D) \geqslant t])dt \quad (2.3)$$

Since the sensitivity of all the queries are bounded by Δ, which means that $|q(D) - q(D')| \leqslant \Delta$ for $\forall q \in O$. Thus, we have

$$Pr[\tilde{q}(D) < t] \leqslant Pr[\tilde{q}(D') < t + \Delta] \quad (2.4)$$

Let $\tilde{q}(D) = q(D) + v$, where v is the independent Laplace noise injected in the line

4 of Algorithm 1. We can derive that

$$
\begin{aligned}
& Pr[\mathcal{A}(D) = O] \\
= & \int_{-\infty}^{\infty} Pr[\tilde{\theta} = t](\prod_{q \in O_\perp} Pr[\tilde{q}(D) < t])(\prod_{q \in O_\top} Pr[\tilde{q}(D) \geqslant t])dt \\
\leqslant & \int_{-\infty}^{\infty} e^{\epsilon/2} Pr[\tilde{\theta} = t + \Delta](\prod_{q \in O_\perp} Pr[\tilde{q}(D) < t])(\prod_{q \in O_\top} Pr[\tilde{q}(D) \geqslant t])dt \\
\leqslant & \int_{-\infty}^{\infty} e^{\epsilon/2} Pr[\tilde{\theta} = t + \Delta](\prod_{q \in O_\perp} Pr[\tilde{q}(D') < t + \Delta])(\prod_{q \in O_\top} Pr[v \geqslant t - q(D)])dt \\
\leqslant & \int_{-\infty}^{\infty} e^{\epsilon/2} Pr[\tilde{\theta} = t + \Delta](\prod_{q \in O_\perp} Pr[\tilde{q}(D') < t + \Delta]) \\
& \times (\prod_{q \in O_\top} e^{\epsilon/2c} Pr[v \geqslant (t + \Delta) - q(D')])dt \\
= & e^{\epsilon} \int_{-\infty}^{\infty} Pr[\tilde{\theta} = t + \Delta](\prod_{q \in O_\perp} Pr[\tilde{q}(D') < t + \Delta])(\prod_{q \in O_\top} Pr[\tilde{q}(D') \geqslant t + \Delta])dt \\
= & e^{\epsilon} Pr[\mathcal{A}(D') = O] \qquad (2.5)
\end{aligned}
$$

□

2.2.3 Utility of Sparse Vector Technique

We analyze the accuracy in terms of the outputs from *Sparse Vector Technique* proposed in Algorithm 1.

Definition 4 ((α, β)-accuracy)**.** *Suppose an algorithm outputs a stream of* $o_1, o_2, \cdots \in \{\perp, \top\}^*$ *in the response to a stream queries* $q_1, q_2, \ldots$. *We say the output is* (α, β)*-accurate with respective to a threshold* θ*, if with probability at most* β*, we have for all* $o_i = \perp$*:*

$$q_i(D) \geqslant \theta - \alpha \qquad (2.6)$$

and for all $o_i = \top$*:*

$$q_i(D) \leqslant \theta + \alpha \tag{2.7}$$

Theorem 3. *The outputs from* Sparse Vector Technique *is* (α, β)*-accurate for*

$$\alpha = \frac{8c\Delta(\log k + \log(2/\beta)}{\epsilon}, \tag{2.8}$$

where k *is the total number of queries tested.*

Proof. In terms of the noise added to the threshold θ, we know that $\tilde{\theta} - \theta \sim Lap(\frac{2\Delta}{\epsilon})$. Then we have that

$$\begin{aligned} & Pr[|\tilde{\theta} - \theta| \geqslant t \times \frac{2\Delta}{\epsilon}] = e^{-t} \\ \Rightarrow \quad & Pr[|\tilde{\theta} - \theta| \geqslant \frac{\alpha}{2}] = e^{-\frac{\epsilon\alpha}{4\Delta}} \end{aligned} \tag{2.9}$$

When setting the quantity to be at most $\beta/2$, we require

$$\begin{aligned} & e^{-\frac{\epsilon\alpha}{4\Delta}} \leqslant \frac{\beta}{2} \\ \Rightarrow \quad & \alpha \geqslant \frac{4\Delta \log(2/\beta)}{\epsilon} \end{aligned} \tag{2.10}$$

Similarly, for each query q, we know that $\tilde{q} - q \sim Lap(\frac{4c\Delta}{\epsilon})$. Thus we will have

$$Pr[|\tilde{q} - q| \geqslant \frac{\alpha}{2}] = e^{-\frac{\epsilon\alpha}{8c\Delta}} \tag{2.11}$$

By using a union bound, we have

$$Pr[\max_{i \in [k]} |\tilde{q}_i - q_i| \geqslant \frac{\alpha}{2}] \leqslant k \times e^{-\frac{\epsilon\alpha}{8c\Delta}} \tag{2.12}$$

We also set the quantity to be at most $\beta/2$ and we will have

$$\begin{aligned} & k \times e^{-\frac{\epsilon\alpha}{8c\Delta}} \leqslant \frac{\beta}{2} \\ \Rightarrow \quad & \alpha \geqslant \frac{8c\Delta(\log k + \log(2/\beta)}{\epsilon} \end{aligned} \tag{2.13}$$

By combining the above two analysis, we will have

$$Pr[|\tilde{\theta} - \theta| + \max_{i \in [k]} |\tilde{q}_i - q_i| \geqslant \alpha] \leqslant \beta, \tag{2.14}$$

when $\alpha \geqslant \frac{8c\Delta(\log k + \log(2/\beta)}{\epsilon}$, which proves Theorem 3. □

3

Privacy Properties of Variants on Sparse Vector Technique

In this Chapter, I will introduce multiple variants of *Sparse Vector Technique* proposed in the previous literatures and critically analyze the privacy properties of these variants, showing that these variants actually do not ensure differential privacy for any finite ϵ. We identify a subtle error in their privacy analysis and further show that an adversary can use the variants to recover counts from input datasets with high probability. We will also introduce a new correct generalized version of *Sparse Vector Technique*.

3.1 Variants of Sparse Vector Technique

As shown in Algorithm 1, the privacy budget spent in the original *Sparse Vector Technique* depends mainly on the number of tested queries that are above the given threshold (that can be called "positive" queries). Recent works have explored the possibility of extending the *Sparse Vector Technique* to eliminate this dependence even on the number of "positive" queries.

Algorithm 2 The Variant of SVT in [37]

Input: Dataset D, query set Q with bounded sensitivity Δ, threshold θ and privacy budget ϵ

Output: a stream of answers $\{\perp, \top\}^{|Q|}$

$\epsilon_1 \leftarrow \epsilon/4$, $\epsilon_2 \leftarrow 3\epsilon/4$
$\tilde{\theta} \leftarrow \theta + Lap(\Delta/\epsilon_1)$
for each query $q_i \in Q$ **do**
    $v_i \leftarrow Lap(\Delta/\epsilon_2)$
    if $q_i(D) + v_i \geqslant \tilde{\theta}$ **then**
        Output $\top$
    else
        Output $\perp$
    end if
end for

J. Lee et al. proposed one variant of *Sparse Vector Technique* (shown in Algorithm 2) for achieving differentially private frequent itemsets mining [37]. In their work, they used the *Sparse Vector Technique* variant for checking whether the targeted itemset has the frequency more than a given threshold θ. In Algorithm 2, the noisy threshold is computed by using $\frac{\epsilon}{4}$ privacy budget, and each single query is perturbed by using an independent $\frac{3\epsilon}{4}$ privacy budget, which does not depend on the number of "positive" queries.

Algorithm 3 The Variant of SVT in [55]

Input: Dataset D, query set Q with bounded sensitivity Δ, threshold θ and privacy budget ϵ

Output: a stream of answers $\{\perp, \top\}^{|Q|}$

$\tilde{\theta} \leftarrow \theta + Lap(\Delta/\epsilon)$
for each query $q_i \in Q$ **do**
    if $q_i(D) \geqslant \tilde{\theta}$ **then**
        Output $\top$
    else
        Output $\perp$
    end if
end for

B. Stoddard et al. extended the *Sparse Vector Technique* (displayed in Algorithm 3) on doing feature selection for classification tasks [55]. In their work, the authors applied their proposed variant of *Sparse Vector Technique* for determining the choice of features by selecting ones whose scores are above certain pre-defined

threshold. In Algorithm 3, the entire privacy budget is used for perturbing the threshold and the real answers to the queries are compared with the noisy threshold.

Algorithm 4 The Variant of SVT in [14]

Input: Dataset D, query set Q with bounded sensitivity Δ, threshold θ and privacy budget ϵ

Output: a stream of answers $\{\perp, \top\}^{|Q|}$

1: $\epsilon_1 \leftarrow \epsilon/2$, $\epsilon_2 \leftarrow \epsilon/2$
2: $\tilde{\theta} \leftarrow \theta + Lap(\Delta/\epsilon_1)$
3: **for** each query $q_i \in Q$ **do**
4: $\quad v_i \leftarrow Lap(\Delta/\epsilon_2)$
5: $\quad$ **if** $q_i(D) + v_i \geqslant \tilde{\theta}$ **then**
6: $\quad\quad$ Output $\top$
7: $\quad$ **else**
8: $\quad\quad$ Output $\perp$
9: $\quad$ **end if**
10: **end for**

Another variant of *Sparse Vector Technique* (shown in Algorithm 4) was proposed by R. Chen et al. in [14] for generating synthetic high-dimensional datasets. They make use of use their *Sparse Vector Technique* variant for generating the dependency graph among all attributes. Specifically, they choose a set of new attributes whose mutual information score with the previous attribute is above certain threshold. In the Algorithm 4, half of the privacy budget is spent on computing noisy threshold. Also, independent privacy budget of the remaining half is used for perturbing each mutual information score query. The noisy query answers are then compared with the noisy threshold.

3.2 Generalized Private Threshold Testing

3.2.1 Description of Generalized Private Threshold Testing

All the variants of *Sparse Vector Technique* introduced in the previous section (Algorithm 2, 3, 4) are trying to eliminate the dependence of the privacy cost on the number of "positive" queries. The only difference among these variants is the amount of noise injected to either threshold or each individual query.

Algorithm 5 Generalized Private Threshold Testing (GPTT)

Input: Dataset D, query set Q with bounded sensitivity Δ, threshold θ and privacy budget ϵ

Output: a stream of answers $\{\perp, \top\}^{|Q|}$

$\epsilon_1, \epsilon_2 \leftarrow f(\epsilon)$
$\tilde{\theta} \leftarrow \theta + Lap(\Delta/\epsilon_1)$
for each query $q_i \in Q$ **do**
    $v_i \leftarrow Lap(\Delta/\epsilon_2)$
    if $q_i(D) + v_i \geqslant \tilde{\theta}$ **then**
        Output $\top$
    else
        Output $\perp$
    end if
end for

We summarize and describe a method called *Generalized Private Threshold Testing* (GPTT) that generalize all the variants of the *Sparse Vector Technique* that do not require a limit on the number of "positive" queries. The details of GPTT is displayed in Algorithm 5.

GPTT takes as input a dataset D, a set of queries $Q = \{q_1, q_2, \ldots, q_n\}$ with bounded sensitivity Δ, a threshold θ and a privacy budget ϵ. For every query $q_i \in Q$, GPTT outputs either $\top$ or $\perp$ that approximates whether or not the query q_i is above or below the given threshold θ. GPTT works exactly like the original *Sparse Vector Technique* - the threshold is perturbed by injecting noise from Laplace noise with parameter Δ/ϵ_1 and every query answer is also perturbed by adding noise from independent Laplace noise with parameter Δ/ϵ_2. The output is determined by comparing the noisy threshold with each noisy query answer. You can see the difference is that there is no limits on the number of "positive" queries to be tested and output by the algorithm.

GPTT is a generalization of all the variants presented in the previous section in terms of the function $f(\epsilon)$ for determining the value of both ϵ_1 and ϵ_2. [37] used GPTT for private itemset mining with $\epsilon_1 = \epsilon/4$ and $\epsilon_2 = 3\epsilon/4$. R. Chen et al. in [14] instantiate GPTT with $\epsilon_1 = \epsilon_2 = \epsilon/2$, for high-dimensional data release.

[55] indicated that the privacy guarantee does not depend on the setting of ϵ_2 and proposed an instantiation of GPTT with $\epsilon_1 = \epsilon$ and $\epsilon_2 = \infty$, for private feature selection.

3.2.2 Privacy Analysis of GPTT based on Previous Works

In this section, we derive and extend the privacy analysis from previous works [37], [14], [55] on our proposed GPTT algorithm. We will show, in Section 3.2.3, that this privacy analysis is flawed and GPTT (as well as all the instantiations presented in previous works) does not ensure differential privacy.

Given any set of queries $Q = \{q_1, q_2, \ldots, q_n\}$, we let the vector $o =< v_1, v_2, \ldots, v_n > \in \{\perp, \top\}^n$ denote the output from GPTT. Given any two neighboring datasets D_1 and D_2, we assume P_1 and P_2 be the output distribution of v given the input dataset D_1 and D_2, respectively. We use $o^{<t}$ to denote $t-1$ previous answers (i.e., $o^{<t} =< o_1, o_2, \ldots, o_{t-1} >$). Then, we have

$$\frac{P_1(o)}{P_2(o)} = \frac{\prod_{i=1}^{n} P_1(o_i \mid o^{<i})}{\prod_{i=1}^{n} P_2(o_i \mid o^{<i})} = \prod_{i:o_i=\top} \frac{P_1(o_i = \top \mid o^{<i})}{P_2(o_i = \top \mid o^{<i})} \times \prod_{i:o_i=\perp} \frac{P_1(o_i = \perp \mid o^{<i})}{P_2(o_i = \perp \mid o^{<i})} \quad (3.1)$$

Let $H_i(x)$ be the probability that q_i is classified as "positive" query (i.e. $o_i = \top$), when the noisy threshold is x. That is,

$$H_i(x) = Pr[o_i = \top \mid x, o^{<i}] = Pr[o_i = \top \mid x]. \quad (3.2)$$

Since the noisy query is derived by injecting noise from a Laplace distribution with parameter Δ/ϵ_2, we can denote that the distribution of the noisy query answer of q_i is from a distribution of $f(y; \mu, \lambda) = \frac{1}{2\lambda} exp(-\frac{|y-\mu|}{\lambda})$, where $\mu = q_i$ and $\lambda = \Delta/\epsilon_2$. Therefore, we have

$$H_i(x) = \int_x^{\infty} f(y; q_i, \Delta/\epsilon_2)dy = \int_{x+\Delta}^{\infty} f(y; q_i + \Delta, \Delta/\epsilon_2)dy. \quad (3.3)$$

Previous works use the property of $H_i(x)$ above to show that their variants ensure $2\epsilon_1$-differential privacy. Let $S = \{i \mid o_i = \top \; and \; q_i(D_1) = q_i(D_2)\}$ and $\bar{S} = \{i \mid o_i = \top \; and \; q_i(D_1) \neq q_i(D_2)\}$. Thus,

$$\prod_{i:o_i=\top} P_1(o_i = \top \mid o^{<i}) = \prod_{i \in S} P_1(o_i = \top \mid o^{<i}) \times \prod_{i \in \bar{S}} P_1(o_i = \top \mid o^{<i}). \tag{3.4}$$

Let $H_i^1(x)$ and $H_i^2(x)$ be the corresponding probability given the input datasets D_1 and D_2, respectively. Then, we have

$$\begin{aligned}
\prod_{i \in S} P_1(o_i = \top \mid o^{<i}) &= \int_{-\infty}^{\infty} Pr[\tilde{\theta} = \theta] \prod_{i \in S} H_i^1(x) dx \\
&= \int_{-\infty}^{\infty} Pr[\tilde{\theta} = \theta] \prod_{i \in S} H_i^2(x) dx \qquad (q_i(D_1) = q_i(D_2) \; for \; i \in S) \\
&= \prod_{i \in S} P_2(o_i = \top \mid o^{<i}). \qquad (3.5) \\
\prod_{i \in \bar{S}} P_1(o_i = \top \mid o^{<i}) &= \int_{-\infty}^{\infty} Pr[\tilde{\theta} = \theta] \prod_{i \in S} H_i^1(x) dx \\
&\leqslant e^{\epsilon_1} \int_{-\infty}^{\infty} Pr[\tilde{\theta} = \theta - \Delta] \prod_{i \in S} H_i^2(x - \Delta) dx \\
&= e^{\epsilon_1} \prod_{i \in \bar{S}} P_2(o_i = \top \mid o^{<i}). \qquad (3.6)
\end{aligned}$$

Thus, we can directly get

$$\prod_{i:o_i=\top} P_1(o_i = \top \mid o^{<i}) \leqslant e^{\epsilon_1} \prod_{i:o_i=\top} P_2(o_i = \top \mid o^{<i}). \tag{3.7}$$

Similarly, we have

$$\prod_{i:o_i=\perp} P_1(o_i = \top \mid o^{<i}) \leqslant e^{\epsilon_1} \prod_{i:o_i=\perp} P_2(o_i = \top \mid o^{<i}). \tag{3.8}$$

Therefore, $P_1(o)/P_2(o) \leqslant e^{2\epsilon_1}$. We will mention that injecting noise to the queries is not really required. The proof above will go through even if ϵ_2 goes to infinity.

3.2.3 The Failure Privacy Guarantee of GPTT

In this section, we give constructive proof, displaying examples of neighboring datasets and a sequence of queries for which GPTT violates differential privacy.

Theorem 4. *GPTT does not ensure ϵ-differential privacy for any finite ϵ.*

Proof. Consider two queries q_1 and q_2 with sensitivity $\Delta = 1$. For special case of GPTT, where $\epsilon_2 = \infty$, we suppose there are two neighboring datasets D_1 and D_2 such that $q_1(D_1) = q_2(D_2) = 0$ and $q_1(D_2) = q_2(D_1) = 1$. Let the threshold $\theta = 0$. Given the output that $o_1 = \perp$ and $o_2 = \top$, we have

$$Pr[o_1 = \perp, o_2 = \top \mid D_1] = \int_{x=-\infty}^{\infty} Pr[x]Pr[0 \leqslant x]Pr[1 > x]dx > 0 \tag{3.9}$$

$$Pr[o_1 = \perp, o_2 = \top \mid D_2] = \int_{x=-\infty}^{\infty} Pr[x]Pr[1 \leqslant x]Pr[0 > x]dx = 0 \tag{3.10}$$

Apparently, when $\epsilon = \infty$, GPTT does not satisfy differential privacy of any ϵ.

To prove Theorem 4 when $\epsilon_2 < \infty$, we construct a similar counterexample as above, except that the sequence of queries Q contains t copies of q_1 and t copies of q_2. Thus, we let $Q = \{q_1, q_2, \ldots, q_{2t}\}$, and assume there are two neighboring datasets D_1 and D_2 such that $q_1(D_1) = \cdots = q_t(D_1) = q_{t+1}(D_2) = \cdots = q_{2t}(D_2) = 0$ and $q_1(D_2) = \cdots = q_t(D_2) = q_{t+1}(D_1) = \cdots = q_{2t}(D_1) = 1$. We also assume the threshold $\theta = 0$ and the output is $o = \{o_1 = o_2 = \cdots = o_t = \perp, o_{t+1} = o_{t+2} = \cdots = o_{2t} = \top\}$.

Then we have

$$\begin{aligned} P_1(o) &= Pr[GPTT(D_1) = o] \\ &= \int_{-\infty}^{\infty} Pr[\tilde{\theta} = z] \prod_{i=1}^{t} Pr[\tilde{q}_i \leqslant z] \prod_{i=t+1}^{2t} Pr[\tilde{q}_i > z] dz \\ &= \int_{-\infty}^{\infty} f_{\epsilon_1}(F_{\epsilon_2}(z))^t (1 - F_{\epsilon_2}(z-1))^t dz \\ &= \int_{-\infty}^{\infty} f_{\epsilon_1}(F_{\epsilon_2}(z) - F_{\epsilon_2}(z)F_{\epsilon_2}(z-1))^t dz, \end{aligned} \tag{3.11}$$

where f_ϵ and F_ϵ are the PDF and CDF, respectively, of the Laplace distribution with parameter $1/\epsilon$. Similarly, we can derive, in terms of datasets D_2, that

$$P_2(o) = Pr[GPTT(D_2) = o] = \int_{-\infty}^{\infty} f_{\epsilon_1}(F_{\epsilon_2}(z-1) - F_{\epsilon_2}(z-1)F_{\epsilon_2}(z))^t dz. \tag{3.12}$$

Let $P_2(o) = \alpha$ and $\delta = |F_{\epsilon_1}^{-1}(\alpha/4)|$. Since $\alpha \leqslant 1$, δ is greater than the 75^{th} percentile of a Laplace distribution with scale $1/\epsilon_1$. That means,

$$\frac{\alpha}{2} \geqslant \int_{-\infty}^{-\delta} f_{\epsilon_1}(t)dt + \int_{\delta}^{\infty} f_{\epsilon_1}(t)dt. \tag{3.13}$$

Moreover, note that $F_{\epsilon_2}(z-1) < F_{\epsilon_2}(z)$ for all z, we have

$$\kappa(z) = \frac{F_{\epsilon_2}(z) - F_{\epsilon_2}(z)F_{\epsilon_2}(z-1)}{F_{\epsilon_2}(z-1) - F_{\epsilon_2}(z-1)F_{\epsilon_2}(z)} > 1. \tag{3.14}$$

Let $\kappa = \min_{z \in [-\delta, \delta]} \kappa(z)$ and $\kappa > 1$. We can get

$$\begin{aligned} \alpha = P_2(o) &= \int_{-\infty}^{\infty} f_{\epsilon_1}(F_{\epsilon_2}(z-1) - F_{\epsilon_2}(z-1)F_{\epsilon_2}(z))^t dz \\ &< \int_{z \notin [-\delta, \delta]} f_{\epsilon_1} dz + \int_{-\delta}^{\delta} f_{\epsilon_1}(F_{\epsilon_2}(z-1) - F_{\epsilon_2}(z-1)F_{\epsilon_2}(z))^t dz \\ &\leqslant \frac{\alpha}{2} + \int_{-\delta}^{\delta} f_{\epsilon_1}(F_{\epsilon_2}(z-1) - F_{\epsilon_2}(z-1)F_{\epsilon_2}(z))^t dz. \end{aligned} \tag{3.15}$$

Thus, we can derive

$$\int_{-\delta}^{\delta} f_{\epsilon_1}(F_{\epsilon_2}(z-1) - F_{\epsilon_2}(z-1)F_{\epsilon_2}(z))^t dz > \frac{P_2(o)}{2}. \tag{3.16}$$

Therefore, we get

$$\begin{aligned} P_1(o) &= \int_{-\infty}^{\infty} f_{\epsilon_1}(F_{\epsilon_2}(z) - F_{\epsilon_2}(z)F_{\epsilon_2}(z-1))^t dz \\ &> \int_{-\delta}^{\delta} f_{\epsilon_1}(F_{\epsilon_2}(z) - F_{\epsilon_2}(z)F_{\epsilon_2}(z-1))^t dz \\ &> \int_{-\delta}^{\delta} f_{\epsilon_1}(F_{\epsilon_2}(z-1) - F_{\epsilon_2}(z-1)F_{\epsilon_2}(z)\kappa(z))^t dz \\ &> \frac{\kappa^t}{2} P_2(o). \end{aligned} \tag{3.17}$$

Since $\kappa > 1$, for every finite $\epsilon > 1$, there exists a value of t such that $\kappa^t \geqslant 2\epsilon$, which means $P_1(o) > e^{\epsilon} P_2(o)$, violating differential privacy.

□

We now indicate the subtle error in the privacy analysis shown in Section 3.2.2, where the previous works ([37], [55], [14]) made.

In the previous works, they split probability of $P(o)$ into the following two parts:

$$\begin{aligned} P(o) &= \prod_{i:o_i=\perp} Pr[o_i = \perp \mid o^{<i}] \prod_{i:o_i=\top} Pr[o_i = \top \mid o^{<i}] \\ &= \int_{-\infty}^{\infty} Pr[\tilde{\theta} = x] \prod_{i:o_i=\perp} Pr[o_i = \perp \mid x, o^{<i}]dx \\ &\quad \times \int_{-\infty}^{\infty} Pr[\tilde{\theta} = x] \prod_{i:o_i=\top} Pr[o_i = \top \mid x, o^{<i}]dx \\ &= \int_{-\infty}^{\infty} Pr[\tilde{\theta} = x] \prod_{i:o_i=\perp} Pr[o_i = \perp \mid x]dx \\ &\quad \times \int_{-\infty}^{\infty} Pr[\tilde{\theta} = x] \prod_{i:o_i=\top} Pr[o_i = \top \mid x]dx. \end{aligned} \tag{3.18}$$

This decomposition step is actually incorrect. The main problem comes from the fact that the distribution of the noisy threshold would be affected given the previous outputs.

To take a simple example, we make $q_1 = m > 0$, $q_2 = 0$ and $\theta = 0$. We assume the output is $o_1 = \perp$ and $o_2 = \top$. For ease of explanation, we assume $\epsilon_2 = \infty$ (but the argument would still work for any finite ϵ_2. Now we can compute the probability of GPTT with outputs o_1 and o_2 as

$$\begin{aligned} Pr[o_1 = \perp, o_2 = \top] &= Pr[o_1 = \perp] Pr[o_2 = \top \mid o_1 = \perp] \\ &= Pr[o_1 = \perp] Pr[m \leqslant \tilde{\theta} \mid 0 > \tilde{\theta}] = 0 \end{aligned} \tag{3.19}$$

However, if we use the incorrect decomposition shown in Expression (3.18), we

will get

$$\begin{aligned} P(o) &= \int_{-\infty}^{\infty} Pr[\tilde{\theta} = x] \prod_{i:o_i=\perp} Pr[o_i = \perp \mid x]dx \\ &\quad \times \int_{-\infty}^{\infty} Pr[\tilde{\theta} = x] \prod_{i:o_i=\top} Pr[o_i = \top \mid x]dx \\ &= \int_{-\infty}^{\infty} f(x)Pr[m \leqslant x]dx \int_{-\infty}^{\infty} f(x)Pr[0 > x]dx \\ &= F(m)(1 - F(0)) > 0, \end{aligned} \tag{3.20}$$

where $F(x)$ is the CDF of the noisy threshold.

We give a correct decomposition as the following:

$$\begin{aligned} P(o) &= \prod_{i:o_i=\perp} Pr[o_i = \perp \mid o^{<i}] \prod_{i:o_i=\top} Pr[o_i = \top \mid o^{<i}] \\ &= \int_{-\infty}^{\infty} Pr[\tilde{\theta} = x \mid o^{<i}] \prod_{i:o_i=\perp} Pr[o_i = \perp \mid x, o^{<i}]dx \\ &\quad \times \int_{-\infty}^{\infty} Pr[\tilde{\theta} = x \mid o^{<i}] \prod_{i:o_i=\top} Pr[o_i = \top \mid x, o^{<i}]dx \\ &= \int_{-\infty}^{\infty} Pr[\tilde{\theta} = x \mid o^{<i}] \prod_{i:o_i=\perp} Pr[o_i = \perp \mid x]dx \\ &\quad \times \int_{-\infty}^{\infty} Pr[\tilde{\theta} = x \mid o^{<i}] \prod_{i:o_i=\top} Pr[o_i = \top \mid x]dx. \end{aligned} \tag{3.21}$$

3.2.4 Reconstructing Data using GPTT

In the previous section, we showed that the GPTT does not satisfy ϵ-differential privacy for any finite ϵ. While this is an interesting result, it still leaves an open question whether GPTT indeed leaks a significant amount of information about the input datasets and allows attacks like re-identification of individuals based on quasi-identifiers.

In this section, we answer this question in the affirmative, and show that GPTT can disclose the exact counts of domain values with high probability. Exact disclosure of cells with small counts (especially, cells with counts 0, 1 and 2) reveals the presence of unique individuals in the data who can be susceptible to re-identification attacks.

We will consider the special case of GPTT where $\epsilon_2 = \infty$. The general case with finite ϵ_2 can be similarly analyzed.

In the attack we designed, we make use of a new set of queries, called *differerence query*, which compute the difference between the counts of one selected pair of domain elements.

Definition 5 (Difference Query)**.** *Let $u_1, u_2 \in \Gamma$ be a pair of domain elements, and let x_1 and x_2 be the frequency of u_1 and u_2 in the input dataset D. The difference query, $diff(u_1, u_2)$ is then defined as*

$$diff(u_1, u_2) = x_1 - x_2, \tag{3.22}$$

where the sensitivity of query $diff$ is $\Delta(diff) = 1$.

Algorithm 6 Attack Algorithm for GPTT

Input: Dataset D with domain Γ, privacy budget ϵ, failure rate δ
Output: a partition of the domain $P = \{P_0, P_1, \ldots, P_p\}$

$Q \leftarrow \{diff(u, v) \mid u, v \in \Gamma \text{ and } u \neq v\}$
$\theta \leftarrow \lceil \frac{1}{\epsilon} \log(\frac{1}{\delta}) \rceil$
Run $GPTT(D \mid \epsilon_1 = \epsilon, \epsilon_2 = \infty)$ with query Q and threshold θ
for each $v \in \Gamma$ **do**
    $Larger(v) \leftarrow \{u \in \Gamma \mid o_{diff(u,v)} = \top\}$
end for
Construct the ordered partition of domain $P = \{P_1, \ldots, P_p\}$, such that
        $\forall u, v \in P_i$, $Larger(v) = Larger(u)$
        $\forall u \in P_i, v \in P_{i+1}$, $Larger(v) \subsetneq Larger(u)$
return P

We now design an attack algorithm described in Algorithm 6. Given the input dataset D with domain Γ, we apply GPTT ($\epsilon_2 = \infty$) to the set of all difference queries in terms of every pair of domain elements $u, v \in \Gamma$. We set the threshold

$\theta = \lceil \frac{1}{\epsilon} \log(\frac{1}{\delta}) \rceil$. Pairs of domain elements $u, v \in \Gamma$ are then grouped together if for each $w \in \Gamma$, GPTT gives the same output for $diff(u, w)$ and $diff(v, w)$. Algorithm 6 will result in a partition of the entire domain Γ. Furthermore, for each domain element $u \in \Gamma$, we define $Larger(u)$ to be the set of domain elements from Γ, where for each $v \in Larger(u)$, GPTT outputs $\top$ for $diff(v, u)$. These are the domain elements that ensure $x_v - x_u > \tilde{\theta}$, where $\tilde{\theta}$ is the noisy threshold. We order the partitions at the end such that every element $u \in P_i$ has a bigger $Larger(u)$ set than any element $v \in P_j$, for $j > i$.

We can show that the ordered partitions P imposes an ordering on the counts in the database D.

Lemma 5. *Let D be a dataset from domain Γ. Let $P = \{P_0, P_1, \ldots, P_p\}$ be the ordered partitions of Γ output by Algorithm 6. Then, with probability $1 - \delta$, for all $0 \leqslant l < m \leqslant p$, $u \in P_l$, $v \in P_m$, we have $x_u < x_v$.*

Proof. Let $\tilde{\theta}$ be the noisy threshold. Since $\theta = \lceil \frac{1}{\epsilon} \log(\frac{1}{\delta}) \rceil$, with probability at least $1 - \delta$, $\tilde{\theta} > 0$. For any $u \in P_l$ and $v \in P_m$, we know $Larger(v) \subsetneq Larger(u)$. Thus, there exists $w \in \Gamma$ such that $x_w - x_u > \tilde{\theta}$ but $x_w - x_v \leqslant \tilde{\theta}$. Therefore, $x_u < x_v$. □

Let $S_i \subset \Gamma$ denote the set of domain elements whose frequency in D is equal to i. It is easy to see that for every S_i, there is some $P^* \in P$ output by Algorithm 6 such that $S_i \subset P^*$. We next show that, for certain datasets, there exists an $m > 0$ such that the sets of domain elements with frequency between [0, m] can be exactly reproduced in the partition from Algorithm 6.

Theorem 6. *Let $P = \{P_0, P_1, \ldots, P_p\}$ be the ordered partitions output by Algorithm 6 based on the input D with parameter ϵ. Let D be a dataset such that $S_i \neq \varnothing$ for all $i \in [0, k]$. That is, D contain at least one domain element with frequency equal to $0, 1, \ldots, k$. Let $\alpha = \lceil \frac{1}{\epsilon} \log(\frac{1}{\delta}) \rceil$. If $k > 2\alpha$, with probability at least $1 - \delta$, for all $i \in [0, m]$, $P_i = S_i$, where $m = k - 2\alpha$.*

Proof. Since $\theta = \alpha = \lceil \frac{1}{\epsilon} \log(\frac{1}{\delta}) \rceil$, with probability at least $1 - \delta$, the noisy threshold will be within $[0, 2\alpha]$. For the dataset D such that $S_i \neq \varnothing$ for all $i \in [0, k]$, where $k > 2\alpha$, let $i \in [0, m-1]$. Suppose $u \in S_i$ and $v \in S_{i+1}$, then we have

$$\begin{aligned} Larger(u) &= \{z \mid x_z > \tilde{\theta} + i\} \\ Larger(v) &= \{z \mid x_z > \tilde{\theta} + i + 1\} \end{aligned} \tag{3.23}$$

Since $i \in [0, m-1]$, with probability at least $1-\delta$, $\tilde{\theta} + i \leqslant m - 1 + 2\alpha < k$. Thus, we have

$$S_{\lceil \tilde{\theta}+i \rceil} \subset Larger(u) - Larger(v) \neq \varnothing \tag{3.24}$$

Therefore, we have $Larger(u) \subsetneq Larger(v)$ and S_i, S_{i+1} will not appear in the same partition $P^* \in P$.

Furthermore, since we know $S_0, S_1, \ldots, S_m$ belong to separate partitions from P and we have $P = \{P_0, P_1, \ldots, P_p\}$ be the ordered partitions. Thus, we have $P_i = S_i$ for $i \in [0, m]$. □

Theorem 6 shows that, for datasets that have domain elements with frequency equal to i for all $i \in [0, k]$, we can exactly tell the domain elements with frequency within $[0, k-2\alpha]$ with high probability. A number of datasets satisfy this assumption. For instance, the datasets that are drawn from a Zipfian distribution, where the size of S_i is in expectation inversely proportional to the count i, and thus all small counts will have support for datasets of sufficiently large size. We also tested our attack algorithm on a number of real-world datasets.

Table 3.1 displays the overview of some real-world datasets. **Adult** is a histogram constructed from U.S. Census data [40] on the attribute of "capital loss". **Medical-Cost** is a histogram of personal medical expenses from the survey of United States department of health, human services in 2007. **Income** is the histogram on "personal income" attribute from [36]. **HepPh** is a histogram constructed using the citation

Table 3.1: Overview of Real Datasets for Reconstruction

Datasets	Domain Size	Scale	k
Adult	4096	17665	15
MedicalCost	4096	9415	26
Income	4096	20787122	98
HepPh	4096	347414	275

network among high energy physics pre-prints on arXiv. The attributes *DomainSize* and *Scale* in Table 3.1 correspond to the size of Γ and the number of tuples in the datasets. The attribute k for each dataset indicates that $S_i \neq \varnothing$ for all $i \in [0, k]$.

Algorithm 7 Data Reconstruction Algorithm

Input: Dataset D with domain Γ, privacy budget ϵ, failure rate δ
Output: A private reconstruction of D

1: $\epsilon \leftarrow \epsilon_1 + \epsilon_2$
2: $P \leftarrow Attack\ Algorithm(D, \epsilon_1, \delta)$
3: **for** each $P^* \in P$ **do**
4: $\quad \tilde{c}_{P^*} \leftarrow count(P^*) + Lap(2/\epsilon_2)$
5: $\quad a_{P^*} \leftarrow \frac{\tilde{c}_{P^*}}{|P|}$
6: $\quad$ We estimate the counts of cells in P^* is round(a_{P^*})
7: **end for**

Empirical Data Reconstruction

We designed Algorithm 7 for reconstructing a dataset using GPTT and differentially private access to the dataset. Algorithm 7 split the total privacy budget into $\epsilon = \epsilon_1 + \epsilon_2$. We use ϵ_1 privacy budget to run Algorithm 6 in terms of GPTT, which will output a set of ordered partitions P of the domain. We use the remaining privacy budget ϵ_2 to compute the noisy total number of tuples within the domain for each partition $P^* \in P$, and estimate the average counts for each domain cell. We round this average to the nearest integer as the estimate of the frequency for each domain element in P^*.

Table 3.2 presents the fraction of domain elements in each dataset whose frequency are corrected estimated under Algorithm 7 with $\epsilon \in \{1.0, 0.5, 0.1\}$. Each value is the average among 10 repetitions. For all the experiments, we set $\epsilon_1 = \epsilon_2 = \epsilon/2$

Table 3.2: Success Rate of Empirical Datasets Reconstruction under Different ϵ

	$\epsilon = 1.0$	$\epsilon = 0.5$	$\epsilon = 0.1$
Adult	0.994	0.991	0.981
MedicalCost	0.985	0.977	0.949
Income	0.798	0.741	0.636
HepPh	0.904	0.795	0.477

Table 3.3: Success Rate of Empirical Datasets Reconstruction for small counts ([0,5]) under Different ϵ

	# of cells with counts in [0,5]	$\epsilon = 1.0$	$\epsilon = 0.5$	$\epsilon = 0.1$
Adult	4062	0.999	0.997	0.992
MedicalCost	3878	1.0	1.0	0.960
Income	2369	1.0	1.0	0.979
HepPh	1153	1.0	1.0	0.970

and $\delta = 0.05$.

You can see, when the ϵ is not small, the counts of a large portion of the domain cells can be correctly reconstructed (Over 90% of the domain counts can be reconstructed in 3 datasets). With the decreasing of the ϵ, the ratio becomes smaller. Two reasons explain this result: (1) Smaller ϵ_1 leads to coarser partitions from Algorithm 6. (2) Smaller ϵ_2 introduces much noise to the counts of each partition, resulting inaccurate estimates.

Table 3.3 displays the accuracy of reconstruction on domain cells with small counts with $[0, 5]$. Note that more than 25% of the domain have frequency within $[0, 5]$ for all these datasets. Again, each value is the average among 10 repetitions.

You can see that more than 95% of all domain elements with small counts within [0,5] can be accurately reconstructed for all these 4 datasets under all the ϵ settings. Especially, when ϵ is not small (i.e., $\epsilon = 1.0$), nearly all these cells can be reconstructed by using Algorithm 7.

All these results show that not only does GPTT not ensure differential privacy, but it can lead to significant loss of privacy as well. Since cells with small counts

can be reconstructed with very high probability ($> 95\%$) from the output of GPTT, accessing to the data via GPTT allows re-identification attacks. Hence, we believe that systems whose privacy stems from GPTT are not safe to use.

3.3 Generalized Version of Sparse Vector Technique

We have discussed the incorrect variants of *Sparse Vector Technique* proposed in the previous works. In the section, we present another generalized version of *Sparse Vector Technique*, which relaxed the conditions in terms of the threshold settings. Specifically, we allow different thresholds to be compared with different queries. Also the compared threshold can also be dependent on the input dataset.

3.3.1 Description of Generalized Sparse Vector Technique

Algorithm 8 Generalized Sparse Vector Technique

Input: Dataset D, a stream of queries $Q = q_1, q_2, \dots$ with bounded sensitivity Δ_1, a set of threshold functions $\Theta = \theta_1, \theta_2, \dots$ with bounded sensitivity Δ_2 , a cutoff point c and privacy budget ϵ
Output: a stream of answers $\in \{\perp, \top\}$

$\alpha \leftarrow Lap(\frac{2(\Delta_1+\Delta_2)}{\epsilon}$
for each query $i = 1, 2, \dots$ **do**
    $v_i \leftarrow Lap(\frac{4c(\Delta_1+\Delta_2)}{\epsilon})$, $count \leftarrow 0$
    if $q_i(D) + v_i \geqslant \theta_i(D) + \alpha$ **then**
        Output $\top$, $count \leftarrow count + 1$
    else
        Output $\perp$
    end if
    if $count == c$ **then**
        Abort
    end if
end for

Algorithm 8 displays our proposed generalized *Sparse Vector Technique*. It takes as inputs a stream of queries $Q = \{q_1, q_2, \dots\}$, where each $q \in Q$ has sensitivity bounded by Δ_1, another stream of threshold functions $\Theta = \{\theta_1, \theta_2, \dots, \}$, where each $\theta \in \Theta$ has bounded sensitivity Δ_2, a cutoff point c limiting the number of queries with identity above the threshold to output and a privacy budget ϵ. Algorithm 8 will

give a stream of outputs from $\{\perp, \top\}$ with at most c $\top$. Algorithm 8 works based on the following two steps:

1. Generate a random noise α from the Laplace distribution with scale $\frac{2(\Delta_1+\Delta_2)}{\epsilon}$.

2. Perturb each query q_i by adding Laplace noise with scale $\frac{4c(\Delta_1+\Delta_2)}{\epsilon}$, getting $\tilde{q}_i$. Then it outputs $\top$, when $\tilde{q}_i(D) \geqslant \theta_i(D) + \alpha$, and $\perp$ otherwise.

Algorithm 8 will stop, when it outputs c $\top$ values.

3.3.2 Privacy Analysis of Generalized Sparse Vector Technique

Theorem 7. *The generalized* Sparse Vector Technique *(Algorithm 8) ensures ϵ-differential privacy.*

Proof. Suppose D and D' are any two neighboring datasets. Let any output $O = O_\perp \cap O_\top$, where $O_\perp$ and $O_\top$ contain the set of queries labeled as $\perp$ and $\top$, respectively. The cutoff point c indicates that $|O_\top| = c$. Then we have

$$\begin{aligned} & Pr[\mathbf{A}(D) = O] \\ = & \int_{-\infty}^{\infty} Pr[\alpha = t](\prod_{i \in O_\perp} Pr[\tilde{q}_i(D) < \theta_i(D) + \alpha)(\prod_{i \in O_\top} Pr[\tilde{q}_i(D) \geqslant \theta_i(D) + \alpha])dt. \end{aligned} \tag{3.25}$$

Let $\tilde{q}_i(D) = q_i(D) + v_i$, where v_i is the independent Laplace noise injected to each query in the line 4 of Algorithm 8. Then, we can derive

$$
\begin{aligned}
& Pr[\mathbf{A}(D) = O] \\
= & \int_{-\infty}^{\infty} Pr[\alpha = t](\prod_{i \in O_\perp} Pr[\tilde{q}_i(D) < \theta_i(D) + t)(\prod_{i \in O_\top} Pr[\tilde{q}_i(D) \geqslant \theta_i(D) + t])dt \\
\leqslant & \int_{-\infty}^{\infty} e^{\epsilon/2} Pr[\alpha = t + \Delta_1 + \Delta_2](\prod_{i \in O_\perp} Pr[\tilde{q}_i(D) < \theta_i(D) + t) \\
& \times(\prod_{i \in O_\top} Pr[\tilde{q}_i(D) \geqslant \theta_i(D) + t])dt \\
\leqslant & \int_{-\infty}^{\infty} e^{\epsilon/2} Pr[\alpha = t + \Delta_1 + \Delta_2](\prod_{i \in O_\perp} Pr[\tilde{q}_i(D') < \theta_i(D') + t + \Delta_1 + \Delta_2) \\
& \times(\prod_{i \in O_\top} Pr[v_i \geqslant \theta_i(D) + t + \Delta_1 + \Delta_2 - q_i(D)])dt \\
\leqslant & \int_{-\infty}^{\infty} e^{\epsilon/2} Pr[\alpha = t + \Delta_1 + \Delta_2](\prod_{i \in O_\perp} Pr[\tilde{q}_i(D') < \theta_i(D') + t + \Delta_1 + \Delta_2) \\
& \times(\prod_{i \in O_\top} e^{\epsilon/2c} Pr[v_i \geqslant \theta_i(D') + t + \Delta_1 + \Delta_2 - q_i(D')])dt \\
= & e^{\epsilon} \int_{-\infty}^{\infty} Pr[\alpha = t](\prod_{i \in O_\perp} Pr[\tilde{q}_i(D') < \theta_i(D') + t)(\prod_{i \in O_\top} Pr[\tilde{q}_i(D') \geqslant \theta_i(D') + t])dt. \\
= & e^{\epsilon} Pr[\mathbf{A}(D') = O] \qquad (3.26)
\end{aligned}
$$

□

3.3.3 Utility Analysis of Generalized Sparse Vector Technique

We also analyze the utility of the generalized *Sparse Vector Technique*.

Theorem 8. *The outputs from generalized* Sparse Vector Technique *is* (γ, β)*-accurate for*

$$\gamma = \frac{8c(\Delta_1 + \Delta_2)(\log k + \log(2/\beta)}{\epsilon}, \qquad (3.27)$$

where k is the total number of queries tested.

Proof. The noise $\alpha \sim Lap(\frac{2(\Delta_1+\Delta_2)}{\epsilon})$, and we have

$$Pr[|\alpha| \geqslant t \times \frac{2(\Delta_1+\Delta_2)}{\epsilon}] = e^{-t}$$

$$\Rightarrow \quad Pr[|\alpha| \geqslant \frac{\gamma}{2}] = e^{-\frac{\epsilon\gamma}{4(\Delta_1+\Delta_2)}}. \tag{3.28}$$

When setting this quantity to be at most $\beta/2$, we require

$$e^{-\frac{\epsilon\gamma}{4(\Delta_1+\Delta_2)}} \leqslant \frac{\beta}{2}$$

$$\Rightarrow \quad \gamma \geqslant 4(\Delta_1+\Delta_2)\log(2/\beta)/\epsilon. \tag{3.29}$$

Similarly, for each query q_i, the injected noise $v_i \sim Lap(\frac{4c(\Delta_1+\Delta_2)}{\epsilon})$. Thus, we have

$$Pr[|v_i| \geqslant \frac{\gamma}{2}] = e^{-\frac{\epsilon\gamma}{8c(\Delta_1+\Delta_2)}}. \tag{3.30}$$

By using a union bound, we have

$$Pr[\max_{i\in[k]} |v_i| \geqslant \frac{\gamma}{2}] \leqslant k \times e^{-\frac{\epsilon\gamma}{8c(\Delta_1+\Delta_2)}}. \tag{3.31}$$

We also set the quantity to be at most $\beta/2$ and we have

$$k \times e^{-\frac{\epsilon\gamma}{8c(\Delta_1+\Delta_2)}} \leqslant \frac{\beta}{2}$$

$$\Rightarrow \quad \gamma \geqslant \frac{8c(\Delta_1+\Delta_2)(\log k + \log(2/\beta)}{\epsilon}. \tag{3.32}$$

By combining the above two analysis, we will have

$$Pr[|\alpha| + \max_{i\in[k]} |v_i| \geqslant \alpha] \leqslant \beta, \tag{3.33}$$

when $\gamma \geqslant \frac{8c(\Delta_1+\Delta_2)(\log k+\log(2/\beta)}{\epsilon}$, which proves the theorem. □

4

Differentially Private Regression Diagnostics

In this Chapter, we focus on solving the problem of doing regression diagnostics under differential privacy. We will use the *Sparse Vector Technique* as a key differentially private primitive for designing some components of the algorithms.

4.1 Introduction

The increasing digitization of personal information increased concerns over ensuring the confidentiality of the individuals from whom such data are collected. And differential privacy was proposed for protecting the sensitive information of individuals from being inferred by attackers.

We consider contexts where analysts, working under privacy constraints, seek to explain or predict some outcome y from a multi-variate set of variables $\mathbf{x}$. Absent privacy constraints, the go-to tool for this task is regression modeling.

For continuous outcomes, the most popular model is *linear regression*, in which we assume that $y = \beta_0 + \boldsymbol{\beta} \cdot \mathbf{x} + \xi$, where the βs are called *coefficients*, and ξ is the *error* assumed to be from the normal distribution $N(0, \sigma^2)$ with zero mean and some unknown fixed standard deviation σ.

For binary outcomes ($y \in \{0, 1\}$), the most popular model is *logistic regression*, in which we assume that y, given $\mathbf{x}$, has a Bernoulli distribution with probability $P(y = 1 \mid \mathbf{x})$ where $\log[P(y = 1 \mid \mathbf{x})/(1 - P(y = 1 \mid \mathbf{x}))] = \beta_0 + \boldsymbol{\beta} \cdot \mathbf{x}$. One can use non-linear functions of $\mathbf{x}$ in the predictor function in either model.

Regression is used for two primary tasks – *explain* associations between the outcome and the explanatory variable and *predict* the outcome on unseen data. The former is especially important in public health, epidemiology, and the social sciences, where the scientific questions typically focus on whether specific variables are important (e.g., statistically significant) or unimportant predictors of the outcome. For example, in a linear regression seeking to explain salaries from demographic variables, a social scientist might be interested in whether or not the coefficient (β) for an explanatory variable indicating "sex is female" is statistically significant, given the other variables in the model. The validity of such inferences depends critically on the reasonableness of the assumptions underpinning the model. For example, if the analyst specifies a linear relationship between salary and age (in years) in the model, but the true generative relationship is quadratic (as it generally is), it is pointless to interpret the inferences for the coefficient for age from the ordinary least squares fit. The fitted model completely misrepresents the association. Thus, besides estimating the parameters of the model, analysts must be able to evaluate how well the posited model assumptions fit the theoretical assumptions underpinning regression analyses for the data at hand. Similarly, when regression is used for prediction, analysts must be able to assess the extent to which the model can predict outcomes accurately for unseen data (e.g., out of sample records).

Absent privacy constraints, analysts have a variety of tools to diagnose the fit and predictive power of regression models. Model fit for linear regression is typically assessed by examining the distribution of *residuals*, which are observed values minus predicted values (defined formally in Section 4.2). For logistic regression, model

fit is often evaluated using binned residual plots. These plots partition the data into disjoint bins based on the fitted values (the predicted probabilities), compute the average residual in each bin, and plot the average residuals versus the averaged fitted values across all bins. To evaluate predictive power for logistic regression, a popular tool is the receiver operating characteristic (ROC) curve that plots the true positive rate against the false positive rate of predictions on unseen data. The area under the ROC curve (or AUC, a value in [0,1]) is typically used to compare models.

Assessing model fit and predictive power are especially crucial for analysts working under privacy constraints. While there are a number of algorithms that compute regression coefficients (βs) in a differentially private manner (typically by adding noise) [13, 52, 59, 63, 64], there is no guarantee that the underlying model accurately describes the relationships in the confidential data, nor that it yields high-quality predictions. Similarly, a model that fits or predicts well on synthetic data may not do so for the confidential data. Unfortunately, analysts working under privacy constraints cannot generate residual plots, binned residual plots and ROC curves computed directly from the confidential data, as this can disclose sensitive properties about individual records [48].

Despite its importance, there has been little work on designing private regression diagnostics. We are not aware of any work on assessing model fit in a differentially private manner, especially using residual plots or binned residual plots. Other than Boyd et al. [5], which considers differentially private computation of the AUC and a preliminary version of our work [55], there isn't any prior work on plotting ROC curves.

In this part, we develop the first algorithms for differentially private regression diagnostics. Our contributions are:

- We design differentially private algorithms for constructing residual plots, binned

residual plots and ROC curves. While the algorithms are based on well-known building blocks for answering range queries on databases, their application to assessing regression models is novel.

- Plotting residuals for linear regression under differential privacy is challenging, since *a priori* there is no bound on residuals. Hence, we first design a differentially private algorithm for linear regression to estimate bounds on residuals, and then use private space partitioning techniques to visualize the density of the residuals in the bounded region. Using synthetic datasets, we perform controlled experiments showing that an analyst can determine whether or not the regression assumptions are satisfied when the product of the dataset size and ϵ exceeds 1000.

- On real datasets, our private ROC curves approximate the true ROC curves well. Moreover, when the product of the privacy loss ϵ and dataset size is at least 1000, our techniques can distinguish ROC curves with 0.025 difference in AUC.

- Using synthetic datasets, we display that our generated private binned residual plot preserves the pattern observed in the real binned residual plot. Empirically, we show the differentially private binned residual plots provide useful diagnostic information when the product of data scale and ϵ is more than 20000.

- Analysts are trained to assess model fit by visual inspection. In addition to having low error, our algorithms can preserve the visual characteristics of these plots.

Organization: In Section 4.2, we present our differentially private algorithm for computing residual plots and a comprehensive empirical evaluation of its perfor-

mance. In Section 4.3, we present the differentially private algorithm for plotting ROC curves and evaluate its utility. In Section 4.4, we present the algorithm for computing differentially private binned residual plots and empirically evaluate its performance. We conclude in Section 4.5.

4.2 Residual Plot

In this section, we introduce a differentially private algorithm for computing distributions of residuals for linear regression, which are used to verify model fit for linear regression. As far as we know, this is the first work on generating plots of residuals under differential privacy.

4.2.1 Review of Residual Diagnostics

Let $D = \{(y_i, \mathbf{x}_i) : i \in [n]\}$ be a confidential training dataset. Linear regression models the outcome as:

$$y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} + \xi_i, \quad \xi_i \sim N(0, \sigma^2), \ \forall i. \tag{4.1}$$

Here, $\beta = (\beta_0, \ldots, \beta_p)$ is the $(p+1) \times 1$ vector of true regression coefficients; these are unknown and estimated from D, typically using ordinary least squares (OLS). The model is based on four key assumptions. One is that each ξ_i is independent, so that individual outcomes are independent; we take this as given. The remaining assumptions are listed below in order of decreasing importance.

1. At any value of $\mathbf{x}_i$, $E(y_i \mid \mathbf{x}_i) = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$.

2. At any value of $\mathbf{x}_i$, the true variance of y values around the regression line is constant, i.e., $Var(y_i \mid \mathbf{x}_i) = \sigma^2$.

3. At any value of $\mathbf{x}_i$, ξ_i follows a normal distribution.

The ideal check of these assumptions is to examine the distribution of ξ_i to see if it lines up with $N(0, \sigma^2)$. Since ξ_i is unknown, we instead examine the distribution of residuals, $r_i = y_i - \mathbf{x}_i\mathbf{b}$, where $\mathbf{b}$ is the estimate of β computed from OLS. When the regression model assumptions are valid, one can show that $E(r_i \mid \mathbf{x}_i) = 0$ and $Cov(r_i, \mathbf{x}_i) = 0$ for any i. Hence, when the assumptions are reasonable, the residuals should be centered around zero and exhibit no systematic pattern with any function of $\mathbf{x}$. For example, the plot of r versus the predicted values, $\hat{y} = \mathbf{x}\mathbf{b}$, should look like a randomly scattered set of points (see Figure 4.1(a)). On the other hand, when linearity is violated, the plot exhibits systematic curvatures or increasing (decreasing) trends of r with $\hat{y}$ (see Figure 4.1(c)). When constant variance is violated but linearity holds, we see a fan shape to the residuals, e.g., the spread of the residuals increases with $\hat{y}$ (see Figure 4.1(b)). This phenomenon is called *heteroscedasticity*. When normality is violated but linearity and constant variance hold, we can see a few residuals that are outliers. With bad fitting models, it is often the case that multiple assumptions fail simultaneously, and the plots exhibit non-random patterns. It is important to note, however, that residuals only can reveal problematic models; a random pattern only indicates lack of evidence the model is mis-specified, not proof that it is correctly specified.

Of course, when D is confidential, one cannot release the residual plot as is. For example, given $\mathbf{b}$ (or a noisy version of $\mathbf{b}$), and the values of $\mathbf{x}_i$ for some target record i, an adversary can deduce the confidential value of y_i simply by computing $r_i + \mathbf{x}_i\mathbf{b}$. Hence, we propose to release a noisy version of the plot of residuals versus predicted values that reveals whether or not one can trust the results from the specified regression.

Algorithm 9 PriRP

Input: Dataset D, a safe model β, ϵ
Output: a private residual plot
1: Step 1: Compute the bound of residuals and predicted values with ϵ_1 budget.
2: Step 2: Perturb the residual plots within the bound above with the left $\epsilon_2 = \epsilon - \epsilon_1$ budget.

4.2.2 Private Residual Plots

An important challenge in releasing differentially private residual plots is that we do not have a prior knowledge of bounds on either the predicted $\hat{y}$ or the residuals r (other than that they are in $(-\infty, \infty)$). Deriving tight upper and lower bounds on $\hat{y}$ and r is extremely important:

- Recent work [6] has proven the impossibility of releasing distributions (specifically CDFs) under differential privacy on infinite domains.
- Differentially private algorithms need to add noise to regions in the domain that have no support. Thus, even if one knew upper and lower bounds on the values, it is beneficial to obtain tighter bounds where most of the data reside in order to minimize noise.
- We also need the bounds to not exclude a large fraction of the $(\hat{y}, r)$ pairs in order to correctly observe trends.

Hence, we propose *PriRP* (see Algorithm 9) for constructing ϵ-differentially private residual plots that proceeds in two steps: 1) compute bounds on the range of predicted $\hat{y}$ and residuals r that cover most of the records in the original dataset using ϵ_1 of the total ϵ budget, and 2) release the distribution of $\hat{y}$ versus r within the bounds using the remaining budget $\epsilon_2 = \epsilon - \epsilon_1$. We next describe these steps in detail.

Computing private bounds: Given a list of real numbers $D = \{z_1, z_2, \ldots, z_n\}$, we would like to identify bounds $(-b, b)$ such that at least a θ fraction of the numbers

Algorithm 10 Private Bounds

Input: Dataset D, unit μ, private budget ϵ, threshold θ
Output: a value d (indicating a bound $[-d, d]$)

$\tilde{\theta} \leftarrow \theta \times n + Lap(2/\epsilon), d \leftarrow \mu$
while True **do**
    $q \leftarrow$ number of $x \in D$ within bound $[-d, d]$
    if $q + Lap(4/\epsilon) < \tilde{\theta}$ **then** $d \leftarrow d * 2$
    else return d
    end if
end while

are contained in $(-b, b)$ with probability of at least $(1 - \alpha)$. We can not use prior work on releasing quantiles under differential privacy (e.g., Algorithm 2 in [53]) since they assume knowledge of a weak upper and lower bound on the input data.[1]

Our approach, shown in Algorithm 10, is an application of the *Sparse Vector Technique* [19]. The algorithm successively poses queries of the form $q_i =$"number of observations in D within the bounds $[-\mu * 2^i, \mu * 2^i]$", where $i = 0, 1, \ldots$ and μ is a scaling constant. The algorithm stops when the noisy answer to query $\tilde{q}_i = q_i + Lap(4/\epsilon)$ exceeds about a θ fraction of the data (i.e. $\tilde{q}_i > \tilde{N} = \theta \cdot n + Lap(2/\epsilon)$).

Theorem 9. *Let D denote a list of real numbers. Algorithm 10 satisfies ϵ-differential privacy. Moreover, if a θ fraction of elements in D lie in $[-\mu \cdot 2^k, \mu \cdot 2^k]$, then with probability at least $1 - \beta$, the output bounds will contain at least $\theta - \alpha$ fraction of elements in D, where $\alpha = \frac{8(\log(k-1)+\log(2/\beta))}{n \cdot \epsilon}$.*

Proof. From [19], we know that Algorithm 10 satisfies ϵ-differential privacy. Moreover, suppose the algorithm stops after answering $\ell + 1$ queries, then we can show (using [19]) that with probability at least $1 - \beta$, $q_{\ell+1} \geqslant n \cdot (\theta - \alpha_\ell)$, where $\alpha_\ell = \frac{8(\log(\ell)+\log(2/\beta))}{n \cdot \epsilon}$. The algorithm outputs bounds $[-\mu \cdot 2^{\ell+1}, \mu \cdot 2^{\ell+1}]$. If $\ell + 1 \geqslant k$, then the output bounds are guaranteed to contain a θ fraction of the elements in D. Otherwise, $\ell + 1 < k$ implies $\alpha_\ell < \alpha$ giving us the desired result. □

As an illustration of Theorem 9, suppose a θ fraction of D lie within $[-\mu \cdot 2^{10}, \mu \cdot$

[1] Moreover, the quality of these algorithms also depends on how tight the assumed bounds are.

2^{10}], and let $n \cdot \epsilon \geqslant 470$. With probability $\geqslant 0.95$, the bounds output by Algorithm 10 contains at least a $\theta - 0.1$ fraction of the elements in D. Note that the error depends on (i) the product of n and ϵ, as well as (ii) μ. The former is a proxy for the amount of signal that can be extracted from the data. μ represents a unit quantity for the elements in D, and should be set neither too large nor too small.

Application to residual plots: We use Algorithm 10 to estimate bounds on $\hat{y}$ and r separately, expending $\epsilon_1/2$ privacy budget each time. We set μ based on the type of y. For example, when y describes an integer-valued outcome with few levels (e.g., a test score), it makes sense to set $\mu = 1$. When the attribute describes a long-tailed variable (e.g., salary), it is better to set $\mu = 1000$. In our experiments, n is known and θ is usually set to 0.95. We set $\epsilon_1 = \min\{0.3, \frac{470}{\epsilon n}\} \times \epsilon$. Based on the illustration above, $\epsilon_1 = \frac{470}{n}$ results in bounds covering around θ fraction of records with high probability. To avoid ϵ_1 getting too large, we set it to 0.3ϵ for small n.

Perturbing residual plots: We visualize the relationships between $\hat{y}$ vs r by privately estimating the 2D probability density inside the bounded region output by Algorithm 10. This can be done using private space partitioning techniques in three steps – *discretization*, *perturbation* and *sampling*.

Discretization: We first discretize the bounded region into m^2 grid cells of equal size. We then construct a histogram $\mathbf{h}$ with m^2 counts, where the count in each cell represents the number of records in D with values $(\hat{y}_i, r_i)$ contained in that cell. Based on analysis similar to [39], we set $m = \sqrt{\frac{N_0 \epsilon}{10}}$, where N_0 is the number of records in D that have $(\hat{y}_i, r_i)$ in the bounded region. Rather than estimating N_0 noisily (with some loss in privacy budget), we approximate N_0 using $\theta^2 \times n$, the expected number of records that are captured in the bounded region by Algorithm 10.

Perturbation: Recent work [31] shows that *DAWA* is one of the best algorithms for

privately answering range queries over 2D histograms. *DAWA* works by first ordering the cells in the histogram according to a Hilbert curve, and then grouping together contiguous ranges of cells (in this ordering) that have similar counts. *DAWA* noisily computes the total counts for each of these groups, and reconstructs the original cell counts by assuming that all cell counts within a group are the same. Partitioning the cells into groups of similar counts allows *DAWA* to minimize the total noise added.

The noisy histogram counts $\tilde{\mathbf{h}}$ are postprocessed to ensure non-negativity (by setting negative values to 0), and integrality (by rounding to the nearest integer) to get $\hat{\mathbf{h}}$. This is the only step that consults the private database D with ϵ_2 budget.

Sampling: The perturbed residual plot is constructed by sampling N_0 points uniformly from each grid cell based on $\hat{\mathbf{h}}$.

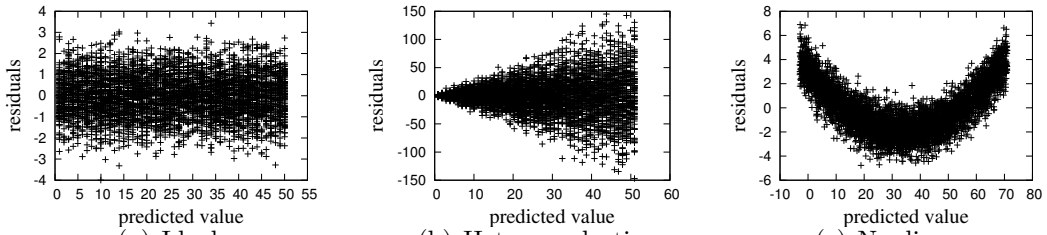
Theorem 10. PriRP *ensures ϵ-differential privacy.*

Proof. Computing bounds on $\hat{y}$ and r each consumes $\frac{\epsilon_1}{2}$ privacy budget. For the process of perturbing residual plots, the discretization and perturbation process are implemented using a ϵ_2-differentially private algorithm for computing 2D histogram of counts. The sampling step does not access the sensitive data and hence will not incur any loss of privacy. By applying sequential composition property of differential privacy, *PriRP* satisfies $\epsilon_1 + \epsilon_2 = \epsilon$-differential privacy. □

4.2.3 Evaluation

In this section, we comprehensively evaluate the quality of residual plots output by our algorithm *PriRP*, and whether or not an analyst can determine model fit using the noisy residual plots. We design the following experiments.

1. We show that noisy residual plots can help analysts correctly identify whether or not the confidential data satisfy the regression assumptions (linearity and equal variance) using synthetic datasets.



(a) Ideal (b) Heteroscedastic (c) Nonlinear

FIGURE 4.1: Residual plots for one confidential dataset randomly sampled from each of generative scenarios ($n = 5000$).

2. We investigate the degree to which linearity and equal variance must be violated before an analyst can detect the violation using *PriRP*.

3. We examine how well *PriRP* allows us to evaluate the quality of differentially private regression output.

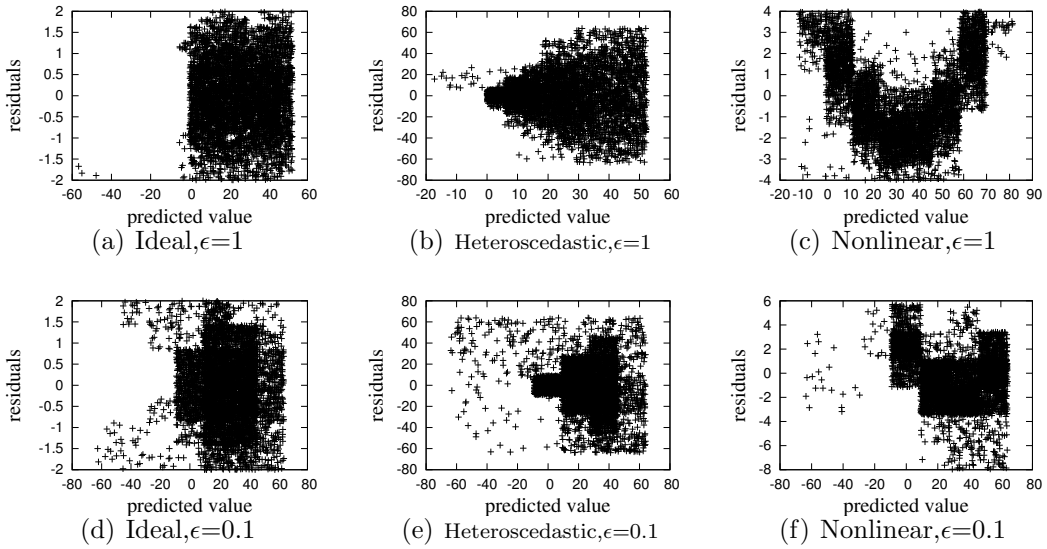
4. We present a case study of using *PriRP* to verify regression assumptions on a real dataset.

5. We evaluate how different choices of the algorithm used to perturb 2D histogram counts affect the end-to-end error of *PriRP*

Quality measures: We evaluate *PriRP* using visual comparisons of residual plots. We quantify the similarity between each perturbed residual plot (PRP) and the residual plot generated from confidential data (RP) as follows.

- Suppose RP contains points $(\hat{y}_i, r_i)$ for every record i. Let $min_x = \min\{\hat{y}_i\} - 0.1$, $max_x = \max\{\hat{y}_i\} + 0.1$, $min_y = \min\{r_i\} - 0.1$, $max_y = \max\{r_i\} + 0.1$. We discretize the area between $[min_x, max_x] \times [min_y, max_y]$ into 10×10 equal-width grid cells.

- For each grid cell c_i, where $i = 1, \ldots, 100$, we compute the percentages $P(c_i)$ and $P_p(c_i)$ that points fall in grid cell i in RP and PRP, respectively.



(a) Ideal,ϵ=1 (b) Heteroscedastic,ϵ=1 (c) Nonlinear,ϵ=1

(d) Ideal,ϵ=0.1 (e) Heteroscedastic,ϵ=0.1 (f) Nonlinear,ϵ=0.1

FIGURE 4.2: Differentially private versions of the residual plots from Figure 4.1 generated via *PriRP* with $\epsilon \in \{0.1, 1\}$.

- The similarity between RP and PRP is defined as the *total variation distance* between the two distributions:

$$Sim(RP, PRP) = \frac{1}{2}(\sum_{i=1}^{100} |P(c_i) - P_p(c_i)| + (1 - \sum_{i=1}^{100} P_p(c_i))).$$

The value of $Sim(\cdot, \cdot)$ is between [0,1]. Values close to zero indicate that the two plots have similar distributions.

Ability to verify regression assumptions

Setup: To illustrate and evaluate the performance of the *PriRP* algorithm, we generate three sets of artificial datasets. The first set, which we call "Ideal," perfectly follows the linear model assumptions; we set $y_i = x_i + \xi_i$, where $\xi_i \sim N(0, 1)$. The second set, which we call "Heteroscedastic," satisfies linearity but fails the constant variance assumption; we set $y_i = x_i + \xi_i$, where $\xi_i \sim N(0, x_i)$. The third set, which we call "Nonlinear," fails the linearity assumption; we set $y_i = 0.01 \times x_i^2 + x_i + \xi_i$, where $\xi_i \sim N(0, 1)$. We generate each x_i from independent uniform distributions

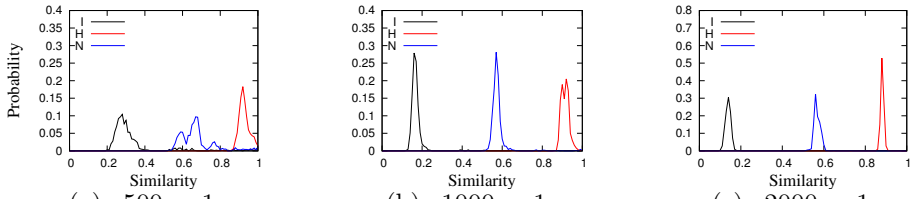
on $[1, 50]$. For each generative model, we create multiple confidential datasets of different size $n \in \{500, 1000, 2000, 5000\}$.

For each generated dataset, we use the OLS estimates, $\mathbf{b} = (b_o, b_1)$, for the linear model $y_i = \beta_0 + \beta_1 x_i + \tau_i$, where $\tau_i \sim N(0, \sigma^2)$. This model is not appropriate for the "Heteroscedastic" or "Nonlinear" generating distributions; we fit these to examine whether the private residual plots can help analysts identify the lack of fit. We use $\mathbf{b}$ to compute each value of $\hat{y}_i$ and $r_i = y_i - \hat{y}_i$, which form the inputs to *PriRP*. Using $\mathbf{b}$ facilitates evaluation of the ability of *PriRP* to provide useful diagnostic information for simulation contexts, without confounding its performance with the methods for generating a private $\mathbf{b}$.

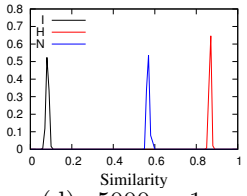
Results: Figure 4.1 displays the confidential residual plots for one dataset ($n = 5000$) randomly sampled from "Ideal", "Heteroscedastic" and "Nonlinear". The usefulness of residual plots is clearly evident: the fanning pattern in the "Heteroscedastic" plot indicates the increasing spread in y with x, and the hook pattern in the "Nonlinear" plot indicates the quadratic relationship between y and x. In contrast, the "Ideal" case shows the classic random scatter that should be present when the assumptions of the posited model fit the data well.

Figure 4.2 displays exemplary private residual plots constructed from single runs of *PriRP* on the datasets in Figure 4.1. When $\epsilon = 1$, the overall pattern in the plots is well preserved—one can easily diagnose the "heteroscedasticity" and "nonlinearity". When $\epsilon = 0.1$, although these signals are diluted, the nonlinearity continues to be diagnosable.

We now examine how well private residual plots reveal that linear model assumptions are violated when they in fact are. Rather than varying n and ϵ independently, we only vary $n \cdot \epsilon$. A number of differentially private algorithms satisfy *scale-epsilon exchangeability* [31]; i.e., increasing n and ϵ have equivalent effects on error. In particular, this is true for our bounds algorithm (see Theorem 9) as well as the perturbation



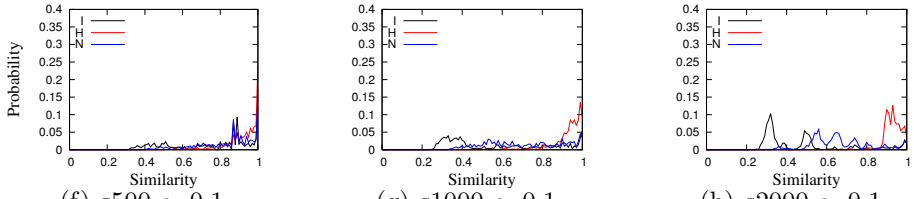
(a) s500,ϵ=1 (b) s1000,ϵ=1 (c) s2000,ϵ=1



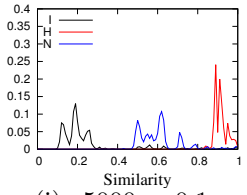
(d) s5000,ϵ=1

Scale	TVD(I,H)	TVD(I,N)
500	0.959	0.902
1000	0.997	0.996
2000	1	1
5000	1	1

(e) TVD, ϵ=1



(f) s500,ϵ=0.1 (g) s1000,ϵ=0.1 (h) s2000,ϵ=0.1



(i) s5000,ϵ=0.1

Scale	TVD(I,H)	TVD(I,N)
500	0.471	0.276
1000	0.658	0.433
2000	0.857	0.676
5000	0.974	0.908

(j) TVD, ϵ=0.1

FIGURE 4.3: Comparison of similarity values for simulated confidential and private residual plots.

algorithm DAWA [31], and is reflected in our experiments.

We compute the distribution of similarity values between the real "Ideal" residual plot and the noisy plots from different models ("Ideal", "Heteroscedastic" and "Nonlinear") using 1000 independently generated noisy plots in each case. In each plot in Figure 4.3, the distributions labeled I, H and N represent the empirical distribution of the similarity of the noisy "Ideal", "Heteroscedastic" and "Nonlinear" residual plot, respectively, with the true "Ideal" residual plot. When $n\epsilon \geqslant 1000$, H and N are well separated from I. Even at $n\epsilon = 500$, we see little overlap between H

and I as well as N and I. This means that at $n\epsilon \geqslant 500$, an analyst is still likely to correctly tell whether it is in the "Ideal" case by using the noisy residual plot.

Figure 4.3(e), 4.3(j) confirm this using total variation distance (TVD) between I vs. H and I vs. N. The TVD exceeds 0.95 when $n\epsilon \geqslant 1000$ and is $\geqslant 0.9$ when $n\epsilon \geqslant 500$.

Discriminatory power

We investigate the degree to which linearity and equal variance assumptions must be violated to be detectable using *PriRP*.

Setup: We generate data from the following three models, sampling each x_i from a uniform distribution on [0,50].

1. M_1: $y_i = x_i + \xi_i$, $\xi_i \sim N(0, 1)$
2. M_2: $y_i = x_i + \xi_i$, $\xi_i \sim N(0, \alpha x_i + 1)$
3. M_3: $y_i = x_i + \gamma x_i^2 + \xi_i$, $\xi_i \sim N(0, 1)$

When α and γ are close to zero, the violation from the standard linear model assumptions can be considered minor, so that using M_1 (instead of M_2 or M_3) is not unreasonable.

First, in the non-private setting, for any dataset generated from model $M_k, k \in \{1, 2, 3\}$ with fixed α and γ, let T_k represent the probability distribution of the counts in the 100 grid cells overlaid on the confidential residual plot. Let $D_k = \text{TVD}(T_k, E[T_1])$ be the TVD between T_k and the expected value of T_1. Using 1000 simulated datasets, we generate 1000 values of (D_1, D_2, D_3) for the specified values of α and γ.

Using these 1000 draws, we find the minimum values of α and γ such that the $\text{TVD}(D_1, D_2)$ and $\text{TVD}(D_1, D_3)$ are at least 0.95. We should be able to differentiate

Table 4.1: Discriminatory power of *PriRP*

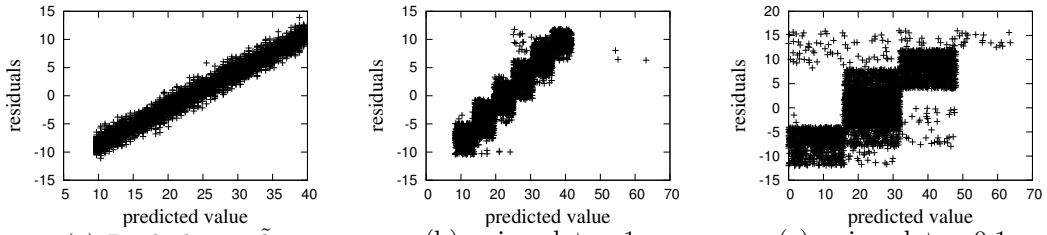
Non-Private			Private		
n	min α	min γ	$n \times \epsilon$	min α	min γ
1000	0.0097	0.000301	1000	0.02	0.002
2000	0.0067	0.000215	2000	0.01	0.0015
5000	0.0041	0.000135	5000	0.008	0.001

M_1 from M_2 or M_3 in these cases. Now we repeat this process using the noisy residual plots. For any specific α and γ, let $\tilde{T}_k$ be the probability distribution of the counts in the 100 grid cells overlaid on the noisy residual plot. Let $\tilde{D}_k = \text{TVD}(\tilde{T}_k, T_1)$. Using 1000 values of $(\tilde{D}_1, \tilde{D}_2, \tilde{D}_3)$ for the specified values of α and γ, we find the minimum values of α and γ such that the $\text{TVD}(\tilde{D}_1, \tilde{D}_2)$ and $\text{TVD}(\tilde{D}_1, \tilde{D}_3)$ are $\geqslant 0.95$.

Results: The first panel in Table **??** displays the minimum α and γ at which violation of model assumptions can be detected using the confidential plots for increasing values of n. As n increases, one can detect violations even at smaller degrees (i.e., smaller α and γ). The second panel of Table **??** represents the minimum α and γ at which residual plots output by *PriRP* can detect violations. While adding noise to the residuals makes it harder to detect violations, we still are able to detect even small violations (i.e., $\alpha = 0.02$ and $\gamma = 0.002$) from the linear regression assumptions when $n \cdot \epsilon \geqslant 1000$.

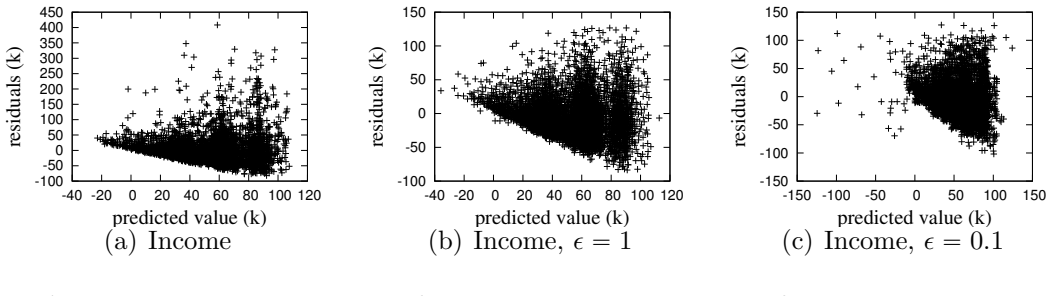
Illustration with differentially private model

We now consider evaluating a noisy model, e.g. noisy coefficients $\tilde{\mathbf{b}}$, obtained from a differentially private regression algorithm. There is a key difference in the interpretation of the noisy plots. The residual plots reflect lack of fit potentially from two sources. First, the posited linear model may be a poor fit to D, even in the fortunate case where $\tilde{\mathbf{b}} \approx \mathbf{b}$ (e.g., with large scale). Second, $\tilde{\mathbf{b}}$ may be a poor approximation of $\mathbf{b}$, even when the posited linear model would fit reasonably well if estimated on the confidential data. As such, the noisy residual plot based on $\tilde{\mathbf{b}}$ provides an omnibus



(a) Real plot on $\tilde{b}$ (b) noisy plot,ϵ=1 (c) noisy plot,ϵ=0.1

FIGURE 4.4: Residual plots based on private model $\tilde{b}$ with $n = 5000$



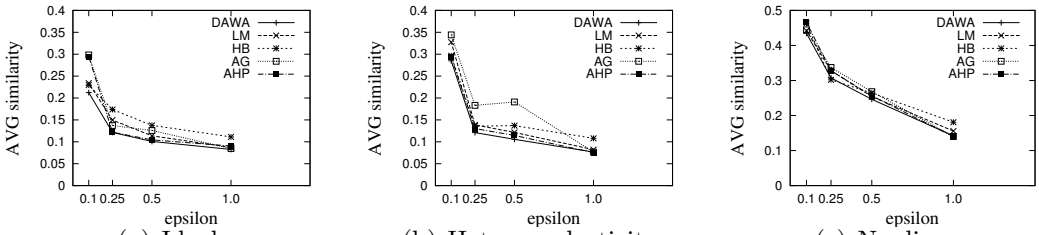
(a) Income (b) Income, $\epsilon = 1$ (c) Income, $\epsilon = 0.1$

(d) log(Income) (e) log(Inc), $\epsilon = 1$ (f) log(Inc), ϵ=0.1

FIGURE 4.5: Residual plots for a linear regression of income and log(income) on several explanatory variables for 2000 Current Population Survey data.

assessment of the quality of the model and the approximation $\tilde{\mathbf{b}}$, as opposed to a focused check of the regression model assumptions.

Figure 4.4 shows an example of evaluating differentially private model output. We use a state of the art algorithm that applies output perturbation [59] to compute the differentially private linear model coefficients $\tilde{b}$ with $\epsilon = 1$ on the "Ideal" dataset. The pattern is still evident in the noisy plots (Figure 4.4(b), 4.4(c)).



(a) Ideal (b) Heteroscedasticity (c) Nonlinear

FIGURE 4.6: Performance on different choice of perturbation algorithm

Table 4.2: Average Regret in terms of Similarity Values

Perturb. Alg.	DAWA	AHP	LM	HB	AG	Uniform	MWEM
Regret	1.00395	1.0756	1.108	1.212	1.215	5.238	5.258

Illustration with real data

We illustrate the use of *PriRP* on a subset of data from the March 2000 Current Population Survey public use file released by the U.S. Census Bureau. The data comprise $n = 49436$ heads of households, each with non-negative income. We seek to predict household income from age in years, education (16 levels), marital status (7 levels), and sex (2 levels). We start by fitting the OLS model of income on main effects only for each variable, using indicator variable coding for the categorical variables. As evident in Figure 4.5(a), the residual plot based on the confidential data reveals an obvious fan-shaped pattern, reflecting non-constant variance. This pattern is also evident in the noisy plots (Figures 4.5(b), 4.5(c)). To remedy the non-constant variance, we instead fit the regression using the natural logarithm of income as the outcome variable. As evident in Figure 4.5(d), this transformation has made the constant variance assumption more believable (as it often does). The noisy residual plots also reveal the improvement in model fit (Figures 4.5(e), 4.5(f)).

Choice of perturbation algorithm

Finally, we evaluate how different choices of perturbation algorithms in the second step of *PriRP* for perturbing residuals affect the performance of *PriRP*.

The problem of releasing the joint probability density over 2 dimensions has received a lot of attention in the differential privacy literature. Table 4.2 lists a set of algorithms which can be used to privately compute 2D densities (as well as answer range queries over the 2D domain). *LM* represents *Laplace Mechanism*. *Uniform* only perturbs the total count and assumes uniform distribution on the domain. *HB* [**?**] uses noisy estimates of multiple hierarchical equiwidth histograms (i.e., discretized at different grid sizes). *DAWA* [38], *AHP* [66] and *AG* [47] partition the space and then use the *Laplace Mechanism* to estimate the fraction of points in each partition. *MWEM* [29] assumes uniform distribution on the domain first and then updates the distribution by sampling the querie which has the poorest answer based on current distribution.

Figure 4.6 shows the comparison results on three different datasets ("Ideal", "Heteroscedasticity" and "Nonlinear") with scale = 5000. In each figure, we compare the average similarity value among 20 trials of using *PriRP* with different 2D perturbation algorithms on various ϵ settings. *Uniform* and *MWEM* are not shown because their results are much worse than the 5 algorithms shown in the figures. From Figure 4.6, we can see *DAWA* performs the best in terms of all datasets and ϵ settings.

The performance of algorithms across different datasets and ϵ values can be summarized using a measure called *regret*, introduced in prior works [31] and [35]. Let $\mathbf{M}$ be a set of algorithms, and let $\mathbf{S}$ be a set of pairs of (dataset, ϵ). Then, the *regret* of algorithm $M^* \in \mathbf{M}$ on a specific dataset D at a specific ϵ is defined as

$$regret(M^*) = \frac{Error(M^*(D,\epsilon))}{\min_{M \in \mathbf{M}}\{M(D,\epsilon)\}}. \tag{4.2}$$

The (dataset, ϵ) specific *regret* measure computes how much worse algorithm M^*'s error is compared to the error of the algorithm with the least error on that dataset at that ϵ. The average *regret* of an algorithm M^* on a set $\mathbf{S}$ is given by

$$regret(M^*) = avg_{(D,\epsilon)\in S}\frac{Error(M^*(D,\epsilon))}{\min_{M\in\mathbf{M}}\{M(D,\epsilon)\}}. \tag{4.3}$$

The *regret* value is greater or equal to 1. The smaller the *regret* is, the better the algorithm performs. *Regret* equals 1 means the corresponding algorithm always performs the best in terms of all (dataset, ϵ) settings. Table 4.2 validates our choice of *DAWA* as the algorithm of choice in *PriRP*, as it has the least *regret* amongst all the algorithms we considered.

4.3 ROC Curves

We now present an algorithm of computing differentially private ROC curves for logistic regression. The algorithm could be used to evaluate the predictive power of any binary classifier.

4.3.1 Review of ROC curves

Let $D_{raw} = \{(y_i, \mathbf{x}_i) : i \in [n]\}$ be a confidential dataset. The logistic regression model posits $P(y_i = 1 \mid \mathbf{x}_i) = \pi_i$, where $\log(\frac{\pi_i}{1-\pi_i}) = \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j$. Let $\mathbf{b} = (b_0, \ldots, b_p)$ be the maximum likelihood estimate of $\beta = (\beta_0, \ldots, \beta_p)$. For any $\mathbf{x}_i$, we compute predicted probabilities,

$$p(\mathbf{x}_i) \;=\; P(y_i = 1 \mid \mathbf{x}_i) \;=\; \frac{exp(b_0 + \sum_{j=1}^{p} x_{ij}b_j)}{1 + exp(b_0 + \sum_{j=1}^{p} x_{ij}b_j)}. \tag{4.4}$$

Analysts often evaluate the fit of a logistic regression model by comparing $p(\mathbf{x}_i)$ to y_i. Ideally, $p(\mathbf{x}_i)$ tends to be large when $y_i = 1$, and tends to be small when $y_i = 0$. To facilitate these comparisons, it is convenient to convert $p(\mathbf{x}_i)$ into a predicted

outcome, $\hat{y}_i \in \{0, 1\}$, by setting $\hat{y}_i = 1$ when $p(\mathbf{x}_i) > \theta$ and $\hat{y}_i = 0$ otherwise. Here, θ is an analyst-defined threshold.

For any given θ, using all n values of $(y_i, \hat{y}_i)$, we can quantify the accuracy of the logistic regression model on a test set D as follows. *True positives*, $TP(\theta)$, are the individuals in D with true and predicted outcomes equal to 1, i.e., $(y_i = 1, \hat{y}_i = 1)$. *False positives*, $FP(\theta)$, are individuals with $(y_i = 0, \hat{y}_i = 1)$. We use the notation $TP(\theta), FP(\theta)$ to denote both the set of individuals and the cardinality of them.

Let n_1 and n_0 be the number of individuals with $y_i = 1$ and $y_i = 0$, respectively. The true-positive rate $TPR(\theta)$ is the probability that an individual in D with $y_i = 1$ is correctly classified with $\hat{y}_i = 1$. The false-positive rate $FPR(\theta)$, is the probability that an individual in D with $y_i = 0$ is wrongly classified with $\hat{y}_i = 1$. Thus, we have

$$TPR(\theta) = \frac{TP(\theta)}{n_1} \quad \text{and} \quad FPR(\theta) = \frac{FP(\theta)}{n_0}. \tag{4.5}$$

The ROC curve is defined by plotting $TPR(\theta)$ versus $FPR(\theta)$ over all possible θ. It starts at $(0, 0)$ for $\theta = 1$ and ends at $(1, 1)$ for $\theta = 0$. It can be approximated numerically by computing Equation 1 over a large set of candidate θ values, say Θ. The area under the ROC curve, abbreviated as AUC, is often used to evaluate the accuracy of the logistic regression. When the regression predicts the outcomes in D accurately, the ROC curve is close to the left and upper boundary, so that AUC is close to 1. When the regression predicts the outcomes in D poorly, the ROC curve is close to the 45° line from (0,0) to (1,1), and the AUC is around 0.5.

Recent work [43] shows that releasing actual ROC curves computed directly from Equation 4.5 can allow attackers with prior knowledge to reconstruct D. For an extreme yet illustrative example, suppose an attacker knows all of D except for $(y_i, \mathbf{x}_i)$. Given the ROC curve based on Equation 4.5, the attacker can determine the unknown y_i by simply enumerating over all values of $(y_i, \mathbf{x}_i)$, and finding the value that reproduces the given ROC curve. Hence, directly releasing the ROC curve may

Algorithm 11 PriROC

Input: Dataset D, a safe model β, ϵ
Output: a private ROC curve

1: Step 1: Select a set of thresholds in terms of predictons of D on β with ϵ_1 privacy budget.
2: Step 2: Compute the private TPR and FPR values based on the selected thresholds above with the left $\epsilon_2 = \epsilon - \epsilon_1$ budget.
3: Step 3: Post-process private TPR and FPR values to make the private ROC curve consistent with the monotonicity of real ROC curves.

leak information, inspiring us to create a differentially private method for generating ROC curves.

4.3.2 Private ROC curves

To generate ROC curves, we need to determine Θ, to compute TPR and FPR values on all chosen thresholds, and to ensure the monotonicity of TPR and FPR values for a valid ROC curve. Under differential privacy, we cannot directly use $\Theta = \{p(\mathbf{x}_i) : \mathbf{x}_i \in D, i \in [n]\}$ (the predicted probabilities for each individual). Simply using LM to compute TPR and FPR values may introduce large noise due to its high sensitivity $(2|\Theta| + 1)$.

We propose an algorithm *PriROC* (Algorithm 11) that addresses these concerns in three steps: 1) Select a set of thresholds that are used for computing the TPR and FPR values with ϵ_1 privacy budget. 2) Compute the private TPR and FPR values in terms of all the chosen thresholds with the left privacy budget $\epsilon_2 = \epsilon - \epsilon_1$. 3) Post-process the results in order to make it a valid ROC curve. We next describe these steps in details.

Choosing thresholds: There are two important considerations when choosing Θ. The number of thresholds must (a) be large enough to accurately approximate the ROC curve, and (b) be small enough to ensure accuracy (despite perturbation) and efficiency (as we will see later, the runtime of our algorithm is $O(|\Theta|^2 log|\Theta|)$).

A data-independent strategy for picking Θ is to choose the thresholds uniformly from $[0, 1]$. More precisely, if we fix $|\Theta| = N$, we set $\Theta = \{0, \frac{1}{N}, \ldots, \frac{N-1}{N}, 1\}$. We

call this strategy N-FixedSpace. This strategy works well when the predictions $P = \{p(\mathbf{x}_i)|\mathbf{x}_i \in D, i \in [n]\}$ are uniformly spread out in $[0, 1]$. Since, N-FixedSpace is data independent, $\epsilon_1 = 0$, and all the privacy budget can be used for computing the TPR and FPR values. However, N-FixedSpace may be ineffective when predictions are skewed. For instance, suppose all the predictions $p(\mathbf{x}_i)$ are either $< \frac{1}{N}$, or $> 1 - \frac{1}{N}$. The ROC curve is then approximated with just 2 points ($TPR(\frac{1}{N})$, $FPR(\frac{1}{N})$) and ($TPR(1 - \frac{1}{N})$, $FPR(1 - \frac{1}{N})$), possibly resulting in a significant loss in accuracy in the AUC.

For this potential shortcoming, we present s-RecursiveMedians (Algorithm 12), a data-dependent strategy that recursively selects thresholds by finding the median among the current data points such that there are roughly the same number of individuals between any two thresholds. Algorithm 12 takes as input the privacy budget for choosing thresholds ϵ_1, depth of the recursion s, and the multiset of predictions P. Each of the s recursive steps uses a privacy budget of $\gamma = \epsilon_1/s$.

The algorithm recursively calls a subroutine FindMedians to compute the noisy median of all predictions within the range (ℓ, r). Initially, $(\ell, r) = (0, 1)$. Since the median function f_{med} has a high global sensitivity (equal to $r - \ell$), we use the smooth sensitivity framework [45] to compute the noisy median. In Algorithm 12, $S^*_{f_{med},\gamma/2}(P)$ represents the smooth sensitivity of f_{med} on P with privacy budget $\gamma/2$. When the global sensitivity of a function $\Delta(f)$ is high, we can use a data dependent sensitivity function $S^*_{f,\epsilon}(D)$ instead to ensure privacy. $S^*_{f,\epsilon}(D)$ is an upper bound on the (local) sensitivity of f at the input database (i.e. the maximum change in function output on between D and its neighbors) and is smooth (i.e., $S^*_{f,\epsilon}(D)$ is close to $S^*_{f,\epsilon}(D')$, when D and D' are neighbors). Given $S^*_{f,\epsilon}(\cdot)$, one way to ensure ϵ-differential privacy is by perturbing the true output $f(D)$ using $\frac{8}{\epsilon} \cdot S^*_{f,\epsilon}(D) \cdot \eta$, where η is random noise sampled from the distribution $\propto 1/(1 + |\eta|^2)$. We refer the reader

Algorithm 12 s-RecursiveMedians

function s-RecursiveMedians(P, ϵ_1, s)
  $\gamma \leftarrow \frac{\epsilon_1}{s}$
  return FindMedians($P, \gamma, s, 0, 1$)
end function
function FindMedians(P, γ, s, ℓ, r)
  if $s = 0$ **then return** $\varnothing$
  end if
  $m \leftarrow median(P)$
  $\tilde{m} \leftarrow m + \frac{8S^*_{f_{med},\gamma/2}(P)}{\gamma/2} * \eta$, η is random noise $\propto \frac{1}{1+\eta^2}$
  if $\tilde{m} \leqslant \ell$ or $\tilde{m} \geqslant r$ **then** $\tilde{m} \leftarrow (\ell + r)/2$
  end if
  $P_1 \leftarrow \{P[i] \mid P[i] < \tilde{m}\}$
  $P_2 \leftarrow \{P[i] \mid P[i] > \tilde{m}\}$
  return FindMedians($P_1, \gamma, s-1, \ell, \tilde{m}$) $\cup$ $[\tilde{m}]$ $\cup$ FindMedians($P_2, \gamma, s-1, \tilde{m}, r$)
end function

to [45] for proofs and details.

For the median function f_{med}, the smooth sensitivity $S^*_{f_{med},\epsilon}(P)$ can be computed as [45]:

$$S^*_{f_{med},\epsilon}(P) = \max_{k=0,\ldots,n} (e^{-k\epsilon} \cdot \max_{t=0,\ldots,k+1} (P[m+t] - P[m+t-k-1])) \tag{4.6}$$

where $P[t]$ is the t^{th} value in P when sorted in increasing order. We generate samples η by drawing U uniformly from $(0, 1)$ and computing $tan(\pi(U - 0.5))$ (as the CDF of the noise distribution is proportional to $arctan(\eta)$).

When the resulting noisy median $\tilde{m}$ falls out of the current range (ℓ, r), we directly use the middle point (e.g., $(\ell + r)/2$) as the partition point (as well as one chosen threshold). The smooth sensitivity of the median query is higher when the points are uniformly distributed in an interval than when they are skewed. A noisy median $\tilde{m}$ out of range indicates that the sensitivity is large when compared to the interval size, and hence picking the midpoint is a good choice. The algorithm completes after s levels of recursions, and the number of chosen thresholds by s-RecursiveMedians is 2^s.

Theorem 11. *Algorithm 12 (s-RecursiveMedians) satisfies ϵ_1-differential privacy.*

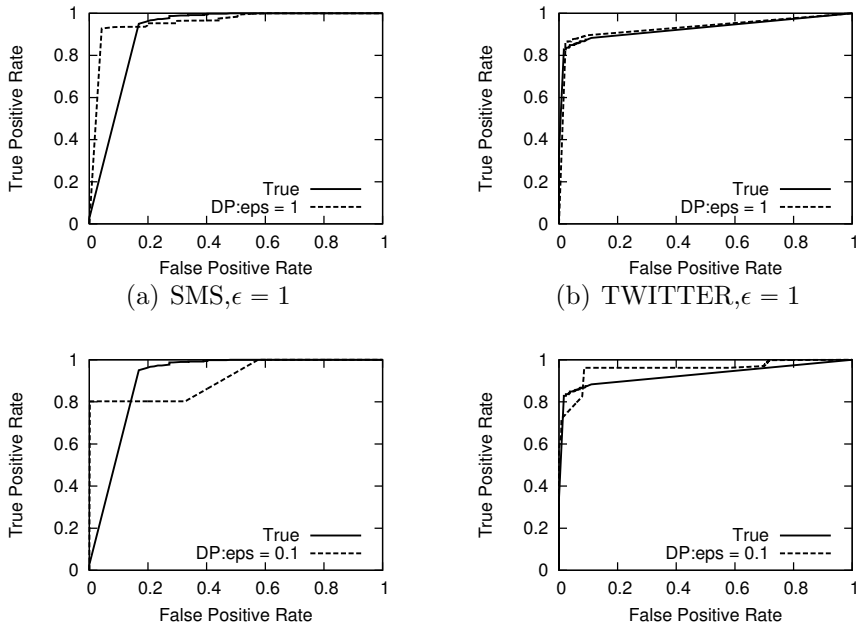
Proof. Computing the median of a set of points by using FindMedians satisfies $\gamma/2 = \epsilon_1/2s$-differential privacy. In each recursive step, there are at most two partitions different in neighboring databases. Since the depth of the recursion is bounded by s, s-RecursiveMedians satisfies $\frac{\epsilon_1}{2s} \times 2 \times s = \epsilon_1$-differential privacy by sequential composition. □

Computing noisy TPR and FPR values: Suppose we are given a set of thresholds $\Theta = \{\theta_1, \ldots, \theta_\ell\}$ to use for approximating the ROC curve, where $\theta_k > \theta_{k+1}$ for all k, $\theta_0 = 1$, and $\theta_\ell = 0$. By definition, $TP(\theta_\ell) = n_1$ and $FP(\theta_\ell) = n_0$. The sets $\{TP(\theta_k) : k \in [\ell]\}$ and $\{FP(\theta_k) : k \in [\ell]\}$ each correspond to a set of prefix range queries.

Let $R^{TP}_\Theta = \{r^{TP}_1, \ldots, r^{TP}_\ell\}$, where r^{TP}_k is the number of individuals $i \in D$ with $y_i = 1$ and $\theta_{k-1} \geqslant p(\mathbf{x}_i) > \theta_k$. Clearly, $TP(\theta_k)$ is the sum of the first k elements in R^{TP}_Θ. We can define R^{FP}_Θ similarly, thereby showing each $FP(\theta_k)$ is also a prefix range query on R^{FP}_Θ.

A recent experimental study of differentially private algorithms for answering range queries [31] indicates that *DAWA* [38] is a good algorithm for answering prefix range queries. *DAWA* additionally approximates the histograms R^{TP}_Θ and R^{FP}_Θ by grouping together cells with similar counts. We compute the TP and FP values privately using half of the remaining privacy budget for each one. Prefix range queries are computed on R^{TP}_Θ to get the TP values and on R^{FP}_Θ to get the FP values, which in turn are used to construct the noisy $TPR(\theta)$ and $FPR(\theta)$ values. Since all subsequent steps do not use the original data, the fact that releasing $TPR(\theta)$ and $FPR(\theta)$ satisfies ϵ-differential privacy follows from the privacy of *DAWA*. We spend the remaining $\epsilon_2 = \epsilon - \epsilon_1$ budget on computing private TPR and FPR values, $\frac{\epsilon_2}{2}$ for each.

Ensuring monotonicity: The noisy TPR and FPR values generated using the al-



(a) SMS,$\epsilon = 1$ (b) TWITTER,$\epsilon = 1$

(c) SMS,$\epsilon = 0.1$ (d) TWITTER,$\epsilon = 0.1$

FIGURE 4.7: Confidential and differentially private ROC curves using *PriROC* at $\epsilon = 1$ (top) and $\epsilon = 0.1$ (bottom).

gorithms above are not guaranteed to satisfy monotonicity. We leverage the ordering constraint between the TPR and FPR values to boost accuracy by using isotonic regression [30]. Since this is a post-processing step, there is no impact on privacy.

Theorem 12. PriROC *satisfies ϵ-differential privacy.*

Proof. ϵ_1 budget is spent on computing thresholds. Theorem 12 proves it ensures ϵ_1-differential privacy. Any existing private 1D perturbation algorithms to compute private TPR and FPR values with $\frac{\epsilon_2}{2}$ budget will satisfy $\frac{\epsilon_2}{2}$-differential privacy for each value. The post-processing step for ensuring monotonicity will not incur extra loss of privacy. By using sequential composition property of differential privacy, *PriROC* ensures $\epsilon_1 + \frac{\epsilon_2}{2} \times 2 = \epsilon$-differential privacy. □

4.3.3 Evaluation

We now evaluate the quality of the ROC curves output by *PriROC*. Our experiments focus on:

1. Visually inspecting whether the ROC curves output by *PriROC* are close to the true ROC curves generated using the confidential predictions.

2. Measuring the average absolute difference between the AUC of the true ROC curve and the AUC of noisy ROC curves output by *PriROC*.

3. Measuring the similarity between true and noisy ROC curves by computing symmetric difference between them.

4. Quantifying the discriminatory power of private ROC curves; i.e., whether they can discriminate between "good" and "bad" classifiers.

5. Evaluating the sensitivity of *PriROC*'s performance to changing the amount of privacy budget allocated to choosing thresholds (ϵ_1) and estimating counts (ϵ_2).

6. Evaluating the sensitivity of *PriROC*'s performance to the choice of the algorithm used to compute noisy TPR and FPR values.

Dataset: We use two text classification datasets - RAW-TWITTER and RAW-SMS. The RAW-TWITTER dataset [27] was collected for the task of sentiment classification. Each tweet is associated with a binary sentiment outcome – positive or negative. The dataset contains 1.6 million tweets from which we randomly sampled 6840 tweets for our experiments. The RAW-SMS dataset [2] contains 5574 SMS messages associated with spam/ham outcome variable. For both datasets, we use a vector of binary explanatory variables representing presence or absence of individual words (excluding stop words).

RAW-TWITTER and RAW-SMS are randomly divided into training sets (containing 90% records) and test sets (containing 10% records). We use differentially private empirical risk minimization [13] to compute a logistic regression model on the training set. We apply the private model on the two test datasets to obtain predictions, which we call SMS and TWITTER. SMS contains 558 records, and each record has a true label $y_i \in \{0, 1\}$ as well as a prediction $P(y_i = 1 \mid \mathbf{x}_i) \in [0, 1]$. In SMS, 481 out of 558 records have label 1. TWITTER contains 684 records, 385 of which have label 1.

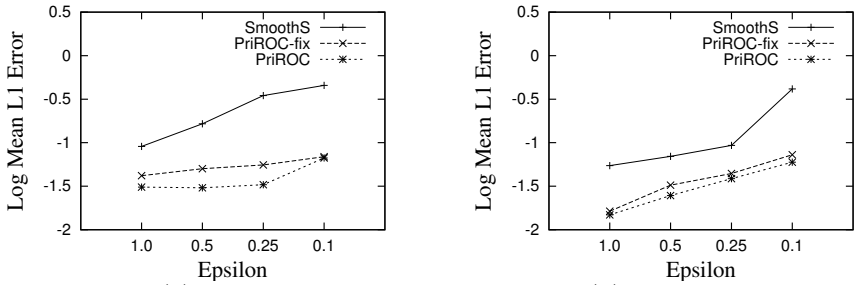
Algorithms: We consider our proposed two strategies for selecting Θ. The first uses the s-RECURSIVEMEDIANS strategy, setting $|\Theta| = 2^{10}$ and spending 0.2ϵ budget to select thresholds. These parameters were empirically chosen (we fix them in our experiment) to trade off error with efficiency, which is not shown in the paper. We refer to the resulting private algorithm for ROC curves as *PriROC*. The second strategy for selecting Θ uses fixed thresholds selected uniformly from $[0, 1]$. Here, we again set $|\Theta| = 2^{10}$, and we dedicate the full ϵ to perturb the TPR and FPR values. We call the resulting differentially private algorithm for ROC curves *PriROC-fix*.

For both *PriROC* and *PriROC-fix*, we apply many different algorithms to perturb the TPR and FPR values. In section 4.3.6, we will show that *DAWA* is the best perturbation algorithm in terms of both datasets and different ϵ settings. So we will use *DAWA* as the default perturbation algorithm in *PriROC*.

We also compare with Boyd et al's [5] algorithm for directly computing the AUC, and call that *SmoothS*.

Visualizing ROC curves

Figure 4.7 displays the ROC curves computed on confidential data and one randomly generated differentially private ROC curve at $\epsilon = 1$ and 0.1 using *PriROC* on both SMS and TWITTER. The *PriROC* curves track the confidential ROC curves quite



(a) SMS (b) TWITTER

FIGURE 4.8: Comparison of AUC error.

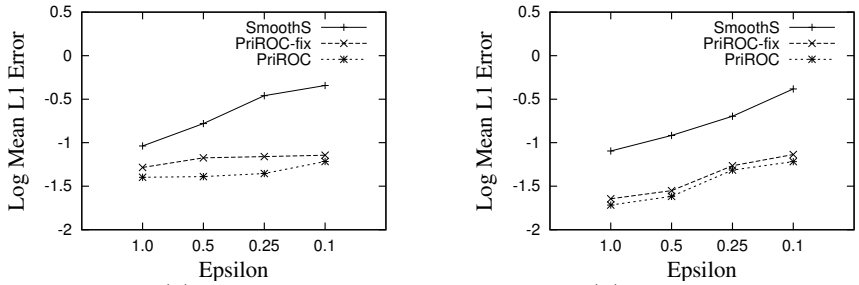
well for all datasets and ϵ settings. The reason why *PriROC* is more accurate for TWITTER than for SMS is that the prediction distribution of SMS is more skewed than TWITTER, so s-RECURSIVEMEDIANS performs better on TWITTER.

AUC error

Figure 4.8 reports the comparison of AUC error for *PriROC*, *PriROC-fix* and *SmoothS* on both SMS (Figure 4.8(a)) and TWITTER (Figure 4.8(b)). For each figure, the x-axis represents different ϵ settings and the y-axis shows the $\log_{10}$ of the mean L_1 error between the confidential AUC and the private AUC over 20 repetitions. Both *PriROC* and *PriROC-fix* have significantly lower errors than *SmoothS* under both datasets and all ϵ settings. Using s-RECURSIVEMEDIANS to choose thresholds results in better AUC accuracy compared to using fixed thresholds.

ROC similarity

Even when two ROC curves have the same AUC, they may look quite different. Here, we directly examine how close the real and private ROC curves are by computing the symmetric difference between them; i.e., the mean area between the true and private curves. We compare three methods: *PriROC*, *PriROC-fix*, and *SmoothS* (*SmoothS* computes data independent, symmetric binormal ROC curves [5]). Figure 4.9 displays the symmetric differences (averaged over 20 trials) between the confidential



(a) SMS (b) TWITTER

FIGURE 4.9: Comparison of symmetric difference between confidential and private ROC curves.

Table 4.3: Discriminatory Power of *PriROC*

Scale $\times$ ϵ	Discriminatory Power	Max p-value
200	0.1	3.7e-3
500	0.05	1.31e-2
1000	0.025	3.29e-3
2000	0.01	1.58e-5

ROC curves and differentially private ROC curves for the three different algorithms on both SMS (Figure 4.9(a)) and TWITTER (Figure 4.9(b)). In each figure, the x-axis represents different ϵ settings, and the y-axis shows the $\log_{10}$ of the mean symmetric difference between the confidential and private ROC curves over 20 repetitions. Both *PriROC* and *PriROC-fix* generate private ROC curves with much lower symmetric differences than those for *SmoothS*. Once again, *PriROC* offers slight gains over *PriROC-fix*.

Discriminatory power

An evaluation methodology is useless if it can't discriminate "good" regression models (or binary classifiers) that have high AUCs from "bad" models that have low AUCs on a given test set. To quantify this discriminatory power, we use synthetic datasets with different AUCs. We generate a pair of synthetic ROC curves R_a and $R_{a+\delta}$ with AUCs a and $a+\delta$, respectively. We construct noisy ROC curves for both R_a and $R_{a+\delta}$ 20 times each, and compute their AUCs. We perform two sample t-tests

Table 4.4: Average Regret in terms of AUC Error and Symmetric Difference

Perturbation Algorithm	DAWA	AHP	HB	LM	MWEM	Uniform
AUC Regret	1	2.126	2.634	3.032	14.472	14.543
SD Regret	1	1.745	2.071	2.605	10.403	10.443

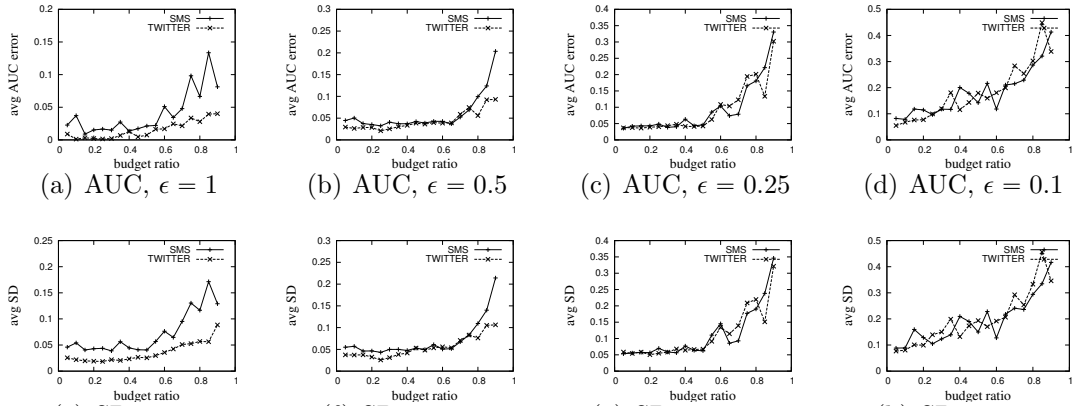
(between noisy AUCs generated from R_a and $R_{a+\delta}$, respectively) and test whether the mean noisy AUC generated from R_a is significantly different (p-value < 0.05) from the mean noisy AUC generated from $R_{a+\delta}$. We say the noisy AUC computed using *PriROC* at data size n and privacy ϵ has discriminatory power δ, if $\forall a$, the noisy AUC values for R_a have a significantly different mean than noisy AUC values for $R_{a+\delta}$.

Table 4.3 displays the discriminatory power of private ROC curves. We generate ROC curves with AUC $a \in A = \{0.95, 0.925, \ldots, 0.7\}$. We report the smallest δ (in multiples of 0.025) at which noisy AUCs have significantly different means for R_a and $R_{a+\delta}$ for $a, a+\delta \in A$. We report the max p-value attained for all such comparisons R_a and $R_{a+\delta}$. Smaller p-values imply the AUC computed using *PriROC* are more discriminating between R_a and $R_{a+\delta}$. Larger $n \cdot \epsilon$ products allow us to discriminate between ROC curves with smaller differences in AUC with more significance.

In summary, ROC curves computed by *PriROC* satisfy ϵ-differential privacy, and closely match the true ROC curves both visually and in terms of qualitative measures like AUC and symmetric difference. Moreover, for $n \cdot \epsilon \geqslant 1000$, private ROC curves can discriminate between "good" and "bad" models even if they differ in AUC by only 0.025.

Sensitivity to budget split

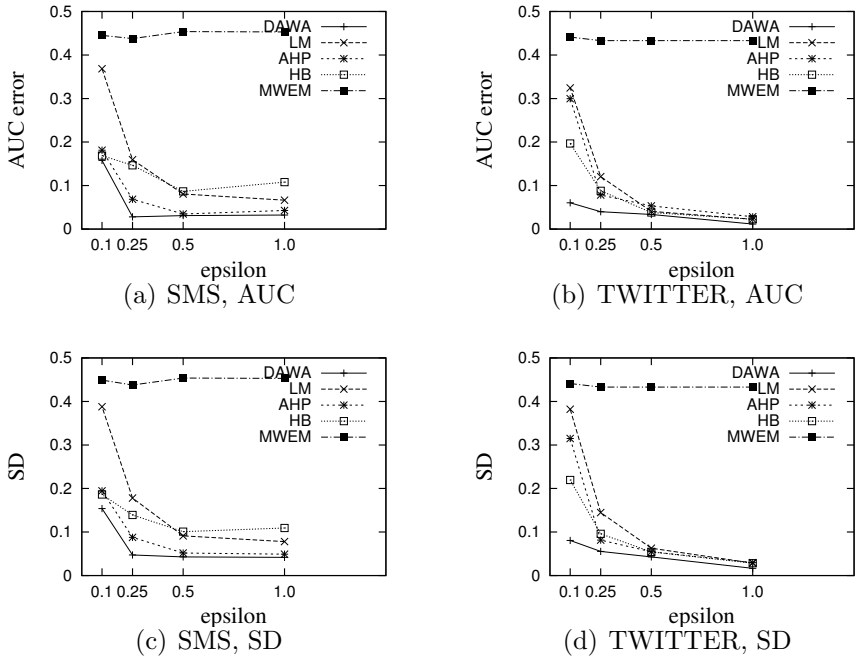
We evaluate the effect of budget split on the performance of *PriROC*. Recall that we split the total privacy budget ϵ into two parts - $\epsilon_1 = \rho\epsilon$ for choosing the set of thresholds, and $\epsilon_2 = (1-\rho)\epsilon$ for computing noisy TPR and FPR values. We call ρ



(a) AUC, $\epsilon = 1$ (b) AUC, $\epsilon = 0.5$ (c) AUC, $\epsilon = 0.25$ (d) AUC, $\epsilon = 0.1$

(e) SD, $\epsilon = 1$ (f) SD, $\epsilon = 0.5$ (g) SD, $\epsilon = 0.25$ (h) SD, $\epsilon = 0.1$

FIGURE 4.10: Effect of Budget Split in terms of AUC error and Symmetric Difference



(a) SMS, AUC (b) TWITTER, AUC

(c) SMS, SD (d) TWITTER, SD

FIGURE 4.11: Performance on different choice of perturbation algorithm

the budget ratio. When the budget ratio ρ is high (close to 1), most of the privacy budget is spent on choosing thresholds resulting in more noise being added to TPR and FPR values. One the other hand, a small budget ratio (close to 0) would result in more noise added to the first step, which may result in thresholds that do not adequately capture the distribution of the TPR and FPR values. Figure 4.10 shows the results in terms of both AUC error and symmetric difference for different budget ratio settings. Each figure includes two curves for SMS and Twitter. Every point reports the average value of 20 trials. We can observe that for all settings of ϵ, large ρ values close to 1 result in an inordinate loss in performance, whereas the performance is relatively stable for a wider range of smaller ρ values. $\rho = 0.2$ is good choice across datasets and across different ϵ values.

Sensitivity to choice of perturbation algorithm

Finally, we evaluate how the choice of perturbation algorithm for perturbing TPR and FPR values affects the performance of *PriROC*. Table 4.4 lists a set of algorithms for perturbing TPR and FPR values. Figure 4.11 shows the results for both datasets (SMS and Twitter) on both error metrics (AUC error and symmetric difference). In each figure, we compare the average corresponding value among 20 trials of using *PriROC* with different perturbation algorithms for computing private FPR and TPR values on various ϵ settings. *Uniform* is not shown in the figures since it works even worse than *MWEM*. From figure 4.11, we can see *DAWA* performs the best in terms of all datasets and ϵ settings. Table 4.4 also displays the optimal choice of *DAWA* by computing the regret in terms of AUC error and symmetric difference for different algorithms among all (dataset, ϵ) settings. The regret of *DAWA* for both AUC error and symmetric difference is 1, which shows the consistent out-performance of choosing *DAWA* as perturbation algorithm in *PriROC*.

4.4 Binned Residual Plot

We now present the first differentially private algorithm for computing binned residual plots for analyzing model fit of logistic regression.

4.4.1 Review of Binned Residual Plots

Let $D_{raw} = \{(y_i, \mathbf{x}_i)) : i \in [n]\}$ be the confidential dataset. As reviewed (in section 4.3.1), in logistic regression, the prediction of each y_i can be computed as

$$p(\mathbf{x}_i) \;=\; P(y_i = 1 \mid \mathbf{x}_i) \;=\; \frac{exp(b_0 + \sum_{j=1}^{p} x_{ij} b_j)}{1 + exp(b_0 + \sum_{j=1}^{p} x_{ij} b_j)}, \tag{4.7}$$

where $\mathbf{b} = (b_0, \ldots, b_p)$ is the maximum likelihood estimate of the coefficients.

The residual of each point is $r_i = y_i - p(\mathbf{x}_i)$. Hence, $r_i \geqslant 0$ when $y_i = 1$ and $r_i \leqslant 0$ when $y_i = 0$. As such, a plot of residuals versus predicted values is not especially informative. It appears as two straight lines, one with non-negative values for cases with $y_i = 1$ and one with non-positive values for cases with $y_i = 0$.

Instead, we use binned residual plots to examine the fit of logistic regression. To make the binned residual plot, the data are split into several bins of roughly equal sample size based on the predictions $p(\mathbf{x}_i)$. Within each bin, the average value of $p(\mathbf{x}_i)$ and average value of r_i are computed. We then plot the average residuals versus the average predicted values across all bins.

A good fit of logistic regression in terms of binned residual plot shows a pattern of randomly scattered set of points (see Figure 4.12(a)). When $\log(\frac{p(\mathbf{x}_i)}{1-p(\mathbf{x}_i)})$ is not linear in terms of $\mathbf{x}_i$, the binned residual plot may show systematic curvatures or bins with extreme average residuals (see Figure 4.12(b)).

Again, when D is confidential, we cannot directly release the true binned residual plot. If an attacker knows the real binned residual plot and all other input data except one record, the attacker can reconstruct the missing record by using the

Algorithm 13 PriBRP

Input: Dataset D, a safe model β, ϵ
Output: a private binned residual plot

1: Step 1: Divide D into bins with roughly equal sample size based on the predicted values using β with ϵ_1 privacy budget.
2: Step 2: Adjust bins above by combining bins of small size with ϵ_2 privacy budget.
3: Step 3: Compute private average residuals and average predicted values across all bins with the left $\epsilon_3 = \epsilon - \epsilon_1 - \epsilon_2$ budget.

Algorithm 14 Private Bins Adjustment

Input: Bins $B = \{q_1, \ldots, q_m\}$, private budget ϵ, threshold θ
Output: New bins $\tilde{B}$

1: $\tilde{\theta} \leftarrow \theta + Lap(2/\epsilon)$
2: $i \leftarrow 1$, $q \leftarrow \varnothing$, $\tilde{B} \leftarrow []$
3: **while** $i \leqslant m$ **do**
4: $\quad q \leftarrow q \cup q_i$, $s \leftarrow |q|$ (size of q)
5: $\quad$ **if** $s + Lap(2/\epsilon) < \tilde{\theta}$ **then** $i \leftarrow i + 1$
6: $\quad$ **else**
7: $\quad\quad \tilde{B} \leftarrow \tilde{B} + [q]$, $q \leftarrow \varnothing$
8: $\quad\quad \tilde{\theta} \leftarrow \theta + Lap(2/\epsilon)$
9: $\quad\quad i \leftarrow i + 1$
10: $\quad$ **end if**
11: **end while**
12: $\tilde{B}[-1] \leftarrow \tilde{B}[-1] \cup q$
13: **return** $\tilde{B}$

similar method from [43]. Therefore, we propose a differentially private algorithm for computing private binned residual plots.

4.4.2 Private Binned Residual Plot

We propose an algorithm called *PriBRP* (see Algorithm 13) that generates binned residual plot under differential privacy in three steps:

Computing bins: Computing bins with roughly equal sample size is similar to the task of selecting thresholds for ROC curves that evenly divides the data. Again, the simple data independent strategy of uniformly selecting the bins within the range of predictions, $[0, 1]$, can be ineffective because the distribution of the predictions for the test data under logistic regression is not uniform. Therefore, we apply our proposed data dependent strategy, namely s-RecursiveMedians (Algorithm 12) for computing the bins. There is no restriction of the number of bins. Usually, the

statisticians set the number of bins to be $\sqrt{n}$ so that each bin has $\sqrt{n}$ data points, where n is the size of input dataset. In our case, since we apply Algorithm 12 for generating the bins and the size of the bins will be in powers of 2, we set $s = \max\{t \mid 2^t \leqslant \sqrt{n}\}$. We get 2^s bins after the first step. In *PriBRP*, we spend ϵ_1 privacy budget for computing bins.

Adjusting bins: Due to the randomized nature of Algorithm 12, there is no guarantee each of the bins contains roughly $\sqrt{n}$ data points. When $n \cdot \epsilon$ is small, it is very likely that some bins will have very few records (or even be empty). In such cases, the noisy average residual $\tilde{r}_{avg}$ and noisy average prediction $\tilde{p}_{avg}$ values computed in each bin (shown in the next step) will have significantly high error. In order to overcome this problem, we adjust the bins by grouping the consecutive bins with small size. We achieve this by applying Algorithm 14 based on the idea of *Sparse Vector Technique* [19]. Our bin adjustment algorithm is described in Algorithm 14. It takes as input a set of initial bins B computed from the first step, privacy budget ϵ, a threshold θ, and outputs a new set of bins $\tilde{B}$. The algorithm greedily attempts to merge bins from left to right till a merged bin has at least θ points. When the bin is large enough, it outputs the bin and continues trying to merge the next set of consecutive bins.

Theorem 13. *Algorithm 14 satisfies ϵ-differential privacy.*

Proof. Computing each merged bin (from line 3 to line 11) is an invocation of *Sparse Vector Technique* (SVT), which ensures ϵ-differential privacy [19]. The release of multiple merged bins are parallel invocations of SVT on disjointed parts of the domain. Thus by parallel composition property of differential privacy, releasing all the merged bins ensures ϵ-differential privacy, which implies Algorithm 14 satisfies ϵ-differential privacy. □

PriBRP's performance critically depends on the threshold θ. Based on the utility

analysis of *Sparse Vector Technique*, the size of output bins will be greater than $\theta - \frac{4(\log(k)+\log(2/\gamma))}{\epsilon}$ with probability $1-\gamma$, where k is the number of answered queries. In the Algorithm 14, we set $\theta = \frac{n}{2^t \times 2} + \frac{4\log(2/\gamma)}{\epsilon}$, ignoring $\log(k)$ term (empirically k is usually small). The goal is that the adjusted bins have size greater than half of the size of real non-private bins with roughly $1-\gamma$ probability (γ is set to be 0.9 in the experiment). In *PriBRP*, we spend ϵ_2 budget on adjusting bins.

Computing average residuals and predictions: Given a set of bins, we directly apply the *Laplace Mechanism* (*LM*) for computing average residuals and predictions within each bin. Suppose the bin under consideration is $q = [l, r]$, where $0 \leqslant l < r \leqslant 1$. Let L_q denote the set of points whose predictions are within $[l, r]$. That is

$$L_q = \{i \in [n] \mid l \leqslant p(\mathbf{x}_i) \leqslant r\} \tag{4.8}$$

Suppose $m = |L_q|$ denoting the size of L_q, then the average residual and prediction values can be computed:

$$r_{avg} = \frac{tot_r}{m}, \quad tot_r = \sum_{i \in L_q} r_i \tag{4.9}$$

$$p_{avg} = \frac{tot_p}{m}, \quad tot_p = \sum_{i \in L_q} p(\mathbf{x}_i). \tag{4.10}$$

We compute the private average residuals and predictions as follows:

1. We compute the private size $\tilde{m} = m + Lap(\frac{\Delta(m)}{\epsilon})$, where the sensitivity $\Delta(m) = 2$ under bounded differential privacy.

2. We compute the private sum of predictions $\tilde{tot}_p = tot_p + Lap(\frac{\Delta(tot_p)}{\epsilon})$, where the sensitivity $\Delta(tot_p) = r - l$. Then the private average prediction is $\tilde{p}_{avg} = \frac{\tilde{tot}_p}{\tilde{m}}$.

3. We compute the private sum of residuals $\tilde{tot}_r = tot_r + Lap(\frac{\Delta(tot_r)}{\epsilon})$, where the sensitivity $\Delta(tot_r) = 1 + r - l$. Then the private average residual is $\tilde{r}_{avg} = \frac{\tilde{tot}_r}{\tilde{m}}$.

We spend $\frac{\epsilon_3}{2}$ budget to compute $\tilde{m}$ and spend each $\frac{\epsilon_3}{4}$ budget for computing $\tilde{tot}_p$ and $\tilde{tot}_r$ all using *LM*. The private average residuals ($\tilde{r}_{avg}$) and predictions ($\tilde{p}_{avg}$) can be computed based on the above private values. When either of $\tilde{r}_{avg}$ and $\tilde{p}_{avg}$ exceeds its own bound, we corret it as

$$\hat{p}_{avg} = \begin{cases} r, & \text{if } \hat{p}_{avg} > r \\ l, & \text{if } \hat{p}_{avg} < l \\ \hat{p}_{avg}, & \text{otherwise} \end{cases} \tag{4.11}$$

$$\hat{r}_{avg} = \begin{cases} 1-l, & \text{if } \hat{r}_{avg} > 1-l \\ -r, & \text{if } \hat{r}_{avg} < -r \\ \hat{r}_{avg}, & \text{otherwise} \end{cases} . \tag{4.12}$$

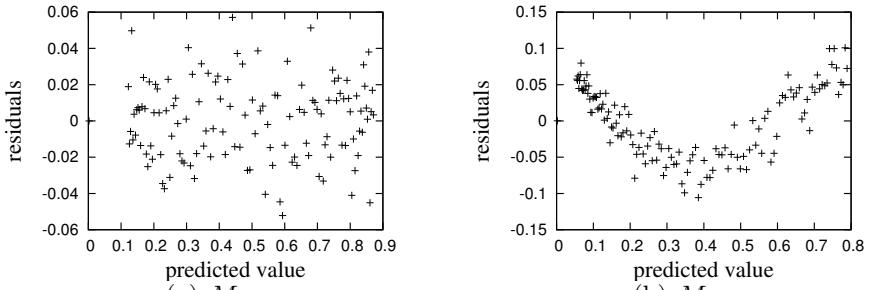
Theorem 14. PriBRP *ensures ϵ-differential privacy.*

Proof. Using Algorithm 12 to compute private bins with ϵ_1 budget ensures ϵ_1-differential privacy due to Theorem 12. Then using Algorithm 14 with ϵ_2 budget ensures ϵ_2-differential privacy. For each bin, computing average residuals and predictions by using *LM* will ensure $\epsilon_3 = \frac{\epsilon_3}{2} + \frac{\epsilon_3}{4} + \frac{\epsilon_3}{4}$-differential privacy based on sequential composition and postprocessing properties of differential privacy. Using parallel composition, computing average residuals and predictions for all bins satisfies ϵ_3-differential privacy. Under sequential composition, *PriBRP* ensures $\epsilon_1 + \epsilon_2 + \epsilon_3 = \epsilon$-differential privacy. □

4.4.3 Evaluation

In this part, we empirically evaluate the performance of our proposed *PrBRP*. We design the following experiments (in the experiment, we set $\epsilon_1 = \epsilon_2 = 0.15\epsilon$ and $\epsilon_3 = 0.7\epsilon$):

1. We visually inspect whether the private binned residual plot by *PrBRP* helps diagnose the model fit of logistic regression using synthetic datasets.



(a) M_1 (b) M_2

FIGURE 4.12: Binned residual plots for one confidential dataset randomly sampled from both models with scale $n = 50000$.

2. We quantitatively evaluate how well *PriBRP* allows us to evaluate the model fit of logistic regression based on the private binned residual plot.

3. We evaluate the sensitivity of *PriBRP*'s performance to the privacy budget allocated to each step.

Dataset: We generate synthetic data from following two models, sampling each x_i from a uniform distribution on [-1,1].

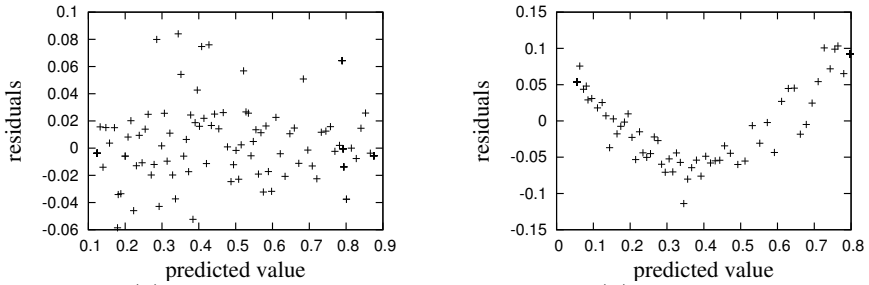
1. $M_1 : p(y_i = 1 \mid \mathbf{x}_i) = \frac{e^{2x}}{1+e^{2x}}$

2. $M_2 : p(y_i = 1 \mid \mathbf{x}_i) = \frac{e^{x^2+2x-1}}{1+e^{x^2+2x-1}}$

The true y_i is generated based on the probability $p(y_i = 1 \mid \mathbf{x}_i)$ computed above. For each scenario, we then fit the model with only the linear term in x_i. This is the correct model for scenario M_1 and the incorrect model for scenario M_2. To be useful, the private binned residual plots should allow us to distinguish these model fits.

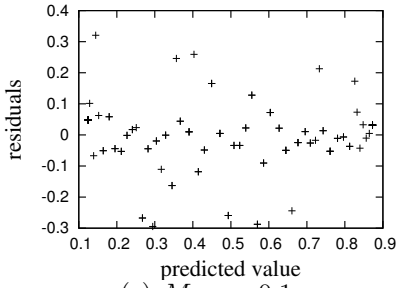
Visualizing binned residual plot

Figure 4.12 displays the real binned residual plots for two synthetic datasets with scale 50000 generated based on model M_1 and M_2. Figure 4.12(a) presents no systematic patterns or extreme values, which show the good fit of logistic regression . Figure 4.12(b) has a hook pattern, indicating the problematic fit.

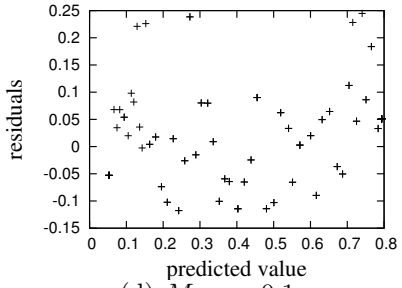


(a) $M_1, \epsilon = 1$

(b) $M_2, \epsilon = 1$



(c) $M_1, \epsilon = 0.1$



(d) $M_2, \epsilon = 0.1$

FIGURE 4.13: Differentially private versions of binned residual plots from Figure 4.12 generated via *PriBRP* with $\epsilon \in \{1, 0.1\}$

Figure 4.13 shows exemplary private binned residual plots constructed from single runs of *PriBRP* on both datasets in Figure 4.12. When $\epsilon = 1$, the overall pattern in the plots is well perserved. When the $\epsilon = 0.1$, the signal becomes less obvious but we can still detect a non-random pattern.

Ability to verify model fit

We now evaluate how well private binned residual plots reveal poor fit of logistic regression. We define the shape difference between real binned residual plot (BRP) and one perturbed binned residual plot (PBRP) as follows:

Table 4.5: Discriminatory Power of Binned Residual Plot under non-private setting

Data Scale	500	1000	1500	2000	2500	3000	3500
TVD($ShapeD_1$,$ShapeD_2$)	0.554	0.69	0.81	0.9	0.94	0.972	0.982

- We divide the points in BRP and PBRP into two sets separately.

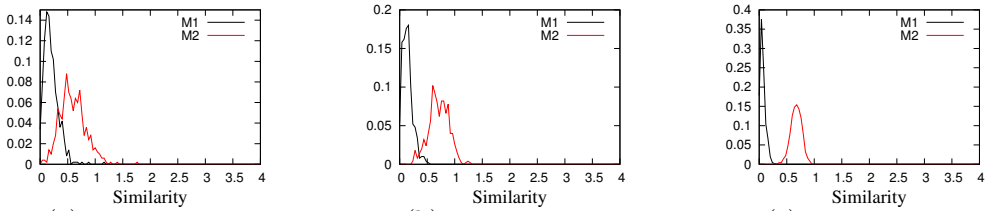
$$\begin{aligned} A_1 &= \{(p_i, r_i) \mid (p_i, r_i) \in BRP \text{ and } p_i \leqslant 0.5\} \\ A_2 &= \{(p_i, r_i) \mid (p_i, r_i) \in BRP \text{ and } p_i > 0.5\} \\ B_1 &= \{(p_i, r_i) \mid (p_i, r_i) \in PBRP \text{ and } p_i \leqslant 0.5\} \\ B_2 &= \{(p_i, r_i) \mid (p_i, r_i) \in PBRP \text{ and } p_i > 0.5\}. \end{aligned}$$

- We compute the regression line for each set by using OLS. The slopes of the regression lines are denoted as Slp_{A_1} and Slp_{A_2},Slp_{B_1} and Slp_{B_2}.

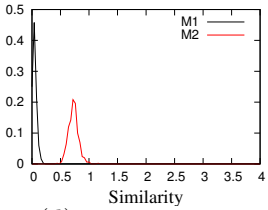
- The shape difference between BRP and PBRP is defined as

$$ShapeD(BRP, PBRP) = abs(Slp_{A_1} - Slp_{B_1}) + abs(Slp_{A_2} - Slp_{B_2}). \quad (4.13)$$

First, in the non-private setting, for any dataset generated from model M_1 and M_2 with a fixed scale, let BRP_1 and BRP_2 be the corresponding real binned residual plot. The ideal binned residual plot for a good model fit of logistic regression should show a random scatter such that the slopes of both regression lines are equal to 0. We compute the shape difference between BRP_i (i = 1 and 2) and the ideal binned residual plot, and the results are denoted as $ShapeD_1$ and $ShapeD_2$. Using 500 simulated datasets, we generate 500 values of ($ShapeD_1$, $ShapeD_2$). Table 4.5 shows the total variation distance (TVD) between the distributions of $ShapeD_1$ and $ShapeD_2$ for different data scale settings. We can see that the TVD $\geqslant 0.9$ when the scale of the data is not less than 2000. Empirically, under non-private setting, we should be able to differentiate M_1 from M_2 by using binned residual plot when the data scale $\geqslant 2000$.



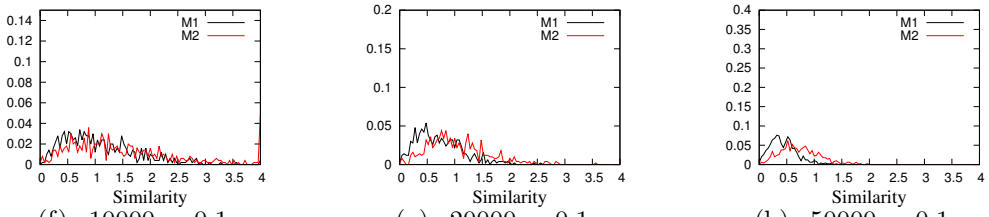
(a) s10000,ϵ=1 (b) s20000,ϵ=1 (c) s50000,ϵ=1



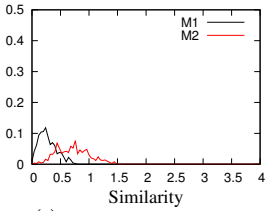
(d) s100000,ϵ=1

Scale	TVD(M1,M2)
10000	0.728
20000	0.936
50000	0.998
100000	1

(e) TVD, ϵ=1



(f) s10000,ϵ=0.1 (g) s20000,ϵ=0.1 (h) s50000,ϵ=0.1



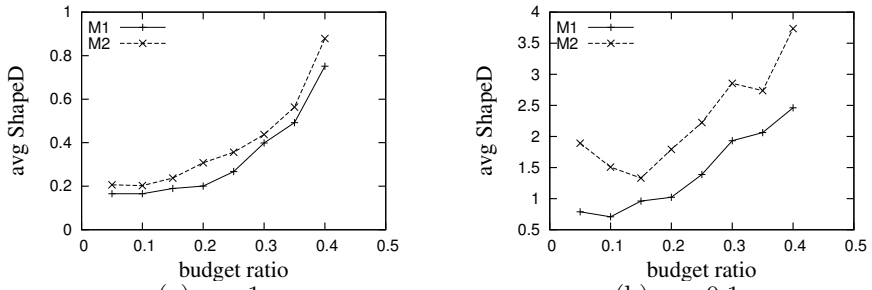
(i) s100000,ϵ=0.1

Scale	TVD(M1,M2)
10000	0.286
20000	0.306
50000	0.39
100000	0.664

(j) TVD, ϵ=0.1

FIGURE 4.14: Comparison of shape difference between simulated confidential and private binned residual plots.

Next, we consider the case of private binned residual plots. Instead of varying n and ϵ independently, we only vary $n \cdot \epsilon$ since the algorithms we use satisfy *scale-epsilon exchangeability* [31]. We compute the shape difference between real binned residual plot from model M_1 and the noisy binned residual plots from both model M_1 and M_2. In each plot of Figure 4.14, the curves M_1 and M_2 represent the empirical distributions of shape difference between real plots from model M_1 and noisy plots from M_1 and M_2 using 500 generated noisy plots in each case. When $n \cdot \epsilon \geqslant 20000$,



(a) $\epsilon = 1$ (b) $\epsilon = 0.1$

FIGURE 4.15: Effect of Budget Split in terms of Shape Difference

two curves are well separated, which means an analyst can distinguish the good fit and poor fit by using noisy binned residual plot. Figure 4.14(e) and 4.14(j) comfirm this finding by using total variation distance between M_1 and M_2 curves. The total variation distance is over 0.9 when $n \cdot \epsilon \geqslant 20000$.

Sensitivity to budget split

We evaluate the effect of budget split on the performance of *PriBRP*. In *PriBRP*, we split the total privacy budget ϵ into three parts - ϵ_1 for computing initial bins, ϵ_2 for adjusting bins and ϵ_3 for computing average residuals and predictions. We let $\epsilon_1 = \epsilon_2 = \rho\epsilon$ and $\epsilon_3 = (1 - 2\rho)\epsilon$, where ρ is called the budget ratio. When ρ is high (close to 0.5), most of the privacy budget is spent on computing bins so that more noise will be injected to average residuals and predictions. On the other hand, a small ratio brings more noise for generating bins so that the generated bins may not equally partition the data. Figure 4.15 displays the results in terms of shape difference for different budget ratio settings. Each figure includes two curves for datasets from model M_1 and M_2 with scale 10000. Each point reports the average shape difference between the real binned residual plot and one perturbed binned residual plot by *PriBRP* among 100 trials under the corresponding budget split. We can observe that for both settings of ϵ, a range of a smaller ρ values lead to relatively better performance in terms of the shape difference for both datasets. $\rho = 0.15$ is a

good choice across datasets and across different ϵ values.

4.5 Conclusions

It is not possible using only the output of private algorithms or estimates based on synthetic data. Thus, we present differentially private diagnostic tools for linear and logistic regression that help analysts assess the quality of the assumptions and predictive capabilities of regression models. Some algorithms (*PriROC* and *PriBRP*) make use of *Sparse Vector Technique* for designing their sub-modules.

More broadly, we believe that this work is the first step toward building provably private verification servers [50], or query systems that allow users to assess the quality of inferences made on synthetic data. Verification servers would permit analysts to gauge whether or not their analyses on synthetic data result in reliable conclusions, thus achieving the twin goals of privacy and utility.

5

Differentially Private Stream Processing

In this Chapter, we design novel differentially private solutions to support queries on streaming datasets, where *Sparse Vector Technique* was also applied as a key primitive in the design of the algorithm.

5.1 Introduction

A number of emerging application domains rely on personal data processed in a streaming manner. Streaming data is the foundation of the Internet of Things [3] and prevalent in domains like environmental sensing, traffic management, health monitoring, and financial technology. Such data is typically captured and analyzed continuously and, because of the volume of the data, it is often processed as it arrives, in real time.

Since this data may report on individuals' location, health status, or other sensitive state, directly releasing the data, or even aggregates computed from the data stream can violate privacy [16, 20]. In particular, continually updating statistics over time leaks more and more information to the attackers, potentially causing harmful privacy leakage [7].

In this part, we propose a novel technique for releasing continuous query answers on real time streams under differential privacy. Our technique combines a *Perturber*, which generates a stream of noisy counts, and an independent module called a *Grouper*, which computes a partition of the data received so far. The *Grouper* privately finds partitions of the data which have small absolute deviations from their average. The final module, called the *Smoother*, combines the output of both the *Perturber* and the *Grouper*, generating the final private estimate of a query answer at each time step. The *Perturb-Group-Smooth* technique (or PGS, the inspiration for "PeGaSus") is data-adaptive: it offers improved accuracy for streams that have sparse or stable counts because the *Grouper* detects these regions and the *Smoother* uses knowledge of these regions to infer better estimates.

PeGaSus not only helps release accurate differentially private streams (individual counts at each time step) but can also simultaneously support multiple alternative query workloads including window sum queries and event monitoring queries like finding jumping and dropping points or detecting low signal points in the stream. These different tasks can be solved by reusing the output of the *Perturber* and the *Grouper*, and just modifying the *Smoother* method, without incurring any additional privacy budget. Surprisingly, for many of these workloads, using our data dependent strategy outperforms state-of-the-art algorithms that are designed specifically for the corresponding query workload.

We propose extensions to PeGaSus to answer counting queries at different hierarchical resolutions on the stream. These extensions allow us to model typical query workloads that appear in building or network monitoring, where analysts are interested in both streams generated by individual sensors (or IP addresses), but also in aggregate streams generated by groups of sensors (or groups of IP addresses).

In summary, we make the following contributions:

- We design PeGaSus, a novel technique for answering a large class of continuous queries over real time data streams under differential privacy.
- PeGaSus uses a combination of a *Perturber*, a data-adaptive *Grouper* and a query specific *Smoother* to simultaneously support a range of query workloads over multiple resolutions over the stream.
- The *Grouper* and *Smoother* in combination offers improved accuracy for streams that have sparse or stable counts.
- A thorough empirical evaluation on a real data stream collected from 4000 WiFi access points from a large educational institution shows that, by using different query specific *Smoother* methods, PeGaSus outperforms the previous state-of-the-art algorithms specialized to given workloads. For example, our data dependent algorithm can compute more accurate sliding window queries than the previous state-of-the-art algorithm that is designed for a specific sliding window workload for all window sizes $w \leqslant 2^8$.

The paper is organized as follows. Section 5.2 reviews the streaming data model, queries on streams, and the semantics of privacy on streams. In Section 5.3, we describe the Perturb-Group-Smooth (PeGaSus) algorithm. In Section 5.4, we show how the framework can support multiple query workload by applying different query specific *Smoother* methods. In Section 5.5, we discuss how to extend PeGaSus to answer counting queries at different hierarchical resolutions on the stream. Comprehensive experiments on a real data stream are presented in Section 5.6. The conclusion is made in Section 5.7.

5.2 Preliminaries

5.2.1 Stream data model

We define the source stream $\mathbf{D}$ as an infinite sequence of tuples. Each tuple is of the form (u, s, t) and is an element from the domain dom $= \mathcal{U} \times \mathcal{S} \times \mathcal{T}$ where $\mathcal{U}$ is set of user identifiers, $\mathcal{S}$ is a set of possible states, and $\mathcal{T}$ is an (infinite) set of timestamps. Each (u, s, t) records an atomic event, namely that user u was observed in state s at time t. Note that this single stream could contain events from multiple sources – these would be encoded as different states (elements of $\mathcal{S}$).

To simplify presentation, we represent time using logical timestamps letting $\mathcal{T} = \{1, 2, 3, \dots\}$. The rationale is that the analysis tasks we consider in Section 5.2.2 emit an aggregate summary of the stream periodically (e.g. every 5 minutes) and thus logical time $t = 1$ can be understood as capturing all events that happened within the first reporting period. Furthermore, this implies that the a tuple (u, s, t) does not describe a specific, instanteous event but rather it encodes the aggregate behavior of user u during the time step t. Therefore, the states $\mathcal{S}$ encode the state of a user during the logical time step t. We illustrate with an example.

Example 3. *Consider a data stream management system that collect data from WiFi access points (APs) distributed across buildings on a campus. Users correspond to mac addresses of individual devices that connect to WiFi access points. The set of time steps could represent the aggregated activity of a user over time intervals of 5 minutes each. Thus, time steps t and $t + 1$ would differ in wall clock time of 5 minutes. Finally, if there are m WiFi access points on campus, then we could have $m + 1$ states: a state $s_\perp$ that represents "user did not make a successful connection to any AP", and m states s_p, one for each AP p, that represents "user made at least one successful connection to the AP p".*

The tuples in $\mathbf{D}$ arrive in order by time. Thus if (u', s', t') arrives after (u, s, t) it

must be that $t \leqslant t'$. We use $\mathbf{D}_t$ to represent a stream prefix: the set of tuples that arrive on or before time t.

5.2.2 Queries on streams

In this paper, we consider a number of queries on the private stream $\mathbf{D}$. The answer to a query on $\mathbf{D}$ is itself a stream. We focus on counting queries as well as other queries that can be derived from counting queries.

Queries on a single target state

A counting query takes a specific target state s and reports, for each time step t, the number of users who were observed in state s at time t. More formally, let $C(s)$ be the infinite stream $C(s) = c_1(s), c_2(s), \ldots$ where $c_t(s) = |\{(u', s', t') \in \mathbf{D}_t \mid t' = t \text{ and } s' = s\}|$. Let $C_t(s)$ denote the prefix stream of $C(s)$ up to time t. When clear from context, we drop the s and just use $C = c_1, c_2, \ldots$.

Note that the answer to the query should be generated in "real time" – i.e., c_t should be produced before any tuple $(\cdot, \cdot, t+1)$ is observed.

Example 4. *An analyst might want to visualize the counts of the number of users who had at least one successful connection in a time step at access point AP1. Hence, the target state is s_{AP1}, $C_t(s_{AP_1})$ represents the number of users with at least one successful connection to AP_1 in time step t, and $C(s_{AP_1})$ represents the stream of counts.*

The counting query defined above is referred to as a *Unit* query. We also support additional queries, all of which can be derived from C.

Sliding Windows A sliding window query with window size w and target state s reports, for each time step t, the total number of times a user has been observed in state s in the most recent w time steps. More formally, let $SW(s, w)$ be an infinite

stream $SW(s,w) = sw_1, sw_2, \ldots$, where $sw_t(s,w) = |\{(u',s',t') \in \mathbf{D}_t \mid t-w < t' \leqslant t \text{ and } s' = s\}|$. Observe that the sliding window query answers can also be derived by summing corresponding counts in query $C(s)$: $sw_t(s,w) = \sum_{t'=t-w+1}^{t} c_{t'}(s)$.

Event Monitoring While each tuple in the stream $\mathbf{D}$ captures an atomic event, the analyst may be interested in monitoring certain patterns in the event stream. We call this task event monitoring and consider monitoring event patterns that can be derived from the counting query stream C.

We define the event monitoring query as follows. Let $EM(s,w,f,B) = b_1, b_2, \ldots$ be an infinite stream of bits where $b_1 = 1$ if the monitored event has occurred at time t and 0 otherwise. The inputs to the query EM are the target state s, a desired window size w, an abitrary function f that computes on a window of w counts and boolean function B that computes on the ouptut of f. The bit b_t is computed as $b_t = B(f(c_{t-w+1}, \ldots, c_t))$. We give two examples of event monitoring queries.

- Jumping and dropping point: This query monitors whether the count has changed by at least δ from the count w time units ago. Thus, f computes the absolute difference between the current count and the count received w time steps before, $f(c_{t-w+1}, \ldots, c_t) = |c_t - c_{t-w+1}|$ and B compares that difference to a threshold δ, $B(x) = 1$ if $x \geqslant \delta$ and is 0 otherwise.
- Low signal: This query monitors whether the total count in a sliding window is smaller than δ. Thus, f computes the total count, $f(c_{t-w+1}, \ldots, c_t) = \sum_{i=t-w+1}^{t} c_i$ and B is again a threshold function, $B(x) = 1$ if $x < \delta$ and is 0 otherwise.

Queries on multiple target states

Our approach also supports queries on multiple target states. Let $\{s_1, \ldots, s_m\} \subseteq \mathcal{S}$ denote the set of states the analyst is interested in. We support three variants. First, the analyst can simply issue multiple queries where each query is on a single target

state (i.e., any one of the queries defined previously). We illustrate this with an example.

Example 5. *An analyst might be interested in the unit query for the states s_p corresponding to all access points p within a specific building, as well as a low signal event monitoring query $EM(s_q, w, \dots)$, for all access points q housed in conference rooms across campus.*

Second, we also support queries on *aggregations* of target states. We denote a single aggregation as $agg \subseteq \{s_1, s_2, \dots, s_m\}$. Any query that is defined for a single target state can also be defined over an aggregation of target states by replacing any equality condition on the state ($s' = s$) with set membership condition ($s' \in agg$). For example, an aggregated counting query is denoted $C(agg)$ and it produces a stream of answers $C(agg) = c(agg)_1, c(agg)_2, \dots$ where $c(agg)_t = \sum_{i \in agg} c_t(s_i)$.

Finally, the analyst may wish to ask a query about more than one aggregation of states. Let $AGG = \{agg_1, agg_2, ...\}$ denote a collection of aggregations. We consider the special case where this collection has a hierarchical structure.

Definition 6. *A set of aggregations $AGG = \{agg_1, agg_2, ...\}$ is* hierarchical *if for any two aggregations $agg_1, agg_2 \in AGG$, they satisfy one of the following two properties:*

1. $agg_1 \subset agg_2$ *or* $agg_2 \subset agg_1$.

2. $agg_1 \cap agg_2 = \varnothing$.

Intuitively, a set of hierarchical aggregations AGG can be represented as a tree or a forest, where any child aggregation contains a subset of states that its parent aggregation covers and any two child aggregations with the same parent aggregation cover two disjoint sets of states. We use $level(agg \mid AGG)$ to denote the level of $agg \in AGG$. $level(agg \mid AGG) = 1$ if agg has no parent aggregation in AGG. If

$agg_1, agg_2 \in AGG$ and agg_1 is the child aggregation of agg_2, $level(agg_1 \mid AGG) = level(agg_2 \mid AGG) + 1$.

Example 6. *An analyst might be interested in aggregate counts of successful connections at the level of individual sensors, as well as for aggregations of sensors within the same room, same floor, and the same building on campus. For each room r, let agg_r denote the aggregation of states s_p, where p is an access point in room r. Similarly, for a floor f and building b, we can define aggregations agg_f and agg_b that aggregate states s_p, where p is in the floor f or building b, respectively. These set of aggregations form a hierarchy. An analyst might be interested in the unit query for aggregate states at floors and building as well as a low signal event monitoring query $EM(agg_r, w, \dots)$, for all room r on campus.*

5.2.3 Privacy for Streams

Two stream prefixes $\mathbf{D}_t$ and $\mathbf{D}'_t$ are considered *neighbors* if they differ by addition or removal of a single tuple; i.e., $|\mathbf{D}_t \oplus \mathbf{D}'_t| = 1$ where $\oplus$ indicates symmetric difference. The algorithms we consider in this paper operate on stream prefixes.

We define privacy for streams as follows, similarly to event differential privacy [4, 23].

Definition 7 (ϵ-differential privacy)**.** *Let $\mathcal{A}$ be a randomized algorithm that takes as input a stream prefix of arbitrary size and outputs an element from a set of possible outputs $\mathcal{O}$. Then $\mathcal{A}$ satisfies ϵ-differential privacy if for any pair of neighboring stream prefixes $\mathbf{D}_t$ and $\mathbf{D}'_t$, and $\forall O \subseteq \mathcal{O}$,*

$$Pr[\mathcal{A}(\mathbf{D}_t) \in O] \leqslant e^{\epsilon} \times Pr[\mathcal{A}(\mathbf{D}'_t) \in O] \tag{5.1}$$

The parameter ϵ controls the privacy risk and smaller ϵ corresponds to stronger privacy protection. The semantics of this privacy guarantee are discussed further in Section 5.2.4.

Remark: We make an assumption that each user can be in at most k different states s during one time period. Without loss of generality, we assume $k = 1$ in our application. If $k > 1$, we can simply normalize the counts (a user who is in k states at time t contributes $1/k$ rather than 1 to the corresponding counting queries) or increase the amount of noise injected in our algorithm in order to provide privacy protection in terms of any single user during one time period.

5.2.4 Privacy Semantics

In this section, we discuss the semantics of privacy ensured by Definition 7 and justify our choice of this privacy goal.

The privacy ensured by Definition 7 can be interpreted in terms of (a) plausible deniability, and (b) disclosure of secrets to adversaries. Let $\phi_u(t)$ and $\phi'_u(t)$ be two mutually exclusive boolean properties about a user u at time step t. Examples of such properties could be that a user was in building $B1$ at time t and building $B2$ at time t, respectively. An algorithm M satisfying Definition 7 allows a user to deny that $\phi'_u(t)$ is true rather than $\phi_u(t)$ since for all neighboring streams $\mathbf{D}_t, \mathbf{D}'_t$ such that $\phi_u(t)$ is true on $\mathbf{D}_t$ and $\phi'_u(t)$ is true on $\mathbf{D}'_t$, and for all output sets $O \in range(\mathcal{A})$, we have:

$$Pr[\mathcal{A}(\mathbf{D}_t) = O] \leqslant e^{\epsilon} Pr[\mathcal{A}(\mathbf{D}'_t) = O] \tag{5.2}$$

Plausible deniability holds even for properties that span larger time windows, albeit to a lesser extent and degrades with the length of the time window. That is, if $\phi_u(t, k)$ and $\phi'_u(t, k)$ are two mutually exclusive boolean properties about a user u that span a time window of $[t - k + 1, t]$, then for all streams $\mathbf{D}_t, \mathbf{D}'_t$ such that $\phi_u(t)$ is true on $\mathbf{D}_t$ and $\phi'_u(t)$ is true on $\mathbf{D}'_t$, and that differ only in the states in the time window $[t - k + 1, t]$, and for all output sets $O \in range(\mathcal{A})$, we have:

$$Pr[\mathcal{A}(\mathbf{D}_t) = O] \leqslant e^{k \cdot \epsilon} Pr[\mathcal{A}(\mathbf{D}'_t) = O] \tag{5.3}$$

Thus our definition also captures the more general w-event privacy [33]. And, if a time step corresponds to 5 minutes, and an algorithm $\mathcal{A}$ satisfies Definition 7 with $\epsilon = 0.1$, then for properties that span 10 minutes, we get privacy at a level $\epsilon = 0.2$, and for properties that span 1 hour, we get privacy at a level of $\epsilon = 1.2$. If the protected window size goes to infinity (k is unbounded), one can extend existing negative results [16] to show that it is impossible to release accurate statistics at each time and offer privacy.

Next we explore the semantics of Definition 7 in terms of disclosure of secrets to adversaries, which can be done in terms of the Pufferfish framework [**?**]. One can show that if an algorithm $\mathcal{A}$ satisfies Definition 7, then the adversary's posterior odds that $\phi_u(t)$ is true vs $\phi'_u(t)$ is true after seeing the output of $\mathcal{A}$, for any pair of mutually exclusive secrets that span a single time step, is no larger than e^ϵ times the adversary's prior odds. However, this strong privacy guarantee only holds under the restrictive assumptions that an adversary is not aware of possible correlations between a user's states across time steps. With knowledge of correlations, an adversary can learn sensitive properties of a user within a time step even from outputs of differentially private algorithms [34, 41, 61].

Recent work [9, 54] has provided methods for deriving an $\epsilon' > \epsilon$ (but no more than $k \times \epsilon$ in terms of any protected window with size k), such that algorithms satisfying Definition 7 with parameter ϵ offer a weaker ϵ' bound on privacy loss, even when records are correlated across time steps. The effective privacy guarantee is closer to ϵ and much smaller than $k \times \epsilon$.

Therefore, for the rest of the paper, we only concern ourselves with developing algorithms that ensure Definition 7 while minimizing error. We emphasize that our proposed algorithms that satisfy Definition 7 simultaneously ensure all of the above provable privacy guarantees, but with different value of ϵ. Hence, algorithms that satisfy Definition 7 are important building blocks for releasing streaming data with

provable guarantees, and allows those guarantees to be configured.

5.3 Private Stream Release under PeGaSus

In this section we describe a novel, data-dependent method for private, real-time release of query answers on data streams. Our algorithm consists of a novel combination of data perturbation and online partitioning, followed by post-processing.

We first present the algorithm for computing a unit counting query in a single target state. In 5.4, we explain how the algorithm can be adapted to answer other kinds of queries on a single target state and in 5.5, we explain an extension to the algorithm to support multiple queries over hierarchical aggregations of states. The input to the algorithm is the true answers to the unit counting query C. The output of the algorithm is $\hat{C} = \hat{c}_1, \hat{c}_2, \ldots$, an infinite stream where $\hat{c}_t$ is an estimate of the true answer c_t.

The three main modules of our algorithm are as follows:

- ***Perturber*:** The *Perturber* consumes the input stream, adds noise to each incoming element of the stream, and releases a stream of noisy counts.
- ***Grouper*:** The *Grouper* consumes the input stream and groups elements of the stream seen so far.
- ***Smoother*:** The *Smoother* performs post-processing using the output of both above modules.

The *Perturber* is a standard noise-addition mechanism, but the *Grouper* carries out an important role of partitioning data into regions that can be well-approximated by a uniform sub-stream. The *Smoother* then combines this information with the output of the *Perturber*. The result is a data-dependent algorithm which can reduce error for streams with properties that are commonly witnessed in practice.

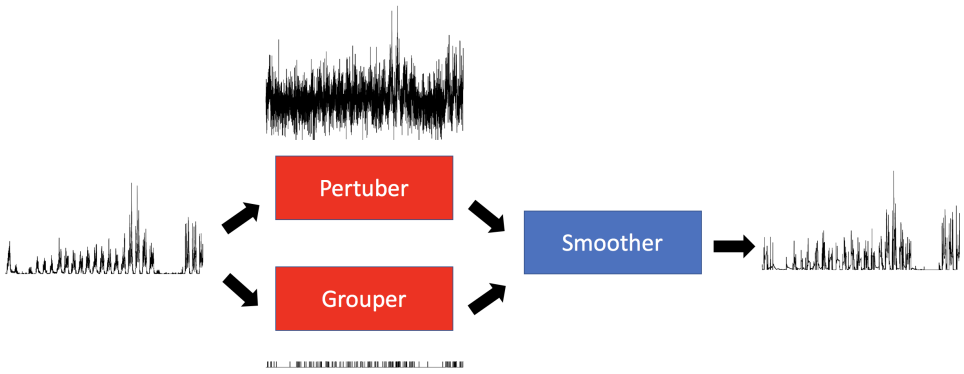


FIGURE 5.1: Overview of PeGaSus.

Algorithm 15 Perturb-Group-Smooth based Stream Release (PeGaSus)

Input: $C = c_1, c_2, \ldots,$ privacy budget $\epsilon = \epsilon_p + \epsilon_g$

Output: Private stream $\hat{C} = \hat{c}_1, \hat{c}_2, \ldots,$

1: **for** each time t **do**
2: $\quad \tilde{c}_t \leftarrow Perturber(c_t, \epsilon_p)$
3: $\quad P_t \leftarrow Grouper(C_t, P_{t-1}, \epsilon_g)$
4: $\quad \hat{c}_t \leftarrow Smoother(\tilde{C}_t, P_t)$
5: $\quad$ Release $\hat{c}_t$
6: **end for**

The combination of the above three modules is formalized in 15, called Perturb-Group-Smooth stream release (PeGaSus). When a new count c_t arrives at time t, the *Perturber* takes the input c_t and outputs a noisy version $\tilde{c}_t$, using ϵ_p portion of the overall ϵ privacy budget (line 2). The *Grouper* takes as input all of the data received so far $C_t = c_1, \ldots, c_t$ and the partition from the previous time P_{t-1}. (Partition P_{t-1} is a partition over the integers $\{1, \ldots, t-1\}$ and represents a grouping of the first $t-1$ counts.) At each time step, the *Grouper* outputs an updated partition P_t with a portion of the privacy budget ϵ_g (in line 3). The *Smoother* then computes a final estimate $\hat{c}_t$ of c_t based on all the initial noisy counts $\tilde{C}_t = \tilde{c}_1, \ldots, \tilde{c}_t$ and the current partition P_t (in line 4).

The execution of 15 is illustrated in 5.1. Both the *Perturber* and the *Grouper* are colored red because they consume the input stream and use the privacy budget. The *Smoother* is colored blue because it only uses the output of the *Perturber* and the

Grouper.

Theorem 15. *When the* Perturber *satisfies ϵ_p-differential privacy and the* Grouper *satisfies ϵ_g-differential privacy, 15 ensures $\epsilon_p + \epsilon_q = \epsilon$-differential privacy.*

The above theorem follows directly from the sequential composition and post-processing properties of differential privacy.

15 forms the basis of a number of algorithm variants we consider throughout the paper. In the remainder of this section, we describe below the design of each module, and basic variants for the *Smoother*. We also include theoretical analysis that illustrates cases where smoothing can reduce error. **??** describe extensions to the algorithm for other kinds of queries.

5.3.1 Design of the Perturber

The *Perturber* takes input c_t at each time t and outputs a noisy version $\tilde{c}_t$. We use the Laplace Mechanism as the implementation of the *Perturber*.

5.3.2 Design of the Grouper

Next we describe the Deviation based Grouper (DBG) (16), a differentially private method for online partitioning which chooses partitions that approximately (subject to distortion introduced for privacy) minimizes a quality score based on deviation.

Recall that this module runs independently of the *Perturber* and does not consume its output. Instead the *Grouper* takes as input all of the data received so far, $C_t = c_1, \ldots, c_t$ at time t, and outputs a partition of C_t. At any time t, the stream seen so far has been partitioned into contiguous groups. All but the most recent group are *closed* and will not be changed, but the most recent group may continue to grow as new elements arrive, until it is eventually closed.

To evaluate the quality of a potential group, we use the deviation function, which measures the absolute difference of a set of counts from its average. Let G be a

Algorithm 16 Deviation based Grouper (DBG)

Input: $C_t = c_1, \ldots, c_t$, the previous partition P_{t-1}, ϵ_g, θ
Output: P_t (a partition of $\{1, \ldots, t\}$)

1: **if** $t = 1$ **then**
2: $P_{t-1} \leftarrow \varnothing$ and let G be closed, empty group.
3: **else**
4: $G \leftarrow$ Last group from P_{t-1}
5: **end if**
6: **if** G has been closed **then**
7: $P_t \leftarrow P_{t-1} \cup \{\{t\}\}$ and let the last group $\{t\}$ be open.
8: $\tilde{\theta} \leftarrow \theta + Lap(4/\epsilon_g)$
9: **else**
10: $\tilde{\theta} \leftarrow \tilde{\theta}_{prev}$ ▷ $\tilde{\theta}_{prev}$ cached from previous call.
11: **if** $(dev(C_t[G \cup \{t\}]) + Lap(8/\epsilon_g)) < \tilde{\theta}$ **then**
12: $P_t \leftarrow P_{t-1}$ with G replaced by $G \cup \{t\}$ and G still open.
13: **else**
14: $P_t \leftarrow P_{t-1} \cup \{\{t\}\}$ and close both group G and $\{t\}$.
15: **end if**
16: **end if**
17: $\tilde{\theta}_{prev} \leftarrow \tilde{\theta}$ ▷ cache $\tilde{\theta}_{prev}$ for subsequent calls.
18: Return P_t

group of indexes of data $C_t = c_1, \ldots, c_t$, $C_t[G]$ be the set of corresponding counts in C_t and denote by $|G|$ the size of the group. Then the deviation of $C_t[G]$ is denoted $dev(C_t[G])$ and is defined:

$$dev(C_t[G]) = \sum_{i \in G} \left| c_i - \frac{\sum_{i \in G} c_i}{|G|} \right|. \tag{5.4}$$

When the $dev(C_t[G])$ is small, the counts in $C_t[G]$ are approximately uniform and the Smoother can exploit this.

Algorithm 16 takes as input the stream seen so far, C_t, the latest partition, P_{t-1}, along with ϵ_g and a user-defined deviation threshold θ which influences the acceptable deviation in a partition. Because the algorithm uses the sparse vector technique [19], it maintains a noisy threshold $\tilde{\theta}_{prev}$ used at the previous time. When a new data c_t arrives at time t, we check the status of the last group G from the previous partition P_{t-1}. If G is closed, we put c_t into a new open group and reset the noisy threshold (in line 6-8). Otherwise, we compute the deviation of the $C_t[G \cup \{t\}]$ and compare a

noisy version of the deviation value with the noisy threshold. If the noisy deviation is smaller than the noisy threshold, we add {t} into the open G (in line 11-12). Otherwise, we add a new group {t} into the partition and close both G and $\{t\}$ (in line 14).

The following example explains how the *Grouper* defined in Algorithm 16 would run, assuming for simplicity an infinite privacy budget (so that the noise added in lines 8 and 11 is zero).

Example 7. *Consider a stream of counts* $C_5 = [5, 5, 6, 9, 10]$ *and a threshold* $\theta = 2$. *At time* 1, *the* Grouper *generates a partition* P_1 *containing a single open group* G *with a single index:* $G = \{1\}$ *and* $P_1 = \{G\}$. *At time 2, the latest group* $G = \{1\}$ *is open and* $dev(C_2[\{1, 2\}]) = dev([5, 5]) = 0$. *Because the deviation is less than* θ, *we add the current data into* G *and keep* G *open.* P_2 *still contains a single group* G. *At time 3, the latest group* $G = \{1, 2\}$ *is open and* $dev([5, 5, 6]) = \frac{4}{3} < \theta$. *We still add the current data into* G *and keep* G *open. At time 4, the latest group* $G = \{1, 2, 3\}$ *is open but* $dev([5, 5, 6, 9]) = 5.5 > \theta$. *Thus we close* $G = \{1, 2, 3\}$ *and create the second group* $\{4\}$ *which is also closed. Thus,* $P_4 = \{\{1, 2, 3\}, \{4\}\}$. *At time 5, the latest group* $G = \{4\}$ *is closed, so we start with a new open group* $G = \{5\}$. *The final output at time 5 is* $P_5 = \{\{1, 2, 3\}, \{4\}, \{5\}\}$.

With a realistic (non-infinite) setting for ϵ_g, the Grouper runs in a similar manner but noise is added to both the threshold and to the deviations that are compared with the threshold. Therefore, the output partitions will differ.

To prove the privacy of Algorithm 16 we make use of the following lemma:

Lemma 16 (Sensitivity of deviation [38])**.** *The global sensitivity of the deviation function* $\Delta(dev)$ *is bounded by 2.*

Theorem 17. *Using Algorithm 16 to generate a partition at every time ensures* ϵ_g*-differential privacy.*

Proof. The algorithm is an implementation of the *Sparse Vector Technique* [19] on multiple disjoint sub-streams applied to the *dev* function. Because the sensitivity of *dev* is bounded by 2 (Lemma 16), the noise added to the threshold, in line (4), is computed as $Lap(4/\epsilon_g) = Lap(2\Delta(dev)/\epsilon_g)$. In line (7), the noise added to the deviation value is $Lap(8/\epsilon_g) = Lap(4\Delta(dev)/\epsilon_g))$ and ϵ_g-differential privacy follows from the original definition of the Sparse Vector Technique. In addition, the derived streams from two neighboring source streams can only differ by 1 at one time. By parallel composition, Algorithm 16 satisfies ϵ_g-differential privacy. □

5.3.3 Design of the Smoother

The *Smoother* computes the final estimate $\hat{c}_t$ for each c_t received at time t based on all the noisy counts $\tilde{C}_t$ from the *Perturber*, in combination with the current partition P_t. Suppose the last group $G = \{t-k, \ldots, t-1, t\}$ from P_t contains current index t with the previous k indexes. Given this output from the *Grouper*, there are several post-processing methods we may apply to generate an estimate, $\hat{c}_t$, that improves upon $\tilde{c}_t$. These are alternatives for the *Smoother* in Algorithm 15:

1. AverageSmoother: We use the average noisy counts of the data indexed in group G to be the estimate of the current data.

$$\hat{c}_t = \frac{\sum_{i \in G} \tilde{c}_i}{|G|}. \tag{5.5}$$

2. MedianSmoother: We use the median noisy counts of the data indexed in group G to be the estimate of the current data.

$$\hat{c}_t = median\{\tilde{c}_i \mid i \in G\}. \tag{5.6}$$

3. JSSmoother: We apply James-Stein estimator [**?**] to update the noisy count

of the current data based on the noisy counts of the data indexed in group G.

$$\hat{c}_t = (1 - \frac{(|G| - 1)\tau^2}{\sum_{i \in G}(\tilde{c}_i - avg)^2}) \times (\tilde{c}_t - avg) + \tilde{c}_t, \tag{5.7}$$

where $avg = \frac{\sum_{i \in G} \tilde{c}_i}{|G|}$ and τ^2 is the variance. We assume uniformity on each group and apply James-Stein estimator to let each estimate shrink to the mean (we can only use the noisy mean here).

We theoretically analyze the effect of AverageSmoother in the next section and empirically evaluate all three variants in Section 5.6. We conclude this section with an example.

Example 8. *Continuing from Example 7, we have a stream of true counts* $C_5 = \{5, 5, 6, 9, 10\}$ *with noisy counts from the* Perturber *of* $\tilde{C}_5 = \{5.6, 4.4, 6.7, 9.5, 10.2\}$ *and a final partition* $P_5 = \{\{1, 2, 3\}, \{4\}, \{5\}\}$ *from the* Grouper. *We now illustrate how the final estimates* $\hat{C}_5$ *would have been produced using MedianSmoother as* Smoother. *Recall that each* $\hat{c}_t$ *is released in real-time based on* P_t*, the groups at time* t*, and not the final grouping* P_5*. At time 1,* $P_1 = \{\{1\}\}$ *and the last group is* $G = \{1\}$*,* $\hat{c}_1 = median\{5.6\} = 5.6$*. At time 2, the last group is* $G = \{1, 2\}$*,* $\hat{c}_2 = median\{5.6, 4.4\} = 5$*. At time 3, the last group is* $G = \{1, 2, 3\}$*,* $\hat{c}_3 = median\{5.6, 4.4, 6.7\} = 5.6$*. At time 4,* $P_4 = \{\{1, 2, 3\}, \{4\}\}$ *and the last group is* $G = \{4\}$*, so* $\hat{c}_4 = median\{9.5\} = 9.5$*. At time 5, the last group* $G = \{5\}$*,* $\hat{c}_5 = median\{10.2\} = 10.2$*. Thus, the final estimates are* $\hat{C}_5 = \{5.6, 5, 5.6, 9.5, 10.2\}$.

5.3.4 Error analysis of smoothing

We now formally analyze how the online grouping and post-processing smoother may help for improving the output utility.

Theorem 18. *Suppose one group* G *in the resulting partition from* Grouper *contains* n *indexes,* $i + 1, i + 2, \ldots, i + n$*. Assume that* $\hat{C}$ *is produced using AverageSmoother*

as the Smoother. *Then* $\hat{C}[G]$*, the resulting estimate for group* G*, will have lower expected error than* $\tilde{C}[G]$*, formally stated as*

$$\mathbb{E}\left\|\hat{C}[G] - C[G]\right\|_2 \leqslant \mathbb{E}\left\|\tilde{C}[G] - C[G]\right\|_2$$

provided that the deviation of $C[G]$ *satisfies*

$$dev(C[G]) \leqslant \frac{\sqrt{2(n - \ln n - 1)}}{(1 + \ln(n-1))\epsilon_p}$$

Proof. In terms of $\tilde{C}$ which are generated from *Perturber* by applying *Laplace Mechanism*, we have $\mathbb{E}\left\|\tilde{C}[G] - C[G]\right\|_2 = n \times \frac{2}{\epsilon_p^2}$. At each time $i+k$, $G = \{i+1, \ldots, i+k\}$. By using AverageSmoother, $\hat{c}_{i+k} = \frac{\tilde{c}_{i+1} + \cdots + \tilde{c}_{i+k}}{k}$. Then we have

$$\begin{aligned}
\mathbb{E}\left\|\hat{C}[G] - C[G]\right\|_2 &= \mathbb{E}\big(\sum_{k=1}^{n}(\frac{\tilde{c}_{i+1} + \cdots + \tilde{c}_{i+k}}{k} - c_{i+k})^2\big) \\
= \quad & \sum_{k=2}^{n}(avg_k - c_{i+k})^2 + \sum_{k=1}^{n}\frac{1}{k} \times \frac{2}{\epsilon_p^2} \\
< \quad & (\ln n + 1) \times \frac{2}{\epsilon_p^2} + \sum_{k=2}^{n}(avg_k - c_{i+k})^2 \\
\leqslant \quad & (\ln n + 1) \times \frac{2}{\epsilon_p^2} + \sum_{k=2}^{n}(|avg_k - avg_n| + |avg_n - c_{i+k}|)^2 \\
\leqslant \quad & (\ln n + 1) \times \frac{2}{\epsilon_p^2} + (\sum_{k=2}^{n}|avg_k - avg_n| + \sum_{k=2}^{n}|avg_n - c_{i+k}|)^2 \\
\leqslant \quad & (\ln n + 1) \times \frac{2}{\epsilon_p^2} + (\sum_{k=2}^{n}|avg_k - avg_n| + dev(C[G]))^2 \\
\leqslant \quad & (\ln n + 1) \times \frac{2}{\epsilon_p^2} + (\sum_{k=2}^{n-1}\frac{1}{k} \times dev(C[G]) + dev(C[G]))^2 \\
< \quad & (\ln n + 1) \times \frac{2}{\epsilon_p^2} + (1 + \ln(n-1))^2 dev(C[G])^2 \qquad (5.8)
\end{aligned}$$

where $avg_k = \frac{c_{i+1}+\cdots+c_{i+k}}{k}$ for $k \in [1, n]$.

Therefore, when $dev(C[G]) \leqslant \frac{\sqrt{2(n-\ln n-1)}}{(1+\ln(n-1))\epsilon_p}$, we have

$$\mathbb{E}\left\|\hat{C}[G] - C[G]\right\|_2 \leqslant \mathbb{E}\left\|\tilde{C}[G] - C[G]\right\|_2 \tag{5.9}$$

□

Theorem 18 implies that when the *Grouper* finds a group with large size but small deviation value, using the AverageSmoother can reduce the error of the estimates. Many realistic data streams are either sparse or have stable counts, which suggests that smoothing can help reduce error. Although we only theoretically analyze the AverageSmoother as the *Smoother*, we empirically compare the three different post-processing strategies on many real streams, which will be shown in Section 5.6. We find that *Grouper* often finds large groups with low deviation and MedianSmoother consistently generates the most accurate noisy streams under all settings. Thus, in our algorithms, we apply the MedianSmoother as default to be the *Smoother* to compute the final estimate for each receive data.

5.4 Multiple Query Support under PeGaSus

The previous section describes how the PeGaSus algorithm (Algorithm 15) can be used to generate $\hat{C}$, a differentially private answer to a given unit counting query C. In this section, we describe how to adapt the algorithm to answer the other queries described in Section 5.2.2, specifically the Sliding Window query and Event Monitoring queries.

A distinctive feature of our approach is that we use the same basic PeGaSus algorithm for these queries and change only the *Smoother*. This is possible because the answers to these queries can be derived from C. Our approach uses the noisy counts $\tilde{C}_t$ along with the group information P_t to then do appropriate smoothing for

Algorithm 17 *Window Sum Smoother* (WSS)

Input: $\tilde{C}_t = \tilde{c}_1, \ldots, \tilde{c}_t$, P_t, w
Output: $\hat{sw}_t$

1: $\hat{sw}_t \leftarrow 0$
2: **for** each $G \in P_t$ such that $G \cap \{t - w + 1, \ldots, t\} \neq \varnothing$ **do**
3: $\quad \hat{c} \leftarrow median\{\tilde{c}_i \mid i \in G\}$
4: $\quad \hat{sw}_t \leftarrow \hat{sw}_t + \hat{c} \times |G \cap \{t - w + 1, \ldots, t\}|$
5: **end for**
6: Return $\hat{sw}_t$

the specific query. Recall that the *Smoother* does not take private data as input and only post-processes the outputs of the differentially private *Perturber* and *Grouper* subroutines. Therefore, we can swap out *Smoother* without impacting the privacy guarantee. An added benefit of this approach is that we can *simultaneously* support all three kinds of queries – counting, sliding window, and event monitors – all using a single privacy budget.

For **sliding window queries**, we propose a new post-processing strategy called *Window Sum Smoother* (WSS), which is shown in Algorithm 17. The basic idea is that, at time t, for every $c_{t'}$ contained in the sliding window ($t' \in [t - w + 1, t]$), we use the MedianSmoother to compute an estimate $\hat{c}_{t'}$ for the count at time t'. To compute the answer to the sliding window query, we simply sum up the counts within the window w.

Note that this is subtly different from the approach described in Section 5.3 because the estimate $\hat{c}_{t'}$ for some t' in the sliding window is based on its group G which may include counts received after t' and up to time t. Thus, *Window Sum Smoother* may provide a better estimate than just using MedianSmoother as described in Section 5.3.3. We make this comparison empirically in Section 5.6 and we also compare against the state of the art technique for sliding window queries [4].

For **event monitoring queries**, the *Smoother* depends on the particular event monitor. For detecting jumps or drops, since we need the count at time t and the count received at time $t - w + 1$, we simply use the MedianSmoother in the

experiment. Actually, we may do better by also smoothing $\tilde{c}_{t-w+1}$ using its group G as defined at time t. For detecting low signal points, we need a sliding window query at each timestamp. Thus we use the *Window Sum Smoother*. Once the counts have been appropriately smoothed, we pass the estimated counts into the event monitor functions B and f to generate an event stream (as described in Section 5.2.2).

5.5 Query on a Hierarchy of Aggregated States

In this section, we describe an extension to PeGaSus to support queries on multiple target states as well as aggregations of states. Recall from Section 5.2.2 that the analyst can request queries on a set of states $\{s_1, \ldots, s_m\} \subseteq \mathcal{S}$ and can also ask queries about aggregations of states. We focus in particular on the setting in which the analyst has specified a hierarchy of aggregations AGG and a query on each $agg \in AGG$.

For ease of presentation, we describe our approach assuming that the analyst has requested a unit counting query on each $agg \in AGG$. However, our approach extends easily to sliding windows and event monitoring using an extension analogous to what was described in Section 5.4.

5.5.1 Hierarchical-Stream PGS

First, we observe that we can answer the queries over hierarchical aggregations by simply treating each aggregation as a separate stream and running any single-stream algorithm on each input stream. Therefore, our first solution is to run PeGaSus on each stream, an algorithm we refer to as *Hierarchical-Stream PeGaSus* (HS-PGS). Formally, given a set of aggregations AGG and a corresponding set of input streams $C(agg) = c_1(agg), c_2(agg), \ldots$ for each $agg \in AGG$, the algorithm HS-PGS executes $PeGaSus(C(agg), \epsilon/h)$ for each $agg \in AGG$ where h is the height of AGG.

Theorem 19. Hierarchical-Stream PeGaSus *satisfies ϵ-differential privacy.*

Proof. The proof follows from (a) the privacy guarantee of PeGaSus (Theorem 15), (b) parallel composition of differential privacy across each level of the hierarchy, and (c) the sequential composition of differential privacy between the h hierarchy levels. □

5.5.2 Hierarchical-Stream PGS With Pruning

We next describe an enhancement of *Hierarchical-Stream PeGaSus* that is designed to lower error when many of the input streams in the aggregation hierarchy are sparse.

The hierarchy implies a monotonicity constraint on the counts in the streams: the counts cannot decrease as one moves up the hierarchy. More formally, for any $agg_1, agg_2 \in AGG$ such that $agg_1 \subset agg_2$, then for every time step t in streams $C(agg_1)$ and $C(agg_2)$, it must be that $c_t(agg_1) \leqslant c_t(agg_2)$. Therefore, if the aggregated stream $C(agg_2)$ is observed to have a "small" count at time t, all the aggregated streams $C(agg_1)$ will also have small counts at time t if $agg_1 \subset agg_2$. In such cases, it will be wise to prune the count $c_t(agg_1)$ to be zero rather than consume privacy budget trying to estimate a smaller count. Pruning small counts also has the benefit that privacy budget that would have been spent on these counts can be saved and spent on non-pruned counts.

We use this pruning idea to modify our approach as follows. At each time step, the hierarchy is traversed and streams with small counts at this time step are pruned; any streams that remain unpruned are fed into the usual PeGaSus algorithm. Note that pruning only affects the current time step; a pruned stream at time t may become unpruned at time $t + 1$.

Algorithm 18 presents our solution, which is called *Hierarchical-Stream PeGaSus with Pruning*. The function PRUNE (lines 1-17) describes a differentially private algorithm for pruning the hierarchy. This function is based on the idea of the *Sparse*

Vector Technique [19]. Essentially, it checks each aggregation $agg \in AGG$ from level 1 to level h. If the current agg has been pruned, all its children are automatically pruned. Otherwise, we compare this aggregation's current count, $c_t(agg)$, against a user-specified threshold β (line 8). If the count is below threshold, it prunes all the children of agg. Further, the privacy budget that would have been spent on the descendants is saved (line 10). To ensure privacy, Laplace noise is added to both the count $c_t(agg)$ and the threshold β.

The *Hierarchical-Stream PeGaSus with Pruning* algorithm itself is described on lines 18-29. At each time step, it calls PRUNE. Then, for each aggregation, if it has been pruned, it simply outputs a count of 0 (line 21). Otherwise, it applies the PeGaSus algorithm to the stream (lines 23-25) where the privacy budget passed to *Perturber* and *Grouper* is adjusted based on what has been pruned and the height of the tree.

In our implementation, there is a small modification to *Grouper* in that we skip over past time steps that were pruned and therefore consider potentially non-contiguous groups.

To prove the privacy guarantee of Algorithm 18, we analyze the PRUNE function.

Theorem 20. *The* PRUNE *function in Algorithm 18 satisfies ϵ-differential privacy.*

Proof. For any two neighboring source streaming datasets, they derive the neighboring multiple streams that only differ by 1 at one timestamp on one single stream. In terms of AGG, there is at most one list of aggregations at different levels that cover this single stream. Function Prune can be treated as an implementation of the *Sparse Vector Technique* from [19], which ensures ϵ-differential privacy. □

Theorem 21. *Algorithm 18 satisfies ϵ-differential privacy.*

Proof. For any two neighboring source streaming datasets, they derive the neighboring multiple streams that only differ by 1 at one timestamp on one single stream.

Algorithm 18 *Hierarchical-Stream PeGaSus with Pruning* (PHS-PGS)

Input: AGG with h levels, streams $C(agg)$ for each $agg \in AGG$, privacy budget $\epsilon = \epsilon_{pr} + \epsilon_p + \epsilon_g$, threshold β

Output: Streams $\hat{C}(agg)$ for each $agg \in AGG$

function PRUNE(AGG, $c_t(agg)$ for each $agg \in AGG$, ϵ, β)
  $Pruned \leftarrow \emptyset$, $\epsilon_{AGG} \leftarrow \emptyset$
  for level $i = 1$ to h **do**
    for each $agg \in AGG$ at level i **do**
      if $agg \in Pruned$ **then**
        Add every child of agg to $Pruned$
        Add $\epsilon_{agg} = 0$ to ϵ_{AGG}
      else if $(c_t(agg) + Lap(2/\epsilon)) < (\beta + Lap(2/\epsilon))$ **then**
        Add every child of agg to $Pruned$
        Add $\epsilon_{agg} = h - i + 1$ to ϵ_{AGG}
      else
        Add $\epsilon_{agg} = 1$ to ϵ_{AGG}
      end if
    end for
  end for
  Return $Pruned$, ϵ_{AGG}
end function
for each timestamp t **do**
  $Pruned, \epsilon_{AGG} \leftarrow$ PRUNE($AGG, c_t(agg)$ for each $agg \in AGG$, ϵ_{pr}, β)
  for each $agg \in AGG$ **do**
    if $agg \in Pruned$ **then** $\hat{c}_t(agg) \leftarrow 0$
    else
      $\tilde{c}_t(agg) \leftarrow Perturber(c_t(agg), \frac{\epsilon_{agg}(\epsilon_p+\epsilon_g)}{h} - \frac{\epsilon_g}{h})$
      $P_t \leftarrow Grouper(C_t(agg), P_{t-1}, \frac{\epsilon_g}{h})$
      $\hat{c}_t(agg) \leftarrow Smoother(\tilde{C}_t(agg), P_t)$
    end if
    Output $\hat{c}_t(agg)$
  end for
end for

Calling function Prune at every timestamp with ϵ_{pr} privacy budget ensures ϵ_{pr}-differential privacy based on Theorem 20. For the pruned aggregated streams, we set the count to be 0, which will not leak any information. For the non-pruned aggregated streams, we apply the similar algorithm with Algorithm 15 to compute noisy counts at each timestamp. Based on the analysis of Theorem 15 and sequential composition of differential privacy on the aggregation set with h levels, releasing noisy counts at every timestamp will satisfy $\epsilon_p + \epsilon_g$-differential privacy. Thus, Algorithm 18 satisfies $\epsilon = \epsilon_{pr} + \epsilon_p + \epsilon_g$-differential privacy. □

Table 5.1: An overview of the streams derived from real WiFi access points connection traces.

Stream Name	# of target states	Length	Total count
Low_5	1	57901	2846
Med_5	1	57901	101843
High_5	1	57901	2141963
Multi_5	128	20000	646254

5.6 Evaluation

In this section we evaluate the proposed algorithms on real data streams for a variety of workloads. We design the following experiments:

1. We evaluate answering unit counting queries on data streams with a single target state.

2. We evaluate answering sliding window queries on data streams with a single target state.

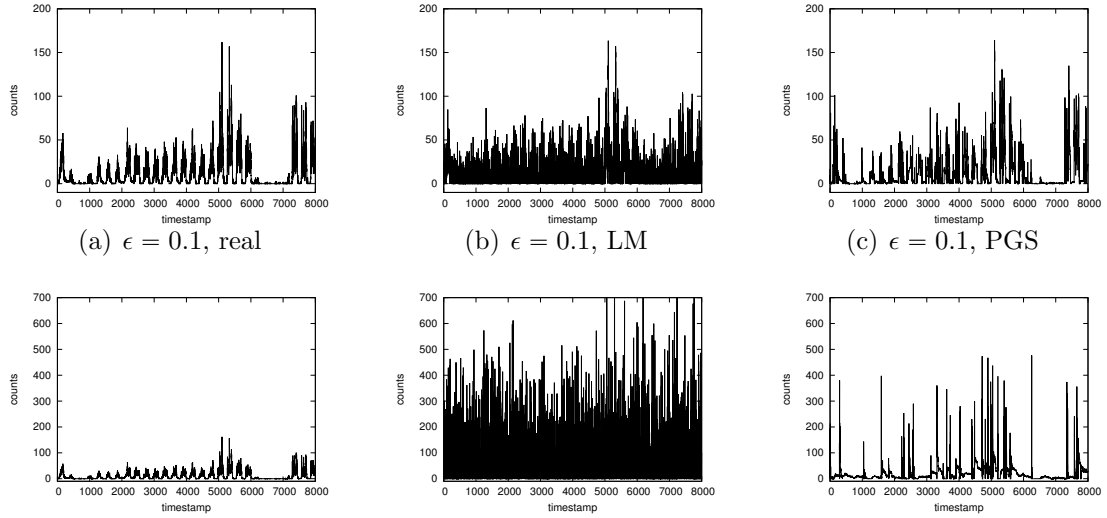
3. We evaluate event monitoring (detecting jumping and dropping points as well as low signal points) on data streams with a single target state.

4. We evaluate answering unit counting queries on a collection of hierarchical aggregations using data streams with multiple target states.

Dataset: Our source data comes from real traces, taken over a six month period, from approximately 4000 WiFi access points (AP) distributed across the campus of a large educational institution. Each time a user device makes a successful connection to any AP, a tuple (u, s, t) will be recorded in the trace, meaning user u successfully connected to AP s at time t. We set the time interval to be 5 minutes and the derived stream, for any single AP s, is denoted $C(s) = c_1(s), c_2(s), \ldots$, and reports the number of users who successfully connected to AP s in each time interval. For the evaluation of data streams with a single target state, we pick three representative

APs with different loads (called "Low_5", "Med_5" and "High_5"). For the evaluation of data streams with multiple target states, we randomly pick 128 APs (called "Multi_5"), and generate a hierarchical aggregation of these states as a binary tree of height 8. An overview of the corresponding derived streams is shown in Table 5.1, where *Length* refers to the number of counts, each representing the number of successful connections in a 5 minute interval. *Total count* is the sum of all counts in the stream.

5.6.1 Unit counting query on a single target state



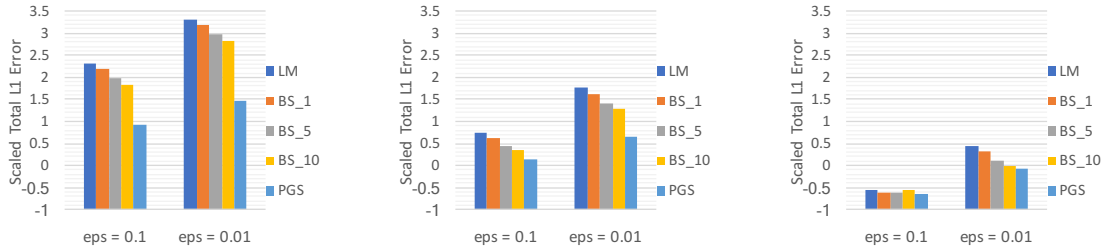
(a) $\epsilon = 0.1$, real (b) $\epsilon = 0.1$, LM (c) $\epsilon = 0.1$, PGS

(d) $\epsilon = 0.01$, real (e) $\epsilon = 0.01$, LM (f) $\epsilon = 0.01$, PGS

FIGURE 5.2: Stream visualizations of the "High_5" for the first 8000 time steps.

Answering a unit counting query on data streams with a single target state is equivalent to releasing a private version of the entire stream. Figure 5.2 shows visually the real and the privately generated streams for the first 8000 time steps of stream "High_5" under $\epsilon = 0.1$ and 0.01. We compare our PeGaSus algorithm with the *Laplace Mechanism* (LM). In PeGaSus, we set $\epsilon_p = 0.8 \times \epsilon$ and $\epsilon_g = 0.2 \times \epsilon$. We set $\theta = \frac{5}{\epsilon_g}$ in our *Grouper*. We use the MedianSmoother method as the *Smoother*. In each figure of a noisy stream, we truncate the negative counts to zero. Clearly,

PeGaSus produces private streams that are more accurate when visualized.



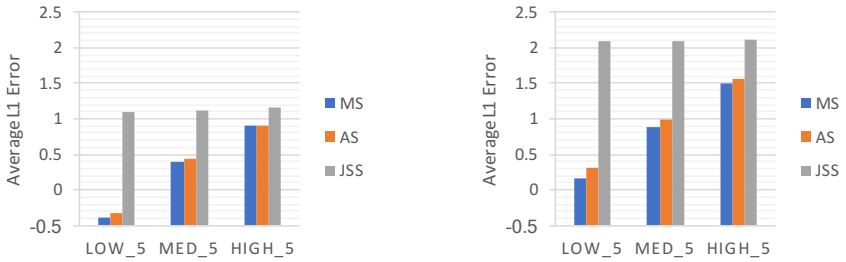
(a) Low_5 (b) Med_5 (c) High_5

FIGURE 5.3: Error for the unit counting query on streams with a single target state. The y-axis reports log_{10} of the scaled total L_1 error.

We also quantitively evaluate the error of answering the unit counting query by using the scaled total L_1 error. For any input data stream $C_t = \{c_1, c_2, \ldots, c_t\}$ and the output private stream $\hat{C}_t = \{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_t\}$, the scaled total L_1 error is defined to be $\frac{\sum_{i=1}^{t} |c_i - \hat{c}_i|}{\sum_{i=1}^{t} c_i}$.

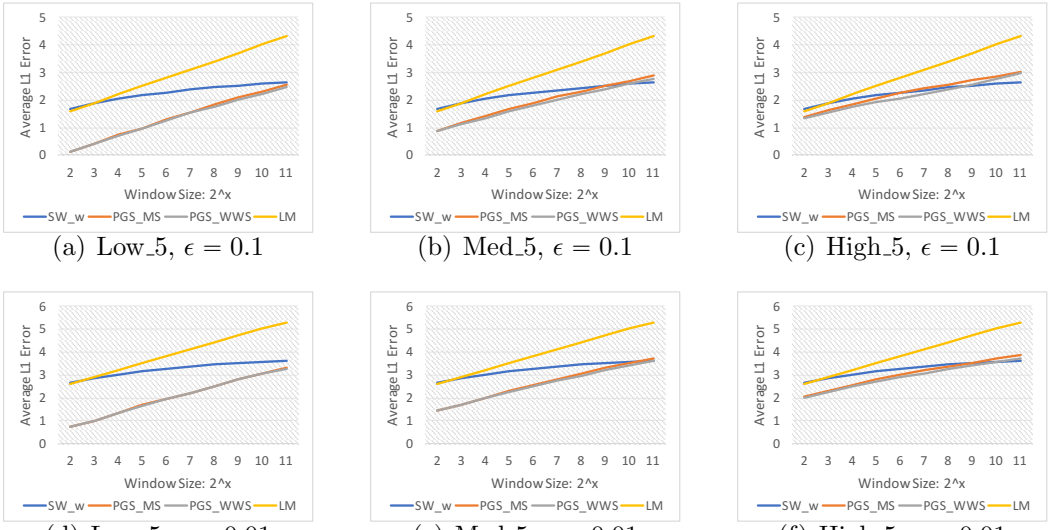
Figure 5.3 reports the evaluation results on the data streams with the single target state from Table 5.1. In each figure, LM means *Laplace Mechanism*, and PeGaSus is PeGaSus with the MedianSmoother as *Smoother*. In addition, as a simple comparison method, we use BS_t to mean a method where we do backward smoothing of results from *Laplace Mechanism*. Given a backward smoothing time k, for any $t \geqslant k$, $\hat{c}_k$ is updated as $\frac{\sum_{i=t-k}^{t} \hat{c}_i^{LM}}{k+1}$, where $\hat{c}_i^{LM}$ is the output from *Laplace Mechanism*. Each bar in the figures reports the average L_1 error in terms of the unit counting query at all time steps. The value is the average of 20 random trials. PeGaSus consistently performs the best on all the data streams and under both ϵ settings.

Next, we compare the impact of the three different smoothing strategies – JSSmoother, MedianSmoother, and AverageSmoother – in terms of answering the unit counting query. The results are shown in Figure 5.4, where "MS" is MedianSmoother, "AS" is AverageSmoother and "JSS" is JSSmoother. Each bar reports the $\log_{10}$ value of the average L_1 error under 20 trials in terms of each smoothing strategy. We can



(a) $\epsilon = 0.1$ (b) $\epsilon = 0.01$

FIGURE 5.4: Smoothing strategy comparison for unit workload on various streams.



(a) Low_5, $\epsilon = 0.1$ (b) Med_5, $\epsilon = 0.1$ (c) High_5, $\epsilon = 0.1$

(d) Low_5, $\epsilon = 0.01$ (e) Med_5, $\epsilon = 0.01$ (f) High_5, $\epsilon = 0.01$

FIGURE 5.5: Error for the sliding window queries. The x-axis represents the window size as 2^x. The y-axis reports $\log_{10}$ of the average L_1 error.

see JSSmoother is not a good smoothing strategy. Both MedianSmoother and AverageSmoother are good smoothing strategies, and MedianSmoother is consistently better than AverageSmoother for all data streams and ϵ settings.

5.6.2 Sliding window query on a single target state

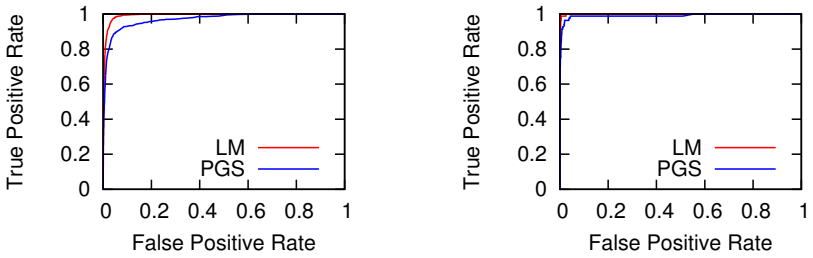
Next we evaluate the sliding window query with window size w. Figure 5.5 presents the results. Each point shows the average L_1 error for answering the sliding window query with size $w = 2^x$ at all timesteps and the value is the average of 20

trials. In each sub-figure, LM represents using *Laplace Mechanism* to answer the unit counting query first, then computing the sliding window query based on the noisy unit counting query answers. SW_w is the state-of-the-art data independent algorithm for computing the sliding window query in terms of a fixed window size w [8]. SW_w generates binary trees on every consecutive window of w counts and perturbs each node query of each binary tree. Then, any sliding window query with size w can be derived as the sum of the prefix and suffix of any two neighboring binary trees. PGS_MS and PGS_WWS are variants of PeGaSus with MedianSmoother as *Smoother* and Algorithm 17 as *Smoother*.

As shown in the figure, LM introduces excessive error. PGS_WWS always performs better than PGS_MS demonstrating the benefit of using a different *Smoother* for this workload. PGS_WWS computes more accurate sliding window queries when w is not greater than 256 compared with the state of the art algorithm SW_w. When the window size becomes larger, SW_w becomes better because SW_w is designed specifically for the sliding window query with window size w while our PeGaSus may introduce a large bias into a large sliding window query. We emphasize that PeGaSus can simultaneously support all sliding window queries instead of having to split the privacy budget and design algorithms for each sliding window workload with a fixed window size.

5.6.3 Event monitoring on a single target state

In this experiment we consider event monitoring queries on the "High_5" data stream. Figure 5.6 displays the ROC curves for detecting jumping and dropping points on streams with a single target state. In each sub-figure, LM represents using *Laplace Mechanism* to generate a noisy stream and doing event monitoring on the noisy stream. PeGaSus uses the MedianSmoother as *Smoother*. We use a fixed window size w and threshold δ to compute the ground truth in terms of the real stream. We



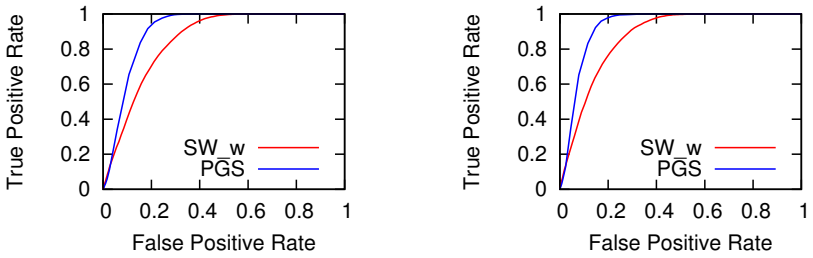
(a) $\epsilon = .1, w = 10, \delta = 100$ (b) $\epsilon = .1, w = 10, \delta = 150$

(c) $\epsilon = .01, w = 10, \delta = 100$ (d) $\epsilon = .01, w = 10, \delta = 150$

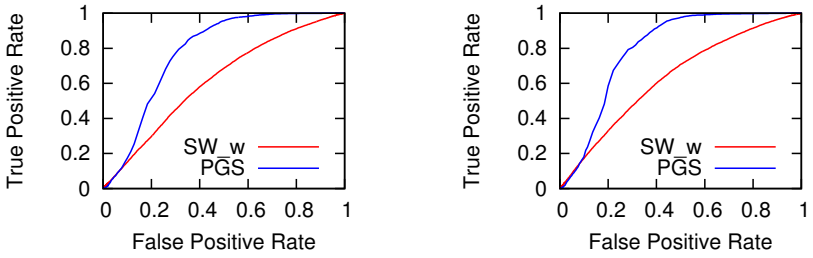
FIGURE 5.6: ROC curve for detecting jumping and dropping points on stream High_5 under different settings.

vary the threshold from 0 to 1000 to do the private event monitoring and compute the corresponding "True Positive Rate" and "False Positive Rate". As shown in the figures, for all w and δ settings, when ϵ is large (= 0.1), both LM and PeGaSus perform very well and LM is slightly better than PeGaSus. However, when ϵ becomes smaller (=0.01), PeGaSus performs much better than LM. Similar results are shown on different w settings and will not be shown due to page limits.

Figure 5.7 shows the ROC curves for detecting low signal points. Since determining the low signal points requires computing the sliding window queries, we compare SW_w from [4] with our proposed PeGaSus with Algorithm 17 as *Smoother*. We use fixed window size w and threshold δ to compute the ground truth in terms of the real stream. We vary the threshold from -4000 to 4000 to do the private event monitoring and compute the corresponding "True Positive Rate" and "False Positive Rate". As shown in the figure, for all the ϵ, w and δ settings, PeGaSus always outperforms SW_w. Similar results are shown on different w and δ settings, which are not shown



(a) $\epsilon = .1, w = 8, \delta = 5$ (b) $\epsilon = .1, w = 8, \delta = 10$



(c) $\epsilon = .01, w = 8, \delta = 5$ (d) $\epsilon = .01, w = 8, \delta = 10$

FIGURE 5.7: ROC curve for detecting low signal points on stream High_5 under different settings.

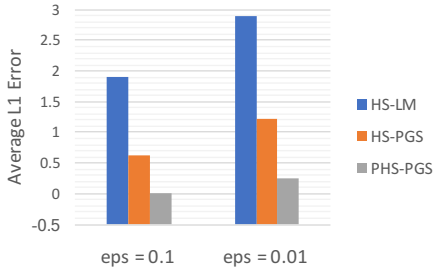


FIGURE 5.8: Unit counting query on hierarchical aggregated streams among multiple target states.

due to page limits

5.6.4 Unit counting query on hierarchical aggregated streams; multiple target states

We evaluate our proposed algorithms in Section 5 for answering unit counting query on a set of hierarchical aggregated streams among multiple target states. We use the "Multi_5" stream from Table 5.1 and consider a set of hierarchical aggregations *AGG* as a binary tree on the total number of states (128 in our case). The total

levels of AGG is $\log_2(128) + 1 = 8$.

Figure 5.8 shows the results under two different ϵ settings. Each bar reports the $\log_{10}$ value of the average L_1 error in terms of the unit counting query on every aggregated stream at all timesteps. The value is also the average of 20 trials. In the figure, HS-LM represents using *Laplace Mechanism* on answering the unit query on each aggregated stream with $\frac{\epsilon}{h}$ privacy budget, where h is the number of levels of AGG. HS-PGS is using our proposed PeGaSus with MedianSmoother as *Smoother* on each aggregated stream. PHS-PGS is the proposed *Hierarchical-Stream PeGaSus*. As shown in the figure, both HS-PGS and PHS-PGS reduce the average L_1 error of answering unit counting query by 1 to 2 orders of magnitude compared with the data-independent HS-LM. In addition, by doing the pruning, the utility of the results are further improved. The average L_1 error of HS-LM is over 78x (445x) times the error of PHS-PGS, and the error of HS-PGS is over 4x (9x) times the error of PHS-PGS for $\epsilon = 0.1$ (0.01).

5.7 Conclusion

We presented a new differentially private algorithm that can simultaneously answer a variety of continuous queries at multiple resolutions on real time data streams. Our novel *Perturber*, data-adaptive *Grouper* and query specific *Smoother* approach helps release counting, sliding window and event monitoring queries with low error even on sparse streams. Specifically, the design of *Grouper* makes use of the idea of *Sparse Vector Technique*. Our empirical results show that our approach outperforms state-of-the-art solutions specialized to individuals queries.

6

Related Works

We discuss the related works from various domains below.

Sparse Vector Technique.

The idea of *Sparse Vector Technique* was first proposed in [22]. After that, both previous works from [51] and [28] made use of *Sparse Vector Technique* for answering queries in an interactive setting.

Multiple variants of *Sparse Vector Technique* [14, 37, 55] were introduced in some previous works in order to eliminate the dependence of the privacy cost in terms of the number of "positive" queries. [37] proposed a variant of *Sparse Vector Technique* for selecting frequent itemsets under differential privacy. [14] designed another variant of *Sparse Vector Technique* for differentially private generation of high-dimensional datasets. Another different variant of *Sparse Vector Technique* was proposed in [55] in order to achieve privacy-preserving feature selections. The only difference among these variants is the amount of noise injected to either threshold or each testing query. In this dissertation, we have shown that all these variants actually do not ensure differential privacy.

There are some following works after our work on critically analyzing the in-

correctness of the previous *Sparse Vector Technique* variants. Both [65] and [42] discussed the violations of differential privacy of the variants. There is also further analysis about optimizing budget allocation for *Sparse Vector Technique* shown in [42].

Private Learning and Evaluation.

Private models for regression and classification has been a popular area of exploration for privacy research. Previous works have produced differentially private training algorithms for Naive Bayes classification [58], decision trees [26], logistic and linear regression [13, 52, 59, 63, 64]. These works only output the noisy model parameters.

Apart from classifier training, Chaudhuri el al. [12] present a generic algorithm for differentially private parameter tuning and model selection. However, this work does not assume a blackbox classifier, and makes strong stability assumptions about the training algorithm. In contrast, our algorithms are classifier agnostic. Additionally Thakurtha et al. [57] present an algorithm for model selection again assuming strong stability assumptions about the model training algorithm. We would like to note that the work in these paper is in some sense orthogonal to the feature selection algorithms we present, and can be used in conjunction with the results in the paper (for instance, to choose the right threshold τ or the right number of features to select). Sheffet's algorithm [52] additionally outputs confidence intervals for the model parameters. No prior work considers model diagnostics.

ROC curves are used to quantify the prediction accuracy of binary classifiers. However, directly releasing the ROC curve may reveal the sensitive information of the input dataset [43]. Kairouz et al. provided inequalities governing the trade-off between false-alarm and missed-detection in [32]. Boyd et al. computed the AUC and a variant of ROC curve called symmetric binormal ROC curve by using smooth sensitivity [5]. Chaudhuri et al. [12] proposes a generic technique for evaluating a

classifier on a private test set. However, they assume that the global sensitivity of the evaluation function is low (a small constant). Their work is inapplicable since the sufficient statistics for generating the ROC curve (the set of true and false positive counts) have a high global sensitivity. The idea of releasing perturbed residual plots was first suggested by [48], who suggested that remote query systems returning regression parameter estimates also return residual plots with random noise added to the residuals from the confidential regression; see also [46]. The noise distribution did not satisfy differential privacy, or any other formal guarantee. Similar ideas were used for logistic regressions by [49].

Private Stream Processing.

Privacy-preserving data release has been studied for the past decade. Lots of differentially private techniques [1, 15, 21, 29, 30, 38, 39, 47, 60–62, 66] have been proposed for counting queries for static datasets.

Dwork adapted differential privacy to a continual observation setting [18], which focused on a 0/1 stream and proposed a cascading buffer counter for counting the number of 1s under event-level differential privacy. Mir et al. studied pan-private algorithms for estimating distinct count, moments and the heavy-hitter count on data streams in [44], which preserves differential privacy even if the internal memory of the algorithms is compromised. Chan et al. studied the application of monitoring the heavy hitters across a set of distributed streams in [11].

Some prior works focus on continual real-time release of aggregated information from streams [4, 8, 10, 25, 33]. Fan et al. proposed *Fast*, an adaptive system to release real-time aggregate statistics under differential privacy by using sampling and filtering. Their algorithm is based on a different privacy model, user-level differential privacy. In [10], Chan et al. used the same privacy model with us. But they focus on a different single task that releasing prefix sums in terms of the streaming data counts. For the input stream with known bounded length T, it generates a binary

tree defined on the stream and perturbs the node counts of the binary tree. Then each prefix range query is computed based on a set of perturbed nodes. For streams with unbounded length, it spends half of the privacy budget on perturbing every range query between $2^i + 1$ and 2^{i+1}. Then it generates a binary tree on every sub-stream between $2^t + 1$ and 2^{t+1} for each $i = 1, 2, \ldots,$ and perturbs the node counts with the remaining half of the privacy budget. Any prefix sum at timestamp k, where $2^i <= k < 2^{i+1}$, can be computed based on the range queries between $2^i + 1$ and 2^{i+1} for all $i \leqslant t$ and the noisy nodes from the binary tree between $2^t + 1$ and 2^{t+1}. The authors proved that the error of each release at timestamp t is $O(\log(t))$.

Bolot et al. used the same privacy model with us and proposed an algorithm for releasing sliding window query on data streams with a fixed window size w. The basic idea is also to generate binary trees on every consecutive w data points and perturbs each node of the binary trees. The sliding window query at each timestamp can be derived as the sum of the suffix and prefix of two consecutive binary trees. This is the state-of-the art algorithm for releasing sliding window query answers. But the algorithm is designed for any fixed window size, which means we may split the budget for answering multiple sliding window queries with different window sizes. Cao et al. [8] studied on a different task of answering a set of special prefix sum queries with at timestamps in the form of $j \times s$, where s is the step size chosen from some pre-defined step size set. The proposed algorithms sample some step sizes from the step size set and then only perturb the window queries in terms of the chosen step sizes. Then the prefix range query from the workload can be computed by composition of these perturbed window queries. Kellaris et al. first proposed another privacy model called w-event differential privacy [33], which is a balance between user-level and event-level differential privacy and designed algorithms for releasing private data under w-event differential privacy.

Finally, Dwork et al. proposed a differentially private online partition algorithm

for counting under continual observations [24]. Like our *Grouper* module, this algorithm also employs the idea of *Sparse Vector Technique* [19]. However, our *Grouper* differs from Dwork et al. in the following important ways: (1) their algorithm is designed for computing partitions such that total counts of each partition are similar while ours is to find groups such that the elements in each group share similar counts. (2) Using deviation function can help the *Grouper* to detect contiguous intervals in the stream that have a stable value (even if the values are high). On the other hand, using the total only lets us group together intervals that have counts close to zero.

7

Conclusion

In this dissertation, we discuss the use of one popular differentially private primitive, *Sparse Vector Technique*, for designing new techniques to solve practical problems under differential privacy.

In the first topic described in Chapter 3, we critically analyze multiple variants of *Sparse Vector Technique* proposed in the previous work. We show that these variants do not ensure differential privacy, and adversaries can even reconstruct real database by accessing to these variants. We also introduce another generalized version of *Sparse Vector Technique*.

In the second topic shown in Chapter 4, we make use of the original *Sparse Vector Technique* primitive, proposing multiple new techniques for achieving regression diagnostics under differential privacy. Specifically, we propose *PriRP* for differentially private evaluation of the model fit for linear regression. We also propose two algorithms *PriROC* and *PriBRP*, achieving the evaluation of logistics regression under differential privacy in terms of the predictive power as well as the model fit.

In the third topic displayed in Chapter 5, we introduce a novel algorithm, PeGaSus, for achieving privacy-preserving streaming data query answering by using

Sparse Vector Technique as a key primitive. Compared with previous works, the new PeGaSus algorithm is data-adaptive, where the injected noise is tuned in terms of the input streaming data. PeGaSus can also simultaneously support multiple workloads without splitting the privacy budget and beat the previous state-of-the-art solutions designed for each specific workload.

We provide comprehensive evaluations for our proposed new techniques. Our works solve several challenging problems and make differential privacy practical on multiple domains.

Bibliography

[1] Acs, G., Castelluccia, C., and Chen, R. (2012), "Differentially Private Histogram Publishing through Lossy Compression," in *2012 IEEE 12th International Conference on Data Mining*, pp. 1–10.

[2] Almeida, T. A., Hidalgo, J. M. G., and Silva, T. P. (2012), "Towards SMS Spam Filtering: Results under a New Dataset," *International Journal of Information Security Science*, pp. 1–18.

[3] Atzori, L., Iera, A., and Morabito, G. (2010), "The Internet of Things: A Survey," *Comput. Netw.*, 54, 2787–2805.

[4] Bolot, J., Fawaz, N., Muthukrishnan, S., Nikolov, A., and Taft, N. (2013), "Private Decayed Predicate Sums on Streams," in *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pp. 284–295, New York, NY, USA, ACM.

[5] Boyd, K., Lantz, E., and Page, D. (2015), "Differential Privacy for Classifier Evaluation," in *ACM Workshop on Artificial Intelligence and Security*, pp. 15–23.

[6] Bun, M., Nissim, K., Stemmer, U., and Vadhan, S. (2015), "Differentially Private Release and Learning of Threshold Functions," in *FOCS*, pp. 634–649.

[7] Calandrino, J. A., Kilzer, A., Narayanan, A., Felten, E. W., and Shmatikov, V. (2011), ""You Might Also Like:" Privacy Risks of Collaborative Filtering," in *2011 IEEE Symposium on Security and Privacy*, pp. 231–246.

[8] Cao, J., Xiao, Q., Ghinita, G., Li, N., Bertino, E., and Tan, K.-L. (2013), "Efficient and Accurate Strategies for Differentially-private Sliding Window Queries," in *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pp. 191–202, New York, NY, USA, ACM.

[9] Cao, Y., Yoshikawa, M., Xiao, Y., and Xiong, L. (2017), "Quantifying Differential Privacy under Temporal Correlations," *ICDE*.

[10] Chan, T.-H. H., Shi, E., and Song, D. (2011), "Private and Continual Release of Statistics," *ACM Trans. Inf. Syst. Secur.*, 14, 26:1–26:24.

[11] Chan, T.-H. H., Li, M., Shi, E., and Xu, W. (2012), "Differentially Private Continual Monitoring of Heavy Hitters from Distributed Streams," in *Proceedings of the 12th International Conference on Privacy Enhancing Technologies*, PETS'12, pp. 140–159, Berlin, Heidelberg, Springer-Verlag.

[12] Chaudhuri, K. and Vinterbo, S. A. (2013), "A Stability-based Validation Procedure for Differentially Private Machine Learning," in *Advances in Neural Information Processing Systems*, pp. 2652–2660.

[13] Chaudhuri, K., Monteleoni, C., and Sarwate, A. D. (2011), "Differentially Private Empirical Risk Minimization," *Journal of Machine Learning Research*, 12, 1069–1109.

[14] Chen, R., Xiao, Q., Zhang, Y., and Xu, J. (2015), "Differentially Private High-Dimensional Data Publication via Sampling-Based Inference," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 129–138, New York, NY, USA, ACM.

[15] Cormode, G., Procopiuc, C., Srivastava, D., Shen, E., and Yu, T. (2012), "Differentially Private Spatial Decompositions," in *2012 IEEE 28th International Conference on Data Engineering*, pp. 20–31.

[16] Dinur, I. and Nissim, K. (2003), "Revealing Information While Preserving Privacy," in *Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '03, pp. 202–210, New York, NY, USA, ACM.

[17] Dwork, C. (2006), "Differential privacy," in *in ICALP*, pp. 1–12, Springer.

[18] Dwork, C. (2010), "Differential Privacy in New Settings," in *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pp. 174–183, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics.

[19] Dwork, C. and Roth, A. (2014), "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, 9, 211–407.

[20] Dwork, C. and Yekhanin, S. (2008), "New Efficient Attacks on Statistical Disclosure Control Mechanisms," in *Proceedings of the 28th Annual Conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, pp. 469–480, Berlin, Heidelberg, Springer-Verlag.

[21] Dwork, C., McSherry, F., Nissim, K., and Smith, A. (2006), "Calibrating Noise to Sensitivity in Private Data Analysis," TCC'06, pp. 265–284, Berlin, Heidelberg, Springer-Verlag.

[22] Dwork, C., Naor, M., Reingold, O., Rothblum, G. N., and Vadhan, S. (2009), "On the Complexity of Differentially Private Data Release: Efficient Algorithms and Hardness Results," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pp. 381–390, New York, NY, USA, ACM.

[23] Dwork, C., Naor, M., Pitassi, T., and Rothblum, G. N. (2010), "Differential Privacy Under Continual Observation," in *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pp. 715–724, New York, NY, USA, ACM.

[24] Dwork, C., Naor, M., Reingold, O., and Rothblum, G. N. (2015), *Pure Differential Privacy for Rectangle Queries via Private Partitions*, pp. 735–751, Springer Berlin Heidelberg, Berlin, Heidelberg.

[25] Fan, L. and Xiong, L. (2012), "Real-time Aggregate Monitoring with Differential Privacy," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pp. 2169–2173, New York, NY, USA, ACM.

[26] Friedman, A. and Schuster, A. (2010), "Data Mining with Differential Privacy," in *KDD*, pp. 493–502, New York, NY, USA, ACM.

[27] Go, A., Bhayani, R., and Huang, L. (2009), "Twitter sentiment classification using distant supervision," *CS224N Project Report, Stanford*, pp. 1–12.

[28] Hardt, M. and Rothblum, G. N. (2010), "A multiplicative weights mechanism for privacy-preserving data analysis," in *In FOCS*, pp. 61–70.

[29] Hardt, M., Ligett, K., and McSherry, F. (2012), "A Simple and Practical Algorithm for Differentially Private Data Release," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pp. 2339–2347, USA, Curran Associates Inc.

[30] Hay, M., Rastogi, V., Miklau, G., and Suciu, D. (2010), "Boosting the Accuracy of Differentially Private Histograms Through Consistency," *Proc. VLDB Endow.*, 3, 1021–1032.

[31] Hay, M., Machanavajjhala, A., Miklau, G., Chen, Y., and Zhang, D. (2016), "Principled Evaluation of Differentially Private Algorithms Using DPBench," in *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pp. 139–154, New York, NY, USA, ACM.

[32] Kairouz, P., Oh, S., and Viswanath, P. (2015), "The Composition Theorem for Differential Privacy," in *Proceedings of the 32nd International Conference in Machine Learning(ICML)*, pp. 1376–1385.

[33] Kellaris, G., Papadopoulos, S., Xiao, X., and Papadias, D. (2014), "Differentially Private Event Sequences over Infinite Streams," *Proc. VLDB Endow.*, 7, 1155–1166.

[34] Kifer, D. and Machanavajjhala, A. (2011), "No Free Lunch in Data Privacy," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pp. 193–204, New York, NY, USA, ACM.

[35] Kotsogiannis, I., Machanavajjhala, A., Hay, M., and Miklau, G. (2017), "Pythia: Data Dependent Differentially Private Algorithm Selection," in *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pp. 1323–1337, New York, NY, USA, ACM.

[36] K.S.Ruggles, J. and R.Goeken (2010), "Integrated public use microdata series: Version 5.0," .

[37] Lee, J. and Clifton, C. W. (2014), "Top-k Frequent Itemsets via Differentially Private FP-trees," KDD '14, pp. 931–940, New York, NY, USA, ACM.

[38] Li, C., Hay, M., Miklau, G., and Wang, Y. (2014), "A Data- and Workload-Aware Query Answering Algorithm for Range Queries Under Differential Privacy," *PVLDB*, 7, 341–352.

[39] Li, N., Yang, W., and Qardaji, W. (2013), "Differentially Private Grids for Geospatial Data," in *ICDE*, pp. 757–768.

[40] Lichman, M. (2013), "UCI Machine Learning Repository," .

[41] Liu, C., Chakraborty, S., and Mittal, P. (2016), "Dependence Makes You Vulnerable: Differential Privacy Under Dependent Tuples," in *NDSS*.

[42] Lyu, M., Su, D., and Li, N. (2017), "Understanding the Sparse Vector Technique for Differential Privacy," *Proc. VLDB Endow.*, 10, 637–648.

[43] Matthews, G. and Harel, O. (2013), "An examination of data confidentiality and disclosure issues related to publication of empirical ROC curves." *Academic radiology*, 20, 889.

[44] Mir, D., Muthukrishnan, S., Nikolov, A., and Wright, R. N. (2011), "Pan-private Algorithms via Statistics on Sketches," in *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pp. 37–48, New York, NY, USA, ACM.

[45] Nissim, K., Raskhodnikova, S., and Smith, A. (2007), "Smooth sensitivity and sampling in private data analysis," in *STOC*, pp. 75–84.

[46] O'Keefe, C. M. and Good, N. M. (2009), "Regression output from a remote analysis system," *Data & Knowledge Engineering*, 68.

[47] Qardaji, W., Yang, W., and Li, N. (2013), "Understanding Hierarchical Methods for Differentially Private Histograms," *PVLDB*, 6.

[48] Reiter, J. P. (2003), "Model diagnostics for remote access servers," *Statistics and Computing*, 13, 371–380.

[49] Reiter, J. P. and Kohnen, C. N. (2005), "Categorical data regression diagnostics for remote servers," *Journal of Statistical Computation and Simulation*, 75, 889–903.

[50] Reiter, J. P., Oganian, A., and Karr, A. F. (2009), "Verification servers: Enabling analysts to assess the quality of inferences from public use data," *Computational Statistics and Data Analysis*, 53, 1475–1482.

[51] Roth, A. and Roughgarden, T. (2010), "Interactive Privacy via the Median Mechanism," in *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pp. 765–774, New York, NY, USA, ACM.

[52] Sheffet, O. (2015), "Differentially Private Least Squares: Estimation, Confidence and Rejecting the Null Hypothesis," *CoRR*, abs/1507.02482.

[53] Smith, A. (2011), "Privacy-preserving Statistical Estimation with Optimal Convergence Rates," in *STOC*, pp. 813–822.

[54] Song, S., Wang, Y., and Chaudhuri, K. (2017), "Pufferfish Privacy Mechanisms for Correlated Data," in *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pp. 1291–1306, New York, NY, USA, ACM.

[55] Stoddard, B., Chen, Y., and Machanavajjhala, A. (2014), "Differentially Private Algorithms for Empirical Machine Learning," *CoRR*, abs/1411.5428.

[56] Sweeney, L. (2000), "Simple Demographics Often Identify People Uniquely," .

[57] Thakurta, A. G. and Smith, A. (2013), "Differentially private feature selection via stability arguments, and the robustness of the lasso," in *Conference on Learning Theory*, pp. 819–850.

[58] Vaidya, J., Shafiq, B., Basu, A., and Hong, Y. (2013), "Differentially Private Naive Bayes Classification," in *International Conferences on Web Intelligence*, pp. 571–576.

[59] Wu, X., Fredrikson, M., Wu, W., Jha, S., and Naughton, J. F. (2015), "Revisiting Differentially Private Regression: Lessons From Learning Theory and their Consequences," *CoRR*, abs/1512.06388.

[60] Xiao, X., Wang, G., and Gehrke, J. (2011), "Differential Privacy via Wavelet Transforms," vol. 23, pp. 1200–1214, Piscataway, NJ, USA, IEEE Educational Activities Department.

[61] Xiao, Y., Xiong, L., Fan, L., Goryczka, S., and Li, H. (2014), "DPCube: Differentially Private Histogram Release Through Multidimensional Partitioning," *Trans. Data Privacy*, 7, 195–222.

[62] Xu, J., Zhang, Z., Xiao, X., Yang, Y., and Yu, G. (2012), "Differentially Private Histogram Publication," ICDE '12, pp. 32–43, Washington, DC, USA, IEEE Computer Society.

[63] Zhang, J., Zhang, Z., Xiao, X., Yang, Y., and Winslett, M. (2012), "Functional mechanism: regression analysis under differential privacy," *Proceedings of the VLDB Endowment*, 5, 1364–1375.

[64] Zhang, J., Xiao, X., Yang, Y., Zhang, Z., and Winslett, M. (2013), "PrivGene: Differentially Private Model Fitting Using Genetic Algorithms," in *SIGMOD*, pp. 665–676.

[65] Zhang, J., Xiao, X., and Xie, X. (2016), "PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions," in *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pp. 155–170, New York, NY, USA, ACM.

[66] Zhang, X., Chen, R., Xu, J., Meng, X., and Xie, Y. (2014), *Towards Accurate Histogram Publication under Differential Privacy*, pp. 587–595.

Biography

Yan Chen was born at July 9th, in Chonqqing, a big city located in the south west of China. He received his B.S. in Computer Science from Peking University, Beijing, China, in 2013. He also got a minor degree in Statistics from Peking University in 2013 as well. In the Fall 2013, Yan entered Duke University as a Ph.D. student in the Department of Computer Science. He worked on the research about data privacy under the supervision of prof. Ashwin Machanavajjhala. Yan received his Ph.D. degree in Computer Science at May, 2018.