DIFFERENTIALLY PRIVATE DATA PUBLISHING
FOR DATA ANALYSIS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Dong Su

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2016

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

I would like to express my deepest gratitude and appreciation to my advisor, Dr. Ninghui Li, for his guidance and supervision and the many opportunies that he has afforded me. He is always available, friendly, and helpful with amazing insights. I am so fortunate to have the opportunity to work with him in my graduate study. Without his help and support, this dissertation would not have been possible. I will be forever indebted to him for what he has given to me.

My appreciation also goes to my prelim exam committee and final exam committee: Dr. Elisa Bertino, Dr. Christopher W. Clifton, Dr. Dan Goldwasser and Dr. Jennifer Neville for their helpful advice and suggestions on my dissertation.

I would also like to express my gratitude to my research collaborators, Dr. Jianneng Cao, Dr. Min Lyu, Dr. Elisa Bertino and Dr. Hongxia Jin. In the past years, I greatly benefited from their constructive suggestions and effective discussions.

I am fortunate to be in Purdue with an amazing group of fellow students. I am grateful to them for their friendship and support throughout.

Last but not least, my hearfelt appreciation goes to my family. Without their limitless love and support, this dissertation cannot be finished. I always keep in my mind on those sparkling moments where they generously helped me, encouraged me to overcome all the challenges. During the whole PhD study, I can always feel their endless love, inspiration and support.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

ABSTRACT

Su, Dong PhD, Purdue University, December 2016. Differentially Private Data Publishing for Data Analysis. Major Professor: Ninghui Li.

In the information age, vast amounts of sensitive personal information are collected by companies, institutions and governments. A key technological challenge is how to design mechanisms for effectively extracting knowledge from data while preserving the privacy of the individuals involved. In this dissertation, we address this challenge from the perspective of differentially private data publishing. Firstly, we propose PrivPfC, a differentially private method for releasing data for classification. The key idea underlying PrivPfC is to privately select, in a single step, a grid, which partitions the data domain into a number of cells. This selection is done using the exponential mechanism with a novel quality function, which maximizes the expected number of correctly classified records by a histogram classifier. PrivPfC supports both the binary classification as well as the multiclass classification. Secondly, we study the problem of differentially private k-means clustering. We develop techniques to analyze the empirical error behaviors of the existing interactive and non-interactive approaches. Based on the analysis, we propose an improvement of the DPLloyd algorithm which is a differentially private version of the Lloyd algorithm and propose a non-interactive approach EUGkM which publishes a differentially private synopsis for k-means clustering. We also propose a *hybrid* approach that combines the advantages of the improved version of DPLloyd and EUGkM. Finally, we investigate the sparse vector technique (SVT) which is a fundamental technique for satisfying differential privacy in answering a sequence of queries. We propose a new version of SVT that provides better utility by introducing an effective technique to improve the performance of SVT in the interactive setting. We also show that in the non-interactive setting (but not the interactive setting), usage of SVT can be replaced by the exponential mechanism.

1. INTRODUCTION

Data collected by organizations and agencies are a key resource in today's information age. The use of sophisticated data mining techniques makes it possible to extract relevant knowledge that can then be used for a variety of purposes, such as research, product development and public policy making. However, the disclosure and exploration of those data pose serious threats to individual privacy. Examples include the identification of the medical record of the governor of Massachusetts from the GIC data [1]; the identification of the search history of an AOL user from the AOL query log data [2]; the identification of Netflix subscribers from the Netflix Prize dataset [3] and the identification of participants from the published aggregated DNA statistics in the Genome-Wide Association Studies (GWAS) [4].

In this dissertation, we consider the problem of private data publication. In this setting, a trusted data curator gathers sensitive information from a large number of respondents, create a microdataset where each tuple corresponds to one entity, such an individual, a household or an organization, and release the sanitized synopsis to the public.

In recent years, differential privacy [5, 6] has emerged as the *de facto* standard privacy notion for private data analysis because it offers a rigorous guarantee of privacy regardless of the adversary's prior knowledge. Differential privacy requires that the output of a data analysis mechanism be approximately identical, even if any single tuple in the input database is arbitrarily added or removed. Differential privacy is parameterized by ϵ, the upper bound of the ratio of the probabilities on getting the same output on the above two database differing in a single tuple. ϵ measures the privacy risk. The smaller the ϵ is, the harder for the adversary to infer the existence of the target tuple in the database.

We aim at developing practical techniques to data analysis under differential privacy. There are two broad approaches for differentially private data analysis. The interactive approach aims at developing customized differentially private algorithms for various data analysis tasks. The non-interactive approach aims at developing differentially private algorithms that can output a synopsis of the input dataset, which can then be used to support various data mining tasks. Most of existing works focus on developing interactive approaches [7–12]. However, the interactive approach is far from being practical since the limited privacy budget has to be shared by all queries issued to the database. On the other hand, non-interactive approaches are free from this limitation. However, very few practical and accurate non-interactive private data publishing algorithms have been proposed. Therefore, in this dissertation, we attempt to provide solutions for differentially private data analysis by proposing new non-interactive algorithms and combining the advantages of two approaches.

We begin in Chapter 3 by introducing PrivPfC, a differentially private method for releasing data for classification. Several state-of-the-art methods follow the structure of existing classification algorithms and are all iterative, which is suboptimal due to the locally optimal choices and division of the privacy budget among many sequentially composed steps. We propose PrivPfC, a new differentially private method for releasing data for classification. The key idea underlying PrivPfC is to privately select, in a single step, a grid, which partitions the data domain into a number of cells. This selection is done using the exponential mechanism with a novel quality function, which maximizes the expected number of correctly classified records by a histogram classifier. PrivPfC supports both the binary classification as well as the multiclass classification. Through extensive experiments on real datasets, we demonstrate PrivPfC's superiority over the state-of-the-art methods.

In Chapter 4, we focus on differentially private k-means clustering. Several state-of-the-art methods follow the *single-workload* approach which adapts an existing machine learning algorithm by making each step private. However, most of them do not have satisfactory empirical performance. In this work, we develop techniques to analyze the empirical error behaviors of one of the state-of-the-art single-workload approaches, DPLloyd, which is a differentially private version of the Lloyd algorithm. Based on the analysis, we propose an improvement of DPLloyd. We also propose a new algorithm for k-means clustering from the perspective of the *non-interactive* approach which publishes a synopsis of the input dataset. After analyzing the empirical error behaviors of EUGkM, we further propose a *hybrid* approach that combines our DPLloyd improvement and EUGkM. Results from extensive and systematic experiments support our analysis and demonstrate the effectiveness of the DPLloyd improvement, EUGkM and the hybrid approach.

In Chapter 5, we focus on the sparse vector technique. The Sparse Vector Technique (SVT) is a fundamental technique for satisfying differential privacy and has the unique quality that one can output some query answers without apparently paying any privacy cost. SVT has been used in both the interactive setting, where one tries to answer a sequence of queries that are not known ahead of the time, and in the non-interactive setting, where all queries are known. Because of the potential savings on privacy budget, many variants for SVT have been proposed and employed in privacy-preserving data mining and publishing. However, most variants of SVT are actually not private. In this dissertation, we analyze these errors and identify the misunderstandings that likely contribute to them. We also propose a new version of SVT that provides better utility, and introduce an effective technique to improve the performance of SVT. These enhancements can be applied to improve utility in the interactive setting. In the non-interactive setting (but not the interactive setting), usage of SVT can be replaced by the Exponential Mechanism (EM); we have conducted analytical and experimental comparisons to demonstrate that EM outperforms SVT.

Our overall contribution can be summarized as follows. On differentially private classification, we propose a non-interactive approach for publishing projected histograms, which results in lower classification error when compared with the current state-of-the-art methods. On differentially private k-means clustering, we propose the EUGkM method for publishing synopsis for k-means clustering, which outperforms existing methods. We also propose a novel hybrid approach to differentially private data analysis, which is so far the best approach to k-means clustering. On SVT, we propose a new version of it that provides better utility, and introduce an effective technique to improve the performance of SVT in the interactive setting. We also showed that in the non-interactive setting, usage of SVT can be replaced by the Exponential Mechanism (EM).

2. PRELIMINARIES AND RELATED WORKS

2.1 The Definition of ϵ-DP

Informally, the DP notion requires any single element in a dataset to have only a limited impact on the output. The following definition is taken from [5, 6].

Definition 2.1.1 (ϵ-Differential Privacy) *An algorithm $\mathcal{A}$ satisfies ϵ-differential privacy (ϵ-DP), where $\epsilon \geq 0$, if and only if for any datasets D and D' that* differ on one element*,* *we have*

$$\forall T \subseteq Range(\mathcal{A}) : \ \mathsf{Pr}\left[\mathcal{A}(D) \in T\right] \leq e^{\epsilon} \, \mathsf{Pr}\left[\mathcal{A}(D') \in T\right], \tag{2.1}$$

where $Range(\mathcal{A})$ denotes the set of all possible outputs of the algorithm $\mathcal{A}$.

The condition (2.1) can be equivalently stated as:

$$\forall t \in Range(\mathcal{A}) : \ \frac{\mathsf{Pr}\left[\mathcal{A}(D) = t\right]}{\mathsf{Pr}\left[\mathcal{A}(D') = t\right]} \leq e^{\epsilon}, \tag{2.2}$$

where we define $\frac{0}{0}$ to be 1.

More generally, ϵ-DP can be defined by requiring Eq. (2.1) to hold on D and D' that are *neighboring*. When applying DP, an important choice is the precise condition under which D and D' are considered to be neighboring. Even when applying DP to relational datasets and interpreting “differing by one element” as “differing by a single record (or tuple)”, there are still two natural choices, which lead to what are called unbounded and bounded DP in [13]. In *Unbounded DP*, D and D' are neighboring if D can be obtained from D' by adding or removing one element. In *Bounded DP*, D and D' are neighboring if D can be obtained from D' by replacing one element in D' with another element. When using bounded DP, two datasets that have different number of elements are not considered to be neighboring; therefore, publishing the exact number of elements in the input dataset satisfies ϵ-DP for any ϵ under bounded DP. However, doing so does not satisfy ϵ-DP for any ϵ in unbounded DP.

One way to understand the intuition of DP is the following “opting-out” analogy. We want to publish $\mathcal{A}(D)$, where D consists of data of many individuals. An individual objects to publishing $\mathcal{A}(D)$ because her data is in D and she is concerned about her privacy. In this case, we can address the individual’s privacy concern by removing her data from D (or replacing her data with some arbitrary value) to obtain D' and publishing $\mathcal{A}(D')$. However, achieving privacy protection by removing an individual’s data is infeasible. Since we need to protect everyone’s privacy, following this approach means that we would need to remove everyone’s data. DP tries to approximate the effect of opting out, by ensuring that any effect due to the inclusion of one’s data is small. This is achieved by ensuring that for any output, one will see the same output with a similar probability even if any single individual’s data is removed (unbounded DP), or replaced (bounded DP).

2.1.1 Bounded DP or Unbounded DP

In the literature, it is generally assumed that using either bounded or unbounded DP is fine, and one can choose whichever one that is more convenient. We point out, however, that using bounded DP is problematic. More specifically, as we show in Section 2.2, bounded DP does not compose under parallel composition (whereas unbounded DP does). This parallel composition property is often used when proving that an algorithm satisfies ϵ-DP.

We also note that any algorithm that satisfies ϵ-unbounded DP also satisfies (2ϵ)-bounded DP, since replacing one element with another can be achieved by removing one element and then adding the other. Therefore, we use unbounded DP in this book.

2.2 Properties of ϵ-DP

DP is an appealing privacy notion in part because it has the following nice properties. These properties are very useful when designing multi-step algorithms that satisfy ϵ-DP.

2.2.1 Post-processing and Sequential Composition

One important property of ϵ-DP is that given an algorithm that satisfies ϵ-DP, no matter what additional processing one performs on the output of the algorithm, the composition of the algorithm and the post-processing step still satisfies ϵ-DP.

Proposition 2.2.1 (Post-processing) *Given $\mathcal{A}_1(\cdot)$ that satisfies ϵ-DP, then for any (possibly randomized) algorithm $\mathcal{A}_2$, the composition of $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e., $\mathcal{A}_2(\mathcal{A}_1(\cdot))$ satisfies ϵ-DP.*

Proof Let D and D' be any two neighboring databases. Let $\mathcal{S}$ be $Range(\mathcal{A}_1)$. For any $t \in Range(\mathcal{A}_2)$, we have

$$\begin{aligned}\mathsf{Pr}\left[\mathcal{A}_2(\mathcal{A}_1(D)) = t)\right] &= \sum_{s\in\mathcal{S}} \mathsf{Pr}\left[\mathcal{A}_1(D) = s\right] \mathsf{Pr}\left[\mathcal{A}_2(s) = t\right] \\ &\leq \sum_{s\in\mathcal{S}} e^{\epsilon}\mathsf{Pr}\left[\mathcal{A}_1(D') = s\right] \mathsf{Pr}\left[\mathcal{A}_2(s) = t\right] \\ &= e^{\epsilon}\mathsf{Pr}\left[\mathcal{A}_2(\mathcal{A}_1(D')) = t\right].\end{aligned}$$

If $\mathcal{S}$ is not countable, $\mathsf{Pr}\left[\mathcal{A}_2(\mathcal{A}_1(D)) = t)\right] = \int_{s\in\mathcal{S}} \mathsf{Pr}\left[\mathcal{A}_1(D) = s\right] \mathsf{Pr}\left[\mathcal{A}_2(s) = t\right] ds$ and the logic of the proof is the same. ■

In the above proposition, the post-processing algorithm $\mathcal{A}_2$ accesses only the output of $\mathcal{A}_1$ and not the input dataset D. The following proposition applies to the case where $\mathcal{A}_2$ also accesses D.

Proposition 2.2.2 (Sequential composition) *Given $\mathcal{A}_1(\cdot)$ that satisfies ϵ_1-DP, and $\mathcal{A}_2(s, \cdot)$ that satisfies ϵ_2-DP for any s, then $\mathcal{A}(D) = \mathcal{A}_2(\mathcal{A}_1(D), D)$ satisfies $(\epsilon_1 + \epsilon_2)$-DP.*

Proof Let D and D' be any two neighboring databases. Let $\mathcal{S}$ be $Range(\mathcal{A}_1)$. For any $t \in Range(\mathcal{A}_2)$, we have

$$\begin{aligned}\mathsf{Pr}\left[\mathcal{A}_2(\mathcal{A}_1(D), D) = t)\right] &= \sum_{s \in \mathcal{S}} \mathsf{Pr}\left[\mathcal{A}_1(D) = s\right] \mathsf{Pr}\left[\mathcal{A}_2(s, D) = t\right] \\ &\leq \sum_{s \in \mathcal{S}} e^{\epsilon_1} \mathsf{Pr}\left[\mathcal{A}_1(D') = s\right] e^{\epsilon_2} \mathsf{Pr}\left[\mathcal{A}_2(s, D') = t\right] \\ &= e^{\epsilon_1 + \epsilon_2} \mathsf{Pr}\left[\mathcal{A}_2(\mathcal{A}_1(D'), D') = t\right].\end{aligned}$$

If $\mathcal{S}$ is not countable, $\mathsf{Pr}\left[\mathcal{A}_2(\mathcal{A}_1(D), D) = t)\right] = \int_{s \in \mathcal{S}} \mathsf{Pr}\left[\mathcal{A}_1(D) = s\right] \mathsf{Pr}\left[\mathcal{A}_2(s, D) = t\right] ds$ and the logic of the proof is the same. ■

Note that Proposition 2.2.1 is a special case of Proposition 2.2.2, where $\mathcal{A}_2$ satisfies 0-DP because it does not look at the input dataset. Proposition 2.2.2 can be further generalized to the case where there are k such algorithms, each taking two inputs, an auxiliary input consisting of the combined outputs of the previous algorithms, and the input dataset, and satisfying ϵ-DP when the auxiliary input is fixed.

Corollary 1 (General Sequential Composition) *Let $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_k$ be k algorithms (that take auxiliary inputs) that satisfy ϵ_1-DP, ϵ_2-DP, $\cdots$, ϵ_k-DP, respectively, with respect to the input dataset. Publishing*

$$\mathbf{t} = \langle t_1, t_2, \cdots, t_k \rangle, \textit{ where } t_1 = \mathcal{A}_1(D), t_2 = \mathcal{A}_2(t_1, D), \cdots, t_k = \mathcal{A}_k(\langle t_1, \cdots, t_{k-1} \rangle, D)$$

satisfies $(\sum_{i=1}^{k} \epsilon_i)$-DP.

This follows from Proposition 2.2.2 via mathematical induction. The ϵ parameter is often referred to as the "**privacy budget**", since it needs to be divided under sequential composition and consumed by individual steps in an algorithm.

2.2.2 Parallel Composition and Convexity

We now consider another form of composition, where k algorithms are applied to an input dataset D, but each algorithm only to a portion of D. We introduce the notion of a partitioning function. Let $\mathbb{D}$ denote the set of all possible data items. A **partitioning algorithm** f takes an item in $\mathbb{D}$ as input and maps it to a positive integer number. Executing f on D once yields a partitioning of D as follows. One executes f on each element of D, each time resulting in a number. Let k be the largest number being outputted, then D is partitioned into k partitions, with D_i including all items mapped to i.

Proposition 2.2.3 (Parallel Composition under Unbounded DP.) *Let* $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_k$ *be* k *algorithms that satisfy* ϵ_1*-DP,* ϵ_2*-DP,* $\cdots$*,* ϵ_k*-DP, respectively. Given a* deterministic partitioning function f*, let* $D_1, D_2, \cdots, D_k$ *be the resulting partitions of executing* f *on* D*. Publishing* $\mathcal{A}_1(D_1), \mathcal{A}_2(D_2), \cdots, \mathcal{A}_k(D_k)$ *satisfies* $(\max_{i\in[1,..,k]} \epsilon_i)$*-DP.*

Proof Given two neighboring datasets D and D', without loss of generality, assume that D contains one more element than D'. Let the result of partitioning of D and D' be $D_1, D_2, \cdots, D_k$ and $D'_1, D'_2, \cdots, D'_k$, respectively. There exists j such that (1) D_j contains one more element than D'_j, and (2) for any $i \neq j$, $D_i = D'_i$. Denote $\mathcal{A}_1(D_1), \mathcal{A}_2(D_2), \cdots, \mathcal{A}_k(D_k)$ by $\mathcal{A}(D)$. Since these k algorithms run on disjoint sets D_i independently, for any sequence $t = (t_1, t_2, \cdots, t_k)$ of outputs of these k algorithms, where $t_i \in Range(\mathcal{A}_i)$, we have

$$\begin{aligned}
\mathsf{Pr}\left[\mathcal{A}(D) = t\right] &= \mathsf{Pr}\left[(\mathcal{A}_1(D_1) = t_1) \wedge (\mathcal{A}_2(D_2) = t_2) \wedge \cdots \wedge (\mathcal{A}_k(D_k) = t_k)\right]. \\
&= \mathsf{Pr}\left[\mathcal{A}_j(D_j) = t_j\right] \prod_{i\neq j} \mathsf{Pr}\left[\mathcal{A}_i(D_i) = t_i\right] \\
&\leq e^{\epsilon_j} \mathsf{Pr}\left[\mathcal{A}_j(D'_j) = t_j\right] \prod_{i\neq j} \mathsf{Pr}\left[\mathcal{A}_i(D'_i) = t_i\right] \\
&\leq e^{\max_{i\in[1,..,k]} \epsilon_i} \mathsf{Pr}\left[\mathcal{A}(D') = t\right].
\end{aligned}$$

■

Example 1 (Publishing histograms based on counts.) *Suppose that we have a method to publish the number of records in a set while satisfying ϵ-DP. We can use the parallel composition to turn that method into one for publishing a histogram. A histogram "bins" the range of values, i.e., divides the entire range of values into a series of intervals, and then counts how many values fall into each interval.*

Recall that publishing the total number of records in a dataset satisfies 0-DP under the bounded DP interpretation. Thus, if parallel composition were to hold for bounded DP as well, then arbitrary histograms can be published accurately while satisfying 0-DP.

Proposition 2.2.4 *Parallel composition* ***does not hold*** *using the bounded DP interpretation.*

Proof When one element in a dataset D is replaced by another element to obtain D', after partitioning D and D', we may be in the situation that there exist $i \neq j$ such tat D_i contains one additional element than D'_i, and D'_j contains one additional element than D_j. Under bounded DP, $\frac{\mathsf{Pr}[A_i(D_i)]}{\mathsf{Pr}\left[A_i(D'_i)\right]}$ can be unbounded because D_i and D'_i contain different numbers of elements. ■

Since parallel composition is frequently used to prove that an algorithm satisfies ϵ-DP, Proposition 2.2.4 suggests that we should use the unbounded interpretation of ϵ-DP wherever possible. If bounded DP is used, one has to be really careful that parallel composition is not used.

Proposition 2.2.3 is only for the case where the partition function f is deterministic. To prove that it also holds when f is randomized, the following convexity property of DP is helpful.

Proposition 2.2.5 (Convexity) *Given two mechanisms $\mathcal{A}_1$ and $\mathcal{A}_2$ that both satisfy ϵ-DP, and any $p \in [0, 1]$, let $\mathcal{A}$ be the mechanism that applies $\mathcal{A}_1$ with probability p and $\mathcal{A}_2$ with probability $1 - p$. Then $\mathcal{A}$ satisfies ϵ-DP.*

Proof Let D and D' be any two neighboring databases. For any $t \in Range(\mathcal{A})$, we have

$$\begin{aligned}
\mathsf{Pr}\left[\mathcal{A}(D)=t\right] &= p\,\mathsf{Pr}\left[\mathcal{A}_1(D)=t\right]+(1-p)\,\mathsf{Pr}\left[\mathcal{A}_2(D)=t\right]\\
&\leq p\,e^{\epsilon}\,\mathsf{Pr}\left[\mathcal{A}_1(D')=t\right]+(1-p)\,e^{\epsilon}\,\mathsf{Pr}\left[\mathcal{A}_2(D')=t\right]\\
&= e^{\epsilon}\left(p\,\mathsf{Pr}\left[\mathcal{A}_1(D')=t\right]+(1-p)\,\mathsf{Pr}\left[\mathcal{A}_2(D')=t\right]\right)\\
&= e^{\epsilon}\mathsf{Pr}\left[\mathcal{A}(D')=t\right].
\end{aligned}$$

■

Again, we can generalize the above to the case of k algorithms.

Corollary 2 (Convexity: General Case) *Given k mechanisms $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_k$ that satisfy ϵ-DP, and $p_1, p_2, \cdots, p_k \in [0,1]$ such that $\sum_{i=1}^{k} p_i = 1$, let $\mathcal{A}$ be the mechanism that applies $\mathcal{A}_i$ with probability p_i. Then $\mathcal{A}$ satisfies ϵ-DP.*

This follows from Proposition 2.2.5 by mathematical induction. With this corollary, we can extend the parallel composition to the case of randomized partition function as well. Note that we require that such a partitioning function f to have an upper-bound on the number of partitions it produces, i.e., there exists b such that $\forall x, f(x) \leq b$.

Proposition 2.2.6 (Parallel composition, Randomized partition function.) *Let $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_k$ be k algorithms that satisfy ϵ_1-DP, ϵ_2-DP, $\cdots$, ϵ_k-DP, respectively. Given a* possibly randomized partitioning function *f, The mechanism of first executing f on D, with $D_1, D_2, \cdots, D_k$ being the resulting partitions, and then publishing $\mathcal{A}_1(D_1), \mathcal{A}_2(D_2), \cdots, \mathcal{A}_k(D_k)$, satisfies $(\max_{i\in[1,..,k]} \epsilon_i)$-DP.*

Proof Let $\epsilon = \max_{i\in[1,..,k]} \epsilon_i$. We can view the result of f as a probabilistic combination of many deterministic partitioning functions. Consider all possible outputs of f on elements in D. The total number of such combinations is finite. Let f_i be the partitioning function that output the i'th such output, and p_i be the probably that executing f results in output f_i. From Proposition 2.2.3, the parallel composition under f_i satisfies ϵ-DP. The behavior under f can be viewed as the convex composition of all f_i's, and thus also satisfies ϵ-DP because of Corollary 2. ■

2.3 The Laplace Mechanism

The Laplace mechanism ([5]) is the first and probably most widely used mechanism for DP. It satisfies ϵ-DP by adding noise to the output of a numerical function. We present first the case where the function outputs a scalar, and then the vector case. We present them separately even though the latter subsumes the former as a special case, because the scalar case is easier to understand.

2.3.1 The Scalar Case

Assume that we have a dataset for patients diagnosed with lung cancer, with one attribute being how many years the patient has been smoking, and another being how many packs of cigarette the patient smokes on average per day. Suppose that we want to know how many patients have been smoking for more than 15 years, how to obtain the answer while satisfying ϵ-DP?

In this case, we want to compute $f(D)$, where f outputs a single scalar value. To satisfy ϵ-DP, one can publish $\tilde{f}(D) = f(D) + X$, where X is a random variable drawn from some distribution. What distribution should one use for X? Intuitively, we want the distribution to have 0 as its mean so that $\tilde{f}(D)$ is an unbiased estimate of $f(D)$. Furthermore, we need to ensure that

$$\forall t, \frac{\mathsf{Pr}\left[\tilde{f}(D) = t\right]}{\mathsf{Pr}\left[\tilde{f}(D') = t\right]} = \frac{\mathsf{Pr}\left[f(D) + X = t\right]}{\mathsf{Pr}\left[f(D') + X' = t\right]} = \frac{\mathsf{Pr}\left[X = t - f(D)\right]}{\mathsf{Pr}\left[X' = t - f(D')\right]} \leq e^{\epsilon},$$

where X and X' are drawn from the same distribution. Let $d = f(D) - f(D')$, we need to ensure that

$$\forall x, \frac{\mathsf{Pr}\left[X = x\right]}{\mathsf{Pr}\left[X' = x + d\right]} \leq e^{\epsilon}. \tag{2.3}$$

We need to ensure that Eq. (2.3) holds for all possible d, and thus need the concept of the global sensitivity of f, which is the maximum change of f between two neighboring datasets D and D'.

Definition 2.3.1 (Global sensitivity) *Let $D \simeq D'$ denote that D and D' are neighboring. The global sensitivity of a function f, denoted by Δ_f, is given below*

$$\Delta_f = \max_{D \simeq D'} |f(D) - f(D')|, \tag{2.4}$$

We want to ensure that Eq. (2.3) holds for all $d \leq \Delta_f$. In other words, the probability density function of the noise should have the property that if one moves no more than Δ_f units on the x-axis, the probability should increase or decrease by a factor of no more than e^ϵ, i.e., if one moves no more than 1 unit on the x-axis, the probability should change by a multiplicative factor of no more than e^{ϵ/Δ_f}.

The distribution that naturally satisfies this requirement is $\mathsf{Lap}\left(\frac{\Delta_f}{\epsilon}\right)$, the Laplace distribution, where $\mathsf{Pr}\left[\mathsf{Lap}\left(\beta\right) = x\right] = \frac{1}{2\beta} e^{-|x|/\beta}$. Note that

$$\frac{\mathsf{Pr}\left[\mathsf{Lap}\left(\beta\right) = x\right]}{\mathsf{Pr}\left[\mathsf{Lap}\left(\beta\right) = x + d\right]} \leq e^{d/\beta} \leq e^{\Delta_f/\beta} = e^{\epsilon}.$$

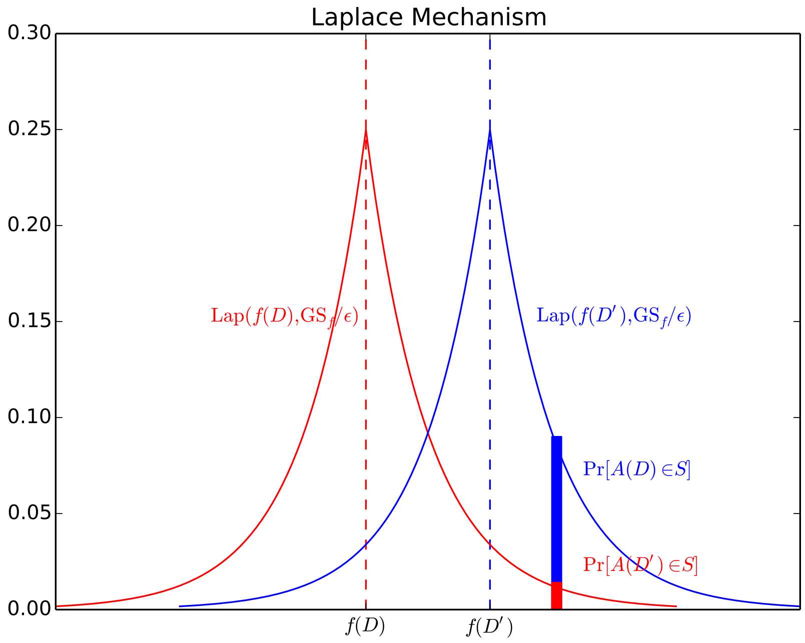


Figure 2.1.: Differential privacy via Laplace noise.

Theorem 2.3.1 (Laplace mechanism, scalar case) *For any function f, the Laplace mechanism $\mathcal{A}_f(D) = f(D) + \mathsf{Lap}\left(\frac{\Delta_f}{\epsilon}\right)$ satisfies ϵ-DP.*

Proof Let X be the noise injected to $f(D)$. So, $X \sim \mathsf{Lap}\left(\frac{\Delta_f}{\epsilon}\right)$.

$$\mathsf{Pr}\left[\mathcal{A}_f(D) = t\right] = \mathsf{Pr}\left[f(D) + X = t\right] = \mathsf{Pr}\left[X = t - f(D)\right] = \frac{\epsilon}{2\Delta_f} \exp\left(\frac{-\epsilon|t - f(D)|}{\Delta_f}\right).$$

Similarly, we have $\mathsf{Pr}\left[\mathcal{A}_f(D') = t\right] = \frac{\epsilon}{2\Delta_f} \exp\left(\frac{-\epsilon|t-f(D')|}{\Delta_f}\right)$. Thus,

$$\begin{aligned}
\frac{\mathsf{Pr}\left[\mathcal{A}_f(D) = t\right]}{\mathsf{Pr}\left[\mathcal{A}_f(D') = t\right]} &= \frac{\exp\left(\frac{-\epsilon|t-f(D)|}{\Delta_f}\right)}{\exp\left(\frac{-\epsilon|t-f(D')|}{\Delta_f}\right)} \\
&= \exp\left(\frac{\epsilon(|t - f(D')| - |t - f(D)|)}{\Delta_f}\right) \\
&\leq \exp\left(\frac{\epsilon|f(D) - f(D')|}{\Delta_f}\right) \\
&\leq \exp(\epsilon).
\end{aligned}$$

The first inequality holds because of the Triangle inequality with absolute value $|a| - |b| \leq |a - b|$ and the second holds due to Eq. (2.4). ■

Example 2 (Counting Queries) *Queries such as "how many patients have been smoking for more than 15 years" are counting queries, as they count how many records satisfy a given condition. In general, counting queries have global sensitivity 1, as adding or removing a single record can change the result of a counting query by at most 1. They can thus be answered by the Laplace mechanism with relatively low noises.*

Example 3 (Sum Queries) *Queries summing up the values of one attribute for the records that satisfy a given condition have sensitivity that equals the size of the domain of that attribute, and can be answered by the Laplace mechanism.*

2.3.2 The Vector Case

The Laplace mechanism can also be applied to a function f that outputs a vector, in which case, the global sensitivity Δ_f is the maximum L_1 norm of the difference between $f(D)$ and $f(D')$, i.e.:

$$\Delta_f = \max_{D \simeq D'} ||f(D) - f(D')||_1. \tag{2.5}$$

And noise calibrated to the global sensitivity should be added to all components of a vector.

Theorem 2.3.2 (Laplace mechanism, the vector case) *The Laplace mechanism for a function f whose value is a k-dimensional vector, defined below, satisfies ϵ-DP.*

$$\mathcal{A}_f(D) = f(D) + \langle X_1, X_2, \cdots, X_k \rangle,$$

where $X_1, X_2, \cdots, X_k$ are i.i.d. random variables drawn from $\mathsf{Lap}\left(\frac{\Delta_f}{\epsilon}\right)$.

Proof Suppose $f(D) = \langle a_1, a_2, \cdots, a_k \rangle$. For any output $t = \langle t_1, t_2, \cdots, t_k \rangle$,

$$\begin{aligned}
\mathsf{Pr}\left[\mathcal{A}_f(D) = t\right] &= \mathsf{Pr}\left[f(D) + \langle X_1, X_2, \cdots, X_k \rangle = t\right] \\
&= \mathsf{Pr}\left[(X_1 = t_1 - a_1) \wedge (X_2 = t_2 - a_2) \wedge \cdots \wedge (X_k = t_k - a_k)\right] \\
&= \prod_{i=1}^{k} \mathsf{Pr}\left[X_i = t_i - a_i\right] = \prod_{i=1}^{k} \frac{\epsilon}{2\Delta_f} \exp\left(\frac{-\epsilon|t_i - a_i|}{\Delta_f}\right) \\
&= \left(\frac{\epsilon}{2\Delta_f}\right)^k \exp\left(\frac{-\epsilon \sum\limits_{i=1}^{k} |t_i - a_i|}{\Delta_f}\right) \\
&= \left(\frac{\epsilon}{2\Delta_f}\right)^k \exp\left(\frac{-\epsilon||t - f(D)||_1}{\Delta_f}\right)
\end{aligned}$$

Similarly, $\mathsf{Pr}\left[\mathcal{A}_f(D') = t\right] = \left(\frac{\epsilon}{2\Delta_f}\right)^k \exp\left(\frac{-\epsilon||t - f(D')||_1}{\Delta_f}\right)$. Thus,

$$\begin{aligned}
\frac{\mathsf{Pr}\left[\mathcal{A}_f(D) = t\right]}{\mathsf{Pr}\left[\mathcal{A}_f(D') = t\right]} &= \frac{\exp\left(\frac{-\epsilon||t - f(D)||_1}{\Delta_f}\right)}{\exp\left(\frac{-\epsilon||t - f(D')||_1}{\Delta_f}\right)} \\
&= \exp\left(\frac{\epsilon(||t - f(D')||_1 - ||t - f(D)||_1)}{\Delta_f}\right) \\
&\leq \exp\left(\frac{\epsilon||f(D) - f(D')||_1}{\Delta_f}\right) \\
&\leq \exp\left(\epsilon\right).
\end{aligned}$$

The first inequality holds because of the triangle inequality for the L_1-norm and the second holds due to Eq. (2.5). ■

Example 4 (Histogram) *Consider again the dataset for patients diagnosed with lung cancer, with one attribute being how many years the patient has been smoking, and another being how many packs of cigarette the patient smokes on average per day. We can publish a one-dimensional histogram that counts how many patients have been smoking for a certain number of years, where the number of years is divided into a few bins, such as* $\{[0-4], [5-9], [10-14], [15-19], [20-29], [30+]\}$. *Publishing such a histogram has global sensitivity* 1*, since adding or removing one patient changes only the count of one bin by* 1*. Thus publishing a noisy histogram with noise drawn from the distribution* $\mathsf{Lap}\left(\frac{1}{\epsilon}\right)$ *added to each bin count satisfies* ϵ*-DP.*

Similarly, we can publish a two-dimensional histogram that also considers how many packs of cigarettes a patient smoke on average per day. The same Laplace mechanism would apply. Note that to satisfy ϵ-DP*, it is important that the way the attribute values are partitioned into bins does not depend on the input dataset. If the partitioning depends on the input dataset, one has to ensure that the partitioning and the histogram together satisfy* ϵ-DP*, using composition properties in Section 2.2.*

Note the the above noisy Histogram method can be viewed either as applying the Laplace mechanism with a vector output, or as a parallel composition of the counting function.

2.4 The Exponential Mechanism

While the Laplace mechanism provides a solution to handle numeric queries, it cannot be applied to non-numeric valued queries. This motivates the development of the exponential mechanism [14], which can be applied whether a function's output is numerical or categorical.

Suppose that one wants to publish $f(D)$, and let O denote the set of possible outputs. To satisfy ϵ-DP, a mechanism should output values in O following some probability distribution. Naturally, some values in O are more desirable than others. For example, the most desirable output is the true value $f(D)$, and one has natural preferences among other values as well. For example, consider a transactional dataset D, and one wants to output the item that appears most frequently in D. Then O is the set of all items, and between two items, we prefer to output the one that appears more often. This preference is encoded using a quality function $q : (\mathbb{D} \times O) \to \mathbb{R}$, where $\mathbb{D}$ denotes the set of all datasets, and $\mathbb{R}$ denotes the set of all real numbers. Without loss of generality, we assume that a higher quality value indicates better utility. For example, in the most frequent item case, a natural choice is to define $q(D, o)$ to be the number of times the item o appears in D.

2.4.1 The General Case of the Exponential Mechanism

Definition 2.4.1 (The Exponential Mechanism) *For any quality function $q : (\mathbb{D} \times O) \to \mathbb{R}$, and a privacy parameter ϵ, the exponential mechanism $\mathcal{M}_q^\epsilon(D)$ outputs $o \in O$ with probability proportional to $\exp\left(\frac{\epsilon q(D,o)}{2\Delta q}\right)$, where*

$$\Delta q = \max_{\forall o, D \simeq D'} |q(D, o) - q(D', o)|$$

is the sensitivity of the quality function. That is,

$$\mathsf{Pr}\left[\mathcal{M}_q^\epsilon(D) = o\right] = \frac{\exp\left(\frac{\epsilon q(D,o)}{2\Delta q}\right)}{\sum_{o' \in O} \exp\left(\frac{\epsilon q(D,o')}{2\Delta q}\right)}$$

Theorem 2.4.1 (The Exponential Mechanism) *The exponential mechanism satisfies ϵ-differential privacy.*

Proof For any two neighboring datasets D and D' and any $o \in O$,

$$\frac{\exp\left(\frac{\epsilon q(D,o)}{2\Delta q}\right)}{\exp\left(\frac{\epsilon q(D',o)}{2\Delta q}\right)} = \exp\left(\frac{\epsilon(q(D,o)-q(D',o))}{2\Delta q}\right) \le \exp\left(\frac{\epsilon}{2}\right), \tag{2.6}$$

Because of the symmetry of neighboring, we also have $\forall o'$, $\exp\left(\frac{\epsilon q(D',o')}{2\Delta q}\right) \le \exp\left(\frac{\epsilon}{2}\right)\exp\left(\frac{\epsilon q(D,o')}{2\Delta q}\right)$.

Now we prove ϵ-DP of the exponential mechanism. For any output o of $\mathcal{M}_q^\epsilon$,

$$\begin{aligned}
\frac{\mathsf{Pr}\left[\mathcal{M}_q^\epsilon(D)=o\right]}{\mathsf{Pr}\left[\mathcal{M}_q^\epsilon(D')=o\right]} &= \frac{\frac{\exp\left(\frac{\epsilon q(D,o)}{2\Delta q}\right)}{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D,o')}{2\Delta q}\right)}}{\frac{\exp\left(\frac{\epsilon q(D',o)}{2\Delta q}\right)}{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D',o')}{2\Delta q}\right)}} \\
&= \left(\frac{\exp\left(\frac{\epsilon q(D,o)}{2\Delta q}\right)}{\exp\left(\frac{\epsilon q(D',o)}{2\Delta q}\right)}\right)\cdot\left(\frac{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D',o')}{2\Delta q}\right)}{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D,o')}{2\Delta q}\right)}\right) \\
&\le \exp\left(\frac{\epsilon}{2}\right)\cdot\left(\frac{\sum_{o'\in O}\exp\left(\frac{\epsilon}{2}\right)\exp\left(\frac{\epsilon q(D,o')}{2\Delta q}\right)}{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D,o')}{2\Delta q}\right)}\right) \\
&\le \exp\left(\frac{\epsilon}{2}\right)\cdot\exp\left(\frac{\epsilon}{2}\right)\left(\frac{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D,o')}{2\Delta q}\right)}{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D,o')}{2\Delta q}\right)}\right) \\
&= \exp(\epsilon).
\end{aligned} \tag{2.7}$$

■

2.4.2 The Monotonic Case of the Exponential Mechanism

In some usages of the exponential mechanism, the quality function $q(D,o)$ is monotonic in the sense that for any D and D' that are neighboring, either $\forall o \in O,\ q(D,o) \ge q(D',o)$, or $\forall o \in O,\ q(D,o) \le q(D',o)$. This is generally the case when the quality function is based on counting the number of records satisfying some condition. For example, this is the case when applying the exponential mechanism to frequent itemsets mining. For such quality functions, the effectiveness of the exponential mechanism can be improved. One can make more accurate selections by choosing each possible output with probability proportional to $\exp(\frac{\epsilon q(D,t)}{\Delta q})$, instead of $\exp(\frac{\epsilon q(D,t)}{2\Delta q})$. To see that doing so satisfies ϵ-DP, observe that Eq. (2.7) of the proof is a product of two terms, and for a monotonic quality function, whenever the first term is ≥ 1, the second term is ≤ 1; thus upper-bounding the first term by e^{ϵ} suffices. See below for details.

The utility benefit of doing is this is equivalent to doubling the privacy budget ϵ. Suppose that under the general Exponential Mechanism, the odds of choosing the best option relative to another less preferable one is $10 : 1$, then under the monotonic Exponential Mechanism, the odds is square to become $100 : 1$.

Corollary 3 *For any monotonic quality function $q : (\mathbb{D} \times O) \rightarrow \mathbb{R}$ and a privacy parameter ϵ, the exponential mechanism $\mathcal{M}_q^{\epsilon}(D)$ outputting $o \in O$ with probability proportional to $e^{\epsilon q(D,o)/(\Delta q)}$ satisfies ϵ-DP.*

Proof Let D and D' be two neighboring datasets. Without loss of generality, assume $D' = D \cup \{r\}$ and the quality function $q(D, o)$ is monotonically increasing when the size of a dataset increases. So, for any output $o' \in O$,

$$\exp\left(\frac{\epsilon q(D, o')}{\Delta q}\right) \leq \exp\left(\frac{\epsilon q(D', o')}{\Delta q}\right).$$

Similarly to Eq.(2.6), we have

$$\exp\left(\frac{\epsilon q(D', o')}{\Delta q}\right) \leq \exp\left(\epsilon\right) \exp\left(\frac{\epsilon q(D, o')}{\Delta q}\right).$$

Now we turn to the privacy proof of the exponential mechanism in the same way as the proof above.

On one hand, we observe that

$$\begin{aligned}\frac{\mathsf{Pr}\left[\mathcal{M}_q^\epsilon(D)=o\right]}{\mathsf{Pr}\left[\mathcal{M}_q^\epsilon(D')=o\right]} &= \left(\frac{\exp\left(\frac{\epsilon q(D,o)}{\Delta q}\right)}{\exp\left(\frac{\epsilon q(D',o)}{\Delta q}\right)}\right)\cdot\left(\frac{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D',o')}{\Delta q}\right)}{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D,o')}{\Delta q}\right)}\right)\\ &\le 1\cdot\exp(\epsilon)\left(\frac{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D,o')}{\Delta q}\right)}{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D,o')}{\Delta q}\right)}\right)=\exp(\epsilon).\end{aligned}$$

On the other hand,

$$\begin{aligned}\frac{\mathsf{Pr}\left[\mathcal{M}_q^\epsilon(D')=o\right]}{\mathsf{Pr}\left[\mathcal{M}_q^\epsilon(D)=o\right]} &= \left(\frac{\exp\left(\frac{\epsilon q(D',o)}{\Delta q}\right)}{\exp\left(\frac{\epsilon q(D,o)}{\Delta q}\right)}\right)\cdot\left(\frac{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D,o')}{\Delta q}\right)}{\sum_{o'\in O}\exp\left(\frac{\epsilon q(D',o')}{\Delta q}\right)}\right)\\ &\le \exp(\epsilon)\cdot 1=\exp(\epsilon).\end{aligned}$$

In a summary, $e^{-\epsilon}\le\frac{\mathsf{Pr}\left[\mathcal{M}_q^\epsilon(D)=o\right]}{\mathsf{Pr}\left[\mathcal{M}_q^\epsilon(D')=o\right]}\le e^\epsilon$ and thus the corollary holds. ■

2.5 Settings to Apply DP

We classify DP mechanisms into the following four settings.

1. *Local Privacy.* In this setting, there is no trusted third party, and each participant perturbs and submits personal data. To apply DP here, one requires that for two arbitrary possible input x_1 and x_2, and any output y: $\mathsf{Pr}\left[y|x_1\right]\le e^\epsilon\mathsf{Pr}\left[y|x_2\right]$.

2. *Interactive query-answering.* For this and the remaining settings, there is a trusted data curator who has access to raw data. In the interactive setting, the data curator sits between the users and the database, and answers queries when they are submitted, without knowing what queries will be asked in the future.

3. *Single workload.* In this setting, there is a single data analysis task one wants to perform on the dataset. Example tasks include learning a classifier, finding k cluster centroids of the data, and so on. The data curator performs the analysis task in a private way, and publishes the result.

4. *Noninteractive publishing.* In this setting the curator publishes a synopsis of the input dataset, from which a broad class of queries can be answered and synthetic data can be generated.

Note that the latter three settings all require a trusted data curator.

Local Privacy. The local privacy setting is closely related to *randomized response* [15], which is a decades-old technique in social science to collect statistical information about embarrassing or illegal behavior. To report a single bit, one reports the true value with probability p and the flip of the true value with probability $1 - p$. In a sense, applying the DP requirement here can be viewed as a generalization of the property from randomized response to a case where one report a non-binary value.

The Interactive Setting. In this setting, the data curator does not know ahead of time what queries will be encountered, and answers queries as they come. One simple method is to divide up the privacy budget and consume a portion of the privacy budget to answer each query [16]. More sophisticated methods (e.g., [17–19]) maintain a history of past queries and answers, and try to use the history to answer new queries whenever the error of doing so is acceptable.

Using the interactive setting in practice, however, has several challenges. First and foremost, answering each query consumes a portion of privacy budget, and after the privacy budget is exhausted, no additional queries can be answered on the data without violating DP. Second, the interactive setting is unsuitable with more than one data users. When a dataset needs to serve the general public such as when the census bureau provides the census data to the public, the number of data users is very large. Because the curator cannot be sure whether any two data users are colluding or not, the privacy budget has to be shared by *all users*. This means that only a few users can be supported and each user can have only a small number of queries answered.

Single Workload. In this setting, the goal is to publish the result from one data mining or machine learning task. Most approaches try to adapt an existing machine learning algorithm by making each step private. An alternative approaches include perturbing the optimization objective function for learning a classifier.

Non-interactive publishing. In this setting, the data curator publishes some summary of the data. It is generally assumed that the set of queries one cares about is known. The most natural set of queries are histogram queries or marginal queries.

Interactive versus Non-interactive. There are a series of negative results concerning differential privacy in the non-interactive mode [5, 20–23], and these results have been interpreted "to mean that one cannot answer a linear, in the database size, number of queries with small noise while preserving privacy" and motivate "an interactive approach to private data analysis where the number of queries is limited to be small — sub-linear in the size n of the dataset" [23]. However, these results are all based on query sets that are broader than the natural set of queries that one is interested in. For example, suppose the dataset is one-dimensional where each value is an integer number in $[1..M]$. Further suppose that the data is sufficiently dense, then publishing a histogram likely gives information that one wants to know about the database. These negative results say that if one also consider subset sum queries (i.e., the sum of an arbitrary set of indices in $[1..m]$), then not all queries can be answered to a high accuracy. Intuitively this is true; however, it does not say much about how accurately we can answer range queries.

2.6 Differentially Private Data Analysis

Many data mining and machine learning problems can be viewed as optimization problems. Examples include k-means clustering, regression, and classification. We use D to denote the input dataset, ω_* to denote the desired output, and $J(D, \omega)$ to denote the objective function to be minimized. That is, we want to output

$$\omega_* = \arg\min_{\omega} J(D, \omega)$$

Several interesting techniques have been developed to perform these optimization tasks while satisfying DP. In this chapter, we group these techniques into the following categories.

1. **Output Perturbation.** One method is to directly perturb the output of the optimization problem. This requires analyzing the sensitivity of the optimization problem; that is, how much ω_* changes when the input dataset D changes by one tuple. Unfortunately, the sensitivities of these optimization problems tend to be so high that such output perturbation destroys utility.

2. **Objective Perturbation.** There exists a class of methods unique to optimization problems. Instead of perturbing the output of the optimization problem, one can perturb the optimization objective function $J(D, \omega)$ to get $J^*(D, \omega)$ in a way such that optimizing according to $J^*(D, \omega)$ is differentially private.

3. **Making Existing Algorithms Private.** Another method is to take an existing optimization algorithm and make each individual step that needs access to the input dataset private.

4. **Iterative Local Search.** There exists another method is to perform an iterative local search to approach ω_*. In each iteration, given the current candidate or candidates, we can generate a pool of new candidates and use the exponential mechanism to select among them.

5. **Publishing Histograms for Optimization.** Finally, one can publish a histogram of D optimized for the purpose of the task, e.g., for clustering or for classification, and then perform optimization using the histogram. Intuitively, this publishes more information than needed for outputting ω_*; however, this appears to outperform the above methods in experiments.

2.6.1 Example Optimization Problems

We now give a brief description of the optimization problems that have been studied in the context of differential privacy, and discuss the feasibility of performing output perturbation for each of them.

k-means Clustering

k-means clustering is a widely used unsupervised machine learning method for data analysis. It has a wide range of applications, including but not limited to nearest neighbor queries, market segment, image processing, and geo-statistics.

The input is a dataset $D = \{x^1, x^2, \ldots, x^N\}$, where each data point x^ℓ is a d-dimensional real vector. Intuitively, the dataset D consists of points in a d-dimensional space. The output is a set of k points $\omega = \{o^1, o^2, \cdots, o^k\}$, known as the centroids. These k centroids partition D into k clusters such that each data point belongs to the cluster defined by the centroid that is closest to the data point. (If there are more than one closest centroids for a data point, the data point is assigned to one of the corresponding clusters.) The objective function to be minimized is the within-cluster sum of squares. We normalize this value and call it Normalized Intra-Cluster Variance (NICV), defined as follows.

$$J_{km}(D, \omega) = \frac{1}{N} \sum_{\ell=1}^{N} \min_{j=1}^{k} ||x^\ell - o^j||^2. \tag{2.8}$$

The standard k-means algorithm is the Lloyd's algorithm ([24]). The algorithm starts by selecting k points as the initial choices for the centroid, and then tries to improve these centroid choices iteratively until no improvement can be made. In each iteration, one first uses the current centroid choices to partition the data points into k clusters $\mathbf{O} = \{O^1, O^2, \cdots, O^k\}$, with each point assigned to the same cluster as the nearest centroid. Then, one updates each centroid to be the center of the data points in the cluster.

$$\forall i \in [1..d]\ \forall j \in [1..k]\ o_i^j \leftarrow \frac{\sum_{x^\ell \in O^j} x_i^\ell}{|O^j|}, \tag{2.9}$$

where x_i^ℓ and o_i^j are the i-th dimension coordinates of x^ℓ and o^j, respectively. The algorithm continues by alternating between data partition and centroid update, until it converges.

The quality of the output computed by the Lloyd's algorithm is subject to the choice of the starting points. `Random Partition` and `Forgy` are two commonly adopted initialization methods. The former randomly partitions the database into k clusters, and takes the centers of the clusters as starting points. The latter randomly selects k data points (seeds) from the database as the starting points. One can run the algorithm multiple times, with different choices of initial centroids, and choose the output that has the minimal NICV.

The global sensitivity of k-means clustering problem is very high, as changing one single data point could completely change the optimal clustering centroids; see [25].

Linear Regression

Linear regression is a fundamental statistical approach for modeling the linear relationship between a dependent variable and several independent variables. It has been used extensively in practical applications, including fitting prediction models and analyzing the relationship between a dependent variable and one or more independent variables.

The input is a dataset $D = \{\langle x^1, y^1\rangle, \langle x^2, y^2\rangle, \ldots, \langle x^N, y^N\rangle\}$, where x^ℓ is a d-dimensional real vector, and y^ℓ is a real scalar value. The output is a d-dimensional vector ω. The optimization objective is

$$J_{\text{lr}}(D, \omega) = \frac{1}{N} \sum_{\ell=1}^{N} \left(y^\ell - \sum_{i=1}^{d} x_i^\ell \omega_i \right)^2 \tag{2.10}$$

In other words, linear regression expresses the value of y as a linear function of the values of $x_1, \ldots, x_d$, such that the sum of square errors of the predicted y values is minimized.

The global sensitivity of linear regression is unbounded. For example, given a dataset where each x is one-dimensional with $N-1$ points at $(0,0)$ and 1 point at $(1/N, 0)$. The optimal line $y = 0\dot{x} + 0$. Adding an additional point $(1, N)$ to the input dataset results in an optimal line $y = N\dot{x} + 0$. Thus, adding noise to the line parameter according to the global sensitivity remove all utilities completely.

Logistic Regression

Logistic regression also learns a vector of linear coefficients; however, the inner product of these coefficients and a data point's independent variables is used to estimate the probability of the dependent variable, using the logistic function.

The input is a dataset $D = \{\langle x^1, y^1\rangle, \langle x^2, y^2\rangle, \ldots, \langle x^N, y^N\rangle\}$, where x^ℓ is a d-dimensional real vector, and y^ℓ has a boolean domain $\{0, 1\}$. The output is a prediction function, which predicts $y = 1$ with probability

$$\rho(\omega_*, x) = \frac{\exp{(\omega_*^T x)}}{1 + \exp{(\omega_*^T x)}}.$$

The model parameter ω_* is computed by minimizing the optimization objective function,

$$J_{log}(D, \omega) = \frac{\Lambda}{2}\|\omega\| + \frac{1}{N}\sum_{\ell=1}^{N} L_\omega(x^\ell, y),$$

where the loss function is defined as

$$L_\omega(x, y) = -y\log(\rho(\omega, x)) - (1 - y)\log(1 - \rho(\omega, x)),$$

and Λ is the regularization parameter.

In [8], it is showed that the sensitivity of the output perturbation approach on logistic regression is $\frac{2}{N\Lambda}$, where Λ is the regularization parameter and N is the dataset size. Note that this means this bound becomes ∞ when no regularization is used.

SVM

Another widely used classification technique is support vector machine (SVM). It has promising empirical performance in many practical applications, and especially works well with high-dimensional data. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An

SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.

The input is a dataset $D = \{\langle x^1, y^1\rangle, \langle x^2, y^2\rangle, \ldots, \langle x^N, y^N\rangle\}$, where x^ℓ is a d-dimensional real vector, and y^ℓ has a boolean domain $\{0, 1\}$. The output is a prediction function,

$$\rho(x) = \begin{cases} 1 & \text{if } \alpha_*^T \cdot x + \beta_* > 0 \\ 0 & \text{otherwise,} \end{cases}$$

where $\alpha_* \in \mathbb{R}^d$ and $\beta_* \in \mathbb{R}$ is computed by minimizing the optimization objective function,

$$J_{svm}(D, \alpha, \beta) = \frac{\Lambda}{2}\|\alpha\|^2 + \frac{1}{N}\sum_{\ell=1}^{N} L_{\alpha,\beta}(x^\ell, y^\ell),$$

where the loss function $L_{\alpha,\beta}(x, y)$ is defined as

$$L_{\alpha,\beta}(x, y) = \max\{1 - 4(y - 0.5)(\alpha^T x + \beta - 0.5), 0\},$$

and Λ is the regularization parameter.

Rubinstein et al. [26] used the same approach for perturbing the parameters outputed by the SVM classifier and showed that the sensivitiy of the SVM learning algorithm can be bounded by $\frac{4L\Lambda\kappa\sqrt{d}}{N}$, where Λ is the regularization parameter, L is the Lipschitz constant of loss function, κ is the bound of kernel, d is dataset dimensionality and N is the dataset size.

2.6.2 Objective Perturbation

We have seen that the global sensitivities of these optimization problems are very high, making direct output perturbation an ineffective method. An interesting approach, first introduced in [8], is to perturb the optimization objective function so that solving it results in a private solution. We now discuss two such techniques.

Adding a Noisy Linear Term to the Optimization Objective Function

One method, proposed by Chaudhuri et al. [8, 10], is to add a Laplacian noise to the optimization objective function. We want to solve

$$\arg\min_{\omega} J(D, \omega), \text{ where } J(D, \omega) = \left(\frac{1}{N} \sum_{i=1}^{N} L(\omega, x_i) \right) + c(\omega),$$

where $c(\omega)$ is the regularizer.

Assuming that both $L(\omega, x_i)$ and $c(\omega)$ are strictly convex and everywhere differentiable for ω. Then define the new objective function to be

$$J^*(D, \omega) = J(D, \omega) + \frac{b^T \omega}{N},$$

where b is a random noise sampled from a distribution with density $\frac{1}{\alpha} e^{-\beta \|b\|}$, α is a normalizing constant and β is a function of ϵ.

The privacy of this method is proved as follows.

Proposition 2.6.1 *Solving* $\arg\min_{\omega} J^*(D, \omega)$ *satisfies* ϵ-DP.

Proof Suppose we have any two neighboring dataset $D = (x_1, y_1), \ldots, (x_{N-1}, y_{N-1}), (a, y)$ and $D' = (x_1, y_1), \ldots, (x_{N-1}, y_{N-1})$. For any ω^* output by our algorithm, we want to show that

$$\frac{\mathsf{Pr}\left[\omega^* | D\right]}{\mathsf{Pr}\left[\omega^* | D'\right]} \leq e^{\epsilon}.$$

Since the regularization function J and the loss function L are strictly convex and differentiable everywhere, unique minimum occurs when the gradient of $J^*(D, \omega) = J(D, \omega) + \frac{b^T \omega}{n}$ is 0. Therefore, for the two neighboring datasets D and D', there are unique values of noise b that maps the different inputs to the same output ω^*.

Let the values of b for the first and second cases respectively, be b_1 and b_2. We have

$$\frac{\partial J(D,\omega)}{\partial \omega} + \frac{b_1}{n} = \frac{\partial J(D',\omega)}{\partial \omega} + \frac{b_2}{n}.$$

Therefore,

$$\begin{aligned} \|b_1 - b_2\| &= \left\| \frac{\partial J(D,\omega)}{\partial \omega} - \frac{\partial J(D',\omega)}{\partial \omega} \right\| \\ &= \left\| \frac{\partial L(\omega,(a,y))}{\partial \omega} - \frac{\partial L(\omega,(a',y'))}{\partial \omega} \right\| \\ &\leq \Delta. \end{aligned}$$

And Δ is the sensitivity of $\frac{\partial J(D,\omega)}{\partial \omega}$.

Finally, we have,

$$\frac{\mathsf{Pr}\left[\omega^*|D\right]}{\mathsf{Pr}\left[\omega^*|D'\right]} = \frac{\mathrm{pdf}(b_1)}{\mathrm{pdf}(b_2)} \leq e^{\frac{\epsilon}{\Delta}\cdot\|b_1-b_2\|} \leq e^{\epsilon}.$$

■

Chaudhuri et al. [8, 10] showed that $\Delta \leq 2$ for both logistic regression and SVM. The loss function of logistic regression is differentiable and can be bounded by 1, Therefore,

$$\begin{aligned} \|b_1 - b_2\| &= \left\| \frac{\partial J(D,\omega)}{\partial \omega} - \frac{\partial J(D',\omega)}{\partial \omega} \right\| \\ &\leq \left\| \frac{\partial L(\omega,(a,y))}{\partial \omega} \right\| + \left\| \frac{\partial L(\omega,(a',y'))}{\partial \omega} \right\| \\ &\leq 2. \end{aligned}$$

Although the loss function of SVM, $L_\omega(x,y) = \max\{1 - y(\alpha^T x + \beta), 0\}$, is not differentiable, Chaudhuri et al. [10] proposed to use a differentiable version of this loss function, and showed that its first order derivative can be bounded by 1 and the noise scale can be bounded by 2.

It is difficult to analyze the impact of adding such linear terms to the objective function on the accuracy of the optimization results; however, experimental results show that this method is not very promising.

The Functional Mechanism

Zhang et al. [11] proposed to perturb the optimization objective function by first approximating the objective function using a polynomial, and then perturbing each and every coefficient of the polynomial.

Given an objective function $J(D, \omega) = \sum_{t_i \in D} L(t_i, \omega)$, the function mechanism first decomposes $J(D, \omega)$ into a series of polynomial basis,

$$J(D, \omega) = \sum_{j=0}^{U} \sum_{\phi \in \Phi_j} \sum_{t_i \in D} \lambda_{\phi t_i} \phi(\omega),$$

and then perturb the aggregated coefficients of each polynomial basis with Laplace noise. In the above, D is the dataset, t_i is the i-th tuple in the dataset and ω is the model parameter. And $\Phi_j (j \in \mathbb{N})$ denote the set of all products of parameter ω's coordinates $\{\omega_1, \dots, \omega_d\}$ with degree j,

$$\Phi_j = \{\omega_1^{c_1} \omega_2^{c_d} \cdots \omega_d^{c_d} | \sum_{l=1}^{d} c_l = j\}.$$

For example, $\Phi_0 = \{1\}$, $\Phi_1 = \{\omega_1, \dots, \omega_d\}$, and $\Phi_2 = \{\omega_i \cdot \omega_j | i, j \in [1, d]\}$.

Algorithm 1 Functional Mechanism

```
INPUT $D$: Dataset, $J(D, \omega)$: objective function, $\epsilon$: privacy parameter
Output $\omega_*$: best parameter vector
  Set $\Delta = 2 \max_t \sum_{j=1}^{U} \sum_{\phi \in \Phi_j} \|\lambda_{\phi t}\|_1$
  for each $0 \leq j \leq U$ do
      for each $\phi \in \Phi_j$ do
          set $\lambda_\phi = \sum_{t_i \in D} \lambda_{\phi t_i} + \text{Lap}\left(\frac{\Delta}{\epsilon}\right)$
      end for
  end for
  Let $\bar{J}(D, \omega) = \sum_{j=1}^{U} \sum_{\phi \in \Phi_j} \lambda_\phi \phi(\omega)$
  Compute $\omega_* = \arg\min_\omega \bar{J}(D, \omega)$ return $\omega_*$
```

The functional mechanism can be applied to linear regression. The expansion of objective function $J(D,\omega)$ for linear regression only involves monomials in Φ_0, Φ_1 and Φ_2.

$$\begin{aligned} J(D,\omega) &= \sum_{t_i \in D} (y_i - t_i^T \omega)^2 \\ &= \sum_{t_i \in D} (y_i)^2 - \sum_{j=1}^{d} \left(2 \sum_{t_i \in D} y_i t_{ij} \right) w_j + \sum_{1 \le j,l \le d} \left(\sum_{t_i \in D} t_{ij} t_{il} \right) w_j w_l. \end{aligned}$$

And the scale of Laplace noise can be bounded by,

$$\begin{aligned} \Delta &= 2 \max_t \sum_{j=1}^{U} \sum_{\phi \in \Phi_j} \|\lambda_{\phi t}\|_1 \\ &\le 2(1 + 2d + d^2). \end{aligned}$$

Note that this sensitivity becomes very large as d increases. Adding noises of this magnitude to every coefficient, and then optimizing for that objective function results in poor performances.

For other regression tasks, e.g. logistic regression, where the objective function are not a polynomial with finite order, Zhang et al. [11] proposed to use the first two order terms of Taylor expansion to approximate this kind of objective function.

Functional Mechanism adds more perturbation to the objective function than the previous method, and thus perform even worse.

2.6.3 Make an Existing Algorithm Private

Another approach is to take a non-private optimization algorithm, and to apply the Laplace Mechanism or the Exponential Mechanism to ensure that every step is private. Often times, one takes an iterative algorithm for an optimization task, and then makes each iteration private. Here, one main parameter is the number of iterations. When the number is too small, then the algorithm is far from converging. On the other hand, when the number is too large, each iteration has very little privacy budget, and too much noise is added to each iteration. Intuitively, this method is sub-optimal because the amount of perturbation added in this approach ensures that outputting all the intermediate results together is private, whereas only the final output is needed.

DPLloyd: Differentially Private Lloyd Algorithm for k-means Clustering

The k-means clustering problem has been used as a motivating application for PINQ ([7]), a platform for interactive privacy preserving data analysis. McSherry implemented k-means clustering using the PINQ system ([27]). We call this the DPLloyd approach. DPLloyd fixes the total number of iterations to be 5 in [7]. It adds Laplacian noise to the iterative update step in the Lloyd algorithm. Each iteration requires computing the total number of points in a cluster and, for each dimension, the sum of the coordinates of the data points in a cluster. Let d be the number of dimensions. Then, each tuple is involved in answering dt sum queries and t count queries. To bound the sensitivity of the sum query to a small number r, each dimension is normalized to $[-r, r]$. Thus, the global sensitivity of DPLloyd is $(dr + 1)t$ [5], and each query is answered by adding Laplacian noise $\mathsf{Lap}\left(\frac{(dr+1)t}{\epsilon}\right)$.

Algorithm 2 NoisyCentroidUpdate

Input: d: number of dimensions, C: cluster, ϵ_s: privacy budget for sum queries, ϵ_c: privacy budget for count queries

```
$count \leftarrow |C| + \mathsf{Lap}\left(\frac{1}{\epsilon_c}\right)$
for each dimension $i$ ($i = 1, 2, \ldots, d$) do
    $sum_i \leftarrow \sum_{x^\ell \in C} x_i^\ell + \mathsf{Lap}\left(\frac{1}{\epsilon_s}\right)$
    $o_i \leftarrow \Pi_{[-r,r]}\left(\frac{sum_i}{count}\right)$
end for
return Cluster centroids $\langle o_1, o_2, \ldots, o_d \rangle$
```

The algorithm is given in Algorithm 3. The overall structure of DPLloyd is to first select initial values, and then iteratively improve them. This same algorithmic structure also applies to many other data analysis tasks, such as linear regression, SVM, etc. When making such an interactive and iterative algorithm differentially private, there are several important decisions one has to make.

Algorithm 3 DPLLOYD

```
Input: $D$: dataset, $d$: number of dimensions, $[-r, r]$: dataset range, $k$: number of clusters, $t$: number of iterations, $IC$: set of initial centroids, $\epsilon$: privacy budget
if $IC$ is empty then
    Randomly select $k$ points $\{o^1, o^2, \ldots, o^k\}$ as initial centroids
else
    $\{o^1, o^2, \ldots, o^k\} \leftarrow IC$
end if
$\epsilon' \leftarrow \frac{\epsilon}{(dr+1)t}$
for Loop $t$ times do
    for each $j$ $(j = 1, 2, \ldots, k)$ do
        Cluster $C^j \leftarrow \{x^\ell : \|x^\ell - o^i\| \leq \|x^\ell - o^j\|, x^\ell \in D, \forall 1 \leq i \leq k\}$
        $\langle o^j_1, o^j_2, \ldots, o^j_d \rangle \leftarrow$ NOISYCENTROIDUPDATE$(d, C^j, \epsilon', \epsilon')$
    end for
end for
return $\{o^1, o^2, \ldots, o^k\}$
```

The first decision is how to select the initial values. In the standard, non-private setting, a purely random choice may suffice, since one could repeat the algorithm multiple times and choose the best result among them. With privacy constraints, however, running the interactive algorithm multiple times means that each run can use only a fraction of the total privacy budget, causing the results to be less accurate. Thus the choice of initial values becomes more important. In the case of k-means clustering, many methods for choosing the initial points have been developed, see, e.g. [28]. However, these methods all need access to the dataset, and it is unclear how to perform them in a differentially privately way. Therefore, DPLloyd randomly generates k points as the initial centroids.

The second decision is how many iterations one runs. A large number of iterations causes too much noises being added. A small number of iterations may be insufficient for the algorithm to converge. Existing approaches fixes a number. However, intuitively the number of rounds would depend both on the available privacy budget ϵ and the quality of the initial values. With a smaller privacy budget, one should run fewer number of rounds, to avoid the results being overwhelmed by too much noise.

The third decision is how to allocate the privacy budget across different rounds. Almost by default existing approaches allocate privacy budget equally across different rounds.

However, intuitively this is not optimal. In later rounds, when one gets closer to the optimal value, it is desirable to have a larger privacy budget.

In the implementation of DPLloyd in PINQ, it is proposed to run 5 iterations, with equal privacy budget allocation for each round. It appears that this setting works quite well across across many datasets. In [29], when a method newly proposed for k-means clustering was compared with DPLloyd, the experiments were done by running DPLloyd with 20, 80, and 200 iterations, resulting in artificially poor performance of DPLloyd.

In [30], it is proposed that when the number of rounds is not fixed, one uses exponentially decreasing allocation of privacy budgets, i.e., $\frac{\epsilon}{2}$ in the first round, $\frac{\epsilon}{4}$ in the second round, and so on. This mostly likely results in deteriorating performance when the number of rounds increases. Using this method, in later rounds, when one hopes to get closer to the optimal value, increasingly larger noises are added due to the exponentially decreasing privacy budget. If one does not allocate privacy budgets equally across all rounds, then one should allocate smaller privacy budgets for the earlier rounds and larger privacy budgets for the later rounds, although one cannot do that without knowing fixing the total number of rounds.

DiffPID3: Differential Private ID3 Algorithm for Decision Tree Classification

In [16], the algorithm for constructing an ID3 decision tree classifier is made differentially private. When the ID3 algorithm needs to get the number of tuples with a specific feature value, it queries the SuLQ interface to get the corresponding noise count. The DiffPID3 algorithm in [9] improved this approach by redesigning the classic ID3 classifier construction algorithm to consider the feature quality function with lower sensitivity and using the exponential mechanism to evaluate all the attributes simultaneously. Specifically, the DiffPID3 algorithm starts with the most general partition of the underlying dataset. Then, the algorithm chooses the attribute that maximizes the purity by using the exponential mechanism and splits the dataset with the selected attribute. The same process is applied recursively on each subset of the dataset until there are no further splits that improve the purity.

Sample and Aggregation

Algorithm 4 DiffPID3

```
INPUT: $D$: Dataset, $\mathcal{A} = \{A_1, \ldots, A_d\}$: set of attributes, $C$: class attribute, $\epsilon$: privacy parameter, $d$: maximal tree depth, $\epsilon' = \frac{\epsilon}{2(d+1)}$: privacy parameter for each recursive invocation
  $t \leftarrow \max_{A \in \mathcal{A}} |A|$
  $N_D \leftarrow \mathrm{NoisyCount}_{\epsilon'}(D)$
  if $\mathcal{A} = \emptyset$ or $d = 0$ or $\frac{N_D}{t|C|} < \frac{\sqrt{2}}{\epsilon'}$ then
      $D_c \leftarrow \mathrm{Partition}(D, \forall c \in C : r_C = c)$
      $\forall c \in C : N_c = \mathrm{NoisyCount}_{\epsilon'}(D_c)$ return a leaf labeled with $\mathrm{argmax}_c(N_c)$
  end if
  $\bar{A} \leftarrow \mathrm{ExpMech}_{\epsilon'}(\mathcal{A}, q)$
  $D_i \leftarrow \mathrm{Partition}(D, \forall i \in \bar{A} : r_{\bar{A}} = i)$
  for each $i \in \bar{A}$ do
      $\mathrm{Subtree}_i \leftarrow \mathrm{DiffPID3}(D_i, \mathcal{A} \setminus \bar{A}, C, d-1, \epsilon')$
  end forreturn a tree with a root node labeled $\bar{A}$ and edges labeled 1 to $|\bar{A}|$ each going to $\mathrm{subtree}_i$.
```

The Sample and Aggregation framework provides another approach to deal with private data analysis, and grows out of the concept of local sensitivity [25]. However, to satisfy differential privacy by adding noises based on the local sensitivity rather than the global sensitivity, one needs to be able to analyze the local sensitivity and come up with a smooth bound of it [25]. Oftentimes the function is too complex for analyzing the local sensitivity. k-means clustering is such an example. While intuitively adding one data point is unlikely to change the result by much for most datasets, it is difficult to analyze the effect of adding one data point, in part because of the iterative nature of the algorithm.

The Sample and Aggregation framework is introduced to handle such cases. Given a dataset D and a function f, SAF first partitions D into m blocks, then it evaluates f on each of the block, and finally it privately aggregates results from all blocks into a single one. The effectiveness of this approach depends on two assumptions. First, $f(D)$ can be approximated by evaluating f on the partitions (i.e., blocks) of D. Second, the aggregation step can be designed to be of low sensitivity, e.g., by taking average of all outputs. Since any single tuple in D falls in one and only one block, adding one tuple can affect at most one block's result, limiting the sensitivity of the aggregation step. Thus one can add less noise in the final step to satisfy differential privacy. Figure 2.2 illustrates the Sample and Aggregation framework.

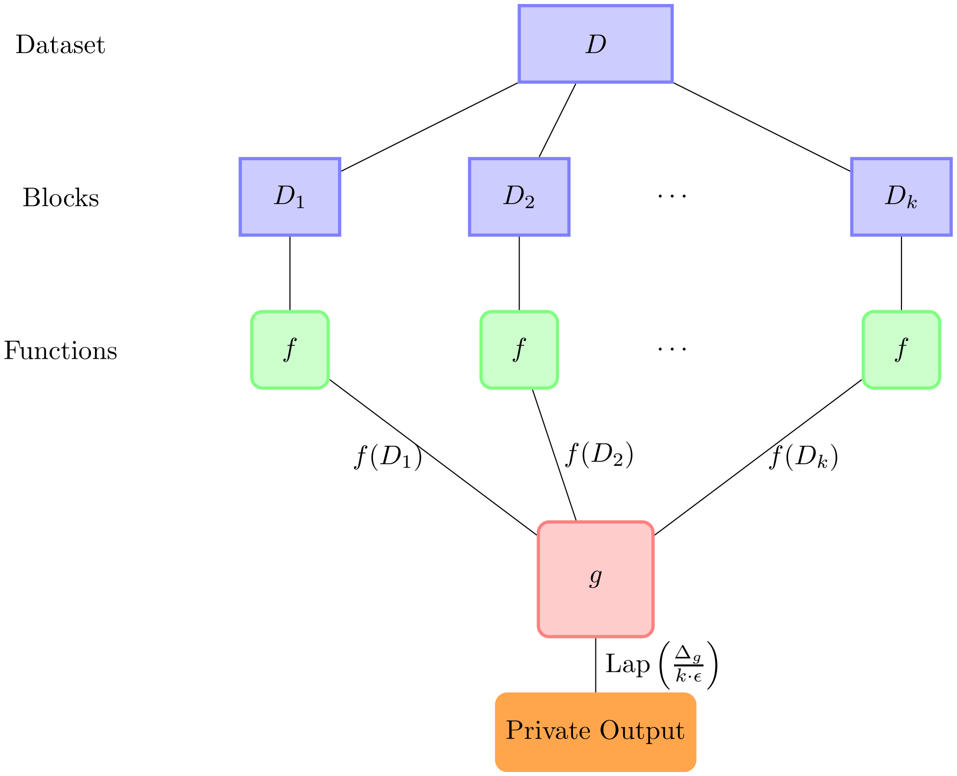


Figure 2.2.: An illustration of the sample and aggregate framework.

```
Algorithm 5 Sample and Aggregate
INPUT Dataset $D$, length of dataset $n$, privacy parameter $\epsilon$, output range (min, max)
  Let $l = n^{0.4}$
  Randomly partition $D$ in to $k$ disjoint blocks, $D_1, \ldots, D_k$
  for each $i \in \{1, \ldots, k\}$ do
      $O_i \leftarrow$ Output of user application on dataset $D_i$
      Clipping $O_i$ to the range $(\min, \max)$
  end for
  $A \leftarrow \frac{1}{k}\sum_{i=1}^{k} O_i + \text{Lap}\left(\frac{|\max - \min|}{k\cdot\epsilon}\right)$
```

The Sample and Aggregation framework was implemented in the GUPT system [29]. Authors of [29] implemented k-means clustering and used it to illustrate the effectiveness of GUPT. We call this algorithm GkM. Given a dataset D, it first partitions D into

$m = N^{0.4}$ blocks $D_1, D_2, \ldots, D_m$. Then, for each block D_j, it calculates its k centroids $o^{j,1}, o^{j,2}, \ldots, o^{j,k}$. Finally, it computes an average on all the blocks to output the j-th centroid

$$\mathbf{c}^j = \frac{1}{m}\sum_{i=1}^{m} o^{j,i} + \mathsf{Lap}\left(\frac{k(max - min)}{m \cdot \epsilon}\right), \tag{2.11}$$

where $[min, max]$ is the estimated output range.

2.6.4 Iterative Local Search via EM

Instead of adding making individual steps in an optimization algorithm private, another approach is to iteratively apply the exponential mechanism to gradually improve the current choice of ω. In order to do this, one has to generate a candidate set, e.g., by generating multiple perturbation of the current ω, and then selects among the set in a private fashion. We now look at some examples of such algorithms.

PrivGene: Differentially Private Model Fitting Using Genetic Algorithms

PrivGene [12] is a general-purpose differentially private model fitting framework based on genetic algorithms. Given a dataset D and a fitting function $f(D, \omega)$ that measures how well the parameter ω fits the dataset D, the PrivGene algorithm initializes a candidate set of possible parameters ω and iteratively refines them by mimicking the process of natural evolution. Specifically, in each iteration, PrivGene selects m' parameters from the candidate set, and generates from them offsprings by crossover and mutation. Then, it creates a new parameter set, which includes all and only the offsprings. At the end of the last iteration, a single parameter is selected and outputted as the final result.

This algorithm is given in Algorithm 6. PrivGene is applied to logistic regression, SVM, and k-means clustering. In the case of k-means clustering, the NICV formula in Equation 2.8, more precisely its non-normalized version, is used as the fitting function f, and the set of k cluster centroids is defined as parameter ω. Initially, the candidate set is populated with 200 sets of cluster centroids randomly sampled from the data space, each set containing exactly k centroids. Then, the algorithm runs iteratively for $N\epsilon/(800m')$ rounds, where m' is empirically set to 10, and N is the dataset size.

Algorithm 6 PrivGene

```
INPUT $D$: Dataset, $J$: objective function, $\epsilon$: privacy parameter, $m$, $m'$ :sizes of candidate set $\Omega$ and selected set $\Omega'$, $r$: number of iterations
Output $\omega$: best parameter vector identified by PrivGene
  Initialize candidate set $\Omega$ with $m$ randomly generated vectors
  for $i = 1$ to $r - 1$ do
      $\Omega' \leftarrow \mathrm{DPSelect}(D, J, \Omega, m', \epsilon/r)$
      $\Omega \leftarrow \{\}$
      for $j = 1$ to $m/2$ do
          Randomly choose two vectors $\omega^1, \omega^2 \in \Omega'$ as parent parameters
          Compute $(v^1, v^2) \leftarrow \mathrm{Crossover}(\omega^1, \omega^2)$
          $\mathrm{Mutate}(v^1)$ and $\mathrm{Mutate}(v^2)$
          Add two offspring parameters $v^1, v^2$ to $\Omega$
      end for
  end for
  $\{\omega\} \leftarrow \mathrm{DPSelect}(D, J, \Omega, 1, \epsilon/r)$ return $\omega$
```

Algorithm 7 DPSelect

```
INPUT $D$: Dataset, $J$: objective function, $\Omega$: parameter candidate set, $m'$: number of parameter vectors to be selected from $\Omega$, $\epsilon_s$: privacy parameter
Output $\Omega'$: set of selected parameter vectors
  $\Omega' \leftarrow \{\}$
  For each $\omega \in \Omega$, compute $J(D, \omega)$
  for $i = 1$ to $m'$ do
      Use privacy budget $\epsilon_s/m'$ to apply the exponential mechanism to select the parameter vector $\omega_*$ from $\Omega$ that aims to minimize $J(D, \omega_*)$.
      Remove $\omega_*$ from $\Omega$, and add $\omega_*$ to $\Omega'$
  end forreturn $\Omega_{\prime}$
```

While the idea of making a genetic programming algorithm differentially private is interesting, the effectiveness of Algorithm 6 is questionable for several reasons. First, the crossover operation often does not result in competitive candidates. Second, with crossover and mutation, the convergence rate is low, which means a larger number of iterations are needed. Third, for each iteration, the algorithm requires making m' selections, with every single one of them consuming some privacy budget.

Iterative Local Search

A more effective local search algorithm can be developed using some ideas from the PrivGene paper, but does not use features of genetic programming. Such an algorithm is implemented in the code accompanying the PrivGene paper [12], even though the algorithm did not appear in the paper. We give the algorithm below.

Algorithm 8 ExpSearch

INPUT D: Dataset, J: objective function, ϵ: privacy parameter, r: number of iterations, ω_0: initial parameter, s: step of search, $\beta < 1$: scaling parameter

Output ω: selected parameter

```
$\omega \leftarrow \omega_0$
for $i = 1$ to $r - 1$ do
    $\Omega \leftarrow \{\}$
    for $j \in \{1..d\}$ do
        $\omega_1 \leftarrow \omega$ with $j$'s attribute $+ s$
        $\omega_2 \leftarrow \omega$ with $j$'s attribute $- s$
        $\Omega \leftarrow \Omega \cup \{\omega_1, \omega_2\}$
        compute $J(D, \omega_1)$ and $J(D, \omega_2)$
    end for
    $\omega \leftarrow$ Use privacy budget $\epsilon/r$ to apply the exponential mechanism to select the parameter vector $\omega_*$ from $\Omega$ that aims to maximize $J(D, \omega_*)$.
    $s \leftarrow s * \beta$
end forreturn $\omega$
```

This algorithm has several interesting ideas. Each round, it uses the exponential mechanism to select a single local perturbation that improves the current solution the best. Compared with PrivGene, which selects multiple candidates (for the purpose of using crossovers to generate the pool of candidates), this means that more privacy budget can be used in each selection. Since only one candidate is selected, there is no crossover. The mutation step takes the form of perturbing the coefficient in one dimension. That is, each iteration can be viewed as moving along one dimension towards a potentially better parameter. Finally, the perturbation step s exponentially decays so that the amount of changes decreases. This makes sense as when one starts to converge to the optimal parameter, smaller adjustments are needed. Also, this feature of exponential decay of the perturbation step can also take advantage of the enhanced exponential mechanism, to be discussed below. This method is most useful when the goal is to find a vector of coefficients, as in the case of logistic regression and SVM.

Enhanced Exponential Mechanism

An Enhanced version of the Exponential Mechanism (EEM) is proposed in [12], which can be used in the iterative local search algorithm. Recall that the quality function we use is the optimization objective function $J(D, \omega)$. In the standard Exponential Mechanism, one considers the maximal difference between the values of the quality function on two neighboring datasets D and D', i.e., $\Delta J = \max_{\forall \omega, D \simeq D'} |J(D, \omega) - J(D', \omega)|$.

EEM is suitable for the case where the dependency of the quality function on the dataset D can be computed by summing up some score for each record $t \in D$, i.e.,

$$J(D, \omega) = c(\omega) + \sum_{x \in D} L_\omega(x).$$

In this case, when making a selection among a set Ω, one could also use as global sensitivity the maximal difference between $L_\omega(x)$ and $L_{\omega'}(x)$ where $\omega, \omega' \in \Omega$, and x is any data element in the input dataset. This is particularly effectively in the local search paradigm where the set of candidates are all minor perturbations, and thus $\max_{x,\omega,\omega'}(|L_\omega(x) - L_{\omega'}(x)|)$ may be small.

In EEM, one selects $\omega \in \Omega$ with probability proportional to $e^{\epsilon J(D,\omega)/(2\Delta_J)}$, where

$$\Delta = \min \left\{ \Delta_1 = \max_{x, x' \in D, \omega \in \Omega} |L_\omega(x) - L_\omega(x')|, \Delta_2 = \max_{x \in D, \omega, \omega' \in \Omega} |L_\omega(x) - L_{\omega'}(x)| \right\}.$$

Δ_1 is exactly the global sensitivity used in the standard exponential mechanism (here the bounded interpretation of DP is used), while Δ_2 is a new global sensitivity designed specifically for additive quality functions.

When EEM is used in PrivGene and iterative local search, Δ_2 is usually smaller than Δ_1. This is because that as more iterations are performed, the quality of the parameter vectors in the candidate set becomes increasingly close to each other as the algorithm converges to (possibly local) optimal. Thus, it is likely that the maximum value of $L_\omega(x) - L_{\omega'}(x)$

gradually decreases with the number of iterations performed, leading to decreasing Δ_2. Δ_1, on the other hand, is not significantly affected by this phenomenon. Therefore, EEM can make more accurate selection in each iteration as the algorithm converges.

2.6.5 Histograms Optimized for Optimization

The final approach we consider is to publish a synopsis of the dataset, often in the form of a noisy histogram, so that synthetic datasets can be generated and optimizers can be learned from these synthetic datasets. Publishing a synopsis enables additional exploratory and predictive data analysis tasks to be performed, and can be argued to be more preferred.

Uniform Griding and Its Extensions

For low-dimensional datasets with numerical attributes, UG and its extension can be applied. UG (Uniform Griding) is a simple algorithm proposed in [31] for producing synopsis of 2-dimensional datasets that can be used to answer rectangular range queries (i.e., how many data points there are in a rectangular range) with high accuracy. The algorithm partitions the space into $M = m \times m$ equal-width grid cells, and then releases the noisy count in each cell. It is observed that for counting queries, a larger M value results in higher errors because more noises are added, and a smaller M value results in higher errors due to the fact that points within cells may be distributed nonuniformly, and queries including a portion of these cells may be answered inaccurately. To balance these two kinds of errors, it is suggested to set

$$m = \sqrt{\frac{N\epsilon}{10}}, \text{ or equivalently, } M = \frac{N\epsilon}{10} \tag{2.12}$$

It has been shown that UG performs quite well for answering rectangular range queries [31].

In [32], UG is extended to higher-dimensional case by setting

$$M = \left(\frac{N\epsilon}{\theta}\right)^{\frac{2d}{2+d}}, \tag{2.13}$$

where $\theta = 10$. And the new algorithm is called extended uniform griding, EUG. When the dimensionality d increases, this approach does not scale very well.

Histogram Publishing for estimating M-Estimators

Lei [33] proposed a scheme to release differentially private histogram tailored for the M-estimator. Similar to UG and EUG, it partitions the data space into equal-width grid cells. However, it uses a different method to determine how many grid cells to use. Given a d-dimensional dataset with N tuples, statistical analysis in [33] suggests that

$$M = \left(\frac{N}{\sqrt{\log(N)}} \right)^{\frac{2d}{2+d}}. \tag{2.14}$$

Theoretical bounds on accuracy for M-estimator is a generalization of maximum likelihood estimation. Given a dataset $D = \{x^1, x^2, \ldots, x^N\}$ and a target function ρ, it determines a parameter ω_*, such that

$$\omega_* = \arg\min_{\omega} \sum_{\ell=1}^{N} \rho(x^{\ell}, \omega).$$

Note that the only difference the above approach has from UG is in how the number of cells is determined. We note that unlike UG, the above approach for choosing M does not depend on ϵ.

DiffGen: Differentially Private Anonymization Based on Generalization

Mohammed et al. [34] proposed DiffGen to publish histogram for classification under differential privacy. It consists of two steps, partition and perturbation. Given a dataset D and taxonomy trees for each predictor attribute, the partition step starts by generalizing all attribute's values into the topmost nodes in their taxonomy trees and then iteratively selects one attribute's taxonomy tree node at a time for specialization by using the standard exponential mechanism. The quality of each candidate specialization is based on the same heuristics used by the decision tree constructions, such as information gain and majority class. The partition step terminates after a given number of specializations. The perturbation step injects Laplace noise into each cell of the partition and outputs all the cells with their noisy counts as the noisy synopsis of the data. Privacy budget needs to be evenly distributed to all the levels in the tree. Thus, only a small portion of budget can be assigned to each node splitting. This would result the histogram structured to be far from optimal and result performance of the algorithm.

3. DIFFERENTIALLY PRIVATE DATA PUBLICATION FOR CLASSIFICATION

3.1 Introduction

Classification is an important tool for data analysis. However, publishing parameters of a classifier learned from a dataset can result in privacy concerns [35], [36]. One way to deal with the privacy concerns is to conduct classification while satisfying differential privacy [5]. Several approaches for learning classifiers while satisfying differential privacy have been proposed in recent years. Some methods compute a classifier as the output [16], [8], [9], [10], [37], [38]. Other methods [34], [39], [12] publish a synopsis of the dataset, often in the form of a noisy histogram, so that synthetic datasets can be generated and classifiers can be learned from these synthetic datasets. Publishing a synopsis enables additional exploratory and predictive data analysis tasks to be performed, and can be argued to be more preferred.

Publishing noisy histograms for one-dimensional or two-dimensional datasets has been studied extensively in recent years, see [40] for a recent survey. However, as observed in [39], [41] these approaches do not work well when the number of attributes/dimensions goes above a few. Many datasets that are of interest have multiple attributes. For a multi-attribute dataset with more than a dozen or so attributes, publishing a histogram with all the attributes results in a sparse histogram where noises may overwhelm the true counts. Therefore it is necessary to select a subset of the attributes that are "useful" for the intended data analysis tasks, and to determine how to discretize the attributes. These selections partition the domain into a number of cells. We call the result a "grid". Once a grid is selected, the next step is straightforward: one adds Laplace noises [5] to the cell counts to produce a noisy histogram.

Thus the key design choice in algorithms for publishing noisy histograms is how to select a suitable grid. Publishing a histogram is similar to performing generalization for the purpose of achieving k-anonymization, since the exact attribute values for records within a cell no longer matter, and only the cell boundary and the number of records in a cell matter. A key challenge studied in research on k-anonymization was also how to find a high-quality grid [42], [43], [44]. However, these proposed methods for k-anonymization have been found to be vulnerable to attacks exploiting background information, e.g., the minimality attack [45], [46]. Fortunately, an approach to select a grid while satisfying differential privacy, as proposed in this chapter, can defend against these attacks.

In this chapter we propose the PrivPfC (Private Publication for Classification) approach for publishing projected histograms. On the key decision of how to select a grid, PrivPfC differs from previous approaches in that it selects a high-quality grid in a single step, whereas previous approaches use an iterative process and as a consequence suffer from two weaknesses. First, an iterative process has to divide the privacy budget among all the iterations, causing the choice made in each iteration to have significant noise. Second, an iterative process is a greedy process and tends to result in a sub-optimal global choice even without considering noises.

The exponential mechanism [14] enables the private selection of a grid in a single step. However, there are a number of challenges to use it effectively. One needs to generate a set of candidate grids that include the high-quality grids, without making the candidate set too large, which affects both running time and accuracy. Furthermore, one needs a quality function that can effectively identify high-quality grids and simultaneously has a low sensitivity.

The first contribution of this chapter is PrivPfC, an algorithm for privately publishing noisy histograms optimized for classification. PrivPfC has two novel ideas. One is a method to enumerate through candidate grids when given a cap on how many grids the algorithm is allowed to consider; and the other is a new quality function that enables the selection of a high-quality "grid". This quality function considers the impact of injected noises on the classification accuracy, adapts to the privacy parameter ϵ, and has a low sensitivity. As demonstrated in [14], any mechanism that satisfies differential privacy can be simulated using the exponential mechanism; thus conceptually any private data analysis problem can be solved by finding a way to enumerate the likely solutions and an effective quality function. PrivPfC solves the problem of finding a suitable grid for publishing histograms optimized for classification.

Our second contribution is that, through extensive experiments on real datasets, we have compared PrivPfC against other state-of-the-art methods for data publishing as well as private classification, demonstrating that PrivPfC improves the state-of-the-art. We also analyze variants of competing algorithms, showing that their weaknesses come from the iterative structure of their algorithms. We note that the fact that PrivPfC outperforms state-of-the-art algorithms specifically designed for privately publishing classifiers is quite counter-intuitive. PrivPfC publishes a histogram, which contains more information than a classifier; thus one would expect the classifiers it produces are less accurate. Experimental results demonstrate otherwise. We believe this points to the posibility of designing better private classification algorithms by using as few steps as possible, avoiding spreading the privacy budget too thin.

The rest of this chapter is organized as follows. Our PrivPfC approach is presented in Section 3.2. The experimental results are shown in Section 3.3.

3.2 PrivPfC Framework

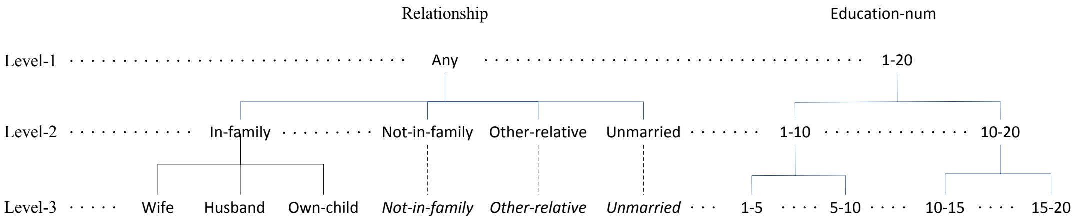


Figure 3.1.: Taxonomy hierarchies of Relationship attribute and Education-num attribute.

We consider a dataset with a set of predictor variables and one response variable. The predictor variables can be numerical or categorical. Following [42], [47], [48], [34], for each predictor variable A_i, we assume the existence of a *taxonomy hierarchy* (also called a *generalization hierarchy* in the literature). Figure 3.1 shows the taxonomy hierarchies for *Relationship*, a categorical variable, and *Education-num*, a numerical variable. In a hierarchy, the root node represents the whole domain of the variable, and a parent node is a generalization (or a cover) of its children. Child nodes under the same parent node are semantically closer to each other than to nodes under a different parent node.

Each *level* of a predictor variable's taxonomy hierarchy forms a partition of its domain. On the basis of the taxonomy hierarchy and its levels, we introduce the notion of a grid.

Definition 3.2.1 (Grid) *Let* $\mathsf{A} = \{A_1, \ldots, A_d\}$ *be the set of predictor variables in a dataset and* $\{T_1, \ldots, T_d\}$ *be their taxonomy hierarchies respectively. Let* h_i *be the height of* T_i*,* $1 \leq i \leq d$*. Then, a* grid g *is given by* $\langle \ell_1, \ldots, \ell_d \rangle$*, where* $1 \leq \ell_i \leq h_i$ *and* $1 \leq i \leq d$*. Such a grid* g *defines a partitioning of the data domain into* cells *where each attribute* A_i *is partitioned into the values at level* ℓ_i*. The number of cells of a grid is* $\Pi_{i=1}^{d} |T_i[l_i]|$*, where* $|T_i[l_i]|$ *is the number of nodes in the level* l_i *of the hierarchy* T_i*. And the number of all possible grids is* $\Pi_{i=1}^{d} h_i$*.*

Definition 3.2.2 (Histogram) *Given a dataset* D *and a grid* g*, a histogram* $\mathsf{H}(D, g)$ *partitions* D *into cells according to* g*, and in each cell outputs the number of records for each value of the response variable.*

PrivPfC publishes $\tilde{\mathsf{H}}(D, g)$, a noisy histogram of the input dataset D, which adds Laplace noise into the counts in the histogram $\mathsf{H}(D, g)$. The key challenge lies in selecting a suitable grid g. Our approach is to define a quality score for each grid, which measures the usefulness for classification of each grid, and apply the exponential mechanism [14] to privately select a grid.

3.2.1 The Quality Function

The quality function needs to satisfy two conditions. First, it needs to accurately measure the desirability of using a particular grid g. Second, it should have a low sensitivity.

Intuitively, we want to ensure that classifiers learned from $\tilde{\mathsf{H}}(D, g)$, a noisy histogram of D using g to partition the data domain, are close to classifiers learned from D directly. Furthermore, we desire this to hold regardless of which particular classification algorithm is used.

We propose to define the quality function to maximize the number of records in D that are classified correctly by the following classifier: for each cell in the grid defined by g, it predicts the class with highest count according to the noisy histogram $\tilde{\mathsf{H}}(D, g)$. This classifier is in the same spirit as histogram classifiers [49], and we use $HC^{\tilde{\mathsf{H}}(D,g)}$ to denote it. When a grid g is fixed, the noisy histogram includes random noises; therefore, the number of correctly classified records is a random variable. We use the expected value of this random variable as the quality function. Therefore, the quality of the grid g can be defined as:

Definition 3.2.3 (Grid Quality) *Given a dataset D with k different class labels, $\mathbb{L} = \{1, 2, \ldots, k\}$, a grid g and ϵ for the parameter of adding Laplace noise to the counts, the grid quality is measured by the expected number of correctly classified records of the histogram classifier $HC^{\tilde{\mathsf{H}}(D,g)}$:*

$$\mathsf{gq}(D, g) = \sum_{c \in g} \sum_{i=1}^{k} n_c^i \cdot p_c^i, \tag{3.1}$$

where $i \in \mathbb{L}$ ranges over the class labels, n_c^i is the number of data points in the cell c with class label i and p_c^i is the probability that class i is the dominant class in cell c (i.e., with the highest noise count) after injecting Laplace noises. The probability p_c^i is given below:

$$\begin{aligned}
p_c^i &= \mathsf{Pr}\left[\textit{Class } i \textit{ is the dominant class after adding noise}\right] \\
&= \mathsf{Pr}\left[n_c^i + Z_i \geq \max_{j \in \mathbb{L}/\{i\}} \left(n_c^j + Z_j\right)\right] \\
&= \int_{-\infty}^{\infty} \left(\mathsf{Pr}\left[n_c^i + Z_i = x\right] \prod_{j \in \mathbb{L}/\{i\}} \mathsf{Pr}\left[n_c^j + Z_j < x\right] \right) \mathrm{d}x \\
&= \int_{-\infty}^{\infty} \left(f\left(x - n_c^i\right) \prod_{j \in \mathbb{L}/\{i\}} F\left(x - n_c^j\right) \right) \mathrm{d}x, \qquad (3.2)
\end{aligned}$$

where Z_i is the Laplace noise added to class i's count, and $f(\cdot)$ and $F(\cdot)$ are, respectively, the probabilistic density function and the cumulative distribution function of the Laplace distribution $\mathrm{Lap}(1/\epsilon)$. The probability p_c^i depends on ϵ, the privacy budget for perturbation, as well as on the counts of the various classes in the cell c.

Intuitively, since the grid quality function gq (Eq. 3.1) counts number of records, it should have a low sensitivity, since adding or removing a record affects only one of the counts, and changes the count by just 1. However, changing the counts also affects the probabilities. Thus, analyzing the sensitivity of the quality function is quite non-trivial.

3.2.2 Sensitivity in the Binary Classification Case

We first study the sensitivity of the grid quality (Eq. 3.1) in the special case where the response variable is binary.

Lemma 1 (Grid Quality for Binary Classification) *Given a dataset D with class labels $\mathbb{L} = \{1, 2\}$, the global sensitivity of the quality function gq is bounded by 1.1.*

More specifically, for each ϵ value, the sensitivity is given by

$$\Delta_{\mathsf{gq}}(\epsilon) = \max_{x \in \{1,2,3,\ldots\}} f_\epsilon(x), \qquad (3.3)$$

where

$$f_\epsilon(x) = \left|(x-1)\cdot\left(\frac{e^{-\epsilon(x-1)}}{2}\left(1+\frac{\epsilon(x-1)}{2}\right)\right.\right.$$
$$\left.\left.-\frac{e^{-\epsilon x}}{2}\left(1+\frac{\epsilon x}{2}\right)\right)+\left(1-\frac{e^{-\epsilon x}}{2}\left(1+\frac{\epsilon x}{2}\right)\right)\right|, \tag{3.4}$$

The global maximum points for $f_\epsilon(x)$, where x ranges over all positive real number, is given below.

$$x^* = \frac{\epsilon e^\epsilon + \sqrt{2-(4-2e^\epsilon)\,e^\epsilon + \epsilon^2 e^\epsilon}}{-\epsilon+\epsilon e^\epsilon}.$$

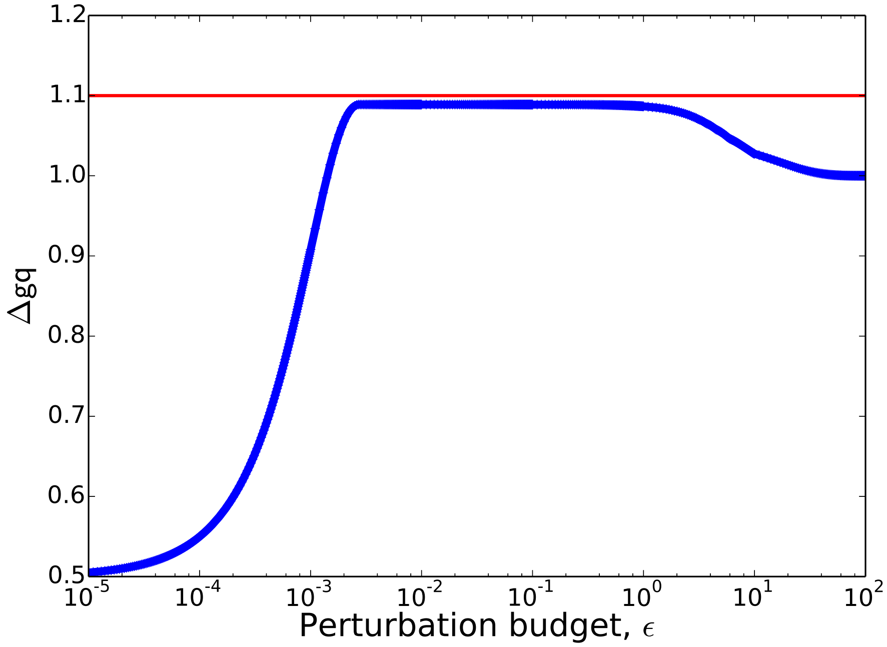


Figure 3.2.: Illustration of the sensitivity of grid quality (Eq. 3.3).

Lemma 2 gives the distribution of the difference of two i.i.d. Laplace random variables, which will be used in the proof of Lemma 1.

Lemma 2 ([50]) *Let Z_1 and Z_2 be two i.i.d. random variables that follow the Laplace distribution with mean 0 and scale $\frac{1}{\epsilon}$. Then the density of their difference $Y = Z_1 - Z_2$ is*

$$f_Y(y) = \frac{\epsilon}{4}e^{-\epsilon|y|}(1+\epsilon|y|) \qquad -\infty < y < \infty,$$

and the corresponding cumulative distribution function is

$$F_Y(y) = \begin{cases} 1 - \dfrac{e^{-\epsilon y}}{2}\left(1 + \dfrac{\epsilon y}{2}\right), & \text{if } y \geq 0, \\ \dfrac{e^{\epsilon y}}{2}\left(1 - \dfrac{\epsilon y}{2}\right), & \text{otherwise.} \end{cases} \tag{3.5}$$

The following is the proof of Lemma 1.

Proof We show the global sensitivity of the grid quality (Eq. 3.1) in the binary classification setting can be safely upperbounded.

Given a dataset D, and without loss of generality we assume that the neighboring dataset of D is $D' = D - t$, where the tuple t has class label 1. The quality value of all cells other than cell e are the same, and $n_e^1 = n'^1_e + 1$ and $n_e^2 = n'^2_e$.

The sensitivity of the grid quality function for binary classification can be computed by,

$$\begin{aligned} \Delta_{\mathsf{gq}} &= |\mathsf{gq}(D, g) - \mathsf{gq}(D', g)| \\ &= |(n_e^1 p_e^1 + n_e^2 p_e^2) - (n_e^{'1} p_e^{'1} + n_e^{'2} p_e^{'2})| \\ &= |p_e^1 + (n_e^1 - 1)(p_e^1 - p_e^{'1}) + n_e^2(p_e^2 - p_e^{'2})| \\ &= |p_e^1 + (n_e^1 - n_e^2 - 1)(p_e^1 - p_e^{'1})|, \end{aligned} \tag{3.6}$$

where p_e^1 is the probability of Class 1 is still the dominant class after adding noise. The last equality holds, because $p_e^2 = 1 - p_e^1$ and $p_e^{'2} = 1 - p_e^{'1}$.

As for p_e^1, by Lemma 2,

$$\begin{aligned} p_e^1 &= \mathsf{Pr}\left[n_e^1 + Z_1 \geq n_e^2 + Z_2\right] \\ &= \mathsf{Pr}\left[Z_2 - Z_1 \leq n_e^1 - n_e^2\right] \\ &= \begin{cases} 1 - \dfrac{e^{-\epsilon(n_e^1 - n_e^2)}}{2}\left(1 + \dfrac{\epsilon(n_e^1 - n_e^2)}{2}\right) & \text{if } n_e^1 - n_e^2 \geq 0 \\ \dfrac{e^{\epsilon(n_e^1 - n_e^2)}}{2}\left(1 - \dfrac{\epsilon(n_e^1 - n_e^2)}{2}\right), & \text{otherwise.} \end{cases} \end{aligned} \tag{3.7}$$

Case 1: $\mathbf{n_e^1 - n_e^2 \geq 1}$.

In this case,

$$n_e^{'1} - n_e^{'2} = n_e^1 - 1 - n_e^2 \geq 0.$$

By Eq. 3.6 and Eq. 3.7, we have

$$\begin{aligned}\Delta_{\mathsf{gq}} = \Bigg|1 - \frac{e^{-\epsilon(n_e^1 - n_e^2)}}{2}\left(1 + \frac{\epsilon(n_e^1 - n_e^2)}{2}\right) + \\ (n_e^1 - n_e^2 - 1)\Bigg[\frac{e^{-\epsilon(n_e^{'1} - n_e^{'2})}}{2}\left(1 + \frac{\epsilon(n_e^{'1} - n_e^{'2})}{2}\right) \\ - \frac{e^{-\epsilon(n_e^1 - n_e^2)}}{2}\left(1 + \frac{\epsilon(n_e^1 - n_e^2)}{2}\right)\Bigg]\Bigg|.\end{aligned}$$

By letting $x = n_e^1 - n_e^2$, we have

$$\begin{aligned}\Delta_{\mathsf{gq}} &= \Bigg|1 - \frac{e^{-\epsilon x}}{2}\left(1 + \frac{\epsilon x}{2}\right) \\ &\quad + (x-1)\Bigg[\frac{e^{-\epsilon(x-1)}}{2}\left(1 + \frac{\epsilon(x-1)}{2}\right) \\ &\quad - \frac{e^{-\epsilon x}}{2}\left(1 + \frac{\epsilon x}{2}\right)\Bigg]\Bigg| \\ &= \Bigg|1 + \frac{(x-1)e^{-\epsilon(x-1)}}{2}\left(1 + \frac{\epsilon(x-1)}{2}\right) \\ &\quad - \frac{xe^{-\epsilon x}}{2}\left(1 + \frac{\epsilon x}{2}\right)\Bigg|,\end{aligned}$$

where $x \geq 1$.

Consider the function

$$g_1(x) = \frac{xe^{-\epsilon x}}{2}(1 + \frac{\epsilon x}{2}).$$

$g_1(x)$ is differentiable and

$$g_1'(x) = \frac{e^{-\epsilon x}}{4}(2 - \epsilon^2 x^2).$$

Thus, by Lagrange's Mean Value Theorem, there exists some ξ between $x-1$ and x (thus $\xi > 0$), so that

$$\begin{aligned}\Delta_{\mathsf{gq}} &= |1 + g_1(x-1) - g_1(x)| \\ &= |1 - g_1'(\xi)| \\ &= \left|1 - \frac{e^{-\epsilon\xi}}{4}\left(2 - \epsilon^2\xi^2\right)\right|.\end{aligned}$$

To bound the expression above, consider another function

$$h(x) = 1 - \frac{e^{-\epsilon x}}{4}\left(2 - \epsilon^2 x^2\right),$$

where $x > 0$. The function $h(x)$ reaches the maximum at the point $\frac{1+\sqrt{3}}{\epsilon}$ with the maximum value 1.1, increases in the interval $\left[0, \frac{1+\sqrt{3}}{\epsilon}\right]$ and decreases in the interval $\left(\frac{1+\sqrt{3}}{\epsilon}, \infty\right)$. When $x \in [0, \frac{1+\sqrt{3}}{\epsilon}]$, $h(0) \leq h(x) \leq h(\frac{1+\sqrt{3}}{\epsilon})$ which means $h(x) \in [0.5, 1.1]$. When $x \in \left(\frac{1+\sqrt{3}}{\epsilon}, \infty\right)$, $h(x)$ decreases and lies in $(1, 1.1]$, because $\lim_{x\to+\infty} h(x) = 1$. Therefore, in this case,

$$\Delta_{\mathsf{gq}} = |h(\xi)| \in [0.5, 1.1].$$

Case 2: $\mathbf{n_e^1 - n_e^2 \leq 0}$

In this case,

$$n_e^{'1} - n_e^{'2} = n_e^1 - 1 - n_e^2 < 0.$$

Similarly, by letting $x = n_e^1 - n_e^2$, we have

$$\begin{aligned}\Delta_{\mathsf{gq}} = &\Bigg| \frac{e^{\epsilon(n_e^1 - n_e^2)}}{2}\left(1 - \frac{\epsilon(n_e^1 - n_e^2)}{2}\right) + \\ &(n_e^1 - n_e^2 - 1)\Bigg[\frac{e^{\epsilon(n_e^1 - n_e^2)}}{2}\left(1 - \frac{\epsilon(n_e^1 - n_e^2)}{2}\right) - \\ &\frac{e^{\epsilon(n_e^1 - n_e^2 - 1)}}{2}\left(1 - \frac{\epsilon(n_e^1 - n_e^2 - 1)}{2}\right)\Bigg]\Bigg| \\ = &\left|\frac{e^{\epsilon x}}{2}\left(1 - \frac{\epsilon x}{2}\right) + \right. \\ &\left.(x-1)\left[\frac{e^{\epsilon x}}{2}\left(1 - \frac{\epsilon x}{2}\right) - \frac{e^{\epsilon(x-1)}}{2}\left(1 - \frac{\epsilon(x-1)}{2}\right)\right]\right| \\ = &\left|\frac{x e^{\epsilon x}}{2}\left(1 - \frac{\epsilon x}{2}\right) - \frac{(x-1)e^{\epsilon(x-1)}}{2}\left(1 - \frac{\epsilon(x-1)}{2}\right)\right|,\end{aligned}$$

where $x \leq 0$.

Let

$$g_2(x) = \frac{x e^{\epsilon x}}{2}\left(1 - \frac{\epsilon x}{2}\right).$$

And its first order derivative is

$$g_2'(x) = \frac{e^{\epsilon x}}{4}\left(2 - \epsilon^2 x^2\right).$$

Then, the sensitivity becomes

$$\Delta_{\mathsf{gq}} = \left|g_2(x) - g_2(x-1)\right|.$$

The derivative $g_2'(x)$ decreases when $x \in \left(-\infty, -\frac{1+\sqrt{3}}{\epsilon}\right)$, increases when $x \in \left[-\frac{1+\sqrt{3}}{\epsilon}, 0\right]$. And when $x = -\frac{1+\sqrt{3}}{\epsilon}$ the function $g_2'(x)$ reaches the minimum value -0.09. Thus, when $x \leq -\frac{1+\sqrt{3}}{\epsilon}$, $g_2'(x) \in [-0.09, 0)$ because $\lim\limits_{x \to -\infty} g_2'(x) = 0$ and $g_2'(x) \in [-0.09, 0.5]$ when $x \in [-\frac{1+\sqrt{3}}{\epsilon}, 0]$. Applying Lagrange's Mean Value Theorem to $g_2(x)$, there exists some η between x and $x-1$, thus $\eta \leq 0$, so that

$$\begin{aligned}\Delta_{\mathsf{gq}} &= |g_2(x) - g_2(x-1)| \\ &= |g_2'(\eta)| \\ &\leq 0.5.\end{aligned}$$

In summary, by considering the above two cases, the global sensitivity for grid quality on binary classification is only determined by the **Case 1**, where the Δ_{gq} reaches its global maximum at

$$x^* = \frac{\epsilon e^\epsilon + \sqrt{2 - (4 - 2e^\epsilon)\, e^\epsilon + \epsilon^2 e^\epsilon}}{-\epsilon + \epsilon e^\epsilon}.$$

The global maximum point x^* is obtained by taking derivative of $1 + g_1(x-1) - g_1(x)$. This completes the proof.

■

Lemma 1 enables us to compute the sensitivity of the quality function for each ϵ value used for adding noises. Figure 3.2 shows the calculated sensitivity for 700 different ϵ values in the range of 0.00001 to 100. We note that each time one invokes PrivPfC, the ϵ value is fixed and one can thus compute the sensitivity to be used in the exponential mechanism. Using this instead of 1.1 slightly improves the utility, while ensuring the satisfaction of differential privacy.

3.2.3 Sensitivity of Grid Quality in the Multiclass Classification Case

For the general multiclass classification case, where there are more than two class labels, deriving an analytical formula similar to Lemma 1 is challenging. Recall that the noisy histogram classifier determines the class label of each cell by ranking all classes according to their noisy counts. The grid quality (Eq. 3.1) models the process by computing the probability that each class is ranked first after adding noises count ranks first for each cell. However, since k independent Laplace random variables are involved in this ranking process, getting the closed form of the density of the joint distribution is very challenging. To get a function for the multiclass case whose global sensitivity is easy to bound, we propose a simple and effective approximation of the grid quality (Eq. 3.1) which for each cell considers only the two classes with the highest number of counts in that cell.

Definition 3.2.4 (Approximation of Grid Quality) *Given a dataset D with class labels $\mathbb{L} = \{1, 2, \dots, k\}$, where $k > 2$, a grid g and ϵ for the parameter of adding Laplacian noises to the counts, the grid quality is*

$$\mathsf{gq}(D, g) = \sum_{c \in g} n_c^{(1)} \cdot p_c^{(1)} + n_c^{(2)} \cdot p_c^{(2)} \tag{3.8}$$

where $n_c^{(1)}$ and $n_c^{(2)}$ are the highest class count and the second highest count in the cell c, $p_c^{(1)}$ is the probability that $n_c^{(1)} + Z_{(1)} >= n_c^{(2)} + Z_{(2)}$ and $p_c^{(2)} = 1 - p_c^{(1)}$.

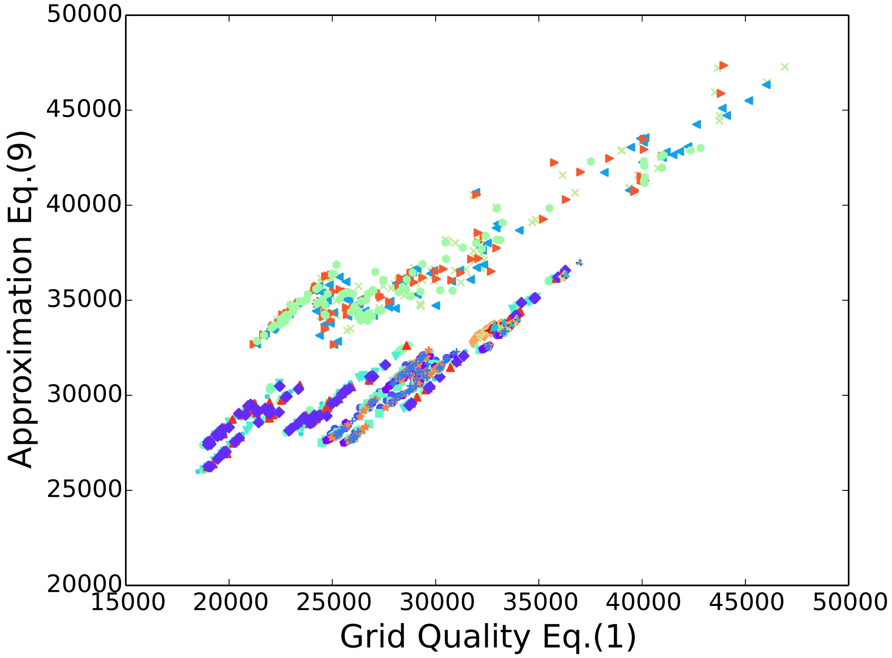


Figure 3.3.: Correlation between grid quality (Eq 3.1) and its approximation (Eq 3.8). Average Pearson correlation coefficient is 0.936 with standard deviation 0.026.

We experimentally study the correlation between the grid quality function (Eq. 3.1) and its approximation (Eq. 3.8) over 4 multiclass real datasets and 5 privacy budgets. In

Figure 3.3, we can see that the simplified multiclass quality function is highly correlated with the original one.

Lemma 3 *For any $\epsilon > 0$, the global sensitivity of the grid quality function for multiclass classification (Eq. 3.8) is bounded by 1.1, that is, $\Delta_{\mathsf{gq}} \leq 1.1$.*

The proof of this lemma is similar to that of Lemma 1 by replacing two class counts n_c^1 and n_c^2 with the two highest class counts $n_c^{(1)}$ and $n_c^{(2)}$.

Algorithm 9 PrivPfC: Differentially Privately Publishing Data for Classification

Input: dataset D, the set of predictor variables A and their taxonomy hierarchies, total privacy budget ϵ, maximum grid pool size Ω.

```
function PrivPfC($D, \mathsf{A}, \epsilon, \Omega$)
    $\epsilon_N \leftarrow 0.03\epsilon$, $\epsilon_{sh} \leftarrow 0.37\epsilon$, $\epsilon_{ph} \leftarrow 0.6\epsilon$
    $\hat{N} \leftarrow |D| + \mathrm{Lap}(1/\epsilon_N)$
    $T \leftarrow 20\% \cdot \hat{N} \cdot \epsilon_{ph}$
    $\mathsf{H} \leftarrow \text{Enumerate}(\mathsf{A}, \Omega, T)$
    $h \leftarrow \mathsf{selectHist}(D, \mathsf{H}, \epsilon_{sh})$
    $\hat{I} \leftarrow \mathsf{perturbHist}(D, h, \epsilon_{ph})$
    return $\hat{I}$
end function
```

Algorithm 10 PrivPfC: Differentially Privately Publishing Data for Classification

```
10: function Enumerate($\mathsf{A}, \Omega, T$)
11:     $L_0 \leftarrow \{\langle 1, 1, \ldots, 1 \rangle\}$
12:     $count \leftarrow 0$
13:     for $k = 0 \rightarrow |\mathsf{A}| - 1$ do
14:         $L_{k+1} \leftarrow \{\}$
15:         for Each grid $g \in L_k$ do
16:             for $j = 1 \rightarrow |\mathsf{A}|$ do
17:                 if $g_j = 1$ then
18:                     for $i = 2 \rightarrow h_j$ do
19:                         $new\_g = g$
20:                         $new\_g_j = i$
21:                         if size$(ng) \leq T$ then
22:                             $L_{k+1} = L_{k+1} \cup ng$
23:                             $count \leftarrow count + 1$
24:                             if $count \geq \Omega$ then go to 34
25:                             end if
26:                         end if
27:                     end for
28:                 end if
29:             end for
30:         end for
31:         if $L_{k+1} == \{\}$ then go to 34
32:         end if
33:     end for
34:     return $\bigcup_{j=1}^{|\mathsf{A}|} L_j$
35: end function
```

3.2.4 Candidate Grids Enumeration

PrivPfC takes as input Ω, the maximum number of candidate grids, and generates a pool of at most Ω candidate grids. We also limit the number of cells in each candidate grid, to prevent the average counts from being dominated by the injected noises. More specifically, we limit the maximum allowed number of cells in any candidate grid g to be $T = 0.2 * \hat{N} * \epsilon$, where ϵ is the privacy budget reserved for adding noises to the histogram,

Algorithm 11 PrivPfC: Differentially Privately Publishing Data for Classification

```
function selectHist($D$, $\mathsf{H}$, $\epsilon_{sh}$)
    for $i = 1 \to |\mathsf{H}|$ do
        $q_i \leftarrow \text{qual}(\mathsf{H}_i)$
        $p_i \leftarrow e^{-(q_i \epsilon_{sh})/2}$
    end for
    $h \leftarrow$ sample $i \in [1..|\mathsf{H}|]$ according to $p_i$
    return $h$
end function

function perturbHist($D$, $h$, $\epsilon_{ph}$)
    Initialize $I$ to empty
    for each cell $c \in h$ do
        $\hat{n}_c^+ \leftarrow n_c^+ + \text{Lap}(1/\epsilon_{ph})$
        $\hat{n}_c^- \leftarrow n_c^- + \text{Lap}(1/\epsilon_{ph})$
        Add $(\hat{n}_c^+, \hat{n}_c^-)$ to $I$
    end for
    Round all counts of $I$ to their nearest non-negative integers.
    return $I$
end function
```

and $\hat{N}$ is a noisy estimate of the total number of tuples. This ensures that the average noise magnitude is no more than the 20% of the average cell count. This non-dominating rule has been used in several differentially private data publishing papers [51], [38].

PrivPfC generates candidate grids by a level-wise search. It starts from the most general grid, $\langle 1, 1, \ldots, 1 \rangle$, where the whole domain is a single cell, and first generates L_1, the list of all grids that have a single attribute going beyond the top level, then generates L_2, the list of all grids that have exactly two attributes going beyond the top level, and so on. It will include a grid as a candidate only when the grid includes no more than T cells. It stops when either it has included all grids with no more than T cells, or it has included Ω grids.

The choice of Ω depends on the amount of computing resources one is willing to spend. When Ω is too large, one runs out of candidate grids that have at most T cells, and increasing Ω further won't increase the size of the pool.

3.2.5 Putting Things Together for PrivPfC

Algorithm 9 shows PrivPfC (Line 1). PrivPfC has three main steps: (1) Enumerate candidate grid (Line 5); (2) Privately select grid (Line 6); (3) Publish noisy counts (Line 7). We divide the privacy budget into three portions: $3\%\epsilon$ is used to privately estimate the dataset size, $37\%\epsilon$ is used for selecting grid (Function selectHist) and $60\%\epsilon$ is used for publishing noisy counts (Function perturbHist). The enumeration step does not access the dataset D and does not consume any privacy budget.

Theorem 3.2.1 PrivPfC *in Algorithm 9 satisfies ϵ-differential privacy.*

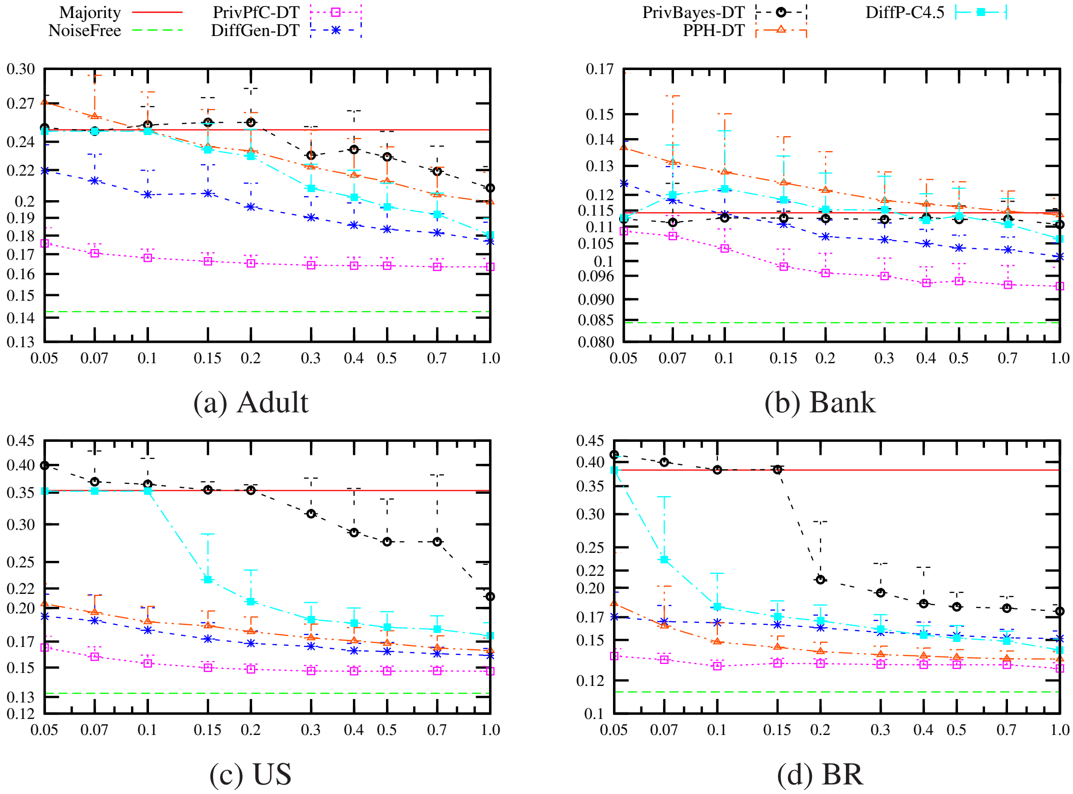
The proof of Theorem 3.2.1 is straightforward based on the analysis above and the sequential composability and parallel composability of differential privacy as discussed in Section 2.5.

Time complexity. The most time consuming step of the algorithm is that of computing the quality of all candidate grids (Line 6), which considers at most Ω candidate grids. Computing the quality of one candidate grid takes time $O(N)$ and therefore selecting the grid takes a total time $O(N \cdot \Omega)$. Once a grid is selected, only a single pass over the dataset is needed to do the perturbation (Line 44). Hence, the total running time for PrivPfC is $O(N \cdot \Omega)$.

3.3 Experiment

3.3.1 Experimental Settings

Datasets. We use 8 real datasets for our experiments, 4 for binary classification and 4 for multiclass classification. They are summarized in Table 5.1. The first one is the Adult dataset from the UCI machine learning repository [52]. It contains 6 numerical attributes and 8 categorical attributes, and is widely used for evaluating the performance of classification algorithms. After removing missing values, the dataset contains 45,222 tuples. We create a multiclass version of the Adult dataset, called Adult-Multiclass, by using the marital-status attribute as the class attribute.



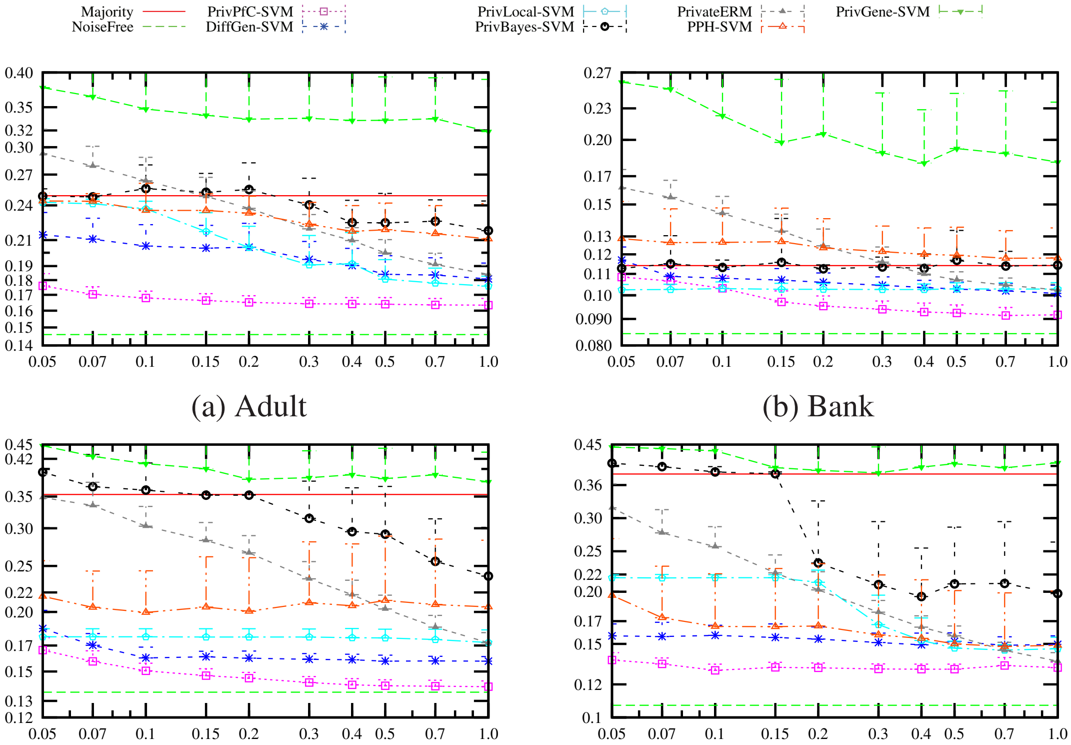
(a) Adult

(b) Bank

(c) US

(d) BR

Figure 3.4.: Comparison of PrivPfC, DiffGen, PrivBayes, PPH and DiffPC-4.5 by decision tree classification. x-axis: privacy budget ϵ in log-scale. y-axis: misclassification rate in log-scale.



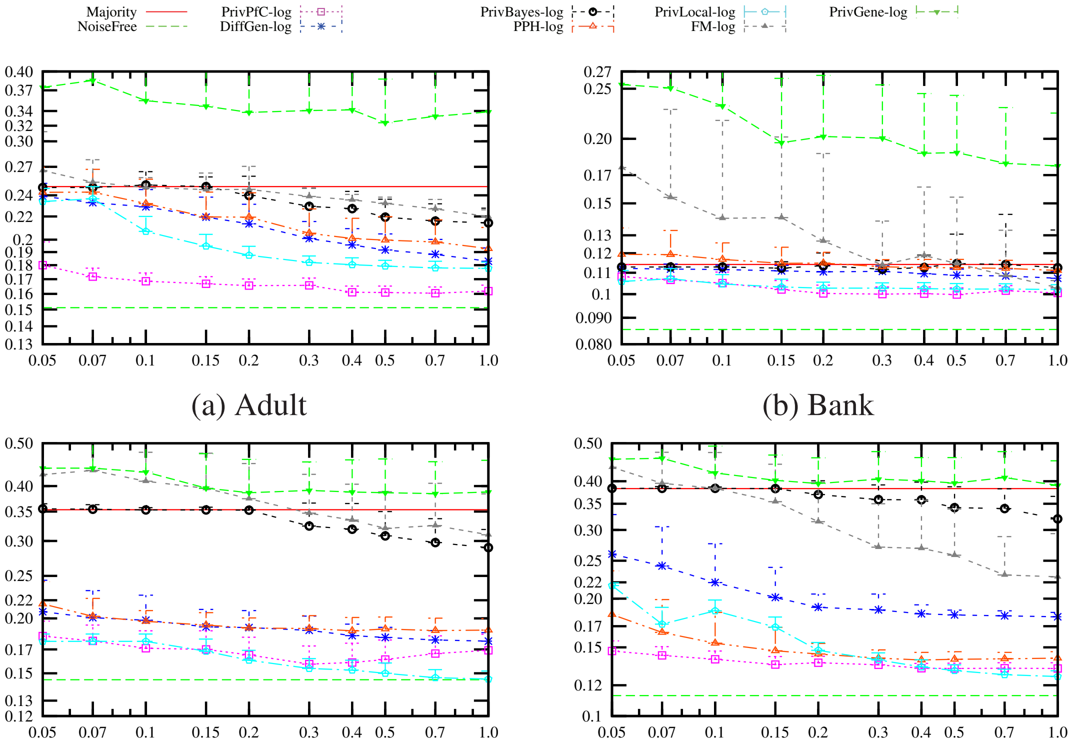
(a) Adult

(b) Bank

(c) US

(d) BR

Figure 3.5.: Comparison of PrivPfC, DiffGen, PrivBayes, PPH, PrivGene and PrivateERM by SVM classification. x-axis: privacy budget ϵ in log-scale. y-axis: misclassification rate in log-scale.



(a) Adult

(b) Bank

(c) US

(d) BR

Figure 3.6.: Comparison of PrivPfC, DiffGen, PrivBayes, PPH and FunctionalMechanism by logistic regression classification. x-axis: privacy budget ϵ in log-scale. y-axis: misclassification rate in log-scale.

The second dataset is the Bank marketing dataset from the same repository. It contains 10 numerical attributes and 10 categorical attributes on 41,188 individuals. The multiclass version of the bank dataset is created by using 3-valued the poutcome attribute as the class attribute. The third is the US dataset from the *Integrated Public Use Microdata Series* (IPUMS) [53]. It has 39,187 United States census records in 2010, with 15 numerical attributes and 31 categorical ones. The multiclass version of the US dataset is created by using the 4-valued SCHLTYPE attribute as the class attribute. We remove one categorical attribute which is highly correlated with the SCHLTYPE attribute in the multiclass version of US dataset. The fourth is the BR dataset (also from IPUMS), which contains 57,333 Brazil census records in 2010 and has 14 numerical attributes and 28 categorical ones. The multiclass version of the BR dataset is created by using the 4-valued EMPSTAT attribute as the class attribute.

Taxonomy Hierarchies. For the Adult and Adult-Multiclass datasets, we use the same taxonomy hierarchies as those in DiffGen [34]. For the remaining 3 datasets, we create taxonomy hierarchies as follows. For numerical attributes, we partition each domain into equal size bins and build hierarchies over them. For categorical attributes, we build taxonomy hierarchies by considering the semantic meanings of the attribute values.

Competing Methods. We compare PrivPfC with 7 state-of-the-art methods in terms of misclassification rate. These include 3 data-publishing methods that publish either a noisy histogram or a noisy Bayesian network, which can be used to generate a synthetic dataset: DiffGen [34], PrivBayes [38], and Private Projected Histogram (PPH) [39]; and 4 methods that directly output a classifier, PrivLocal, PrivGene [12], DiffPC-4.5 [9], and PrivateERM [10]. Table 5.2 summarizes the competing methods mentioned in this chapter.

DiffGen [34]. DiffGen also uses taxonomy and publishes a noisy histogram. However, it chooses the grid in a way different from PrivPfC. In DiffGen, one iteratively selects one attribute at a time for specialization, using the exponential mechanism. The quality function suggested in [34] aims to maximize the number of tuples that have the majority class label in all cells. The number of specialization steps is an important parameter and is an input to the algorithm. As suggested in [34], we set the number of specialization steps to be 10 for the Adult dataset, Adult-Multiclass and the bank dataset. For the US and BR datasets, we set the number to be 6 and 8 respectively, as beyond these numbers, the DiffGen implementation runs into memory problems, because the taxonomy trees for these datasets have larger fan-outs.

PPH [39]. PPH also publishes a noisy histogram. It uses the exponential mechanism to select k attributes, using a quality function that maximizes the discernibility score regarding the label attribute. The grid is determined by the k attributes. For each categorical attribute, the full domain is used. For a numerical attribute, it uses the formula proposed in Lei [33] to decide how many bins to discretize the attribute domain.

PrivBayes [38]. PrivBayes publishes a noisy Bayesian network. It determines the structure of a Bayesian network by first randomly selecting an attribute as the first node, and then iteratively selecting another attribute to create a new node and up to k already created nodes as the new node's parent nodes. After the structure is determined, PrivBayes perturbs the marginals needed for computing the conditional distributions. The performance of the PrivBayes algorithm depends on k. We set $k = 3$ for the Adult dataset and the Bank dataset, which is the same as the one used in [38]. For the US and BR datasets, which were not used in [38], setting $k = 3$ runs out of memory in our experiments because these datasets have more attributes; we set $k = 2$ for them.

Classifier-outputting Methods. *PrivGene* [12] is a general-purpose private model fitting framework based on genetic algorithms, which can be applied to SVM classification and logistic regression. *PrivLocal* is a differentially private local search algorithm. *DiffPC-4.5* [9] outputs a C-4.5 decision tree classifier differential-privately. *PrivateERM* [10] outputs an SVM classifier by injecting noise into the risk function first and then optimizing the perturbed risk function.

The source codes of DiffGen, PrivBayes, PPH, DiffPC-4.5, PrivLocal were shared by authors of corresponding papers. The source code of PrivateERM was shared by the authors of PrivGene. We implement the PrivGene algorithm by strictly following the paper [12].

Evaluation Methodology. The evaluation is based on 3 classification models: the CART decision tree, the SVM with radial basis kernel and the logistic regression model. For all the experiments, we vary ϵ from 0.05 to 1.0. Similar to the experiment settings of [9], [34], [39], under each privacy budget, we execute 10-fold stratified cross-validation to evaluate the misclassification rate of the above methods. For each train-test pair, we run the target method 10 times. Each time we privately compute a classifier using the training data and evaluates its accuracy on the testing data, which is disjoint from the training data. We report the average measurements over the 10 runs and the 10-fold cross-validations. The implementation and experiments of PrivPfC were done in Python 2.7 and all experiments were conducted on an Intel Core i7-3770 3.40GHz PC with 16GB memory.

For methods that output a classifier, i.e., DiffPC-4.5, PrivateERM, PrivGene, PrivLocal and FunctionalMechanism, we use parameters suggested by the corresponding papers. For other data publishing methods, i.e., PrivPfC, PPH, DiffGen, and PrivBayes, we generate private synthetic datasets and then use standard implementations of classification methods on these datasets. To evaluate their performance in terms of the decision tree model, we use the rpart [54] library to build decision trees on synthetic datasets. For evaluation in terms of SVM model, we use the LibSVM package [55]. For evaluation in terms of logistic regression, we use R's glm (generalized linear model) function. When comparing different approaches, We use the same sets of parameters for these classifiers.

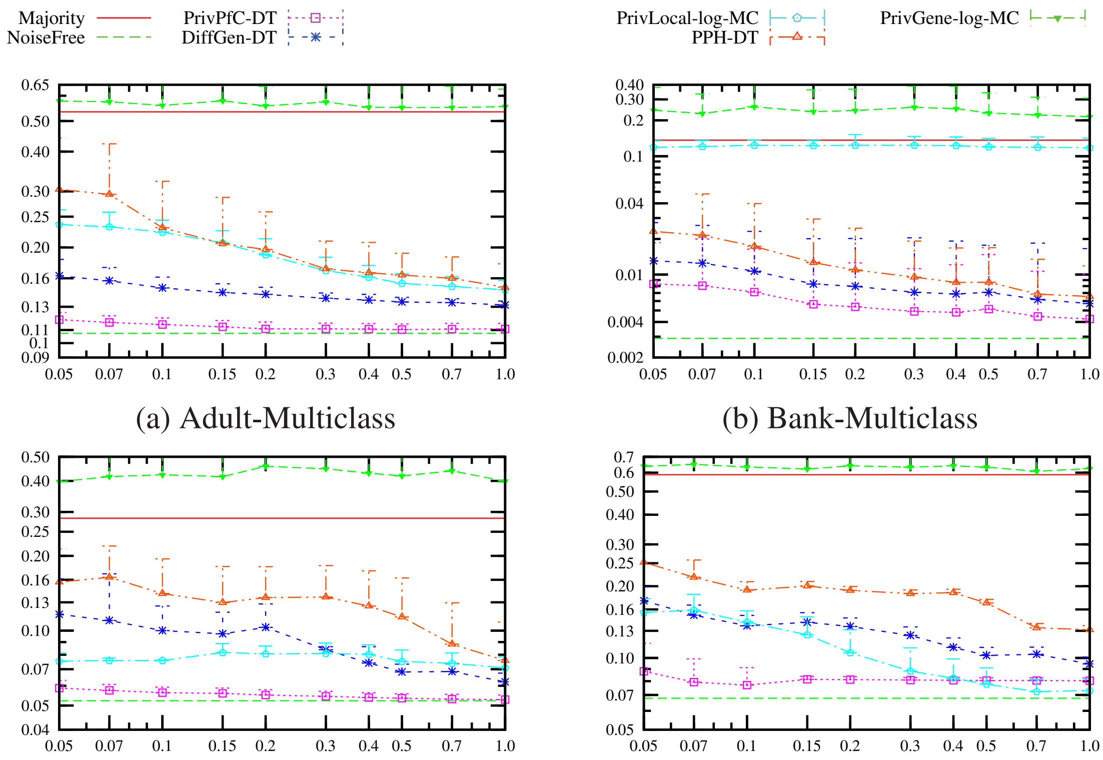
We consider two baselines – *Majority* and *NoiseFree*. *Majority* is the misclassification rate by majority voting on the class attribute, which predicts each test case with the majority class label in the train dataset. *NoiseFree* is the misclassification rate of a decision tree, an SVM classifier or a logistic regression classifier built on the true data. We expect that a good algorithm to perform significantly better than *Majority*, and gets close to *NoiseFree* as ϵ increases.

Table 3.1.: Dataset characteristics

Dataset	# Dim	# Num	# Cate	# Records	# Classes	Classification Task
Adult	15	6	8	45,222	2	Determine whether a person makes over 50K a year.
Adult-Multiclass	15	6	8	45,222	3	Determine the three classes marital status of a person.
Bank	21	10	10	2	41,188	Determine whether the client subscribed a term deposit.
Bank-Multiclass	21	10	10	3	41,188	Determine 3 types of outcome of previous marketing campaign.
US	47	15	31	39,187	2	Determine whether a person makes over 50K a year.
US-Multiclass	46	15	30	39,187	4	Determine the four types of school attended by a person.
BR	43	14	28	57,333	2	Determine whether a person makes over 300 per month.
BR-Multiclass	43	14	28	57,333	4	Determine the four types of employment status of a person.

Table 3.2.: Summary of differentially private classification methods

	Methods	**Description**
Data-publishing	PrivPfC	Our proposed method.
	PrivPfC-SelNF	Our proposed method with noise-free grid selection.
	DiffGen [34]	Private data release for classification via recursive partitioning.
	DiffGen-Struct-NF	DiffGen with noise-free partitioning procedure.
	PrivBayes [38]	Private Data Release via Bayes network.
	PrivBayes-Struct-NF	PrivBayes with noise-free network learning procedure.
	PPH [39]	Private data release for classification by projection and perturbation.
Classifier-outputting	DiffPC-4.5 [9]	Privately construct C4.5 decision tree classifier.
	PrivGene [12]	Private model fitting based on genetic algorithms.
	PrivLocal [12]	Private local search algorithm.
	FunctionalMechanism [11]	Private model fitting by perturbing the fitting function.
	PrivateERM [10]	Private classifier construction based on empirical risk minimization.



(a) Adult-Multiclass

(b) Bank-Multiclass

(c) US-Multiclass

(d) BR-Multiclass

Figure 3.7.: Comparison of PrivPfC, DiffGen, PPH, PrivLocal and PrivGene by decision tree classification and logistic regression classification on the multiclass datasets. y-axis: misclassification rate in log-scale.

3.3.2 Comparison with Existing Solutions

For each classification method, we compare PrivPfC with $\Omega = 10,000$, with three existing data-publishing methods DiffGen, PrivBayes, PPH and any classifier-outputting method that can be applied to this classification method. We note that PrivBayes is not designed to be optimized for a single classification task; thus in some sense is not expected to perform well.

Figure 3.4 reports the average misclassification rates and the corresponding standard deviations for decision tree. Clearly, PrivPfC has the best performance in most cases, followed by DiffGen, PPH, DiffPC-4.5 and PrivBayes. The performance of PrivPfC is also the most robust, as can be seen from the fact that the standard deviations of its misclassification rates are always the lowest.

Figure 3.5 shows similar experimental results for SVM classification. PrivPfC has the best performance, followed by DiffGen, PrivLocal, PPH, PrivateERM, PrivBayes and PrivGene. PrivGene performs the worst, because the crossover operation in each iteration significantly destroys the structure selected SVM parameter by misaligning the parameter values to their corresponding dimensions. On the other hand, PrivLocal only uses perturbation to generate offsprings and the structure of SVM parameters can be largely kept. This result also confirms our remarks on the effectiveness of PrivGene.

Figure 3.6 reports the experimental results on Logistic regression. Overall PrivPfC has the best performance, followed by PrivLocal, DiffGen, PPH, PrivBayes, FunctionalMechanism and PrivGene. PrivGene performs the worst again. Note that, in the US and BR dataset, when the privacy budget is large, PrivLocal outperforms PrivPfC with a slight advantage. This is because PrivLocal has a tighter sensitivity bound when applying to logistic regression. Besides, when the privacy budget is large, PrivPfC selects a subset of features to build histogram, whereas the PrivLocal can use the full set of dimensions to build the classifier.

Comparison on multiclass classification. We compare 5 approaches: PrivPfC, DiffGen, PPH, PrivLocal and PrivGene on multiclass classification on 4 real datasets, Adult-Multiclass, Bank-Multiclass, US-Multiclass and BR-Multiclass. The evaluations of the three non-interactive data publishing methods, PrivPfC, DiffGen and PPH are done by the decision tree classification, since these methods privately generate synthetic datasets and decision tree can naturally supports multiclass classification. The PrivLocal and PrivGene methods only produce one classifier, SVM or Logistic regression at a time. We therefore use the One-vs.-rest approach [56] to reduce the multiclass classification problem into the binary classification problem and use ϵ/k budget to train each classifier, where k is the number of classes. Figure 3.7 shows the experimental results. PrivPfC is again the winner in most cases.

3.3.3 Varying Parameters in PrivPfC

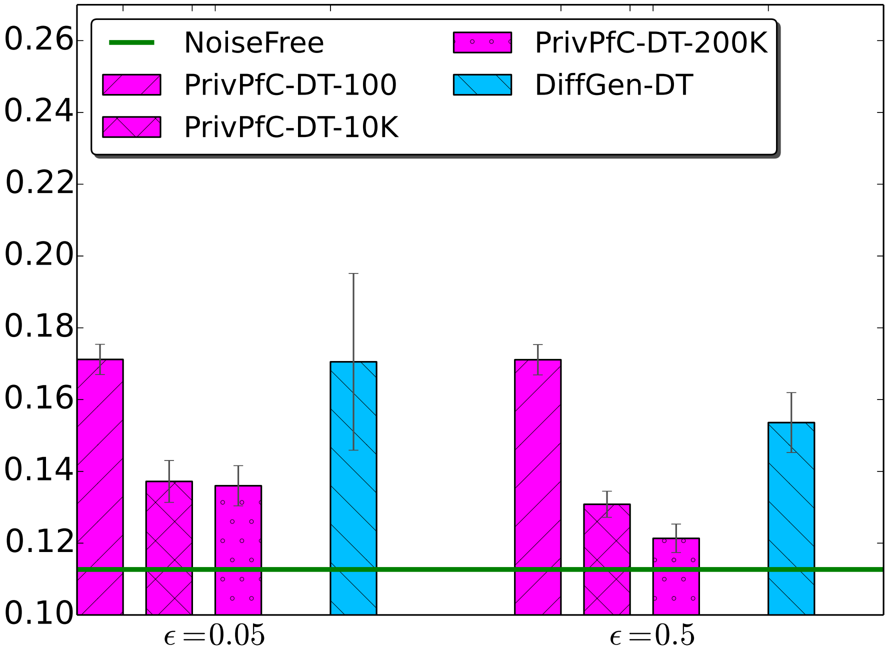


Figure 3.8.: Varying the maximum pool size Ω on PrivPfC by decision tree classification on the BR dataset. y-axis: misclassification rate.

We now explore the effect of changing Ω, the maximum grid pool size, and the effect of using different privacy budget allocation plans in PrivPfC. Figure 3.8 reports the results of PrivPfC's performance under three Ω values, 100, 10,000 and 200,000. The evaluation is done on the BR dataset with two privacy budgets, 0.05 and 0.5. We can see that with the increasing of the maximum pool size, PrivPfC's performance gets significant improvement from $\Omega = 100$ to $\Omega = 10,000$. When setting Ω to the larger value, 200,000, PrivPfC also gets a small amount of improvement.

PrivPfC distributes the privacy budget among three steps, size estimation, grid selection and perturbation, in a 3%-37%-60%. While these ratios are somewhat arbitrary, we have experimentally evaluated other ratios, allocating between 20% and 60% to each of the step other than size estimation. We have found that the differences among different budget allocations are minor, so long as the last step receives at least 30% of the privacy budget.

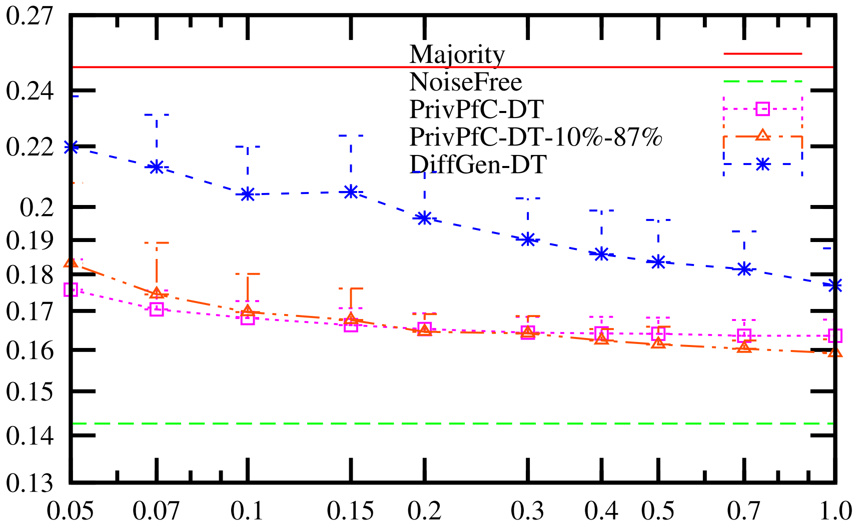
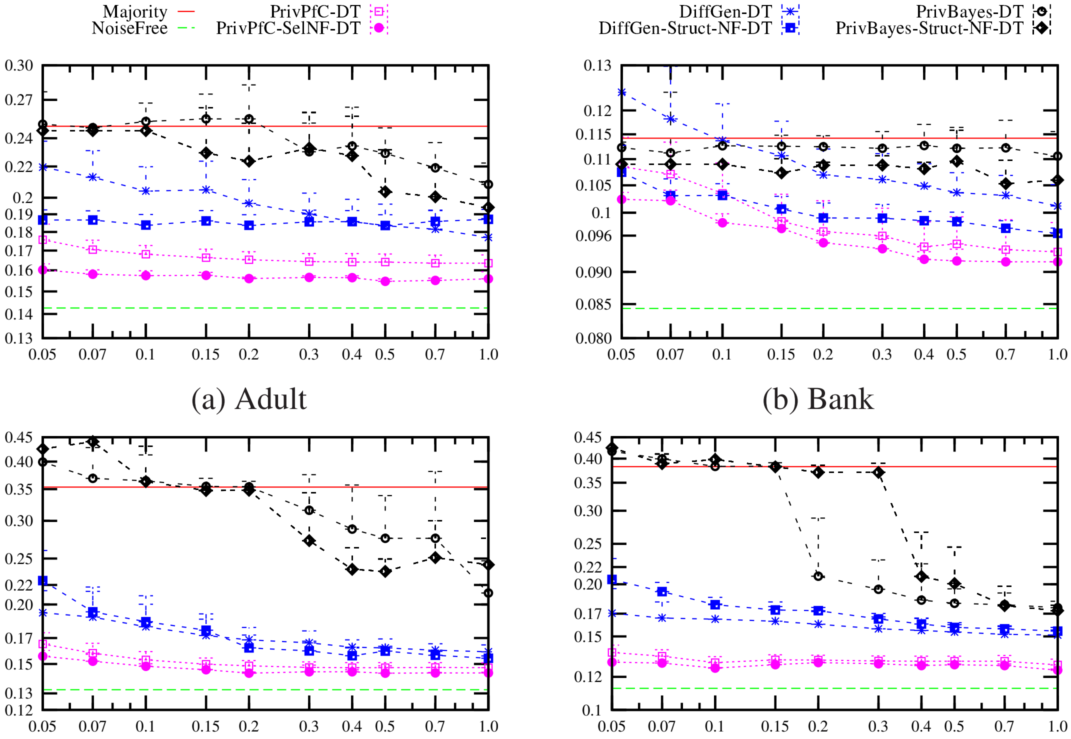


Figure 3.9.: Comparison of two different privacy budget allocations on PrivPfC by decision tree classification on the Adult dataset. y-axis: misclassification rate in log-scale.

Figure 3.9 compares the result of PrivPfC with 3%-10%-87%, with PrivPfC (both when $\Omega = 10,000$, and DiffGen; it shows that PrivPfC 3%-10%-87% performs reasonably well, and in fact slightly better than the standard PrivPfC when $\epsilon \geq 0.2$.

3.3.4 Analyses of Sources of Errors

We have seen that PrivPfC outperforms the other data-publishing methods such as DiffGen and PrivBayes. The key difference in PrivPfC is that we choose the grid g in a single step, instead of arriving at the final grid through a series of decisions. For example, DiffGen iteratively chooses the attributes and ways to partition them, and PrivBayes iteratively builds a Bayesian network. There are two reasons why such an iterative approach does not perform well. The first is that the decisions made in each iteration may be sub-optimal because of the randomization necessary for satisfying differential privacy. The second is that even if the decision made in each iteration is locally optimal, the combination of them is not globally optimal. To see to what extent the latter factor affects accuracy, we consider variants of them respectively, DiffGen-Struct-NF and PrivBayes-Struct-NF. In these



(a) Adult

(b) Bank

(c) US

(d) BR

Figure 3.10.: Analyses of PrivPfC, DiffGen and PrivBayes by decision tree classification. x-axis: privacy budget ϵ in log-scale. y-axis: misclassification rate in log-scale.

variants, the decisions in each iteration are performed without any perturbation, but noises are still added when publishing the counts. We also consider a variant of PrivPfC, called PrivPfC-SelNF, in which the histogram selection step is noise-free. All these variants are not private; they are used to understand the source of errors only.

Figure 3.10 reports the experimental results of comparing these methods, using Decision Tree. We first observe that PrivPfC-SelNF indeed outperforms PrivPfC, although the differences tend to be smaller than the difference between PrivPfC and DiffGen. We also observe that PrivBayes-Struct-NF still performs poorly; in fact, it performs significantly worse than PrivPfC. Again, this is not surprising given that the iterative Bayes network construction approach is not designed to optimize one classification task. Similarly DiffGen-Struct-NF still underperforms PrivPfC. This suggests that the inherent iterative structure of DiffGen is suboptimal.

3.3.5 Scalability over Dimensions and Runtime

We study the scalability of dimensions of our algorithm as well as our competitors. This experiment is performed on the US dataset. First, we sort all of its predictor variables by their degrees of correlation to the response variable in descending order. The correlation is measured by the χ^2 statistic, which is one of the most effective methods of feature selection for classification [57], [58]. We then generate the set of datasets with lower number of dimensions by projecting the US dataset to dimensions defined by the first $d - 1$ predictor variables and the response variable, where $d = 10, 15, 20, 25, 30, 35, 40, 47$.

Figure 3.11 shows the results. We can see that as the increasing of dimensionality, PrivPfC, DiffGen and PPH offer stable classification performance. PrivPfC is still the best among them. PrivBayes is the poorest in all cases and its performance goes worse as the dimensionality increases.

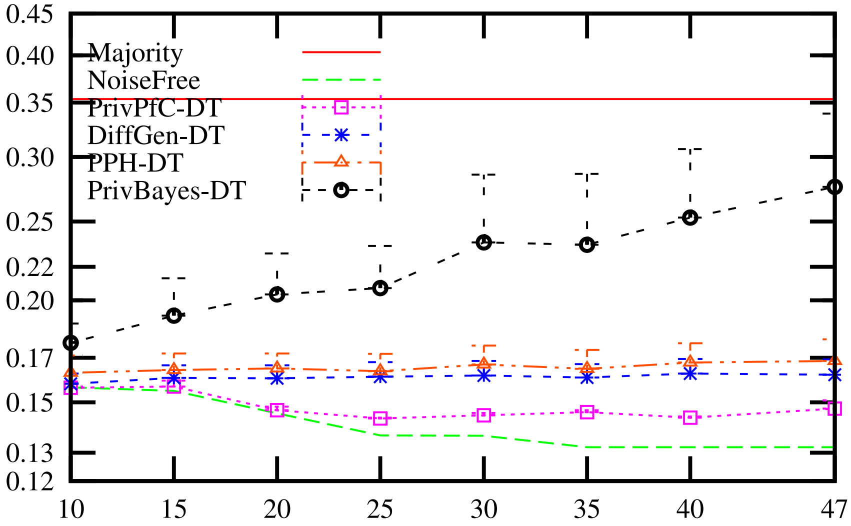


Figure 3.11.: Comparison of PrivPfC, DiffGen, PrivBayes and PPH by varying dimensions (decision tree classification). $\epsilon = 0.5$. x-axis: dimensions. y-axis: misclassification rate in log-scale.

We also compare the running time of 4 data publishing algorithms, PrivPfC, DiffGen, PrivBayes and PPH, on the US dataset with privacy budget 0.05 and 0.5 respectively. Figure 3.12 shows the comparison results. PrivBayes is the most inefficient one, followed by

PrivPfC, PPH and DiffGen. By considering runtime comparison results and accuracy comparison results (Figure 3.4, 3.5, 3.6 and 3.7) together, we can see that PrivPfC trades more runtime for accuracy improvement. From the runtime comparison result, we can also see that under different privacy budgets, PrivBayes, PrivPfC and PPH have close runtime while DiffGen needs longer time when the privacy budget gets larger. This is because with more privacy budget DiffGen is likely to choose finer partitions in attribute taxonomy hierarchy, which results in more time to project data into the partition structure.

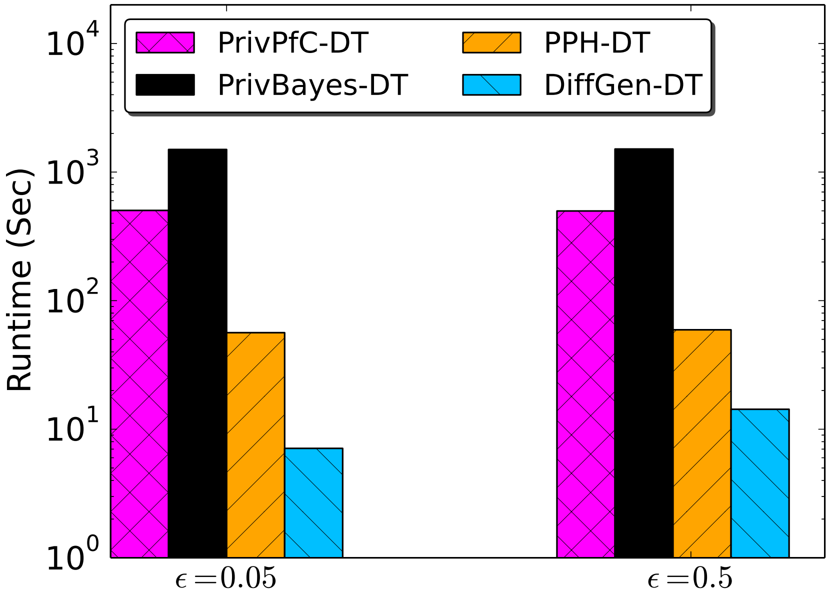


Figure 3.12.: Runtime comparison of PrivPfC, DiffGen, PPH and PrivBayes on decision tree classification. x-axis: privacy budget. y-axis: runtime in seconds.

4. DIFFERENTIALLY PRIVATE K-MEANS CLUSTERING

4.1 Introduction

In recent years, differential privacy [5] has been increasingly adopted as the privacy notion of choice of data analysis while preserving individual privacy. Several broad classes of approaches exist for developing differentially private techniques for data analysis. In this chapter we study differentially private k-means clustering. Clustering analysis plays an essential role in data analysis tasks. Clustering under differential privacy has also been studied in [7, 12, 16, 25, 29, 30].

Our study has two goals. The first is to improve the techniques for performing k-means clustering. The second is to use k-means clustering as a case study to compare several classes of methods for private data analysis, and to identify the strengths and weaknesses of these methods.

There are three state-of-the-art differentially private algorithms on k-means clustering. The first method, which we call DPLloyd, makes the iterative Lloyd algorithm [7, 16] differentially private by adding noises to each step. The second method, which we call PGkM, uses PrivGene [12], a framework for differentially private model fitting based on genetic algorithms. These two use an iterative optimization approach that tries to gradually improve the choice of the centroids. The third algorithm uses the sample and aggregation framework [25] and is implemented in the GUPT system [29], which we call GkM. This algorithm uses a noisy average of the centroids computed from many subsamples of the dataset.

An alternative approach is to publish a synopsis of the dataset in a way that satisfies differential privacy. Then one can perform any k-means clustering algorithm on a synthetic dataset generated from the synopsis. One immediate benefit is that one can run a k-means clustering algorithm such as the Lloyd on the synthetic dataset many times and choose the best, since we are only accessing the synopsis produced while satisfying differential privacy. This also enables one to select an appropriate k value. An additional benefit is that one can perform other analysis tasks beyond k-means clustering on the synthetic dataset at the same time without affecting privacy.

In this chapter, we propose to combine the following differentially private synopsis algorithms with k-means clustering. The dataset is viewed as a set of points over a d-dimensional domain, which is divided into M equal-size cells, and a noisy count is obtained from each cell. A key decision is to choose the parameter M. A larger M value means lower average counts for each cell, and therefore noisy counts are more likely to be dominated by noises. A smaller M value means larger cells, and therefore one has less accurate information of where the points are. We propose a method that sets $M = \left(\frac{N\epsilon}{10}\right)^{\frac{2d}{2+d}}$, which is derived based on extending the analysis in [31], which aims to minimize errors when answering rectangular range queries for 2-dimensional data, to higher dimensional case. We call the resulting k-means algorithm EUGkM, where EUG is for Extended Uniform Grid.

We conducted extensive experimental evaluations for these algorithms on 6 datasets used in the literature as well as 81 datasets that we synthesized by varying the dimension d from 2 to 10 and the number of clusters from 2 to 10. Our experimental results contradict findings in the literature. GkM was introduced after DPLloyd and was claimed to have accuracy advantage over DPLloyd, and PGkM was introduced after and compared GkM. However, we found that DPLloyd is the best method among these three methods, and GkM is by far the worst. In the comparison of DPLloyd and GkM in [29], DPLloyd was run using much larger number of iterations than necessary, and thus perform poorly. We are also able to explain why DPLloyd is better. Our error analysis shows that errors due to GUPT's sample and aggregation approach are asymptotically worse than errors for DPLloyd, as the number of data points increases. The reason why DPLloyd outperforms PGkM is that the genetic programming style of PGkM needs more iterations to converge. When making these algorithms differentially private, the privacy budget is divided among all iterations, thus having more iterations means more noise is added to each iteration. Therefore, the more direct DPLloyd outperforms PGkM.

The most intriguing results are those comparing DPLloyd with EUGkM. For most datasets, EUGkM performs much better than DPLloyd. However, for a few, DPLloyd outperforms EUGkM. Through further analytical and empirical analysis, we found that, while the performance of both algorithms are greatly affected by the two key parameters: the number of dimensions d and the number of clusters k, the ways they are affected by these two parameters are different, due to the different structures of these two algorithms. DPLloyd scales worse when k increases, while EUGkM scales worse when d increases.

An intriguing question is can we further improve upon DPLloyd and EUGkM? We note that the accuracy of DPLloyd is affected by two key factors: the number of iterations and the choice of initial centroids. In fact, these two are closely related. If the initially chosen centroids are very good and close to the true centroids, one only needs perhaps one iteration to improve it, and this reduction in the number of iterations would mean less noise is added.

This leads us to propose a novel hybrid method that combines the non-interactive EUGkM with the single-workload DPLloyd. We first use a portion of the total privacy budget to run EUGkM, and then use the centroids outputted by EUGkM as the initial centroids for one round of DPLloyd with the remaining privacy budget. Such a method, however, may not actually outperform EUGkM, especially when the privacy budget ϵ is small, since then one round of DPLloyd may actually worsen the centroids. We use our error analysis formulas to determine whether there is sufficient privacy budget for such a hybrid approach to outperform EUGkM. We then experimentally validate the effectiveness of the Hybrid approach.

The hybrid approach is also applicable to other private data analysis tasks which have an iterative/incremental algorithm structure. For example, a wide range of machine learning algorithms, such as support vector machines and logistic regression, employ stochastic gradient descent to train models. When making such algorithms differentially private, there is always the tradeoff between the quality of the initial values and the number of iterations. Applying the hybrid approach is potentially beneficial. One can first publish a private synopsis of the input data, on which one can find an optimizer, and uses it as the starting choice for iterative improvement.

In this chapter we advance the state-of-the-art on differentially private data mining in several ways. First, we have introduced a new non-interactive method, EUGkM, for differentially private k-means clustering, which are highly effective and often outperform the state-of-the-art single-workload methods and non-interactive methods. Second, we have developed techniques to analyze the error behaviors of DPLloyd and EUGkM. This kind of empirical error analyses are missed in most differentially private data analysis papers. Third, based on the error analysis of DPLloyd, we proposed an improved version of DPLloyd which significantly reduces the clustering error. Fourth, we introduce the novel concept of hybrid approach to differentially private data analysis, which is so far the best approach to k-means clustering. We conjecture that such a hybrid approach may prove useful in other analysis tasks as well. Finally, we have extensively evaluated existing methods for k-means clustering, and analyzed their strengths and weaknesses.

The rest of the chapter is organized as follows. In Section 4.2, we describe the DPLloyd approach, systematically analyze its error behavior and propose an improvement of DPLloyd. In Section 4.3, we describe and analyze other existing single-workload approaches, GkM and PGkM. In Section 4.4, we describe two non-interactive approaches UGkM and MkM and derive a new non-interactive approach EUGkM. In Section 4.5 we study the error behavior of DPLloyd and EUGkM, introduce the hybrid approach. In Section 4.6, we show the experimental results on the performance comparison among the single-workload, non-interactive and the proposed hybrid approaches, and analyze their strengths and weaknesses.

4.2 Differentially Private Lloyd Algorithm and Its Improvements

In this section, we describe the DPLloyd approach for differentially private k-means clustering, analyze its error behavior and propose an improvement of DPLloyd based on the analysis.

4.2.1 DPLloyd

A differentially private version of the Lloyd's algorithm was first proposed by Blum et al. [16]. We call this the DPLloyd approach. As shown in Section 2.6.3, DPLloyd differs from the standard Lloyd algorithm in the following ways. First, Laplace noises are added to the iterative update steps in the Lloyd algorithm. Second, the number of iterations needs to be fixed in order to decide how much noise needs to be added in each iteration.

Optimization Issues

The overall structure of DPLloyd is to first select initial values, and then iteratively improve them. This same algorithmic structure also applies to many other data analysis tasks, such as linear regression, SVM, etc. When making such a single-workload algorithm differentially private, there are two important decisions one has to make.

The first decision is how to select the initial values. In the standard, non-private setting, a purely random choice may suffice, since one could repeat the algorithm multiple times and choose the best result among them. With privacy constraints, however, running the single-workload algorithm multiple times results in each run can use only a fraction of the total privacy budget, and make the results being even less accurate.

The second decision is how many iterations one runs. A large number of iterations causes too much noises being added. A small number of iterations may be insufficient for the algorithm to converge. Existing approaches fix a number. However, intuitively the number of rounds would depend on the available privacy budget ϵ. With a smaller privacy budget, one should run fewer number of rounds, to avoid the results being overwhelmed by too much noise.

How to choose these parameters has not been carefully considered in the literature. In the implementation of DPLloyd in PINQ [27], it is proposed to run 5 iterations, with equal privacy budget allocation for each round. In [29], comparison of GkM with DPLloyd was done by running DPLloyd with 20, 80, and 200 iterations, resulting in incorrect claim that GkM outperforms DPLloyd. Dwork [30] considered the possibility of running k-

means clustering without knowing the number of rounds in advance, and proposed to use exponentially decreasing allocation of privacy budgets, i.e., $\frac{\epsilon}{2}$ in the first round, $\frac{\epsilon}{4}$ in the second round, and so on. This mostly likely results in deteriorating performance when the number of rounds increases. The main reason is that in later rounds, when one gets closer to the optimal value, it is desirable to have a larger privacy budget.

Below, we propose an approach to improve the selection of initial centroids for k-means clustering, design a general framework for deciding the number of iterations and apply it to improve DPLloyd. The improved version of DPLloyd is called DPLloyd-Impr.

Selecting Initial Centroids

The quality of initial centroids greatly affects the accuracy of DPLloyd. A poor choice of initial centroids can result in converging to a local optimum that is far from global optimum, or not converging after the given number of iterations. While many methods for choosing the initial points have been developed [28], these methods were developed without the privacy concern and need access to the dataset. In [7], k points at uniform random from the domain are chosen as the initial centroids. We have observed empirically that this can perform poorly in some settings, since some randomly chosen initial centroids are close together. We thus introduce an improved method for choosing initial centroids that is similar to the concept of sphere packing. Given a radius a, we randomly generate k centroids one by one such that each new centroid is of distance at least a away from every corner of the domain $[-r, r]^d$ and each new centroid is of distance at least $2a$ away from any existing centroid. When a randomly chosen point does not satisfy this condition, we generate another point. When we have failed repeatedly, e.g. failed over 80% of the user defined maximum number of tries, we conclude that the radius a is too large, and try a smaller radius. We use a binary search to find the maximal value for a such that the process of choosing k centroids succeed. This process depends only on the shape of the overall domain and not where the data points are, and thus does not affect privacy. The pseudocode of this algorithm is shown in Algorithm 12.

Algorithm 12 SpherePackingInitialCentroidsGeneration

```
Input: $d$: number of dimensions, $[-r, r]$: dataset range, $k$: number of clusters
$radius_{lo} \leftarrow 0.0$
$radius_{hi} \leftarrow r\sqrt{d}$
return BinarySearch$(k, r, radius_{lo}, radius_{hi})$
Function BinarySearch$(k, r, radius_{lo}, radius_{hi})$
while $radius_{lo} < radius_{hi}$ do
    $radius_{mid} \leftarrow (radius_{lo} + radius_{hi})/2$
    $numTrials, \langle o_1, \ldots, o_k \rangle \leftarrow$ randomly choose $k$ initial centroids with radius $radius_{mid}$ in $[-r, r]^d$
    if $numTrials < 3$ then
        $radius_{lo} \leftarrow radius_{mid}$
    else if $numTrials > 0.8 \cdot \mathrm{MAXNUMTRIES}$ then
        $radius_{hi} \leftarrow radius_{mid}$
    else
        break
    end if
end while
return $\langle o_1, \ldots, o_k \rangle$
```

Optimizing Rounds and Budget Allocation

We introduce the following general approach of determining the number of rounds and privacy budget allocation. Our approach depends on the ability to analyze the amount of noise introduced in each round, manifested as the mean squared error (MSE). Given this, one also specifies a threshold for the maximum MSE. The basic idea is to choose the number of iterations so that we try to ensure that each iteration's MSE is no larger than the threshold, and use a smaller number of rounds if necessary. Below we show how to apply this idea to DPLloyd.

Error Study of DPLloyd

DPLloyd adds noises to each iteration of updating centroids. We now analyze the mean squared error (MSE) between noisy centroids and true centroids in one iteration.

Consider one centroid and its update in one iteration. The true centroid's i'th dimension should be $o_i = \frac{S_i}{C}$, where C is the number of data points in the cluster and S_i is the sum of i'th dimension coordinates of data points in the cluster. Consider the noisy centroid $\widehat{o}$; its i'th dimension is $\widehat{o_i} = \frac{S_i+\Delta S_i}{C+\Delta C}$, where ΔC is the noise added to the count and ΔS_i is the noise added to the S_i. The MSE is thus:

$$\mathsf{MSE}\left(\widehat{o}\right) = \mathsf{E}\left[\sum_{i=1}^{d}\left(\frac{S_i+\Delta S_i}{C+\Delta C} - \frac{S_i}{C}\right)^2\right]. \tag{4.1}$$

Derivation based on the above formula gives the following proposition.

Proposition 4.2.1 *In one round of DPLloyd, the MSE is*

$$\Theta\left(\frac{(kt)^2 d^3}{(N\epsilon)^2}\right).$$

Proof Let us first consider the MSE on the i-th dimension.

$$\begin{aligned}
\mathsf{MSE}\left(\widehat{o_i}\right) &= \mathsf{E}\left[\left(\frac{S_i+\Delta S_i}{C+\Delta C} - \frac{S_i}{C}\right)^2\right] \\
&\approx \mathsf{E}\left[\left(\frac{C\Delta S_i - S_i\Delta C}{C^2}\right)^2\right] \\
&= \frac{\mathsf{E}[(\Delta S_i)^2]}{C^2} + \frac{\mathsf{E}[S_i^2(\Delta C)^2]}{C^4} + \frac{2CS_i\mathsf{E}[\Delta S_i \Delta C]}{C^4} \\
&= \frac{\mathsf{Var}\left(\Delta S_i\right)}{C^2} + \frac{S_i^2\mathsf{Var}\left(\Delta C\right)}{C^4}.
\end{aligned}$$

The last step holds, because ΔS_i and ΔC are independent zero-mean Laplace noises and the following formulas hold:

$$\begin{cases}
\mathsf{E}[\Delta S_i \Delta C] = 0 \\
\mathsf{E}[(\Delta S_i)^2] = \mathsf{E}[(\Delta S_i)^2] - (\mathsf{E}[\Delta S_i])^2 = \mathsf{Var}\left(\Delta S_i\right) \\
\mathsf{E}[(\Delta C)^2] = \mathsf{E}[(\Delta C)^2] - (\mathsf{E}[\Delta C])^2 = \mathsf{Var}\left(\Delta C\right),
\end{cases}$$

where $\mathsf{Var}\left(\Delta S_i\right)$ and $\mathsf{Var}\left(\Delta C\right)$ are the variances of ΔS_i and ΔC, respectively.

Suppose that on average $\frac{|S_i|}{2r \cdot C} = \rho$, where $[-r, r]$ is the range of the i'th dimension. That is, ρ is the normalized coordinate of i-th dimension of the cluster's centroid. Furthermore, suppose that each cluster is about the same size, i.e., $C \approx \frac{N}{k}$. Then, $\mathsf{MSE}\left(\widehat{o_i}\right)$ can be approximated as follows:

$$\mathsf{MSE}\left(\widehat{o_i}\right) \approx \frac{k^2}{N^2}\left(\mathsf{Var}\left(\Delta S_i\right) + (2\rho r)^2 \cdot \mathsf{Var}\left(\Delta C\right)\right). \tag{4.2}$$

DPLloyd adds to each sum/count function Laplace noise $\mathsf{Lap}\left(\frac{(dr+1)t}{\epsilon}\right)$. Therefore, both $\mathsf{Var}\left(\Delta S_i\right)$ and $\mathsf{Var}\left(\Delta C\right)$ are equal to $\frac{2((dr+1)t)^2}{\epsilon^2}$. From Equation (4.2) we obtain

$$\begin{aligned} \mathsf{MSE}\left(\widehat{o_i}\right) &\approx \frac{k^2}{N^2}\left(\mathsf{Var}\left(\Delta S_i\right) + (2\rho r)^2 \cdot \mathsf{Var}\left(\Delta C\right)\right) && (4.3) \\ &= 2(1 + (2\rho r)^2)\left(\frac{kt(dr+1)}{N\epsilon}\right)^2. && (4.4) \end{aligned}$$

As the noise added to each dimension is independent, from Equation 4.1 we know that the MSE is

$$\mathsf{MSE}\left(\widehat{o}\right) = \sum_{i=1}^{d} \mathsf{MSE}\left(\widehat{o_i}\right) \approx 2d(1 + (2\rho r)^2)\left(\frac{kt(dr+1)}{N\epsilon}\right)^2. \tag{4.5}$$

When r is a small constant, this becomes $\Theta\left(\frac{(kt)^2 d^3}{(N\epsilon)^2}\right)$. ■

Proposition 4.2.1 shows that the distortion to the centroid proportional to $t^2k^2d^3$, while inversely proportional to $(N\epsilon)^2$.

Optimizing Privacy Budget Allocation Within Each Round

An issue specific to DPLloyd and may not be shared by all iterative algorithms is that within each round of DPLloyd, the privacy budget needs to be divided among the count and the d sum queries. Suppose ϵ_0 is allocated to the count query, and ϵ_i is allocated to the sum query for the i-th dimension, for each $i = 1, 2, \ldots, d$. While all dimensions should be treated equally, i.e., $\epsilon_1 = \epsilon_2 = \ldots = \epsilon_d$, an interesting question is what should be the right value of $\frac{\epsilon_i}{\epsilon_0}$? The DPLloyd approach allocates the privacy budget according to the sensitivities of different queries; thus $\frac{\epsilon_i}{\epsilon_0} = r$, assuming that each dimension is normalized to $[-r, r]$. Different r values will result in different allocations of privacy budget.

We observe that the analysis in Section 4.2.1 calls for a fixed allocation of $\frac{\epsilon_i}{\epsilon_0}$, independent from how the data ranges are normalized. Plugging $\mathsf{Var}\left(\Delta S_i\right) = \frac{2r^2}{\epsilon_i^2}$ and $\mathsf{Var}\left(\Delta C\right) = \frac{2}{\epsilon_0^2}$ into Equation (4.3), one obtains

$$\begin{aligned}\sum_{i=1}^{d} \mathsf{MSE}\left(\widehat{o_i}\right) &\approx \frac{k^2}{N^2} \sum_{i=1}^{d} \big(\mathsf{Var}\left(\Delta S_i\right) + (2\rho r)^2 \cdot \mathsf{Var}\left(\Delta C\right)\big) \\ &= \frac{2r^2k^2}{N^2} \left(\sum_{i=1}^{d} \frac{1}{\epsilon_i^2} + \frac{4d\rho^2}{\epsilon_0^2} \right). \end{aligned} \tag{4.6}$$

Minimization of the above subject to $\sum_{i=1}^{d} \epsilon_i + \epsilon_0 = z$ can be solved using the method of *Lagrange multipliers*, where z is the privacy budget allocated to the current round. The optimal proportion is

$$\epsilon_1 : \epsilon_2 : \cdots : \epsilon_d : \epsilon_0 = 1 : 1 : \cdots : 1 : \sqrt[3]{4d\rho^2}. \tag{4.7}$$

To compute $\sqrt[3]{4d\rho^2}$, we need an estimation of ρ, the normalized coordinate of i-th dimension of the cluster's centroid. We note that $0 \leq \rho \leq 0.5$. If a cluster includes points perfectly balanced between the negative side and the positive side, then $\rho = 0$. If all points have r $(-r)$ as its i-th coordinate, then $\rho = 0.5$. We empirically compute ρ from 81 synthetic datasets that are not used for purpose of evaluation. We use $\rho = 0.225$ in this chapter, and conjecture that it provides a good enough approximation for most scenarios.

We note that in the DPLloyd approach, if one chooses $r = 1$, i.e., normalizes each dimension to the range of $[-1, 1]$, one would allocate the privacy budget with a ratio of $\epsilon_i : \epsilon_0 = 1 : 1$, which is suboptimal in most cases.

Algorithm 13 shows the privacy budget allocation improvement of DPLloyd within each round.

```
Algorithm 13 DPLloydOptimizationForOneIteration
Input: $D$: dataset, $d$: number of dimensions, $[-r, r]$: dataset range, $k$: number of clusters, $IC$: set of initial centroids, $\epsilon$: privacy budget
$\{o^1, o^2, \ldots, o^k\} \leftarrow IC$
Budget allocation ratio $\gamma_s$ for sum query, $\gamma_c$ for count queries by Eq. 4.7
for each $j$ ($j = 1, 2, \ldots, k$) do
    Cluster $C^j \leftarrow \{x^\ell : \|x^\ell - o^i\| \leq \|x^\ell - o^j\|, x^\ell \in D, \forall 1 \leq i \leq k\}$
    $\langle o_1^j, o_2^j, \ldots, o_d^j \rangle \leftarrow$ NoisyCentroidUpdate$(d, C^j, \gamma_s \cdot \epsilon, \gamma_c \cdot \epsilon)$
end for
return $\{o^1, o^2, \ldots, o^k\}$
```

Determining the Number of Rounds

Based on our analysis in Section 4.2.1, we make several observations. First, it makes no sense to run DPLloyd with a large number of rounds. From Proposition 4.2.1, the distortion on the centroid is on the order of $\Theta\left(\frac{t^2}{(N\epsilon)^2}\right)$. Thus, running DPLloyd with too many rounds results in large distortion on the cluster centroids. Second, one should dynamically determine the number of rounds based on parameters such as N and ϵ, since the distortion on the centroid is inversely proportional to $(N\epsilon)^2$.

By exploiting these observations, we propose a way to determine the number of iterations. We first determine a minimum privacy budget ϵ^m that needs to be allocated to each iteration (see below). Then, the privacy budget allocation across the iterations is decided by the following two cases. *Case 1:* $\epsilon \leq 2\epsilon^m$. In this case, the total privacy budget is inadequate. If we distribute it to more than 2 iterations, then as stated before the added noise in each round would easily dominate the centroid improvement. Therefore, we decide that DPLloyd runs for two iterations only, each with privacy budget of $\frac{\epsilon}{2}$. *Case 2:* $\epsilon > 2\epsilon^m$. In this case, the total privacy budget is able to meet the requirement of assigning minimum budget to each iteration. We require that the total number of iterations is at most 7. Thus, the total number of iterations $t^- = \min\{\frac{\epsilon}{\epsilon^m}, 7\}$, and the privacy budget allocated to each of them is $\frac{\epsilon}{t^-}$.

We now come to the calculation of ϵ^m. The intuition is that if the centroid improvement of one iteration is effective, then the MSE value should not be too big. We use the heuristic that the MSE of all the centroids improvement should be no more than $0.004 \cdot r^d$. It follows from Equation 4.6 that

$$\frac{2r^2k^3}{N^2}\left(\sum_{i=1}^{d}\frac{1}{\epsilon_i^2}+\frac{4d\rho^2}{\epsilon_0^2}\right)\leq 0.004r^d, \tag{4.8}$$

where $\sum_{i=0}^{d}\epsilon_i=\epsilon^m$. According to the optimized ratio in Equation 4.7, the privacy budget ϵ^m is distributed between ϵ_i's as follows:

$$\begin{cases}\epsilon_0=\frac{\sqrt[3]{4d\rho^2}}{d+\sqrt[3]{4d\rho^2}}\epsilon^m\\ \epsilon_i=\frac{1}{d+\sqrt[3]{4d\rho^2}}\epsilon^m, \text{for } i=1,2,\ldots,d.\end{cases}$$

Plugging the above into Inequality 4.8 we can find the minimal ϵ^m value,

$$\epsilon^m=\left(\frac{500k^3}{N^2}\left(d+\sqrt[3]{4d\rho^2}\right)^3\right)^{1/2}. \tag{4.9}$$

For the Gowalla dataset, $\epsilon^m \approx 0.011$; for the Adult-num dataset, it is approximately equal to 0.096.

Algorithm 14 summarizes our improvement of DPLloyd.

4.3 Other Approaches

In this section, we describe other existing approaches to differentially private k-means clustering. Further analyses of them are presented in Section 4.6.

4.3.1 PGkM

PrivGene [12] is a general-purpose differentially private model fitting framework based on genetic algorithms. Given a dataset D and a fitting-score function $f(D, \theta)$ that measures how well the parameter θ fits the dataset D, the PrivGene algorithm initializes a candidate set of possible parameters θ and iteratively refines them by mimicking the process of natu-

Algorithm 14 DPLloydImprovement

Input: D: dataset, d: number of dimensions, $[-r, r]$: dataset range, k: number of clusters, t: number of iterations, IC: set of initial centroids, ϵ: privacy budget

```
if $IC$ is empty then
    $\{o^1, o^2, \ldots, o^k\} \leftarrow$ SpherePackingInitialCentroidsGeneration$(d, r, k)$
else
    $\{o^1, o^2, \ldots, o^k\} \leftarrow IC$
end if
Compute the minimum budget $\epsilon^m$ by Eq. 4.9
if $\epsilon < 2\epsilon^m$ then
    $t^- \leftarrow 2$
else
    $t^- \leftarrow \min\{\frac{\epsilon}{\epsilon^m}, 7\}$
end if
$\epsilon' \leftarrow \frac{\epsilon}{t^-}$
for Loop $t^-$ times do
    for each $j$ ($j = 1, 2, \ldots, k$) do
        Cluster $C^j \leftarrow \{x^\ell : \|x^\ell - o^i\| \leq \|x^\ell - o^j\|, x^\ell \in D, \forall 1 \leq i \leq k\}$
        $\langle o^j_1, o^j_2, \ldots, o^j_d \rangle \leftarrow$ DPLloydOptimizationForOneIteration$(D, d, [-r, r],$
    $k, \{o^1, o^2, \ldots, o^k\}, \epsilon')$
    end for
end for
return $\{o^1, o^2, \ldots, o^k\}$
```

ral evolution. Specifically, in each iteration, PrivGene uses the exponential mechanism [14] to privately select from the candidate set m' parameters that have the best fitting scores, and generates a new candidate set from the m' selected parameters by crossover and mutation. Crossover regards each parameter as an h-dimensional vector. Given two parameter vectors, it randomly selects a number $\bar{h}$ such that $0 < \bar{h} < h$ and splits each vector into the first $\bar{h}$ dimensions in the vector and the remaining $h - \bar{h}$ dimensions (the lower half). Then, it swaps the lower halves of the two vectors to generate two child vectors. These vectors are then mutated by adding a random noise to one randomly chosen dimension.

In [12], PGkM is applied to logistic regression, SVM, and k-means clustering. In the case of k-means clustering, the NICV formula in Equation 2.8, more precisely its non-normalized version, is used as the fitting function f, and the set of k cluster centroids is defined as parameter θ. Each parameter is a vector of $h = k \cdot d$ dimensions. Initially, the candidate set is populated with 200 sets of cluster centroids randomly sampled from the data space, each set containing exactly k centroids. Then, the algorithm runs iteratively for $\max\{8, (xN\epsilon)/m'\}$ rounds, where x and m' are empirically set to 1.25×10^{-3} and 10, respectively, and N is the dataset size.

We call the approach of applying PrivGene to k-means clustering PGkM, which is similarly to DPLloyd in that it tries to iteratively improve the centroids. However, rather than maintaining and improving a single set of k centroids, PGkM maintains a pool of candidates, uses selection to improve their quality, and crossover and mutation to broaden the pool.

By selecting multiple sets of centroids in each round and applying mutation, PGkM reduces the chance that the iterative process is stuck in a suboptimal solution. At the same time, doing this invariably slows down the converging process. At the same time, if one increases the number of iterations, each iteration becomes highly inaccurate. Thus whether PGkM is a suitable approach for a problem depends on whether the benefit of PGkM can compensate for the slow converging speed. Our experimental results in Section 4.6 show that for k-means clustering, this is not the case and PGkM performs poorly.

4.3.2 GkM

The k-means clustering problem was also used to motivate the *sample and aggregate* framework (SAF) for satisfying differential privacy, which was developed in [25, 59], and implemented in the GUPT system [29].

Given a dataset D and a function f, SAF first partitions D into ℓ blocks, then it evaluates f on each of the block, and finally it privately aggregates results from all blocks into a single one. Since any single tuple in D falls in one and only one block, adding one tuple can affect at most one block's result, limiting the sensitivity of the aggregation step. Thus one can add less noise in the final step to satisfy differential privacy.

As far as we know, GUPT [29] is the only implementation of SAF. Authors of [29] implemented k-means clustering and used it to illustrate the effectiveness of GUPT. We call this algorithm GkM. Given a dataset D, it first partitions D into ℓ blocks $D_1, D_2, \ldots, D_\ell$. Then, for each block D_b ($1 \leq b \leq \ell$), it calculates its k centroids $o^{b,1}, o^{b,2}, \ldots, o^{b,k}$. Finally, it averages the centroids calculated from all blocks and adds noise. Specifically, the i'th dimension of the j'th aggregated centroid is

$$o_i^j = \frac{1}{\ell}\sum_{b=1}^{\ell} o_i^{b,j} + \mathsf{Lap}\left(\frac{2(max_i - min_i)\cdot k \cdot d}{\ell \cdot \epsilon}\right), \tag{4.10}$$

where $o_i^{b,j}$ is the i'th dimension of $o^{b,j}$, $[min_i, max_i]$ is the estimated output range of i'th dimension. One half of the total privacy budget is used to estimate this output range, and the other half is used for adding Laplace noise.

We have found that the implementation downloaded from [60], which uses Equation (4.10), performed poorly. Analyzing the data closely, we found that min_i and max_i often fall outside of the data range, especially for small ϵ. We slightly modified the code to bound min_i and max_i to be within the data domain. This does not affect the privacy but is able to greatly improve the accuracy. In this chapter we use this fixed version.

Here a key parameter is the choice of ℓ. Intuitively, a larger ℓ will result in each block being very small and unable to preserve the cluster information in the blocks, and a smaller ℓ, on the other hand, results in large noise added. (Note the inverse dependency on ℓ in Equation (4.10)). Analysis in [29] suggests to set $\ell = N^{0.4}$. Our experimental results, however, show that the performance of GkM with $\ell = N^{0.4}$ is quite poor.

We can analytically show why GkM performs worse than DPLloyd. There are two sources of errors in GkM. The first is that the cluster centers obtained from different blocks may not be accurate. Such errors increase when the number of blocks increases, since then each block has fewer data points and is less likely to have centroids similar to the whole dataset. The second is due to the added noise in the aggregation step. The MSE due to the added noise is on the order of $\frac{k^2d^2}{\ell^2\epsilon^2}$. Compared with the MSE analysis of DPLloyd, they are comparable when $\ell \approx \frac{N}{t\sqrt{d}}$, that is, when each block contains only a small number of data points. It is unlikely that one could learn k centroids from such small blocks. At the same time, if one chooses $\ell = N^{0.4}$, then MSE will be linear in $\frac{k^2d^2}{N^{0.8}\epsilon^2}$, which is much larger than that of the DPLloyd method. This seems a fundamental limitation of the sample and aggregation approach.

4.4 Using a Private Synopsis

Approaches such as DPLloyd and GkM suffer from two limitations. First, often times the purpose of conducting k-means clustering is to visualize how the data points are partitioned into clusters. The single-workload approaches, however, output only the centroids. In the case of DPLloyd, one could also obtain the number of data points in each cluster; however, it cannot provide more detailed information on what shapes data points in the clusters take. The value of single-workload differentially private k-means clustering is thus limited. Second, as the privacy budget is consumed by the single-workload method, one cannot perform any other analysis on the dataset; doing so will violate differential privacy.

An approach where one first generates a synopsis of a dataset using a differentially private algorithm, and then applies k-means clustering algorithm on the synopsis, avoids these two limitations. In this chapter, we consider the following synopsis method. Given a d-dimensional dataset, one partitions the domain into M equal-width grid cells, and then releases the noisy count in each cell, by adding Laplace noise to each cell count.

The synopsis consists of a set of cells, each of which has a rectangular bounding box and a (noisy) count of how many data points are in the bounding box. The synopsis tells only how many points are in a cell, but not the exact locations of these points. For the purpose of clustering, we treat all points as if they are at the center of the bounding box. In addition, these noisy counts might be negative, non-integer, or both. A straightforward solution is to round the noisy count of a cell to be a non-negative nearest integer and replicate the cell center as many as the rounded count. This approach, however, may introduce a significant systematic bias in the clustering result, when many cells in the synopsis are empty or close to empty and these cells are not distributed uniformly. In this case, simply turning negative counts to zero can produce a large number of points in those empty areas, which can pull the centroid away from its true position. We take the approach of keeping the noisy count unchanged and adapting the centroid update procedure in k-means to use the cell as a whole. Specifically, given a cell with center c and noisy count $\tilde{n}$, its contribution to the centroid is $c \times \tilde{n}$. Using this approach, in one cluster, cells who have negative noisy count can "cancel out" the effect of other cells with positive noise. Therefore, we can have better clustering performance.

For this method, the key parameter is M, the number of cells. When M is large, the average count per cell is low, and the noise will have more impact. When M is small, each cell covers a large area, and treating all points as at the center may be inaccurate when the points are not uniformly distributed. We now describe two existing methods of choosing M and extend one of them.

4.4.1 MkM

Lei [33] proposed a scheme to release differentially private synopses tailored for the M-estimator. Given a d-dimensional dataset with N tuples, statistical analysis in [33] suggests that

$$M = \left(\frac{N}{\sqrt{\log(N)}} \right)^{\frac{2d}{2+d}} \tag{4.11}$$

We name the approach of applying the k-means clustering on this synopsis MkM.

4.4.2 UGkM

UG is a simple algorithm proposed in [31] for producing synopsis of 2-dimensional datasets that can be used to answer rectangular range queries (i.e., how many data points there are in a rectangular range) with high accuracy. The algorithm partitions the space into $M = m \times m$ equal-width grid cells, and then releases the noisy count in each cell. It is observed that for counting queries, a larger M value results in higher errors because more noises are added, and a smaller M value results in higher errors due to the fact that points within cells may be distributed nonuniformly, and queries including a portion of these cells may be answered inaccurately. To balance these two kinds of errors, it is suggested to set

$$m = \sqrt{\frac{N\epsilon}{10}}, \text{ or equivalently, } M = \frac{N\epsilon}{10} \tag{4.12}$$

It has been shown that UG performs quite well for answering rectangular range queries [31]. UG can be easily extended to d-dimensional dataset by setting $m = \sqrt[d]{M}$. We use UGkM to represent the UG-based k-means clustering scheme.

4.4.3 EUGkM

We now analyze the choice of M for higher-dimensional case. Given a d-dimensional rectangular range counting query, suppose that act is its precise answer and est is its estimated answer using the released noisy counts of the cells. We use *mean squared error* (MSE) to measure the accuracy of est with respect to act. That is,

$$\mathsf{MSE}\,(est) = \mathsf{E}\left[(est - act)^2\right] = \mathsf{Var}\,(est) + (\mathsf{Bias}\,(est))^2,$$

where $\mathsf{Var}\,(est)$ is the variance of est and $\mathsf{Bias}\,(est)$ is its bias.

There are two error sources when computing est. First, Laplace noises are added to cell counts to satisfy differential privacy. This results in the variance of est. Since counting a cell size has the sensitivity of 1, Laplace noise $\mathsf{Lap}\left(\frac{1}{\epsilon}\right)$ is added. Thus, the noisy count has the variance of $\frac{2}{\epsilon^2}$. Suppose that the given counting query covers α portion of the total M cells in the data space. Then, $\mathsf{Var}\,(est) = \alpha\frac{2M}{\epsilon^2}$. Second, the given counting query may not fully contain the cells that fall on the border of the query rectangle. To estimate the number of points in the intersection between the query rectangle and the border cells, it assumes that data are uniformly distributed. This results in the bias of est, which depends on the number of tuples in the border cells. The border of the given query consists of $2d$ hyper rectangles, each being $(d-1)$-dimensional. The number of cells falling on a hyper rectangle is in the order of $M^{\frac{d-1}{d}}$. On average the number of tuples in these cells is in the order of $M^{\frac{d-1}{d}} \cdot \frac{N}{M} = \frac{N}{M^{\frac{1}{d}}}$. Therefore, we estimate the bias of est with respect to one hyper rectangle to be $\beta \frac{N}{M^{\frac{1}{d}}}$, where $\beta \geq 0$ is a parameter. We thus estimate $(\mathsf{Bias}\,(est))^2$ to be $2d\left(\beta \frac{N}{M^{\frac{1}{d}}}\right)^2$. Summing the variance and the squared bias, it follows that

$$\mathsf{MSE}\,(est) = \alpha \frac{2M}{\epsilon^2} + \beta^2 \frac{2dN^2}{M^{\frac{2}{d}}}.$$

To minimize the MSE, we set the derivative of the above equation with respect to M to 0. This gives

$$M = \left(\frac{N\epsilon}{\theta}\right)^{\frac{2d}{2+d}}, \tag{4.13}$$

where $\theta = \sqrt{\frac{\alpha}{2\beta^2}}$. We name the above extended approach as EUG (extended uniform griding approach). We use EUGkM to represent the EUG-based k-means clustering scheme. The algorithm of EUGkM is shown in Algorithm 15.

4.5 The Hybrid Approach

As we will show in Section 4.6, DPLloyd still under-performs EUGkM in most settings. Recall that EUGkM publishes a private synopsis of the the dataset, and thus enables other analysis to be performed on the dataset, beyond k-means. An intriguing question is “Whether one can do better for k-means clustering?” In particular, can we further improve DPLloyd? Recall that there are two key issues that greatly affect the accuracy of DPLloyd: the number of iterations and the choice of initial centroids. In fact, these two are closely related. If the initially chosen centroids are very good and close to the true centroids, one only needs perhaps just one iteration to improve it, and this reduction in the number of iterations would mean less noise is added. Now if only we have a method to choose really good centroids in a differentially private way, then we can use part (e.g., half) of the privacy budget to get those initial centroids, and the remaining privacy budget to run one iteration of DPLloyd to further improve it.

In fact, we do have such a method. EUGkM does it. This leads us to propose a hybrid method that combines the synopsis-based EUGkM with the single-workload DPLloyd. We

Algorithm 15 EUGkM

Input: D: dataset, N: dataset size, d: number of dimensions, $[-r, r]$: dataset range, k: number of clusters, IC: set of initial centroids, ϵ: privacy budget

```
if $IC$ is empty then
    $\{o^1, o^2, \ldots, o^k\} \leftarrow$ SpherePackingInitialCentroidsGeneration$(d, r, k)$
else
    $\{o^1, o^2, \ldots, o^k\} \leftarrow IC$
end if
$M = \left(\frac{N\epsilon}{10}\right)^{\frac{2d}{2+d}}$
Construct grid $g$ by partitioning data space $[-r, r]^d$ into $M$ equal width cells
Construct noisy histogram $\hat{h}$ by projecting dataset $D$ into grid $g$
for each cell $c \in \hat{h}$ do
    $\hat{n}_c \leftarrow n_c + \mathrm{Lap}(1/\epsilon)$
end for
repeat
    for each $j$ $(j = 1, 2, \ldots, k)$ do
        Cluster $C^j \leftarrow \{c^\ell : \|c^\ell - o^i\| \leq \|c^\ell - o^j\|,$ cell center $c^\ell \in \hat{h}, \forall 1 \leq i \leq k\}$
        $count \leftarrow \sum_{c^\ell \in C^j} \hat{n}_{c^\ell}$
        for each dimension $i$ $(i = 1, 2, \ldots, d)$ do
            $sum_i \leftarrow \sum_{c^\ell \in C^j} c^\ell_i \cdot \hat{n}_{c^\ell}$
            $o^j_i \leftarrow \Pi_{[-r,r]}\left(\frac{sum_i}{count}\right)$
        end for
    end for
until No more changes on cluster centroids
return Cluster centroids $\{o^1, o^2, \ldots, o^k\}$ and noisy synopsis $\hat{h}$
```

first use a portion of the privacy budget to run EUGkM, and then use the centroids outputted by EUGkM as the initial centroids for one round of DPLloyd. Such a method, however, may not actually outperform EUGkM, especially when the privacy budget ϵ is small, since then one round of DPLloyd may actually worsen the centroids. Therefore, when ϵ is small, we should stick to the EUGkM method, and only when ϵ is large enough should we adopt the EUGkM+DPLloyd approach. In order to determine what ϵ is large enough, we analyze how the errors depend on the various parameters in DPLloyd and in EUGkM.

4.5.1 Error Study of EUGkM

Non-interactive approach partitions a dataset into a grid of M uniform cells. Then, it releases private synopses for the cells, and runs k-means clustering on the synopses to return the cluster centroids. Similar to the error analysis for DPLloyd, we analyze the MSE. Let o be the true centroid of a cluster, and $\widehat{o}$ be its estimator computed by a non-interactive approach. The MSE between $\widehat{o}$ and o is composed of two error sources. First, the count in each cell is inaccurate after adding Laplace noise. This results in the variance (i.e., $\mathsf{Var}\left(\widehat{o}\right)$) of $\widehat{o}$ from its expectation $\mathsf{E}\left[\widehat{o}\right]$. Second, we no longer have the precise positions of data points, and only assume that they occur at the center in a cell. Thus, the expectation of $\widehat{o}$ is not equal to o, resulting in a bias (i.e., $\mathsf{Bias}\left(\widehat{o}\right)$). The MSE is the combination of these two errors,

$$\mathsf{MSE}\left(\widehat{o}\right) = \mathsf{Var}\left(\widehat{o}\right) + \left(\mathsf{Bias}\left(\widehat{o}\right)\right)^2 \tag{4.14}$$

Analyzing the variance. We assume that each cluster has a volume that is $\frac{1}{k}$ of the total volume of the data space, and has the shape of a cube. In d-dimensional case, the width of the cube is $w = \frac{2r}{\sqrt[d]{k}}$. Suppose that the *geometric* center[1] of the cube is τ_i. Let T be the set of cells included in the cluster. For each cell $t \in T$, we use c_t to denote the number of tuples in t, t_i to denote the i'th dimension coordinate of the center of cell t, and ν_t to denote the noise added to the cell size. Let $\widehat{o_i}$ be the i-th dimension of the noisy centroid. Then, the variance of $\widehat{o_i}$ is

$$\begin{aligned}
& \mathsf{Var}\left(\widehat{o_i}\right) = \mathsf{Var}\left(\widehat{o_i} - \tau_i\right) \\
= \ & \mathsf{Var}\left(\frac{\sum_{t\in T} t_i(c_t+\nu_t)}{\sum_{t\in T}(c_t+\nu_t)} - \tau_i\right) \\
= \ & \mathsf{Var}\left(\frac{\sum_{t\in T}(t_i-\tau_i)(c_t+\nu_t)}{\sum_{t\in T}(c_t+\nu_t)}\right) \\
\approx \ & \frac{1}{C^2}\sum_{t\in T}\left((t_i-\tau_i)^2 \cdot \mathsf{Var}\left(c_t+\nu_t\right)\right).
\end{aligned}$$

[1] Note that this is not the cluster centroid.

In the above, the first step follows because τ_i as the cube geometric center is a constant. The last step is derived by assuming $\sum_{t\in T}(c_t+\nu_t) \approx C$, that is, the noisy cluster size is approximately equal to the original cluster size C.

We can see that within the cube, different cells' contribution to the variance is not the same. Basically, the closer a cell is to the cube center, the less its contribution. The contribution is proportional to the squared distance to the cube center. We thus approximate the variance as follows:

$$\begin{aligned} \mathsf{Var}\left(\widehat{o_i}\right) &\approx \frac{1}{C^2}\int_{-\frac{w}{2}}^{\frac{w}{2}} x^2 \left(\frac{M}{(2r)^d} w^{d-1} \frac{2}{\epsilon^2}\right) \mathrm{d}x \\ &= \frac{2Mr^2}{3C^2\epsilon^2 k^{\frac{d+2}{d}}}. \end{aligned}$$

In the above integral, x in the first term is the distance from a cell center to the cube center (i.e., $t_i - \tau_i$). The second term $\frac{M}{(2r)^d}$ is the number of cells per unit volume, and w^{d-1} is the volume of the $(d-1)$-dimensional plane that has a distance of x to the cube center. The last term $\frac{2}{\epsilon^2}$ is the variance of the cell size (i.e., $\mathsf{Var}\left(c_t+\nu_t\right)$). Suppose that clusters are of equal size, that is, $C = \frac{N}{k}$. Then, the variance of the noisy centroid by summing all the d dimensions is

$$\mathsf{Var}\left(\widehat{o}\right) \approx \frac{2dMr^2k^{\frac{d-2}{d}}}{3N^2\epsilon^2}. \tag{4.15}$$

The analysis shows that the variance of the EUGkM is proportional to $\frac{M}{(N\epsilon)^2}$. EUGkM sets M to $\left(\frac{N\epsilon}{10}\right)^{\frac{2d}{2+d}}$. Plugging it into Equation 4.15, we get that the variance of EUGkM is *inversely proportional* to $(N\epsilon)^{\frac{4}{2+d}}$.

Analyzing the bias . Let x_i be the i'th dimension coordinate of a tuple x. Then, the bias of $\widehat{o_i}$ is

$$\begin{aligned} &\mathsf{Bias}\left(\widehat{o_i}\right) = \mathsf{E}\left[\widehat{o_i}\right] - o_i \\ =\ & \mathsf{E}\left[\tfrac{\sum_{t\in T} t_i(c_t+\nu_t)}{\sum_{t\in T}(c_t+\nu_t)}\right] - \tfrac{\sum_{t\in T}\sum_{x\in t} x_i}{\sum_{t\in T} c_t} \\ \approx\ & \tfrac{\sum_{t\in T}\sum_{x\in t}(t_i - x_i)}{C}, \end{aligned}$$

where the last step is developed by approximating $\sum_{t\in T}(c_t+\nu_t)$ to the cluster size C.

The bias developed in the above formula is dependent on data distribution. Its precise estimation requires to access real data. We thus only estimate its upper bound. Let $q_i = t_i - x_i$. Non-interactive approach partitions each dimension into $\sqrt[d]{M}$ intervals of equal length. Hence, q_i falls in the range of $[-\frac{r}{\sqrt[d]{M}}, \frac{r}{\sqrt[d]{M}}]$, and the upper bound of $\mathsf{Bias}\left(\widehat{o_i}\right)$ is $\frac{r}{\sqrt[d]{M}}$. Summing all the d dimensions, we obtain the upper bound of squared bias of noisy centroid

$$\left(\mathsf{Bias}\left(\widehat{o}\right)\right)^2 \leq \frac{dr^2}{M^{\frac{2}{d}}}. \tag{4.16}$$

The estimation shows that the upper bound of squared bias decreases as a function of $M^{\frac{2}{d}}$. This is consistent with the expectation. As M increases, the data space is partitioned into finer-grained cells. Therefore, the distance between a tuple in a cell to the cell center decreases on average.

Comparing DPLloyd and EUGkM. For DPLloyd, its MSE is inversely proportional to $(N\epsilon)^2$ (Equation 4.5). For EUGkM, its MSE consists of variance and squared bias. Plugging $M = \left(\frac{N\epsilon}{10}\right)^{\frac{2d}{2+d}}$ into Equation 4.15 and Inequality 4.16, it follows that the MSE of EUGkM is inversely proportional to $(N\epsilon)^{\frac{4}{2+d}}$. Therefore, the MSE of DPLloyd drops much faster than that of EUGkM as ϵ increases or the dataset size N increases.

The MSE of EUGkM is inversely proportional to $(N\epsilon)^{\frac{4}{2+d}}$. Thus, it increases exponentially as a function of d. Instead, from Equation 4.5, it follows that the MSE of DPLloyd has only cubic growth with respect to d. Therefore, DPLloyd is more scalable to d than EUGkM.

In Section 4.6.2, we will demonstrate the above analysis well explains the empirical performance for DPLloyd and EUGkM.

4.5.2 Hybrid Approach

Our hybrid approach combines EUGkM and DPLloyd. Given a dataset and privacy budget ϵ, the hybrid approach first checks whether it overtakes the DPLloyd method and also the EUGkM method. If this is not the case, the hybrid approach simply falls back to EUGkM. Otherwise, the hybrid approach allocates a portion of privacy budget to EUGkM to output a synopsis and find k intermediary centroids that work well for the synopsis. Then, it runs DPLloyd for one iteration using the remaining privacy budget to refine these k centroids. The hybrid algorithm finally outputs a noisy synopsis as well as the cluster centroids. The full algorithm of the hybrid approach is in Algorithm 16.

We use MSE to heuristically determine the conditions, on which the hybrid approach overtakes the DPLloyd method and also the EUGkM method. Basically, we require that the MSE of the hybrid approach be smaller than those of the other two approaches, since smaller MSE implies smaller error to the cluster centroid. From Equation 4.5, it follows that the MSE of DPLloyd with full privacy budget is

$$\mathsf{MSE}_{\mathsf{DPLloyd}}(\epsilon, t) = 2d(1 + (2\rho r)^2)\left(\frac{kt(dr+1)}{N\epsilon}\right)^2. \tag{4.17}$$

The MSE of the EUGkM method consists of two parts, the variance and the bias. The variance of the approximate variance (Equation 4.15) by setting $M = \left(\frac{N\epsilon}{10}\right)^{\frac{2d}{2+d}}$.

$$\mathsf{Var}_{\mathsf{EUGkM}}(\epsilon) = \frac{2dr^2(k)^{\frac{d-2}{d}}}{3 \times (10)^{\frac{2d}{2+d}}(N\epsilon)^{\frac{4}{2+d}}}. \tag{4.18}$$

Similarly, the bias part is

$$\mathsf{Bias}_{\mathsf{EUGkM}}(\epsilon) = \frac{dr^2}{(N\epsilon/10)^{\frac{4}{2+d}}}. \tag{4.19}$$

Suppose that in the Hybrid approach, f portion of the total privacy budget ϵ is allocated to the EUGkM part, we model the MSE of the hybrid approach,

$$\begin{aligned}\mathsf{MSE}_{\mathsf{Hybrid}}(\epsilon, f) = \omega_1 \cdot \mathsf{Var}_{\mathsf{EUGkM}}(f\epsilon) + \omega_2 \cdot \mathsf{Bias}_{\mathsf{EUGkM}}(f\epsilon)\\ + \omega_3 \cdot \mathsf{MSE}_{\mathsf{DPLloyd}}((1-f)\epsilon, t=1).\end{aligned} \tag{4.20}$$

The LHS of Eq (4.20) is the the best actual MSE values that the hybrid approach can achieve, while the RHS of Eq (4.20) are theoretical values of the EUGkM's variance, bias and DPLloyd's MSE. We use linear regression to estimate the parameters of the above error model on the ideal set of synthetic datasets (Synthe-PT dataset). After building the linear regression model, we have $\omega_1 = 0.14$, $\omega_2 = -0.0019$ and $\omega_3 = 0.42$. Since the parameter for the $\mathsf{Bias}_{\mathsf{EUGkM}}(f\epsilon)$ is very small, we remove it from the regression model.

Algorithm 16 Hybrid Method for k-means Clustering

```
Input: $D$: dataset, $N$: dataset size, $d$: number of dimensions, $[-r, r]$: dataset range, $k$: number of clusters, $IC$: set of initial centroids , $\epsilon$: privacy budget
Optimize Eq (4.20) to get the best $\mathsf{MSE}_{\mathsf{Hybrid}}$ and the best allocation ratio $f$
if $\mathsf{MSE}_{\mathsf{Hybrid}}(\epsilon) < \mathsf{VarEUGkM}(\epsilon)$ then
    $\mathcal{C}_{\text{inter}}, \hat{h} \leftarrow \text{EUGkM}(D, N, d, [-r, r], k, \emptyset, f \cdot \epsilon)$
    $\mathcal{C}_{\text{final}} \leftarrow \text{DPLloydOptimizationForOneIteration}(D, d, [-r, r], k, \mathcal{C}_{\text{inter}}, (1-f)\epsilon)$
else
    $\mathcal{C}_{\text{final}}, \hat{h} \leftarrow \text{EUGkM}(D, N, d, [-r, r], k, \emptyset, \epsilon)$
end if
return $\mathcal{C}_{\text{final}}$
```

4.6 Performance and Analysis

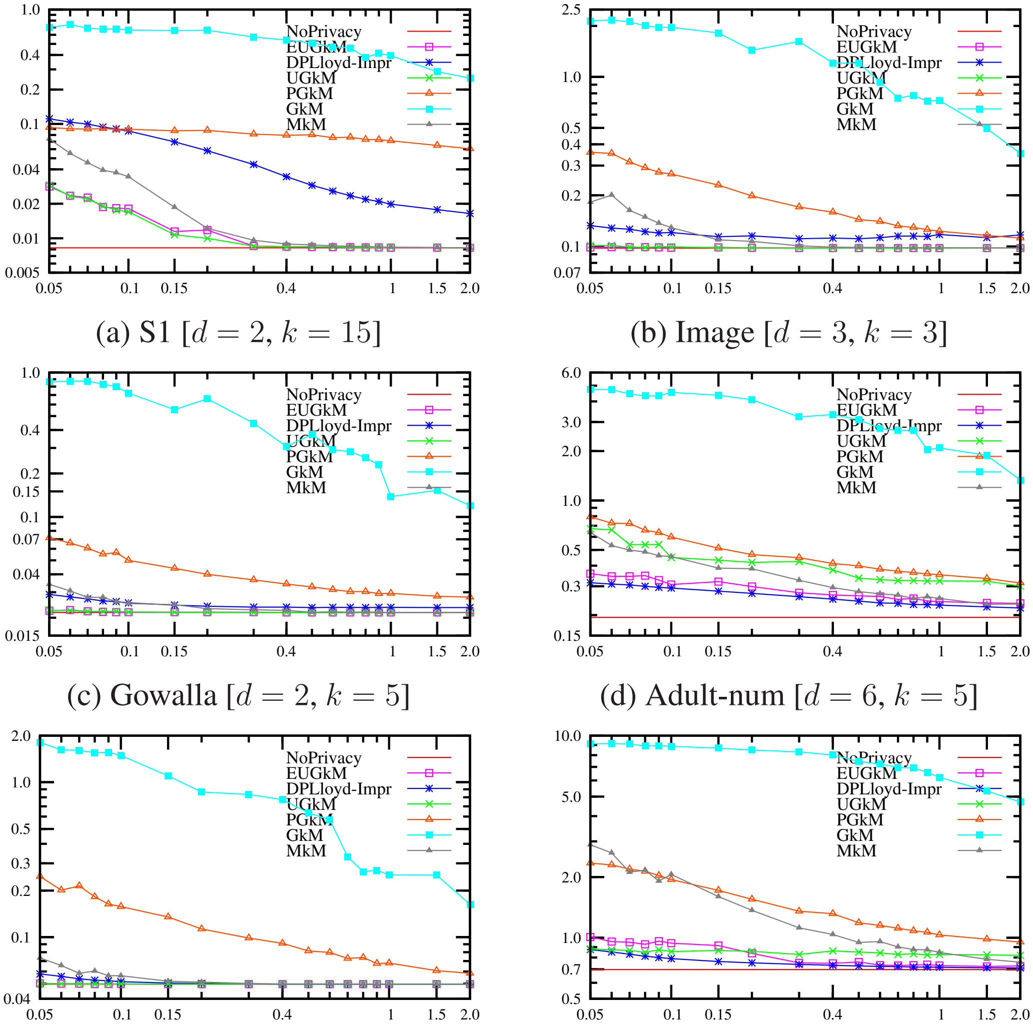
Table 4.1.: Descriptions of datasets.

Dataset	Number tuples	Number of dimensions	Number of clusters
S1	5,000	2	15
Gowalla	107,091	2	5
TIGER	16,281	2	2
Image	34,112	3	3
Adult-num	48,841	6	5
Lifesci	26,733	10	3
Synthe	10,000 + O	$\{2, 3, \ldots, 10\}$	$\{2, 3, \ldots, 10\}$
Synthe-PT	10,000	$\{2, 3, \ldots, 10\}$	$\{2, 3, \ldots, 10\}$

O is # outliers and is uniformly sampled from $[0, 100]$.

Table 4.2.: Summary of differentially private k-means methods

	Methods	**Description**
Non-interactive	EUGkM	Our proposed method to release synopsis for k-means under DP.
	UG [31]	DP release synopsis for answering rectangular range queries on 2D data.
	MkM [33]	DP release synopsis tailored for the M-estimator.
Interactive	DPLloyd [16]	DP Lloyd algorithm.
	DPLloyd Impr	Our proposed improvement of DPLloyd.
	PGkM [12]	Private model fitting based on genetic algorithms.
	GkM [29]	GUPT based k-means.



(a) S1 $[d = 2, k = 15]$

(b) Image $[d = 3, k = 3]$

(c) Gowalla $[d = 2, k = 5]$

(d) Adult-num $[d = 6, k = 5]$

(e) TIGER $[d = 2, k = 2]$

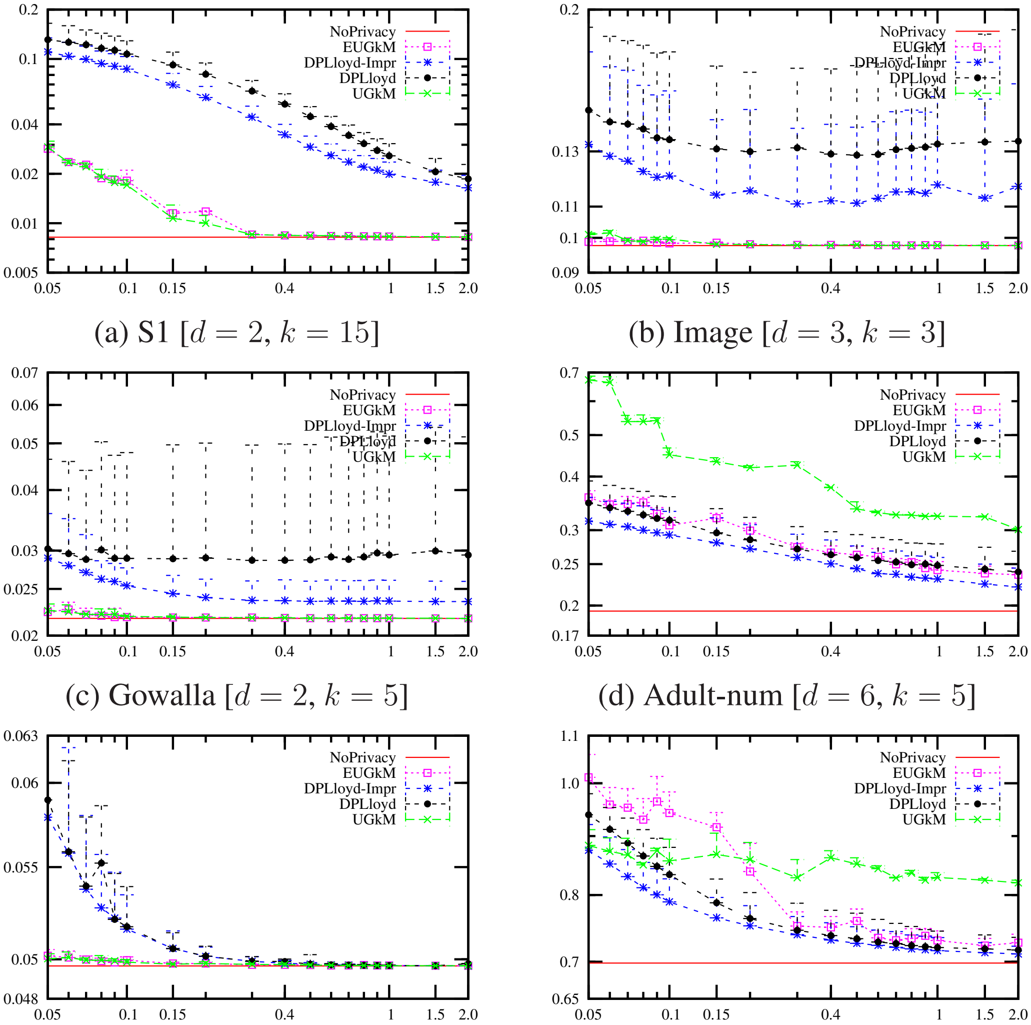
(f) Lifesci $[d = 10, k = 3]$

Figure 4.1.: The comparison of DPLloyd-Impr, PGkM, GkM, EUGkM, UGkM and MkM by varying the privacy budget ϵ. x-axis: privacy budget ϵ in log-scale. y-axis: NICV in log-scale.

In this section, we compare and analyze the performance of the seven methods described in Table 5.2.

4.6.1 Evaluation Methodology

We experimented with six external datasets and two sets of syntheticly generated datasets. The first external dataset is a 2D synthetic dataset S1 [61], which is a benchmark



(a) S1 $[d = 2, k = 15]$

(b) Image $[d = 3, k = 3]$

(c) Gowalla $[d = 2, k = 5]$

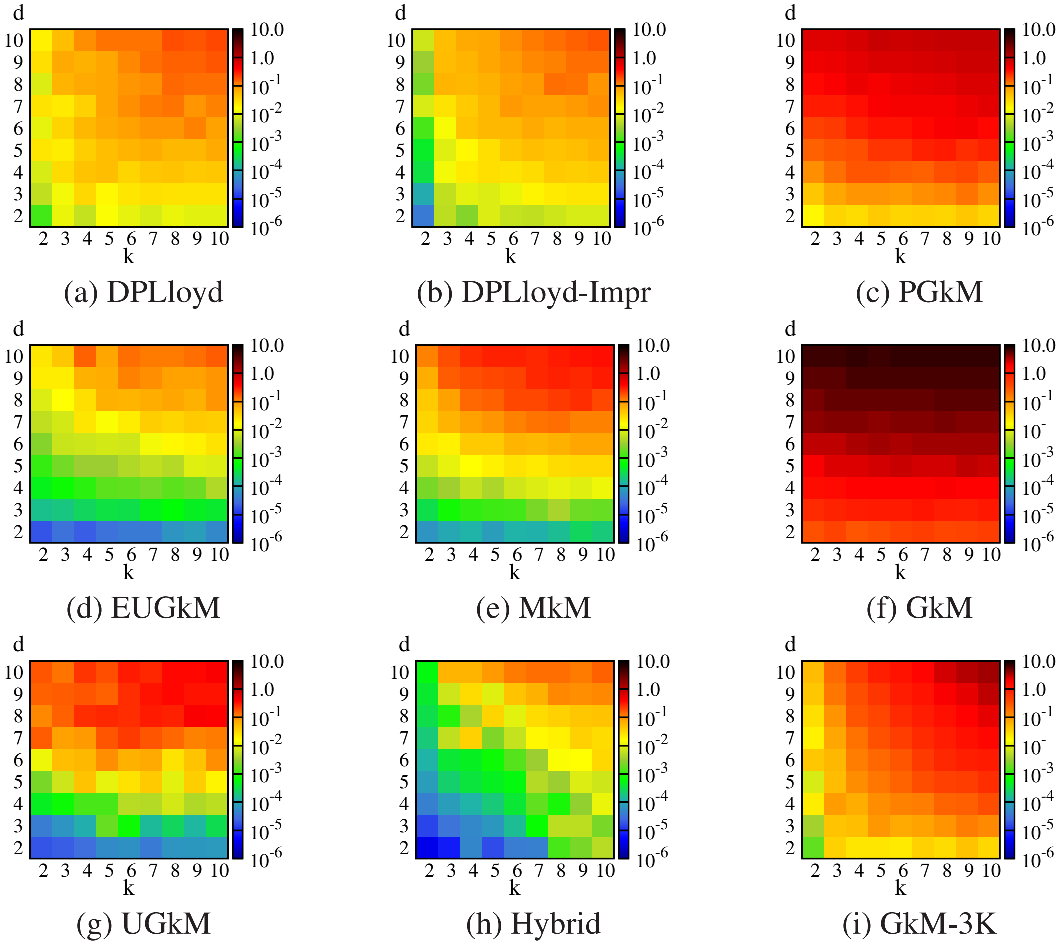
(d) Adult-num $[d = 6, k = 5]$

(e) TIGER $[d = 2, k = 2]$

(f) Lifesci $[d = 10, k = 3]$

Figure 4.2.: The close-up view of the comparison of DPLloyd-Impr, DPLloyd, EUGkM, and UGkM by varying the privacy budget ϵ. x-axis: privacy budget ϵ in log-scale. y-axis: NICV in log-scale.

to study the performance of clustering schemes. S1 contains 5,000 tuples and 15 Gaussian clusters. The Gowalla dataset contains the user checkin locations from the Gowalla location-based social network whose users share their checking-in time and locations (longitude and latitude). We sample one locaiton of each user ID and obtain a 2D dataset of 107,091 tuples. We set the number of clusters, $k = 5$, for this dataset. The third dataset is a 1-percentage sample of road dataset which was drawn from the 2006 TIGER (Topologi-



(a) DPLloyd

(b) DPLloyd-Impr

(c) PGkM

(d) EUGkM

(e) MkM

(f) GkM

(g) UGkM

(h) Hybrid

(i) GkM-3K

Figure 4.3.: The heatmap by varying k and d on the Synthe datasets with $\epsilon = 1.0$.

cally Integrated Geographic Encoding and Referencing) dataset [62]. It contains the GPS coordinates of road intersections in the states of Washington and New Mexico. The fourth is Image [61], a 3D dataset with 34,112 RGB vectors. We set $k = 3$ for it. We also use the well known Adult dataset [52]. We use its six numerical attributes, and set $k = 5$. The last dataset is Lifesci. It contains 26,733 records and each of them consists of the top 10 principal components for a chemistry or biology experiment. As previous approaches [12, 29], we set $k = 3$. Table 4.1 summarizes the datasets. For all the datasets, we normalize the domain of each attribute to [-1.0, 1.0].

We generate two sets of synthetic datasets. The first set of synthetic datasets, which we call Synthe, is generated by using the clusterGeneration [63] R package. It is designed for generating cluster datasets with specified degree of separation which is a quantitative measure of closeness between any cluster and its nearest neighboring cluster. Besides, the clusterGeneration package can generate clusters with arbitrary diameters, shapes and orientations. In this chapter, we generate 81 dataset by varying k and d from 2 to 10. We fix the dataset size to 10,000 and distribute them into k clusters with size proportional to the ratio $1 : 2 : \dots : k$. We also inject few outliers whose number is uniformly sampled from $[0, 100]$. For each dataset, we randomly sample its degree of separation from $[0.16, 0.26]$, which means from clusters with small overlapping to separated-but-close clusters.

The second set of synthetic dataset is mainly for tuning parameters of the EUGkM algorithm. We fix the dataset size to be 10,000, and vary k and d from 2 to 10 respectively. For each dataset, k well separated Gaussian clusters with equal size are generated. We call the second set of synthetic dataset as the Synthe-PT set, where PT stands for parameter tuning.

Implementations for DPLloyd and GkM were downloaded from [27] and [60], respectively. The source code of PGkM [12] was shared by the authors. We implemented EUGkM, UGkM and MkM.

Configuration. Each algorithm outputs k centroids $\mathbf{o} = \{o^1, o^2, \cdots, o^k\}$. The quality of the centroids $\mathbf{o}$ is evaluated by the Normalized Intra-Cluster Variance (NICV) (Eq.2.8).

We note that since both DPLloyd, EUGkM, UGkM and MkM use Lloyd-style iteration, they are affected by the choice of initial centroids. In addition, all algorithms have random noises added somewhere to satisfy differential privacy. To conduct a fair comparison, we need to carefully average out such randomness effects. GkM and PGkM do not take a set of initial centroids as input. GkM divides the input dataset into multiple blocks, and for each block invokes the standard k-means implementation from the Scipy package [64] with a different set of initial centroids to get the result, and finally aggregates the outputs for all the blocks. We run GkM and PGkM 100 times and report the average result.

DPLloyd-Impr generates 30 sets of initial centroids by using the proposed sphere packing method in Section 4.2.1. We run DPLloyd-Impr 100 times on each set of initial centroids, and report the average of the 3,000 NICV values as the final evaluation of DPLloyd-

Impr. For DPLloyd, we randomly generate 30 sets of initial centroids and use the same way to compute the averaged NICV values.

Algorithms based on a private synopsis (such as EUGkM) have the advantage that once a synopsis is published, one can run k-means clustering with as many sets of initial centroids as one wants and choose the result that has the best performance relative to the synopsis. In our experiments, given a synopsis, we use the same 30 sets of initial centroids as those generated for the DPLloyd-Impr method. For each set, we run clustering and output a set of k centroids. Among all the 30 sets of output centroids, we select the one that has the lowest NICV relative to the synopsis rather than to the original dataset. This process ensures selecting the set of output centroids satisfies differential privacy. We then compute the NICV of this selected set relative to the original dataset, and take it as the resulting NICV with respect to the synopsis. To deal with the randomness introduced by the process of generating synopsis, we generate 10 different synopses and take the average of the resulting NICV values.

For EUGkM, we set the the parameter $\theta = 10$. We experimentally compare the EUGkM's performance on different θ choices and find that $\theta = 10$ for EUGkM works well in most cases. This parameter tuning for EUGkM is performed on the Synthe-PT dataset. Therefore, the following evaluation of EUGkM on the Synthe dataset strictly satisfies differential privacy, since the parameter is determined on an independent set of datasets.

As the baseline, we run standard k-means algorithm [24] over the same 30 sets of initial centroids generated in DPLloyd-Impr and take the minimum NICV among all the 30 runs.

4.6.2 Experimental Results.

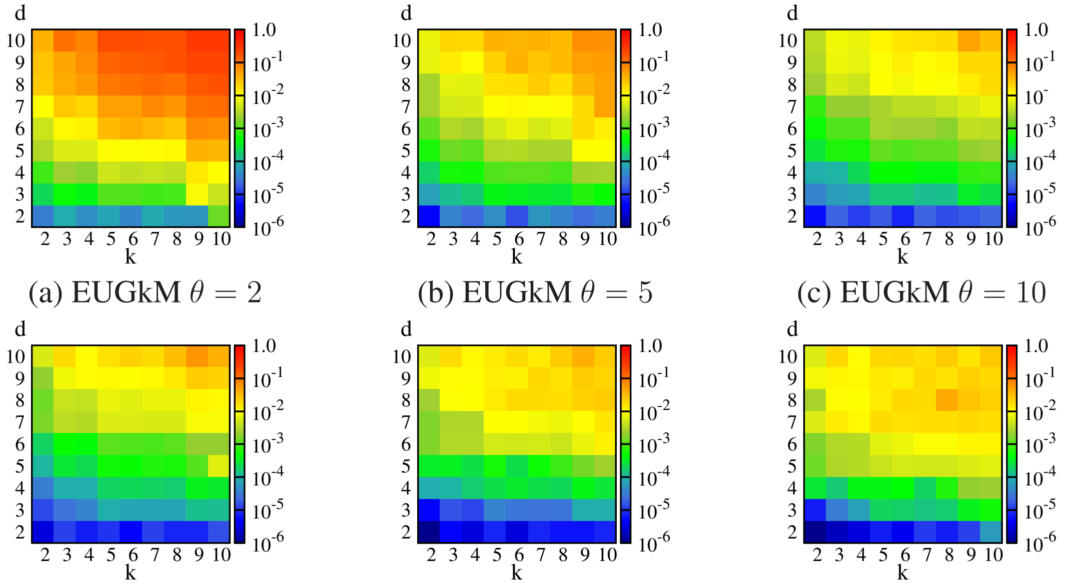
Fig. 4.1 and Fig. 4.2 report the results for the 6 external datasets. For these, we vary ϵ from 0.05 to 2.0 and plot the NICV curves for the methods listed in Table 5.2. This enables us to see how these algorithms perform under different privacy budgets.

Fig. 4.3 reports the results on the Synthe datasets. For these, we fix $\epsilon = 1.0$ and report the difference of NICV values between each approach and the baseline. This enables us to see the scalability of these algorithms when k and d increase.

Among approaches not using a synopsis, DPLloyd-Impr has the best performance in most cases. It also outperforms DPLloyd in most cases. For synopsis-based approaches, both EUGkM and UGkM clearly outperform MkM, especially for small ϵ values. EUGkM and UGkM has close performance on the low dimensional datasets. As the dimensionality increases, the advantage of EUGkM to UGkM becomes obvious. Comparing DPLloyd-Impr and EUGkM (Fig. 4.2), we observe that in the four low dimensional external datasets (S1, Gowalla, TIGER and Image), EUGkM clearly outperforms DPLloyd-Impr at small ϵ value and their gap becomes smaller as ϵ increases. However, in the two high dimensional datasets (Adult-num and Lifesci), DPLloyd-Impr outperforms EUGkM almost in all given privacy budgets. The similar observations can also be found in Fig. 4.3.

Fig. 4.3 also exhibits the effects of the number of clusters and the number of dimensions. The EUGkM's performance is more sensitive to the increase of dimension, while DPLloyd-Impr gets worse quickly as the number of clusters increases. In addition, Figure 4.4 shows the difference of EUGkM's performance on the synthetic dataset for parameter tuning under different θ choices. We can see that setting $\theta = 10$ for EUGkM works well in most cases.

The empirical performance of DPLloyd and EUGkM in Fig. 4.2 and Fig. 4.3 can be well explained by our theoretical analysis in Section 4.2.1 and Section 4.5.1. Recall that the MSE of DPLloyd is proportional to $\left(\frac{(kt)^2 d^3}{(N\epsilon)^2}\right)$ and that of EUGkM is proportional to $\left(\frac{1}{(N\epsilon)^{\frac{4}{2+d}}}\right)$. This explains why the NICV of DPLloyd, which is inversely proportional to $(N\epsilon)^2$, drops much faster than that of EUGkM as ϵ grows in Fig. 4.2. It also explains why DPLloyd has better performance on 'big' dataset (e.g., the TIGER dataset). This also explains that, as the dimensionality of dataset increases, DPLloyd outperforms EUGkM (Fig. 4.2) and why DPLloyd is more scalable to d than EUGkM (Fig. 4.3).



(a) EUGkM $\theta = 2$

(b) EUGkM $\theta = 5$

(c) EUGkM $\theta = 10$

(d) EUGkM $\theta = 20$

(e) EUGkM $\theta = 50$

(f) EUGkM $\theta = 100$

Figure 4.4.: The heatmap by varying k and d on the Synthe-PT datasets. $\epsilon = 1.0$. Varying the θ value in EUGkM.

4.6.3 Performance of the Hybrid Approach

We now compare the hybrid approach with EUGkM and DPLloyd. The configuration for EUGkM and DPLloyd is the same as in Section 4.6.1. For the hybrid approach, we run EUGkM 10 times to output 10 sets of intermediate centroids. Then we run DPLloyd 10 times on each intermediate result. We finally report the average of 100 NICV values. Fig. 4.5 gives the results on the six external datasets. In low dimensional datasets (S1, Gowalla, TIGER, and Image), the hybrid approach simply falls back to EUGkM for small ϵ value. When ϵ increases, both the hybrid approach and EUGkM converge to the baseline with the former having slightly better performance. For example, in the Gowalla dataset for $\epsilon = 0.7$, the average NICV of the hybrid approach is 0.02172 and that of EUGkM is 0.02174.

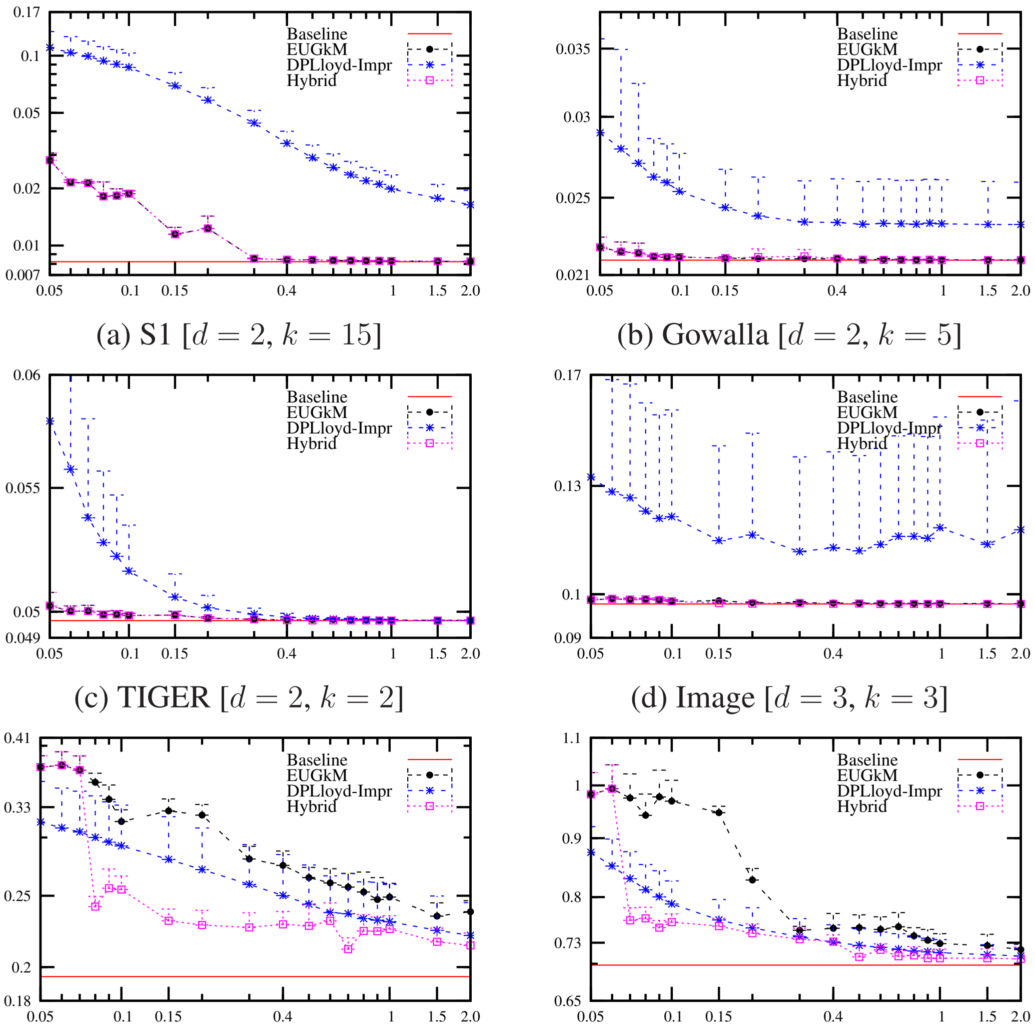
In higher dimensional datasets (Adult-num and Lifesci), the hybrid approach outperforms the other two approaches in most cases. It is worse than DPLloyd only for a few small ϵ values, on which it falls back to EUGkM. There are two possible reasons. The first is that the MSE analysis assumes that datasets are well clustered and each cluster has equal size, but the real datasets are skewed. For example, the baseline approach partitions the Adult-num dataset into 5 clusters, in which the biggest cluster contains 13,894 tuples and the smallest contains 3,160 tuples. The second is that we use the variance of EUGkM as the lower bound of its MSE. Thus, it is possible that the MSE of the hybrid approach (approximated by the MSE of one-iteration DPLloyd with half privacy budget) is larger than the variance of EUGkM, but actually smaller than its MSE. In such cases, the hybrid approach gives lower NICV if it does not fall back to EUGkM. For example, on the Adult-num dataset for $\epsilon = 0.05$, the hybrid approach of falling back to EUGkM has the NICV of 0.370, while its NICV is 0.244, if it applies EUGkM plus one-iteration of DPLloyd.

We also evaluate the hybrid approach using the Synthe datasets as generated in Section 4.6.1. Fig. 4.3 clearly shows that the hybrid approach is more scalable than EUGkM with respect to both k and d. This confirms the effectiveness of the hybrid approach.

4.6.4 The Analysis of the GkM Approach

From Fig. 4.1 and Fig. 4.3, it is clear that GkM is always much worse than others. There are two sources of errors for GkM. One is that GkM is aggregating centroids computed from the subsets of data, and this aggregation may be inaccurate even without adding noise. The other is that the noise added according to Equation (4.10) may be too large. We find that setting $\ell = N^{0.4}$ in GkM, which corresponds to block size of $N^{0.6}$, is far from optimal, as the error GkM is dominated by that from the noise, and is much higher than the error due to sample and aggregation.

Fig. 4.6 shows the effect of varying block size from around $N^{0.1}$ to N on the two sources of errors. In Fig. 4.6, we show error from GkM, error from using the aggregation without noise (SAG), and error from adding noise computed by Equation 4.10) to the best known centroids (Noise). From the figure, it is clear that setting $\ell = N^{0.4}$, which corresponds to block size of $N^{0.6}$ is far from optimal, as the error GkM is dominated by that from the noise, and is much higher than the error due to sample and aggregation. Indeed,



(a) S1 $[d = 2, k = 15]$

(b) Gowalla $[d = 2, k = 5]$

(c) TIGER $[d = 2, k = 2]$

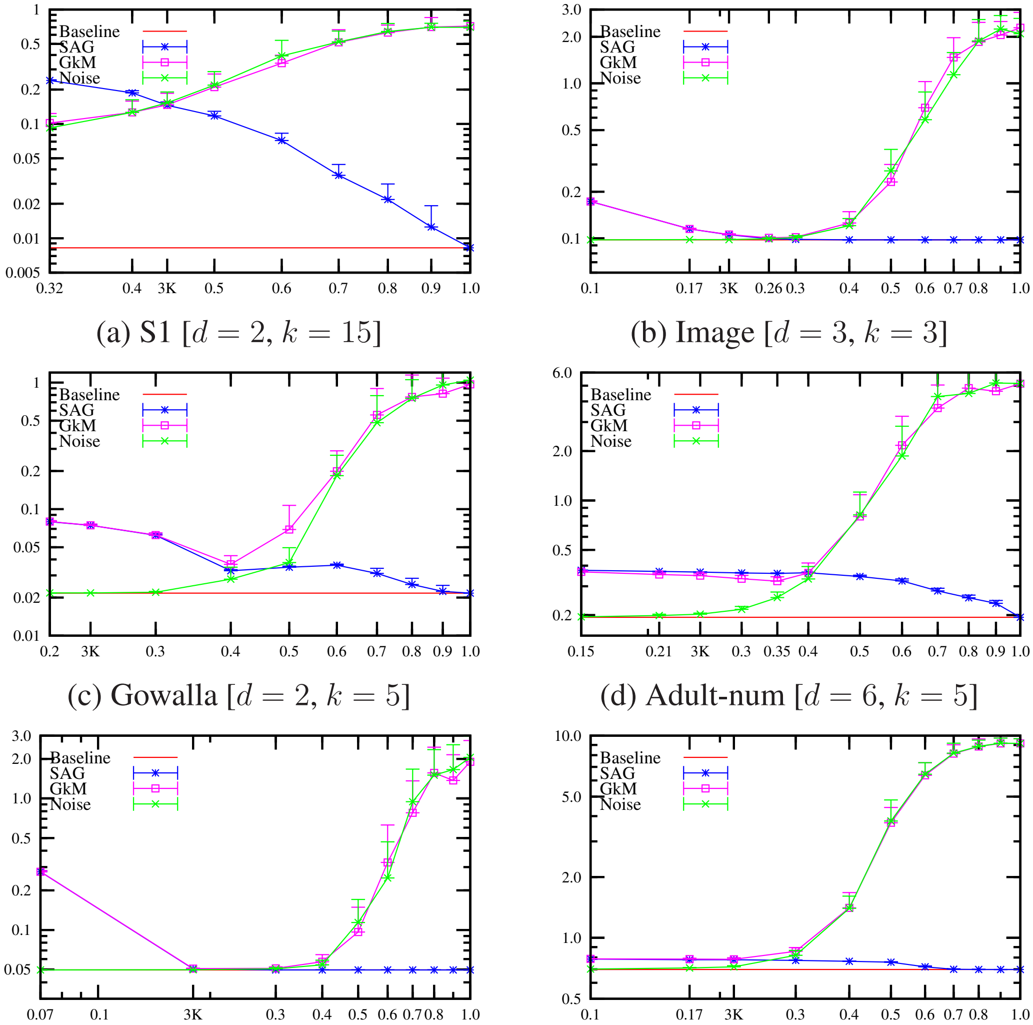
(d) Image $[d = 3, k = 3]$

(e) Adult-num $[d = 6, k = 5]$

(f) Lifesci $[d = 10, k = 3]$

Figure 4.5.: The comparison of the Hybrid approach with EUGkM and DPLloyd-Impr. x-axis: privacy budget ϵ in log-scale. y-axis: NICV in log-scale.

we observed that as the block size decreases the error of GkM keeps decreasing, until when the block size gets close to k. It seems that even though many individual blocks result in poor centroids, aggregating these relatively poor centroids can result in highly accurate centroids. This effect is most pronounced in the Tiger dataset, which consists of two large clusters. The two centroids computed from each small block can be approximately viewed as choosing one random point from each cluster. When averaging these centroids, one gets very close to the true centroids.



(a) S1 $[d = 2, k = 15]$

(b) Image $[d = 3, k = 3]$

(c) Gowalla $[d = 2, k = 5]$

(d) Adult-num $[d = 6, k = 5]$

(e) TIGER $[d = 2, k = 2]$

(f) Lifesci $[d = 10, k = 3]$

Figure 4.6.: The analysis of the GkM Approach. x-axis: block size exponent in log-scale, y-axis: NICV in log-scale.

4.6.5 The Analysis of the PGkM Approach

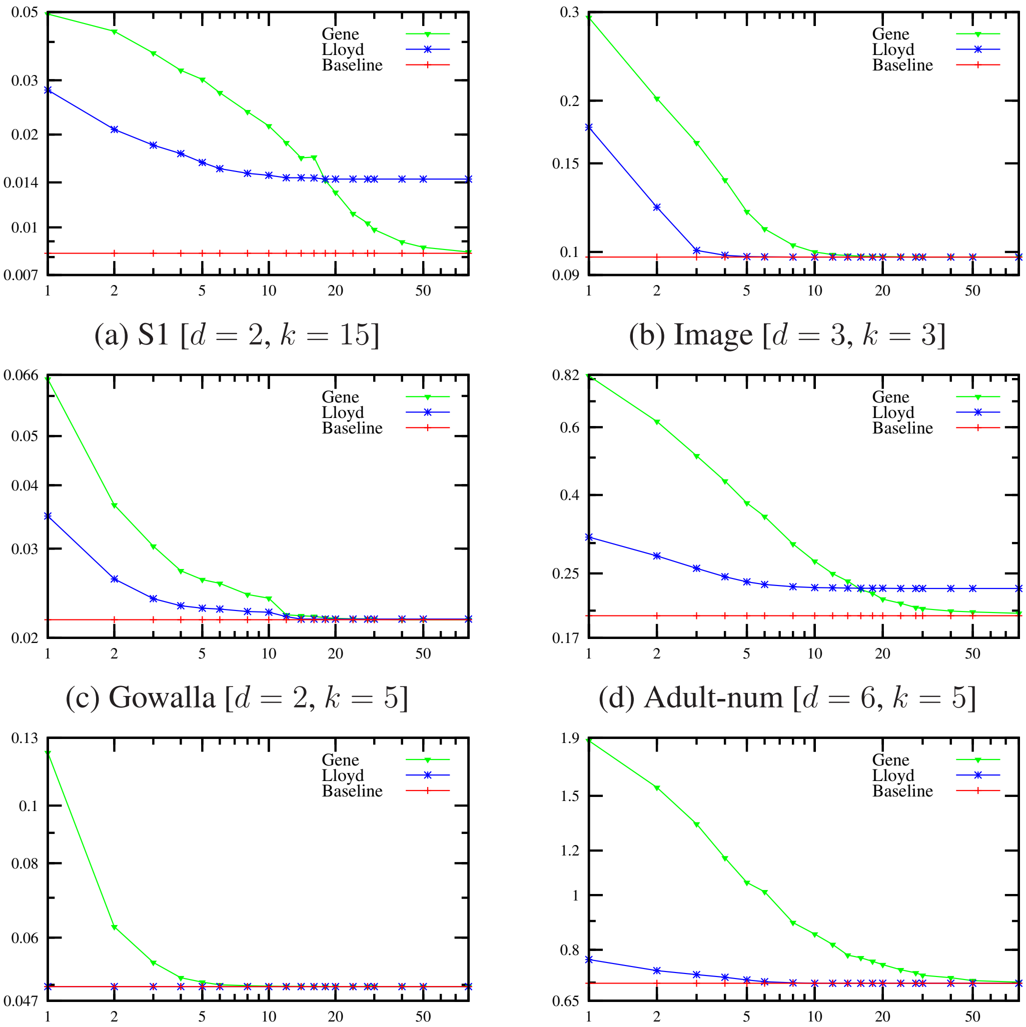
PGkM is a stochastic k-means method based on genetic algorithms. A stochastic method converges to global optimum [65]. On the contrary, DPLloyd is a gradient descent method derived from the standard Lloyd's algorithm [24], which may reach local optimum. However, PGkM is still inferior to DPLloyd in Fig. 4.3.

There are two possible reasons. First, a stochastic approach typically takes a ‘larger’ number of iterations to converge [65]. We compare the Lloyd’s algorithm with Gene (i.e., the non-private version of PGkM without considering differential privacy) in terms of NICV over the number of iterations in Fig. 4.7. For Lloyd, we reuse the initial centroids generated in Section 4.6.1. Given a dataset, we run Lloyd on the 30 sets of initial centroids generated for the dataset, and report the average NICV. Generally, Gene overtakes Lloyd as the number of iterations increases and finally converges to the global optimum. However, Lloyd improves its performance much faster than Gene in the first few iterations, and converges to the global optimal (or local optimum) more quickly. For example, in the Image dataset, Lloyd reaches the best baseline after three iterations, while the Gene needs more than 10 iterations to achieve the same.

The second reason is that the low privacy budget allocated to select a parameter (i.e., a set of k cluster centroids) from the candidate set. In each iteration PGkM selects 10 parameters, and the total number of iterations is at least 8. Thus, the privacy budget allocated to select a single parameter is at most $\epsilon/80$. Therefore, PGkM has reasonable performance only for big ϵ value.

4.6.6 The Analysis of the EUGkM, UGkM and MkM Approaches

The difference between of the three synopsis-based methods, EUGkM, UGkM and MkM is the choice of grid size M. The EUGkM method sets it to $\left(\frac{N\epsilon}{10}\right)^{\frac{2d}{2+d}}$, the UGkM method sets it to $\left(\frac{N\epsilon}{10}\right)$ and the MkM method sets it to $\left(\frac{N}{\sqrt{\log(N)}}\right)^{\frac{2d}{2+d}}$. Fig. 4.1 and Fig. 4.3 show that the performance of UGkM and EUGkM are superior to that of MkM. An important reason is that MkM does not take ϵ as a factor in M. Thus, it is nonadaptive to the variation of ϵ. This explains why EUGkM and UGkM perform much better than MkM for small ϵ values. On the other hand, although UGkM considers the impact of the privacy budget ϵ, it does not produce large enough grids for the high dimensional data. This explains why EUGkM performs better on high dimensional data than UGkM.



(a) S1 $[d = 2, k = 15]$

(b) Image $[d = 3, k = 3]$

(c) Gowalla $[d = 2, k = 5]$

(d) Adult-num $[d = 6, k = 5]$

(e) TIGER $[d = 2, k = 2]$

(f) Lifesci $[d = 10, k = 3]$

Figure 4.7.: The comparison of the convergence rate of the genetic algorithm based k-means and Lloyd algorithm. x-axis: number of iterations in log-scale, y-axis: NICV in log-scale.

4.6.7 Estimating the Number of Clusters.

In cluster analysis, an important problem is to estimate the number of clusters, which has a deterministic effect on the clustering results. Such problem becomes more prominent in the differential privacy setting, since the data analyst cannot access the private database as many times as she/he wants.

Our EUGkM approach can address this problem. Several heuristics and statistics [66, 67] have been proposed to determine the number of clusters k automatically. Suppose we have a list of candidate values of k and one statistics ϕ for determining the best k. Once an EUGkM synopsis is published, we evaluate ϕ for each candidate k value on this noisy synopsis. The k value with the best ϕ score will be selected for the following k-means clustering. All the operations are performed on the released EUGkM synopsis. So the estimation process satisfies the differential privacy. This is another advantage of using a private synopsis.

We also experimentally evaluate the above method on the six external datasets and on six privacy budget values. This method gives very accurate estimations on the k values under most of the privacy budget settings. We omit the experimental results for space reasons.

Fig. 4.8 presents the running time of DPLloyd and EUGkM on the six external datasets. We follow the same experiment configuration as in Section 4.6.1. As expected, the running time of DPLloyd is much lower than that of EUGkM. This is because EUGkM has to run k-means clustering over 30 sets of initial centroids and output the centroids with the best NICV relative to the noisy synopsis. Another reason is that DPLloyd sets the number of iterations to 5 while EUGkM runs k-means clustering until converge.

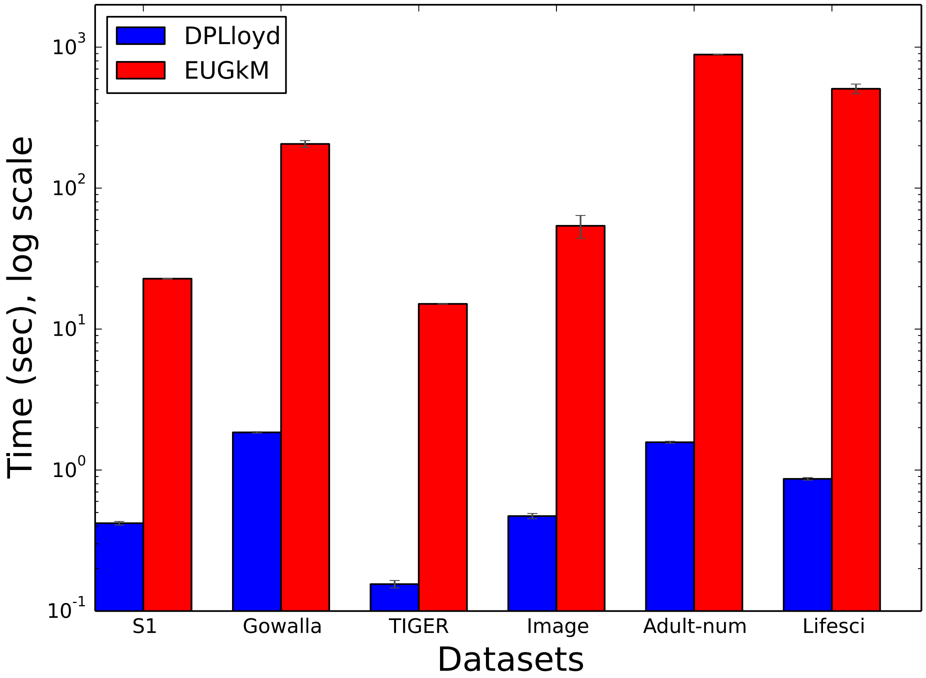


Figure 4.8.: Comparing running time between DPLloyd and EUGkM, $\epsilon = 0.1$

Table 4.3.: Likelihood of the Top-4 Selected k values based on RT-validity over S1 and Gowalla datasets.

Dataset	ϵ	Top-1			Top-2			Top-3			Top-4		
k Range		k	Likelihood	Validity	k	Likelihood	Validity	k	Likelihood	Validity	k	Likelihood	Validity
S1	0.05	16	0.71	0.105676	15	0.28	0.103682	14	0.01	0.087395			
[2, 16]	0.1	7	0.38	0.163604	15	0.29	0.156305	6	0.18	0.162932	4	0.11	0.172983
	0.3	14	0.59	0.12504	15	0.29	0.101673	13	0.07	0.151193	6	0.02	0.163395
	0.5	14	0.62	0.112746	15	0.26	0.084299	16	0.09	0.084636	13	0.03	0.124349
	1.0	14	0.51	0.098129	15	0.49	0.067993						
	2.0	15	0.99	0.066126	14	0.01	0.09665						
	∞	15		0.061424	14		0.092348	13		0.116655	10		0.15266
Gowalla	0.05	5	0.93	0.046851	4	0.04	0.048147	6	0.03	0.047906			
[2, 6]	0.1	5	0.98	0.045948	6	0.02	0.044983						
	0.3	5	0.99	0.04394	6	0.01	0.04508						
	0.5	5	0.99	0.043698	6	0.01	0.043858						
	1.0	5	0.99	0.044303	6	0.01	0.043972						
	2.0	5	0.99	0.044324	6	0.01	0.044382						
	∞	5		0.044348	4		0.055141	3		0.077007	7		0.08466

Table 4.4.: Likelihood of the Top-4 Selected k values based on RT-validity over the TIGER and Image datasets.

Dataset	ϵ	Top-1			Top-2			Top-3			Top-4		
k Range		k	Likelihood	Validity	k	Likelihood	Validity	k	Likelihood	Validity	k	Likelihood	Validity
TIGER	0.05	2	0.98	0.013056	3	0.01	0.016129	4	0.01	0.000899			
[2, 4]	0.1	2	1.0	0.01295									
	0.3	2	1.0	0.012226									
	0.5	2	1.0	0.01224									
	1.0	2	1.0	0.012206									
	2.0	2	1.0	0.012214									
	∞	2		0.012178	4		0.240427	3		0.309628			
Image	0.05	2	0.86	0.112783	3	0.14	0.113939						
[2, 4]	0.1	2	0.72	0.108702	3	0.28	0.107116						
	0.3	2	0.78	0.109119	3	0.22	0.1076						
	0.5	2	0.63	0.10737	3	0.37	0.104306						
	1.0	2	0.54	0.107542	3	0.46	0.104778						
	2.0	3	0.54	0.105789	2	0.46	0.108082						
	∞	3		0.105543	2		0.108335	4		0.15741			

Table 4.5.: Likelihood of the Top-4 Selected k values based on RT-validity over the Adult-num and Lifesci datasets.

Dataset	ϵ	Top-1			Top-2			Top-3			Top-4		
k Range		k	Likelihood	Validity	k	Likelihood	Validity	k	Likelihood	Validity	k	Likelihood	Validity
Adult-num	0.05	2	0.92	0.061897	3	0.05	0.088172	5	0.02	0.3036	4	0.01	0.04727
[2, 6]	0.1	2	0.87	0.059629	3	0.08	0.133769	4	0.04	0.072274	6	0.01	0.357226
	0.3	2	0.89	0.062967	3	0.09	0.233581	4	0.01	0.059019	6	0.01	0.367765
	0.5	2	0.84	0.071696	5	0.06	0.396922	3	0.05	0.110833	6	0.03	0.481508
	1.0	2	0.81	0.062337	3	0.07	0.152268	4	0.07	0.339829	6	0.04	0.43002
	2.0	2	0.82	0.072643	3	0.07	0.099015	5	0.05	0.372042	6	0.05	0.479522
	∞	2		0.090091	3		0.436318	6		0.464136	5		0.471094
Lifesci	0.05	7	0.4	0.820997	6	0.32	0.788941	5	0.11	0.762209	4	0.06	0.790328
[2, 7]	0.1	7	0.76	0.839943	6	0.22	0.882122	5	0.01	0.833441	4	0.01	1.044702
	0.3	7	0.68	0.94046	6	0.21	0.936626	5	0.1	0.880875	3	0.01	0.822615
	0.5	7	0.68	0.991935	6	0.23	0.978901	5	0.07	1.019253	4	0.02	1.020965
	1.0	7	0.43	0.96385	6	0.29	0.977027	5	0.21	0.974339	2	0.05	0.350502
	2.0	7	0.41	0.990653	6	0.38	0.998312	5	0.14	0.984321	4	0.05	0.949489
	∞	6		0.979265	5		1.036223	4		1.04051	7		1.055249

5. UNDERSTANDING THE SPARSE VECTOR TECHNIQUE

5.1 Introduction

Differential privacy (DP) is increasingly being considered the privacy notion of choice for privacy-preserving data analysis and publishing in the research literature. In this chapter we study the Sparse Vector Technique (SVT), a basic technique for satisfying DP, which was first proposed by Dwork et al. [23] and later refined in [19] and [17], and used in [68–72]. Compared with other techniques for satisfying DP, SVT has the unique quality that one can output some query answers without apparently paying any privacy cost. More specifically, in SVT one is given a sequence of queries and a certain threshold T, and outputs a vector indicating whether each query answer is above or below T; that is, the output is a vector $\{\bot, \top\}^\ell$, where ℓ is the number of queries answered, $\top$ indicates that the corresponding query answer is above the threshold and $\bot$ indicates below. SVT works by first perturbing the threshold T and then comparing each perturbed individual query answer against the noisy threshold. When one expects that the predominant majority of queries are on one side, e.g., below the threshold, one can use SVT so that while each output of $\top$ (which we call a **positive outcome**) consumes some privacy budget, each output of $\bot$ (**negative outcome**) consumes none. That is, with a fixed privacy budget and a given level of noise added to each query answer, one can keep answering queries as long as the number of $\top$'s does not exceed a pre-defined cutoff point.

This ability to avoid using any privacy budget for queries with negative outcomes is very powerful for **the interactive setting**, where one answers a sequence of queries without knowing ahead of the time what these queries are. Some well-known lower-bound results [5, 20–22] suggest that "one cannot answer a linear, in the database size, number of queries with small noise while preserving privacy" [23]. This limitation can be bypassed using SVT, as in the iterative construction approach in [17, 19, 68]. In this approach, one maintains a history of past queries and answers. For each new query, one first uses this history to derive an answer for the query, and then uses SVT to check whether the error of this derived answer is below a threshold. If it is, then one can use this derived answer for this new query without consuming any privacy budget. Only when the error of this derived answer is above the threshold, would one need to spend privacy budget accessing the database to answer the query.

With the power of SVT come the subtlety of why it is private and the difficulty of applying it correctly. The version of SVT used in [17,68], which was abstracted into a generic technique and described in Roth's 2011 lecture notes [73], turned out to be not differentially private as claimed. This error in [17,68] is arguably not critical because it is possible to use a fixed version of SVT without affecting the main asymptotic results. Since 2014, several variants of SVT were developed; they were used for frequent itemset mining [69], for feature selection in private classification [70], and for publishing high-dimensional data [71]. These usages are in the **non-interactive setting**, where all the queries are known ahead of the time, and the goal is to find c queries that have large answers, e.g., finding the c most frequent itemsets. Unfortunately, these variants do not satisfy DP, as pointed out in [74]. When using a correct version of SVT in these papers, one would get significantly worse accuracy. Since these papers seek to improve the tradeoff between privacy and utility, the results in them are thus invalid.

The fact that many usages of SVT are not private, even when proofs of their privacy were given, is already known [74, 75]; however, we feel that what led to the erroneous proofs were not clearly explained, and such an explanation can help researchers to avoid similar errors in the future. One evidence of the continuing confusion over SVT appears in [74], the first paper that identifies errors in some SVT variants. In [74], the SVT variants in [69–71] were modeled as a generalized private threshold testing algorithm (GPTT), and a proof showing that GPTT does not satisfy ϵ-DP for any finite ϵ (which we use ∞-DP to denote in this chapter) was given. However, as we show in this chapter, the proof in [74] was incorrect. This error was not reported in the literature. One goal of this chapter is to clearly explain why correct usages of SVT is private, and what are the most likely confusions that caused the myriad of incorrect usages of SVT.

A second goal of this chapter is to improve the accuracy of SVT. A version of SVT with a correct privacy proof appeared in Dwork and Roth's 2014 book [76], and was used in some recent work, e.g., [72]. In this chapter, we present a version of SVT that adds less noise for the same level of privacy. In addition, we develop a novel technique that optimizes the privacy budget allocation between that for perturbing the threshold and that for perturbing the query answers, and experimentally demonstrate its effectiveness.

A third goal of this chapter is to point out that usage of SVT can be replaced by the Exponential Mechanism (EM) [14] when used in the non-interactive setting. Most recent usages of SVT in [69–72] are in the non-interactive setting, where the goal is to select up to c queries with the highest answers. In this setting, one could also use the Exponential Mechanism (EM) [14] c times to achieve the same objective, each time selecting the query with the highest answer. Using analysis as well as experiments, we demonstrate that EM outperforms SVT.

In summary, this chapter has the following novel contributions.

1. We propose a new version of SVT that provides better utility. We also introduce an effective technique to improve the performance of SVT. These enhancements achieve better utility than previous SVT algorithms and can be applied to improve utility in the interactive setting.

2. While previous papers have pointed out most of the errors in usages of SVT, we use a detailed privacy proof of SVT to identify the misunderstandings that likely caused the different non-private versions. We also point out a previously unknown error in the proof in [74] of the non-privacy of some SVT variants.

3. Through analysis and experiments on real datasets, we have evaluated the effects of various SVT optimizations and compared them to EM. Our results show that for non-interactive settings, one should use EM instead of SVT.

The rest of the chapter is organized as follows. We analyze six variants of SVT in Section 5.2. In Section 5.3, we present our optimizations of SVT. We compare SVT with the exponential mechanism in Section 5.4. The experimental results are shown in Section 5.5. Related works are summarized in Section 5.6.

5.2 Variants of SVT

Figure 5.1.: An instantiation of the SVT proposed in this chapter

Input: A private database D, a stream of queries $Q = q_1, q_2, \cdots$ each with sensitivity no more than Δ, either a sequence of thresholds $\mathbf{T} = T_1, T_2, \cdots$ or a single threshold T (see footnote *), and c, the maximum number of queries to be answered with $\top$.
Output: A stream of answers $a_1, a_2, \cdots$, where each $a_i \in \{\top, \bot\} \cup \mathbb{R}$ and $\mathbb{R}$ denotes the set of all real numbers.

Algorithm 17 An instantiation of the SVT proposed in this chapter.

```
Input: $D, Q, \Delta, \mathbf{T} = T_1, T_2, \cdots, c$.
$\epsilon_1 = \epsilon/2, \quad \rho = \mathsf{Lap}(\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1$,   count = 0
for each query $q_i \in Q$ do
    $\nu_i = \mathsf{Lap}(2c\Delta/\epsilon_2)$
    if $q_i(D) + \nu_i \geq T_i + \rho$ then
        Output $a_i = \top$
        count = count + 1, Abort if count $\geq c$.
    else
        Output $a_i = \bot$
    end if
end for
```

Figure 5.2.: SVT in Dwork and Roth 2014 [76]

Input: A private database D, a stream of queries $Q = q_1, q_2, \cdots$ each with sensitivity no more than Δ, either a sequence of thresholds $\mathbf{T} = T_1, T_2, \cdots$ or a single threshold T (see footnote *), and c, the maximum number of queries to be answered with $\top$.
Output: A stream of answers $a_1, a_2, \cdots$, where each $a_i \in \{\top, \bot\} \cup \mathbb{R}$ and $\mathbb{R}$ denotes the set of all real numbers.

Algorithm 18 SVT in Dwork and Roth 2014 [76].

```
Input: $D, Q, \Delta, T, c$.
$\epsilon_1 = \epsilon/2, \quad \rho = \mathsf{Lap}\left(c\Delta/\epsilon_1\right)$
$\epsilon_2 = \epsilon - \epsilon_1, \quad$ count = 0
for each query $q_i \in Q$ do
    $\nu_i = \mathsf{Lap}\left(2c\Delta/\epsilon_1\right)$
    if $q_i(D) + \nu_i \geq T + \rho$ then
        Output $a_i = \top$, $\rho = \mathsf{Lap}\left(c\Delta/\epsilon_2\right)$
        count = count + 1, Abort if count $\geq c$.
    else
        Output $a_i = \bot$
    end if
end for
```

Figure 5.3.: SVT in Roth's 2011 Lecture Notes [73]

Input: A private database D, a stream of queries $Q = q_1, q_2, \cdots$ each with sensitivity no more than Δ, either a sequence of thresholds $\mathbf{T} = T_1, T_2, \cdots$ or a single threshold T (see footnote *), and c, the maximum number of queries to be answered with $\top$.
Output: A stream of answers $a_1, a_2, \cdots$, where each $a_i \in \{\top, \bot\} \cup \mathbb{R}$ and $\mathbb{R}$ denotes the set of all real numbers.

Algorithm 19 SVT in Roth's 2011 Lecture Notes [73].

```
**Input:** $D, Q, \Delta, T, c$.
$\epsilon_1 = \epsilon/2, \quad \rho = \mathsf{Lap}\,(\Delta/\epsilon_1)$,
$\epsilon_2 = \epsilon - \epsilon_1, \quad$ count = 0
**for** each query $q_i \in Q$ **do**
    $\nu_i = \mathsf{Lap}\,(c\Delta/\epsilon_2)$
    **if** $q_i(D) + \nu_i \geq T + \rho$ **then**
        Output $a_i = q_i(D) + \nu_i$
        count = count + 1, **Abort** if count $\geq c$.
    **else**
        Output $a_i = \bot$
    **end if**
**end for**
```

Figure 5.4.: SVT in Lee and Clifton 2014 [69]

Input: A private database D, a stream of queries $Q = q_1, q_2, \cdots$ each with sensitivity no more than Δ, either a sequence of thresholds $\mathbf{T} = T_1, T_2, \cdots$ or a single threshold T (see footnote *), and c, the maximum number of queries to be answered with $\top$.
Output: A stream of answers $a_1, a_2, \cdots$, where each $a_i \in \{\top, \bot\} \cup \mathbb{R}$ and $\mathbb{R}$ denotes the set of all real numbers.

Algorithm 20 SVT in Lee and Clifton 2014 [69].

```
Input: $D, Q, \Delta, T, c$.
$\epsilon_1 = \epsilon/4, \quad \rho = \mathsf{Lap}\left(\Delta/\epsilon_1\right)$
$\epsilon_2 = \epsilon - \epsilon_1,$   count = 0
for each query $q_i \in Q$ do
    $\nu_i = \mathsf{Lap}\left(\Delta/\epsilon_2\right)$
    if $q_i(D) + \nu_i \geq T + \rho$ then
        Output $a_i = \top$
        count = count + 1, Abort if count $\geq c$.
    else
        Output $a_i = \bot$
    end if
end for
```

Figure 5.5.: SVT in Stoddard et al. 2014 [70]

Input: A private database D, a stream of queries $Q = q_1, q_2, \cdots$ each with sensitivity no more than Δ, either a sequence of thresholds $\mathbf{T} = T_1, T_2, \cdots$ or a single threshold T (see footnote *), and c, the maximum number of queries to be answered with $\top$.
Output: A stream of answers $a_1, a_2, \cdots$, where each $a_i \in \{\top, \bot\} \cup \mathbb{R}$ and $\mathbb{R}$ denotes the set of all real numbers.

Algorithm 21 SVT in Stoddard et al. 2014 [70].

```
Input: $D, Q, \Delta, T$.
$\epsilon_1 = \epsilon/2, \quad \rho = \mathsf{Lap}\left(\Delta/\epsilon_1\right)$
$\epsilon_2 = \epsilon - \epsilon_1$
for each query $q_i \in Q$ do
    $\nu_i = 0$
    if $q_i(D) + \nu_i \geq T + \rho$ then
        Output $a_i = \top$

    else
        Output $a_i = \bot$
    end if
end for
```

Figure 5.6.: SVT in Chen et al. 2015 [71]

Input: A private database D, a stream of queries $Q = q_1, q_2, \cdots$ each with sensitivity no more than Δ, either a sequence of thresholds $\mathbf{T} = T_1, T_2, \cdots$ or a single threshold T (see footnote *), and c, the maximum number of queries to be answered with $\top$.
Output: A stream of answers $a_1, a_2, \cdots$, where each $a_i \in \{\top, \bot\} \cup \mathbb{R}$ and $\mathbb{R}$ denotes the set of all real numbers.

Algorithm 22 SVT in Chen et al. 2015 [71].

```
Input: $D, Q, \Delta, \mathbf{T} = T_1, T_2, \cdots$.
$\epsilon_1 = \epsilon/2, \quad \rho = \mathsf{Lap}\left(\Delta/\epsilon_1\right)$
$\epsilon_2 = \epsilon - \epsilon_1$
for each query $q_i \in Q$ do
    $\nu_i = \mathsf{Lap}\left(\Delta/\epsilon_2\right)$
    if $q_i(D) + \nu_i \geq T_i + \rho$ then
        Output $a_i = \top$

    else
        Output $a_i = \bot$
    end if
end for
```

	Alg. 17	**Alg. 18**	**Alg. 19**	**Alg. 20**	**Alg. 21**	**Alg. 22**
ϵ_1	$\epsilon/2$	$\epsilon/2$	$\epsilon/2$	$\epsilon/4$	$\epsilon/2$	$\epsilon/2$
Scale of threshold noise ρ	Δ/ϵ_1	$c\Delta/\epsilon_1$	Δ/ϵ_1	Δ/ϵ_1	Δ/ϵ_1	Δ/ϵ_1
Reset ρ after each output of $\top$ **(unnecessary)**		Yes				
Scale of query noise ν_i	$2c\Delta/\epsilon_2$	$2c\Delta/\epsilon_2$	$c\Delta/\epsilon_1$	Δ/ϵ_2	0	Δ/ϵ_2
Outputting $q_i + \nu_i$ instead of $\top$ **(not private)**			Yes			
Outputting unbounded $\top$'s **(not private)**					Yes	Yes
Privacy Property	ϵ-DP	ϵ-DP	∞-DP	$\left(\frac{1+6c}{4}\epsilon\right)$-DP	∞-DP	∞-DP

Figure 5.7.: Differences among Algorithms 17-22.

* Algorithms 17 and 22 use a sequence of thresholds $\mathbf{T} = T_1, T_2, \cdots$, allowing different thresholds for different queries. The other algorithms use the same threshold T for all queries. We point out that this difference is mostly syntactical. In fact, having an SVT where the threshold always equals 0 suffices. Given a sequence of queries $q_1, q_2, \cdots$, and a sequence of thresholds $\mathbf{T} = T_1, T_2, \cdots$, we can define a new sequence of queries $r_i = q_i - T_i$, and apply the SVT to r_i using 0 as the threshold to obtain the same result. In this chapter, we decide to use thresholds to be consistent with the existing papers.

In this section, we analyze variants of SVT; six of them are listed in Figure 5.1 to Figure 5.6. Alg. 17 is an instantiation of our proposed SVT. Alg. 18 is the version taken from [76]. Alg. 19, 20, 21, and 22 are taken from [69–71, 73] respectively.

The table in Figure 5.7 summarizes the differences among these algorithms. Their privacy properties are given in the last row of the table. Alg. 17 and 18 satisfy ϵ-DP, and the rest of them do not. Alg. 19, 21, 22 do not satisfy ϵ-DP for any finite ϵ, which we denote as ∞-DP.

An important input parameter to any SVT algorithm is the number c, i.e., how many positive outcomes one can answer before stopping. This number can be quite large. For example, in privately finding top-c frequent itemsets [69], c ranges from 50 to 400. In using selective stochastic gradient descent to train deep learning model privately [72], the number of gradients to upload at each epoch ranges from 15 to 140,106.

To understand the differences between these variants, one can view SVT as having the following four steps steps:

1. Generate the threshold noise ρ (Line 1 in each algorithm), which will be added to the threshold during comparison between each query and the threshold (line 5). In all except Alg. 18, ρ scales with Δ/ϵ_1. In Alg. 18, however, ρ scales with $c\Delta/\epsilon_1$. This extra factor of c in the noise scale causes Alg. 18 to be much less accurate than Alg. 17. We show that including the factor of c is an effect of Alg. 18's design to resample ρ each time a query results in a positive outcome (Line 6). When keeping ρ unchanged, ρ does not need to scale with c to achieve privacy.

2. For each query q_i, generate noise ν_i to be added to the query (Line 4), which should scale with $2c\Delta/\epsilon_2$. In Alg. 20 and 22, ν_i scales with Δ/ϵ_2. Removing the factor of c from the magnitude of the noise will result in better utility; however, this is done at the cost of being non-private. Alg. 21 adds no noise to q_i at all, and is also non-private.

3. Compare the perturbed query answer with the noisy threshold and output whether it is above or below the threshold (Lines 5, 6, 9). Here Alg. 17 differs in that it

outputs the noisy query answer $q_i(D) + \nu_i$, instead of an indicator $\top$. This makes it non-private.

4. Keep track of the number of $\top$'s in the output, and stop when one has outputted c $\top$'s (Line 7). This step is missed in Alg. 21 and 22. Without this limitation, one can answer as many queries as there are with a fixed accuracy level for each query. If this was to be private, then one obtains privacy kind of "for free".

5.2.1 Privacy Proof for Proposed SVT

We now prove the privacy of Alg. 17. We break down the proof into two steps, to make the proof easier to understand, and, more importantly, to enable us to point out what confusions likely cause the different non-private variants of SVT to be proposed. In the first step, we analyze the situation where the output is $\perp^\ell$, a length-ℓ vector $\langle \perp, \cdots, \perp \rangle$, indicating that all ℓ queries are tested to be below the threshold.

Lemma 4 *Let $\mathcal{A}$ be Alg. 17. For any neighboring datasets D and D', and any integer ℓ, we have*

$$\mathsf{Pr}\left[\mathcal{A}(D) = \perp^\ell\right] \leq e^{\epsilon_1} \mathsf{Pr}\left[\mathcal{A}(D') = \perp^\ell\right].$$

Proof We have

$$\begin{aligned} \mathsf{Pr}\left[\mathcal{A}(D) = \perp^\ell\right] &= \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho = z\right] f_D(z)\, dz, \\ \text{where } \; f_D(z) &= \mathsf{Pr}\left[\mathcal{A}(D) = \perp^\ell \mid \rho = z\right] && (5.1) \\ &= \prod_{i \in \{1,2,\cdots,\ell\}} \mathsf{Pr}\left[q_i(D) + \nu_i < T_i + z\right]. && (5.2) \end{aligned}$$

The probability of outputting $\perp^\ell$ over D is the summation (or integral) of the product of $\mathsf{Pr}\left[\rho = z\right]$, the probability that the threshold noise equals z, and $f_D(z)$, the conditional probability that $\perp^\ell$ is the output on D given that the threshold noise ρ is z. The step from (5.1) to (5.2) is because, given D, $\mathbf{T}$, the queries, and ρ, whether one query results in $\perp$ or not depends completely on the noise ν_i and is independent from whether any other query results in $\perp$.

The key observation underlying the SVT technique is that for any neighboring D, D', we have $f_D(z) \leq f_{D'}(z+\Delta)$. Suppose that we have $q_i(D) = q_i(D') - \Delta$ for each q_i, then the ratio $f_D(z)/f_{D'}(z)$ is unbounded when $|L|$ is unbounded. However, $f_D(z)$ is upper-bounded by the case where the dataset is D' but the noisy threshold **is increased by** Δ, because for any query q_i, $|q_i(D) - q_i(D')| \leq \Delta$. More precisely, we have

$$\begin{aligned}
\mathsf{Pr}\left[q_i(D) + \nu_i < T_i + z\right] &= \mathsf{Pr}\left[\nu_i < T_i - q_i(D) + z\right] \\
&\leq \mathsf{Pr}\left[\nu_i < T_i + \Delta - q_i(D') + z\right] \\
&= \mathsf{Pr}\left[q_i(D') + \nu_i < T_i + (z + \Delta)\right]. \qquad (5.3)
\end{aligned}$$

Because $\rho = \mathsf{Lap}\left(\Delta/\epsilon_1\right)$, by the property of the Laplace distribution, we have:

$$\forall z,\ \mathsf{Pr}\left[\rho = z\right] \leq e^{\epsilon_1}\, \mathsf{Pr}\left[\rho = z + \Delta\right],\ \text{and thus}$$

$$\begin{aligned}
\mathsf{Pr}\left[\mathcal{A}(D) = \perp^\ell\right] &= \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho = z\right]\ f_D(z)\, dz \\
&\leq \int_{-\infty}^{\infty} e^{\epsilon_1} \mathsf{Pr}\left[\rho = z + \Delta\right]\ f_{D'}(z+\Delta)\, dz \\
&= e^{\epsilon_1} \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho = z'\right] f_{D'}(z')\, dz' \qquad \text{let } z' = z + \Delta \\
&= e^{\epsilon_1} \mathsf{Pr}\left[\mathcal{A}(D') = \perp^\ell\right].
\end{aligned}$$

■

We can obtain a similar result when the output is $\top^\ell$ instead of $\bot^\ell$, i.e., $\Pr\left[\mathcal{A}(D)=\top^\ell\right] \le e^{\epsilon_1}\Pr\left[\mathcal{A}(D')=\top^\ell\right]$, because $\Pr\left[\rho=z\right] \le e^{\epsilon_1}\Pr\left[\rho=z-\Delta\right]$ and $g_D(z) \le g_{D'}(z-\Delta)$, where

$$g_D(z) \;=\; \prod_i \Pr\left[q_i(D)+\nu_i \ge T_i+z\right]. \tag{5.4}$$

The fact that this bounding technique works both for positive outputs and negative outputs likely contributes to the misunderstandings behind Alg. 21 and 22, which treat positive and negative outputs exactly the same way. The error is that when the output consists of both $\bot$ and $\top$, one has to choose one side (either positive or negative) to be bounded by the above technique, and cannot do both at the same time.

We also observe that the proof of Lemma 4 will go through if no noise is added to the query answers, i.e., $\nu_i=0$, because Eq (5.3) holds even when $\nu_i=0$. It is likely because of this observation that Alg. 21 adds no noise to query answers. However, when considering outcomes that include both positive answers ($\top$'s) and negative answers ($\bot$'s), one has to add noises to the query answers, as we show below.

Theorem 5.2.1 *Alg. 17 is ϵ-DP.*

Proof Consider any output vector $\boldsymbol{a} \in \{\bot,\top\}^\ell$. Let $\boldsymbol{a}=\langle a_1,\cdots,a_\ell\rangle$, $\mathbf{I}_\top=\{i : a_i=\top\}$, and $\mathbf{I}_\bot=\{i : a_i=\bot\}$. Clearly, $|\mathbf{I}_\top|\le c$. We have

$$\begin{aligned}
\Pr\left[\mathcal{A}(D)=\boldsymbol{a}\right] &= \int_{-\infty}^{\infty} \Pr\left[\rho{=}z\right]\;\; f_D(z)\;\; g_D(z)\,dz, \qquad (5.5)\\
\text{where}\quad f_D(z) &= \prod_{i\in\mathbf{I}_\bot} \Pr\left[q_i(D){+}\nu_i{<}T_i{+}z\right]\\
\text{and}\quad g_D(z) &= \prod_{i\in\mathbf{I}_\top} \Pr\left[q_i(D){+}\nu_i{\ge}T_i{+}z\right].
\end{aligned}$$

The following, together with $\epsilon = \epsilon_1 + \epsilon_2$, prove this theorem:

$$\begin{aligned} \mathsf{Pr}\left[\rho{=}z\right] &\leq e^{\epsilon_1}\mathsf{Pr}\left[\rho{=}z+\Delta\right] \\ f_D(z) &\leq f_{D'}(z+\Delta) \qquad (5.6) \\ g_D(z) &\leq e^{\epsilon_2} g_{D'}(z+\Delta). \qquad (5.7) \end{aligned}$$

Eq. (5.6) deals with all the negative outcomes. Eq. (5.7), which deals with positive outcomes, is ensured by several factors. At most c positive outcomes can occur, $|q_i(D) - q_i(D')| \leq \Delta$, and the threshold for D' is just Δ higher that for D; thus adding noise $\nu_i = \mathsf{Lap}\left(2c\Delta/\epsilon_2\right)$ to each query ensures the desired bound. More precisely,

$$\begin{aligned} g_D(z) &= \prod_{i\in \mathbf{I}_\top} \mathsf{Pr}\left[\nu_i \geq T_i{+}z{-}q_i(D)\right] \\ &\leq \prod_{i\in \mathbf{I}_\top} \mathsf{Pr}\left[\nu_i \geq T_i{+}z - \Delta - q_i(D')\right] \qquad (5.8) \\ &\leq \prod_{i\in \mathbf{I}_\top} e^{\epsilon_2/c}\mathsf{Pr}\left[\nu_i \geq T_i{+}z{-}\Delta{-}q_i(D'){+}2\Delta\right] \qquad (5.9) \\ &\leq e^{\epsilon_2} \prod_{i\in \mathbf{I}_\top} \mathsf{Pr}\left[q_i(D') + \nu_i \geq T_i + z + \Delta\right] \qquad (5.10) \\ &= e^{\epsilon_2} g_{D'}(z+\Delta). \end{aligned}$$

Eq. (5.8) is because $-q_i(D) \geq -\Delta - q_i(D')$, Eq. (5.9) is from the Laplace distribution's property, and Eq. (5.10) is because there are at most c positive outcomes, i.e., $|\mathbf{I}_\top| \leq c$. ■

We observe that while $g_D(z) \leq g_{D'}(z-\Delta)$ is true, replacing (5.7) with it does not help us prove anything, because (5.6) uses $(z{+}\Delta)$ and (5.7) uses $(z{-}\Delta)$, and we cannot change the integration variable in a consistent way.

5.2.2 Privacy Properties of Other Variants

Alg. 18 is taken from the differential privacy book published in 2014 [76]. It satisfies ϵ-DP. It has two differences when compared with Alg. 17. First, ρ follows $\mathsf{Lap}\left(c\Delta/\epsilon_1\right)$ instead of $\mathsf{Lap}\left(\Delta/\epsilon_1\right)$. This causes Alg. 18 to have significantly worse performance than Alg. 17, as we show in Section 5.5. Second, Alg. 18 refreshes the noisy threshold T after each output of $\top$. We note that making the threshold noise scale with c is necessary for privacy *only if* one refreshes the threshold noise after each output of $\top$; however, such refreshing is unnecessary.

Alg. 19 is taken from [73], which in turn was abstracted from the algorithms used in [17, 68]. It has two differences from Alg. 17. First, ν_i follows $\mathsf{Lap}\left(c\Delta/\epsilon_2\right)$ instead of $\mathsf{Lap}\left(2c\Delta/\epsilon_1\right)$; this is not enough for ϵ-DP (even though it suffices for $\frac{3\epsilon}{2}$-DP). Second, it actually outputs the noisy query answer instead of $\top$ for a query above the threshold. This latter fact causes Alg. 19 to be not ϵ'-DP for any finite ϵ'. A proof for this appeared in Appendix A of [75]; The error in the proof for Alg. 19's privacy in [73] occurs in the following steps:

$$
\begin{aligned}
& \mathsf{Pr}\left[\mathcal{A}(D) = \boldsymbol{a}\right] \\
= & \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho{=}z\right] f_D(z) \prod_{i\in\mathbf{I}_\top} \mathsf{Pr}\left[q_i(D){+}\nu_i{\geq}T{+}z \wedge q_i(D){+}\nu_i{=}a_i\right]\, dz \\
= & \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho{=}z\right] f_D(z) \prod_{i\in\mathbf{I}_\top} \mathsf{Pr}\left[q_i(D){+}\nu_i = a_i\right]\, dz \qquad (5.11) \\
\leq & \int_{-\infty}^{\infty} e^{\epsilon_1} \mathsf{Pr}\left[\rho{=}z+\Delta\right] f_{D'}(z+\Delta)\, dz \prod_{i\in\mathbf{I}_\top} e^{\epsilon_2/c} \mathsf{Pr}\left[q_i(D') + \nu_i = a_i\right]
\end{aligned}
$$

The error occurs when going to (5.11), which is implicitly done in [73]. This step removes the condition $q_i(D){+}\nu_i \geq T{+}z$.

Another way to look at this error is that outputting the positive query answers reveals information about the noisy threshold, since the noisy threshold must be below the outputted query answer. Once information about the noisy threshold is leaked, the ability to answer each negative query "for free" disappears.

Alg. 20, taken from [69], differs from Alg. 17 in the following ways. First, it sets ϵ_1 to be $\epsilon/4$ instead of $\epsilon/2$. This has no impact on the privacy. Second, ν_i does not scale with c. As a result, Alg. 20 is only $\left(\frac{1+6c}{4}\right)\epsilon$-DP in general. In [69], Alg. 20 is applied for finding frequent itemsets, where the queries are counting queries and are monotonic. Because of this monotonicity, the usage of Alg. 20 here is $\left(\frac{1+3c}{4}\right)\epsilon$-DP. Theorem 5.3.1 can be applied to Alg. 20 to establish this privacy property; we thus omit the proof of this.

Alg. 22, taken from [71], was motivated by the observation that the proof in [69] can go through without stopping after encountering c positive outcomes, and removed this limitation.

Alg 21, taken from [70], further used the observation that the derivation of Lemma 4 does not depend on the addition of noises, and removed that part as well. The proofs for Alg. 20, 21, 22 in [69–71] roughly use the logic below.

$$\begin{aligned}\int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho{=}z\right] f_D(z) g_D(z) dz &\leq e^{\epsilon} \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho{=}z\right] f_{D'}(z) g_{D'}(z) dz \\ \text{because} \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho{=}z\right] f_D(z)\, dz &\leq e^{\epsilon/2} \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho{=}z\right]\; f_{D'}(z)\, dz \\ \text{and} \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho{=}z\right] g_D(z)\, dz &\leq e^{\epsilon/2} \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho{=}z\right] g_{D'}(z),\end{aligned}$$

This logic incorrectly assumes the following is true:

$$\int_{-\infty}^{\infty} p(z) f(z) g(z) dz = \int_{-\infty}^{\infty} p(z) f(z) dz \int_{-\infty}^{\infty} p(z) g(z) dz$$

A proof that Alg. 22 does not satisfy ϵ-DP for any finite ϵ is given in Appendix B of [75]. While these proofs also apply to Alg. 21, we give a much simpler proof of this below.

Theorem 5.2.2 *Alg. 21 is not ϵ'-DP for any finite ϵ'.*

Proof Consider a simple example, with $T = 0$, $\Delta = 1$, $\mathbf{q} = \langle q_1, q_2 \rangle$ such that $\mathbf{q}(D) = \langle 0, 1 \rangle$ and $\mathbf{q}(D') = \langle 1, 0 \rangle$, and $a = \langle \bot, \top \rangle$. Then by Eq (5.5), we have

$$\begin{aligned}\mathsf{Pr}\left[\mathcal{A}(D) = a\right] &= \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho = z\right] \mathsf{Pr}\left[0 < z\right] \mathsf{Pr}\left[1 \geq z\right] \, dz \\ &= \int_{0}^{1} \mathsf{Pr}\left[\rho = z\right] \, dz > 0,\end{aligned}$$

which is nonzero; and

$$\mathsf{Pr}\left[\mathcal{A}(D') = a\right] = \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho = z'\right] \mathsf{Pr}\left[1 < z'\right] \mathsf{Pr}\left[0 \geq z'\right] \, dz',$$

which is zero. So the probability ratio $\frac{\mathsf{Pr}[\mathcal{A}(D)=a]}{\mathsf{Pr}[\mathcal{A}(D')=a]} = \infty$. ■

5.2.3 Error in Privacy Analysis of GPTT

In [74], the SVT variants in [69–71] were modeled as a generalized private threshold testing algorithm (GPTT). In GPTT, the threshold T is perturbed using $\rho = \mathsf{Lap}\left(\Delta/\epsilon_1\right)$ and each query answer is perturbed using $\mathsf{Lap}\left(\Delta/\epsilon_2\right)$ and there is no cutoff; thus GPTT can be viewed as a generalization of Algorithm 22. When setting $\epsilon_1 = \epsilon_2 = \frac{\epsilon}{2}$, GPTT becomes Alg. 22.

There is a constructive proof in [74] to show that GPTT is not ϵ'-DP for any finite ϵ'. However, this proof is incorrect. This error is quite subtle. We discovered the error only after observing that the technique of the proof can be applied to show that Alg. 17 (which we have proved to be private) to be non-private.

$$\frac{\mathsf{Pr}\left[\mathsf{GPTT}(D) = \boldsymbol{a}\right]}{\mathsf{Pr}\left[\mathsf{GPTT}(D') = \boldsymbol{a}\right]} = \frac{\int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho = z\right] \left(F_{\epsilon_2}(z) - F_{\epsilon_2}(z) F_{\epsilon_2}(z-1)\right)^t dz}{\int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho = z\right] \left(F_{\epsilon_2}(z-1) - F_{\epsilon_2}(z) F_{\epsilon_2}(z-1)\right)^t dz}$$

where $F_\epsilon(x)$ is the cumulative distribution function of $\mathsf{Lap}\left(1/\epsilon\right)$.

The goal of the proof is to show that the above is unbounded as t increases. A key observation is that the ratio of the integrands of the two integrals is always larger than 1, i.e.,

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

For example when $z = 0$, $F_{\epsilon_2}(0) = 1/2$, and $\kappa(0) = \frac{1 - F_{\epsilon_2}(-1)}{F_{\epsilon_2}(-1)}$. However, when $|z|$ goes to ∞, $\kappa(z)$ goes to 1.

The proof tries to limit the integrals to be a finite interval. It denotes $\alpha = \mathsf{Pr}\left[\mathsf{GPTT}(D') = \boldsymbol{a}\right]$. Then choose parameter $\delta = |F_{\epsilon_1}^{-1}(\frac{\alpha}{4})|$ and thus

$$\alpha \leq 2 \int_{-\delta}^{\delta} \mathsf{Pr}\left[\rho = z\right] \left(F_{\epsilon_2}(z-1) - F_{\epsilon_2}(z) F_{\epsilon_2}(z-1)\right)^t dz.$$

Denote the minimum of $\kappa(z)$ in the closed interval $[-\delta, \delta]$ by κ. Then we have $\frac{\mathsf{Pr}[\mathsf{GPTT}(D) = \boldsymbol{a}]}{\mathsf{Pr}[\mathsf{GPTT}(D') = \boldsymbol{a}]} > \frac{\kappa^t}{2}$. They claimed that for any $\epsilon' > 1$ there exists a t to make the above ratio larger than $e^{\epsilon'}$.

The proof is incorrect because of dependency in the parameters. First, α is a function of t; and when t increases, α decreases because the integrand above is positive and decreasing. Second, δ depends on α, and when α decreases, δ increases. Thus when t increases, δ increases. We write δ as $\delta(t)$ to make the dependency on t explicit. Third, κ, the minimum value of $\kappa(z)$ over the interval $[-\delta(t), \delta(t)]$, decreases when t increases. That is, κ is also dependent on t, denoted by $\kappa(t)$, and decreases while t increases. It is not sure that there exists such a t that $\frac{\kappa(t)^t}{2} > e^{\epsilon'}$ for any $\epsilon' > 1$.

To demonstrate that the error in the proof is fundamental, we point out that following the logic of that proof, one can prove that Alg. 17 is not ϵ'-DP for any finite ϵ'. We now show such a "proof" that contradicts Lemma 4. Let $\mathcal{A}$ be Alg. 17, with $c = 1$. Consider an example with $\Delta = 1$, $T = 0$, a sequence $\mathbf{q}$ of t queries such that $\mathbf{q}(D) = 0^t$ and $\mathbf{q}(D') = 1^t$, and output vector $\boldsymbol{a} = \perp^t$. Let

$$\beta = \mathsf{Pr}\left[\mathcal{A}(D) = \perp^\ell\right] = \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho = z\right]\left(F_{\frac{\epsilon}{4}}(z)\right)^t dz$$

$$\alpha = \mathsf{Pr}\left[\mathcal{A}(D') = \perp^\ell\right] = \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho = z\right]\left(F_{\frac{\epsilon}{4}}(z-1)\right)^t dz,$$

$$\text{where} \quad F_{\frac{\epsilon}{4}}(x) \text{ is the cumulative distribution function of } \mathsf{Lap}\left(\frac{4}{\epsilon}\right).$$

Find a parameter δ such that $\int_{-\delta}^{\delta} \mathsf{Pr}\left[\rho = z\right] \; dz \geq 1 - \frac{\alpha}{2}$. Then $\int_{-\delta}^{\delta} \mathsf{Pr}\left[\rho = z\right]\left(F_{\frac{\epsilon}{4}}(z-1)\right)^t dz \geq \frac{\alpha}{2}$. Let κ be the minimum value of $\frac{F_{\frac{\epsilon}{4}}(z)}{F_{\frac{\epsilon}{4}}(z-1)}$ in $[-\delta, \delta]$; it must be that $\kappa > 1$. Then

$$\beta > \int_{-\delta}^{\delta} \mathsf{Pr}\left[\rho = z\right]\left(F_{\frac{\epsilon}{4}}(z)\right)^t dz \geq \int_{-\delta}^{\delta} \mathsf{Pr}\left[\rho = z\right]\left(\kappa F_{\frac{\epsilon}{4}}(z-1)\right)^t dz$$

$$= \kappa^t \int_{-\delta}^{\delta} \mathsf{Pr}\left[\rho = z\right]\left(F_{\frac{\epsilon}{4}}(z-1)\right)^t dz \geq \frac{\kappa^t}{2}\alpha.$$

Since $\kappa > 1$, one can choose a large enough t to make $\frac{\beta}{\alpha} = \frac{\kappa^t}{2}$ to be as large as needed. We note that this contradicts Lemma 4. The contradiction shows that the proof logic used in [74] is incorrect.

5.2.4 Other Variants

Some usages of SVT aim at satisfying (ϵ, δ)-DP [5], instead of ϵ-DP. These often exploit the advanced composition theorem for DP [77], which states that applying k instances of ϵ-DP algorithms satisfies (ϵ', δ')-DP, where $\epsilon' = \sqrt{2k\ln(1/\delta')}\epsilon + k\epsilon(e^\epsilon - 1)$. In this chapter, we limit our attention to SVT variants to those satisfying ϵ-DP, which are what have been used in the data mining community [69–72].

The SVT used in [17, 19] has another difference from Alg. 19. In [17, 19], the goal of using SVT is to determine whether the error of using an answer derived from past queries/answers is below a threshold. This check takes the form of "**if** $|\tilde{q}_i - q_i(D) + \nu_i| \geq$

$T + \rho$ $\mathbf{then}$ output i," where $\tilde{q}_i$ gives the estimated answer of a query obtained using past queries/answers, and $q_i(D)$ gives the true answer. This is incorrect because the noise ν_i should be outside the absolute value sign. In the usage in [17,19], the left hand of the comparison is always ≥ 0; thus whenever the output includes at least one $\top$, one immediately knows that the threshold noise $\rho \geq -T$. This leakage of ρ is somewhat similar to Alg. 19's leakage caused by outputting noisy query answers that are found to be above the noisy threshold. This problem can be fixed by using "$\mathbf{if}$ $|\tilde{q}_i - q_i(D)| + \nu_i \geq T + \rho$ $\mathbf{then}$ output i" instead. By viewing $r_i = |\tilde{q}_i - q_i(D)|$ as the query to be answered; this becomes a standard application of SVT.

5.3 Optimizing SVT

Alg. 17 can be viewed as allocating half of the privacy budget for perturbing the threshold and half for perturbing the query answers. This allocation is somewhat arbitrary, and other allocations are possible. Indeed, Alg. 20 uses a ratio of $1:3$ instead of $1:1$. In this section, we study how to improve SVT by optimizing this allocation ratio and by introducing other techniques.

5.3.1 A Generalized SVT Algorithm

We present a generalized SVT algorithm in Alg. 23, which uses ϵ_1 to perturb the threshold and ϵ_2 to perturb the query answers. Furthermore, to accommodate the situations where one wants the noisy counts for positive queries, we also use ϵ_3 to output query answers using the Laplace mechanism.

We now prove the privacy for Alg. 23; the proof requires only minor changes from the proof of Theorem 5.2.1.

Theorem 5.3.1 *Alg. 23 is* $(\epsilon_1 + \epsilon_2 + \epsilon_3)$*-DP.*

Proof Alg. 23 can be divided into two phases, the first phase outputs a vector to mark which query is above the threshold and the second phase uses the Laplace mechanism to

Algorithm 23 Our Proposed Standard SVT

```
Input: $D, Q, \Delta, \mathbf{T} = T_1, T_2, \cdots, c$ and $\epsilon_1, \epsilon_2$ and $\epsilon_3$.
Output: A stream of answers $a_1, a_2, \cdots$
$\rho = \mathsf{Lap}\left(\frac{\Delta}{\epsilon_1}\right)$, count = 0
for Each query $q_i \in Q$ do
    $\nu_i = \mathsf{Lap}\left(\frac{2c\Delta}{\epsilon_2}\right)$
    if $q_i(D) + \nu_i \geq T_i + \rho$ then
        if $\epsilon_3 > 0$ then
            Output $a_i = q_i(D) + \mathsf{Lap}\left(\frac{c\Delta}{\epsilon_3}\right)$
        else
            Output $a_i = \top$
        end if
        count = count + 1, Abort if count $\geq c$.
    else
        Output $a_i = \bot$
    end if
end for
```

output noisy counts for the queries that are found to be above the threshold in the first phase. Since the second phase is ϵ_3-DP, it suffices to show that the first phase is $(\epsilon_1+\epsilon_2)$-DP. For any output vector $\boldsymbol{a} \in \{\top,\bot\}^\ell$, we want to show

$$\begin{aligned}
\mathsf{Pr}\left[\mathcal{A}(D)=\boldsymbol{a}\right] &= \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho{=}z\right] \;\; f_D(z) \;\; g_D(z)\, dz \\
&\leq \int_{-\infty}^{\infty} e^{\epsilon_1+\epsilon_2}\mathsf{Pr}\left[\rho{=}z+\Delta\right] \;\; f_{D'}(z+\Delta)\; g_{D'}(z+\Delta)\, dz \\
&= e^{\epsilon_1+\epsilon_2}\mathsf{Pr}\left[\mathcal{A}(D')=\boldsymbol{a}\right].
\end{aligned}$$

This holds because, similarly to the proof of Theorem 5.2.1,

$$\begin{aligned}
\mathsf{Pr}\left[\rho{=}z\right] &\leq e^{\epsilon_1}\mathsf{Pr}\left[\rho{=}z+\Delta\right], \\
f_D(z) &= \prod_{i\in\mathbf{I}_\bot} \mathsf{Pr}\left[q_i(D){+}\nu_i{<}T_i{+}z\right] \leq f_{D'}(z+\Delta), \\
g_D(z) &= \prod_{i\in\mathbf{I}_\top} \mathsf{Pr}\left[q_i(D){+}\nu_i{\geq}T_i{+}z\right] \leq e^{\epsilon_2} g_{D'}(z+\Delta).
\end{aligned}$$

■

5.3.2 Optimizing Privacy Budget Allocation

In Alg. 23, one needs to decide how to divide up a total privacy budget ϵ into $\epsilon_1, \epsilon_2, \epsilon_3$. We note that $\epsilon_1+\epsilon_2$ is used for outputting the indicator vector, and ϵ_3 is used for outputting the noisy counts for queries found to be above the threshold; thus the ratio of $(\epsilon_1+\epsilon_2):\epsilon_3$ is determined by the domain needs and should be an input to the algorithm.

On the other hand, the ratio of $\epsilon_1 : \epsilon_2$ affects the accuracy of SVT. Most variants use $1:1$, without a clear justification. To choose a ratio that can be justified, we observe that this ratio affects the accuracy of the following comparison:

$$q_i(D) + \mathsf{Lap}\left(\frac{2c\Delta}{\epsilon_2}\right) \geq T + \mathsf{Lap}\left(\frac{\Delta}{\epsilon_1}\right).$$

To make this comparison as accurate as possible, we want to minimize the variance of $\mathsf{Lap}\left(\frac{\Delta}{\epsilon_1}\right) - \mathsf{Lap}\left(\frac{2c\Delta}{\epsilon_2}\right)$, which is

$$2\left(\frac{\Delta}{\epsilon_1}\right)^2 + 2\left(\frac{2c\Delta}{\epsilon_2}\right)^2,$$

when $\epsilon_1 + \epsilon_2$ is fixed. This is minimized when

$$\epsilon_1 : \epsilon_2 = 1 : (2c)^{2/3}. \tag{5.12}$$

We will evaluate the improvement resulted from this optimization in Section 5.5.

5.3.3 SVT for Monotonic Queries

In some usages of SVT, the queries are monotonic. That is, when changing from D to D', all queries whose answers are different change in the same direction, i.e., there do not exist q_i, q_j such that $(q_i(D) > q_i(D')) \wedge (q_j(D) < q_j(D'))$. That is, we have either $\forall_i\, q_i(D) \geq q_i(D')$, or $\forall_i\, q_i(D') \geq q_i(D)$. This is the case when using SVT for frequent itemset mining in [69] with neighboring datasets defined as adding or removing one tuple. For monotonic queries, adding $\mathsf{Lap}\left(\frac{c\Delta}{\epsilon_2}\right)$ instead of $\mathsf{Lap}\left(\frac{2c\Delta}{\epsilon_2}\right)$ suffices for privacy.

Theorem 5.3.2 *Alg. 23 with* $\nu_i = \mathsf{Lap}\left(\frac{c\Delta}{\epsilon_2}\right)$ *in line 3 satisfies* $(\epsilon_1+\epsilon_2+\epsilon_3)$*-DP when all queries are monotonic.*

Proof Because the second phase of Alg. 23 is still ϵ_3-DP, we just need to show that for any output vector $\boldsymbol{a}$,

$$\begin{aligned}
\mathsf{Pr}\left[\mathcal{A}(D)=\boldsymbol{a}\right] &= \int_{-\infty}^{\infty} \mathsf{Pr}\left[\rho{=}z\right] \;\; f_D(z) \;\; g_D(z)\, dz \\
&\le e^{\epsilon_1+\epsilon_2} \mathsf{Pr}\left[\mathcal{A}(D')=\boldsymbol{a}\right], \\
\text{where } \; f_D(z) &= \prod_{i\in\mathbf{I}_\perp} \mathsf{Pr}\left[q_i(D){+}\nu_i{<}T_i{+}z\right], \\
\text{and } \; g_D(z) &= \prod_{i\in\mathbf{I}_\top} \mathsf{Pr}\left[q_i(D){+}\nu_i{\ge}T_i{+}z\right].
\end{aligned}$$

It suffices to show that either $\mathsf{Pr}\left[\rho{=}z\right] f_D(z)g_D(z) \le e^{\epsilon_1+\epsilon_2}\mathsf{Pr}\left[\rho{=}z\right] f_{D'}(z)g_{D'}(z)$, or $\mathsf{Pr}\left[\rho{=}z\right] f_D(z)g_D(z) \le e^{\epsilon_1+\epsilon_2}\mathsf{Pr}\left[\rho{=}z+\Delta\right] f_{D'}(z+\Delta)g_{D'}(z+\Delta)$.

First consider the case that $q_i(D) \ge q_i(D')$ for any query q_i. In this case, we have

$$\mathsf{Pr}\left[q_i(D)+\nu_i < T_i+z\right] \le \mathsf{Pr}\left[q_i(D')+\nu_i < T_i+z\right],$$

and thus $f_D(z) \le f_{D'}(z)$. Note that $q_i(D)-q_i(D') \le \Delta$. Therefore, $g_D(z) \le e^{\epsilon_2} g_{D'}(z)$, without increasing the noisy threshold by Δ, because $\mathsf{Pr}\left[q_i(D)+\nu_i \ge T_i+z\right] \le \mathsf{Pr}\left[q_i(D')+\nu_i \ge T_i+z-\Delta\right] \le e^{\frac{\epsilon_2}{c}}\mathsf{Pr}\left[q_i(D')+\nu_i \ge T_i+z\right]$ since $\nu_i = \mathsf{Lap}\left(\frac{c\Delta}{\epsilon_2}\right)$.

Then consider the case in which $q_i(D) \le q_i(D')$ for any query q_i. We have the usual

$$\begin{aligned}
f_D(z) &\le f_{D'}(z+\Delta), \\
\text{and } \; \mathsf{Pr}\left[\rho{=}z\right] &\le e^{\epsilon_1}\mathsf{Pr}\left[\rho{=}z+\Delta\right],
\end{aligned}$$

as in previous proofs. With the constraint $q_i(D) \le q_i(D')$, using $\nu_i = \mathsf{Lap}\left(\frac{c\Delta}{\epsilon_2}\right)$ suffices to ensure that $\mathsf{Pr}\left[q_i(D)+\nu_i \ge T_i+z\right] \le e^{\frac{\epsilon_2}{c}}\mathsf{Pr}\left[q_i(D')+\nu_i \ge T_i+\Delta+z\right]$. Thus $g_D(z) \le e^{\epsilon_2} g_{D'}(z+\Delta)$ holds. ■

For monotonic queries, the optimization of privacy budget allocation (5.12) becomes $\epsilon_1 : \epsilon_2 = 1 : c^{2/3}$.

5.4 SVT versus EM

We now discuss the application of SVT in the non-interactive setting, where all the queries are known ahead of the time. We note that most recent usages of SVT, e.g., [38, 69–72], are in the non-interactive setting. Furthermore, these applications of SVT aim at selecting up to c queries with the highest answers. In [69], SVT is applied to find the c most frequent itemsets, where the queries are the supports for the itemsets. In [71], the goal of using SVT is to determine the structure of a Bayesian Network that preserves as much information of the dataset as possible. To this end, they select attribute groups that are highly correlated and create edges for such groups in the network. While the algorithm in [71] takes the form of selecting attribute groups with score above a certain threshold, the real goal is to select the groups with the highest scores. In [72], SVT is used to select parameters to be shared when trying to learn neural-network models in a private fashion. Once selected, noises are added to these parameters before they are shared. The selection step aims at selecting the parameters with the highest scores.

EM or SVT. In non-interactive setting, one can also use the Exponential Mechanism (EM) [14] to achieve the same objective of selecting the top c queries. More specifically, one runs EM c times, each round with privacy budget $\frac{\epsilon}{c}$. The quality for each query is its answer; thus each query is selected with probability proportion to $\exp\left(\frac{\epsilon}{2c\Delta}\right)$ in the general case and to $\exp\left(\frac{\epsilon}{c\Delta}\right)$ in the monotonic case. After one query is selected, it is removed from the pool of candidate queries for the remaining rounds.

An intriguing question is which of SVT and EM offers higher accuracy. Theorem 3.24 in [76] regarding the utility of SVT with $c = \Delta = 1$ states: For any sequence of k queries $f_1, \ldots, f_k$ such that $|\{i < k : f_i(D) \geq T - \alpha\}| = 0$ (i.e. the only query close to being above threshold is possibly the last one), SVT is (α, β) accurate (meaning that with probability at least $1 - \beta$, all queries with answer below $T - \alpha$ result in $\perp$ and all queries with answers above $T - \alpha$ result in $\top$) for: $\alpha_{\mathrm{SVT}} = 8(\log k + \log(2/\beta))/\epsilon$.

In the case where the last query is at least $T + \alpha$, being (α, β)-correct ensures that with probability at least $1 - \beta$, the correct selection is made. For the same setting, we say that

EM is (α, β)*-correct* if given $k-1$ queries with answer $\leq T-\alpha$ and one query with answer $\geq T+\alpha$, the correct selection is made with probability at least $1-\beta$. The probability of selecting the query with answer $\geq T+\alpha$ is at least $\frac{e^{\epsilon(T+\alpha)/2}}{(k-1)e^{\epsilon(T-\alpha)/2}+e^{\epsilon(T+\alpha)/2}}$ by the definition of EM. To ensure this probability is at least $1-\beta$,

$$\alpha_{\text{EM}} = (\log(k-1) + \log((1-\beta)/\beta))/\epsilon,$$

which is less than $1/8$ of the α_{SVT}, which suggests that EM is more accurate than SVT.

The above analysis relies on assuming that the first $k-1$ queries are no more than $T-\alpha$. When that is not assumed, it is difficult analyze the utility of either SVT or EM. Therefore, we will use experimental methods to compare SVT with EM.

SVT with Retraversal. We want to find the most optimized version of SVT to compare with EM, and note that another interesting parameter that one can tune when applying SVT is that of the threshold T. When T is high, the algorithm may select fewer than c queries after traversing all queries. Since roughly each selected query consumes $\frac{1}{c}$'th of the privacy budget, outputting few than c queries kind of "wasted" the remaining privacy budget. When T is low, however, the algorithm may have selected c queries before encountering later queries. No matter how large some of these later query answers are, they cannot be selected.

We observe that in the non-interactive setting, there is a way to deal with this challenge. One can use a higher threshold T, and when the algorithm runs out of queries before finding c above-threshold queries, one can retraverse the list of queries that have not been selected so far, until c queries are selected. However, it is unclear how to select the optimal threshold. In our experiments, we consider SVT-ReTr, which increases the threshold T by different multiples of the scale factor of the Laplace noise injected to each query, and applies the retraversal technique.

5.5 Evaluation

In this section, we experimentally compare the different versions of the SVT algorithm, including our proposed SVT algorithm with different privacy budget allocation methods. We also compare the SVT variants applicable in the non-interactive setting with EM.

Utility Measures. Since the goal of applying SVT or EM is to select the top queries, one standard metric is *False negative rate (FNR)*, i.e., the fraction of true top-c queries that are missed. When an algorithm outputs exactly c results, the FNR is the same as the False Positive Rate, the fraction of incorrectly selected queries.

The FNR metric has some limitations. First, missing the highest query will be penalized the same as missing the c-th one. Second, selecting a query with a very low score will be penalized the same as selecting the $(c+1)$-th query, whose score may be quite close to the c'th query. We thus use another metric that we call Score Error Rate (SER), which measures the ratio of "missed scores" by selecting S instead of the true top c queries, denoted by Top_c.

$$SER = 1.0 - \frac{\text{avgScore}(S)}{\text{avgScore}(\mathsf{Top}_c)}.$$

We present results for both FNR and SER and observe that the correlation between them is quite stable.

Table 5.1.: Dataset characteristics

Dataset	Number of Records	Number of Items
BMS-POS	515,597	1,657
Kosarak	990,002	41,270
AOL	647,377	2,290,685
Zipf	1,000,000	10,000

Datasets. The performance of different algorithms would be affected by the distribution of query scores, we thus want to evaluate the algorithms on several representative distributions. In the experiments, we use the item frequencies in three real datasets: BMS-POS, Kosarak and AOL as representative distributions of query scores. In addition, we also use

Table 5.2.: Summary of algorithms

Settings	Methods	Description
Interactive	SVT-DPBook	DPBook SVT (Alg. 18).
	SVT-S	Standard SVT (Alg. 23).
Non-interactive	SVT-ReTr	Standard SVT with Retraversal.
	EM	Exponential Mechanism.

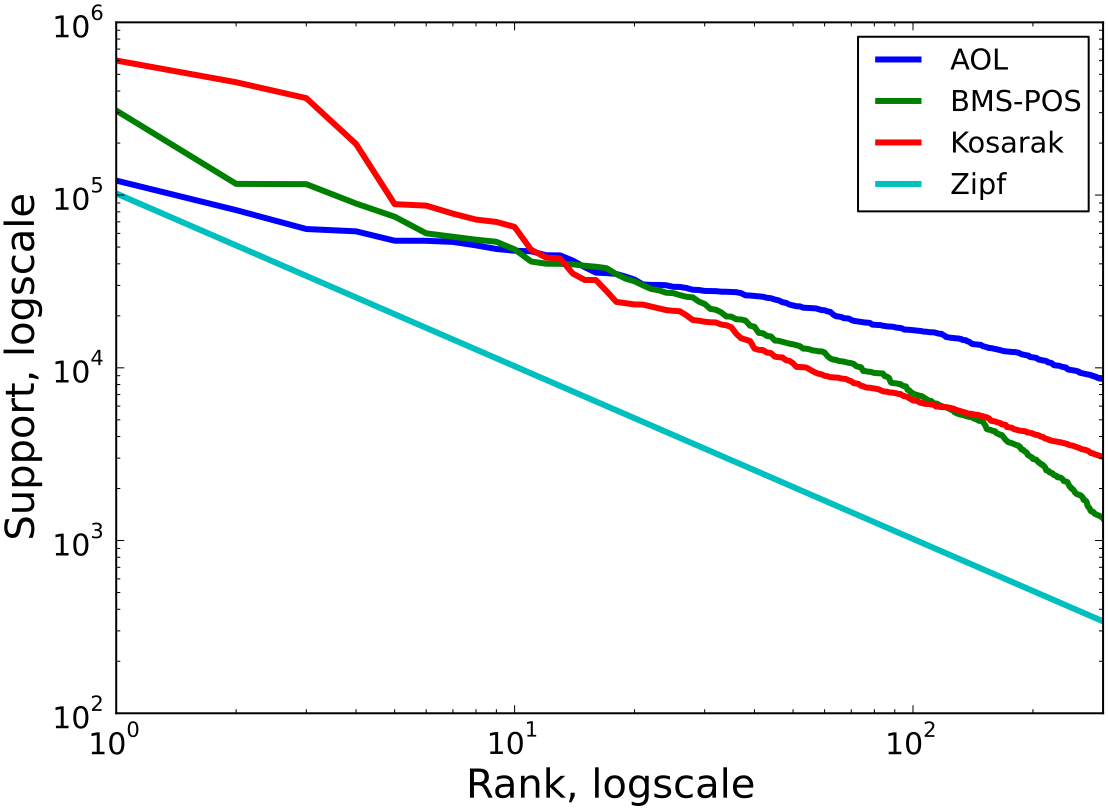


Figure 5.8.: The distribution of 300 highest scores from experiment datasets.

the distribution inspired by the Zipf's law, which states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. Similar phenomenon occurs in many other rankings unrelated to language, such as the population ranks of cities in various countries, corporation sizes, income rankings, ranks of number of people watching the same TV channel, and so on. In this distribution, the i'th query has a score proportional to $\frac{1}{i}$. The characteristics of these datasets are summarized in Table 5.1, and the distribution of the 300 highest scores are shown in Figure 5.8.

Evaluation Setup. We consider the following algorithms. SVT-DPBook is from Dwork and Roth's 2014 book [76] (Algorithm 18). SVT-S is our proposed standard SVT, i.e., Algorithm 23 without numerical outputs ($\epsilon_3 = 0$); and since the count query is monotonic, we use the version for monotonic queries in Section 5.3.3. We consider four privacy budget allocations, 1:1, 1:3, 1:c and 1:$c^{2/3}$, where the last is what our analysis suggests for the monotonic case. These algorithms can be applied in both the interactive and the non-interactive setting.

For the non-interactive setting, we consider EM and SVT-ReTr, which is SVT with the optimizations of increasing the threshold and retraversing through the queries (items) until c of them are selected. We fix the privacy budget allocation to be $1 : c^{2/3}$ and vary the amount we increase the threshold from 1D, 2D, . . ., to 5D, where 1D means adding one standard deviation of the added noises to the threshold.

We vary c from 25 to 300, and each time uses the average score for the c'th query and the $c + 1$'th query as the threshold. We show results for privacy budget $\epsilon = 0.1$ in the paper. We omit results for other ϵ values because of space limitation. We note that varying c have a similar impact of varying ϵ, since the accuracy of each method is mostly affect by $\frac{\epsilon}{c}$; therefore the impact of different ϵ can be seen from different c values. We run each experiment 100 times, each time randomizing the order of items to be examined. We report the average and standard deviation of SER. All algorithms are implemented in Python 2.7 and all the experiments are conducted on an Intel Core i7-3770 3.40GHz PC with 16GB memory.

Results in the Interactive Setting. Figure 5.9, 5.10, 5.11 and 5.12 reports the results for the algorithms that can be applied in the interactive setting. While it is clear that in some settings (such as when $c = 25$) all methods are quite accurate, and in some other settings all methods are very inaccurate (such as when $c \geq 100$ for the Zipf dataset), in the settings in between the two extremes, the differences between these methods are quite large.

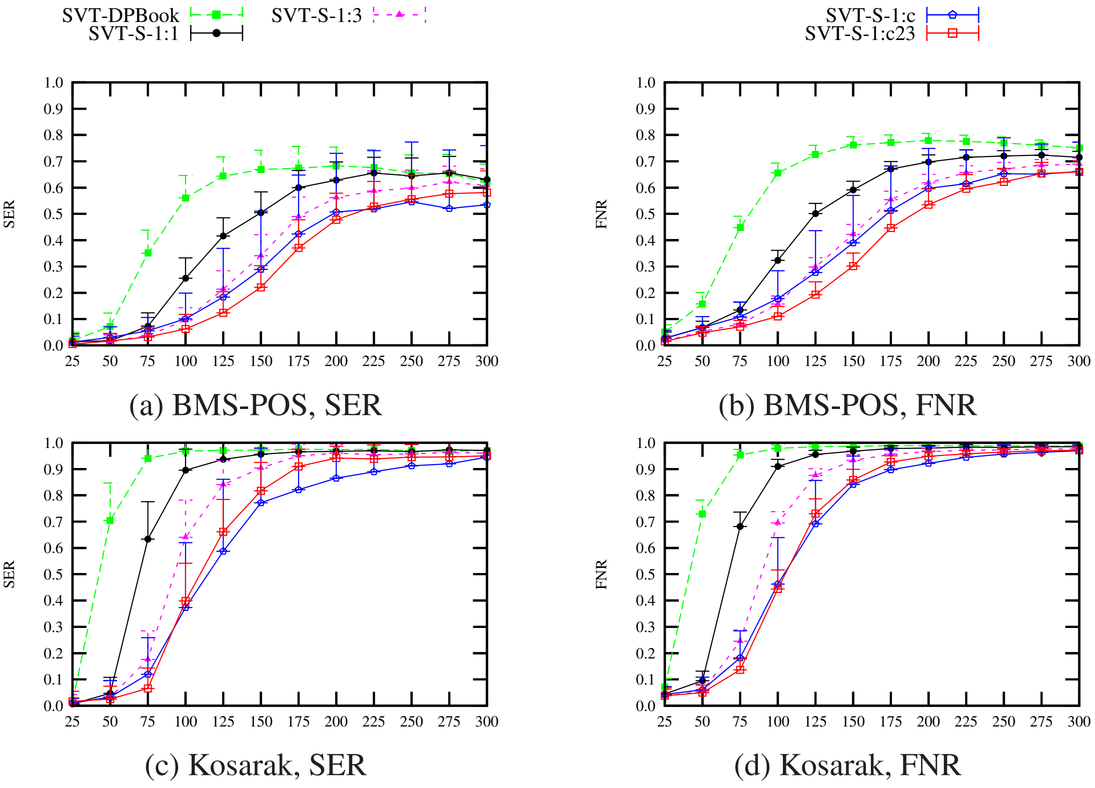
SVT-DPBook performs the worst, followed by SVT-S-1:1, then by SVT-S-1:3, and finally by SVT-S-1:c and SVT-S-1:c23. The differences among these algorithms can be quite pronounced. For example, on the Kosarak dataset, with $\epsilon = 0.1$, $c = 50$, SVT-DPBook's SER is 0.705, which means that the average support of selected items is only around 30% of that for the true top-50 items, which we interpret to mean that the output is meaningless. In contrast, all four SVT-S algorithms have SER less than 0.05, suggesting high accuracy in the selection. SVT-DPBook's poor performance is due to the fact that the threshold is perturbed by a noise with scale as large as $c\Delta/\epsilon$.

For the differences among the four budget allocation approaches, it appears that the performance of $1 : c$ and $1 : c^{\frac{2}{3}}$ are clearly better than the others; and their advantages over the standard $1 : 1$ allocation is quite pronounced. Which of $1 : c$ and $1 : c^{\frac{2}{3}}$ is better is less clear. In general, the former is better for larger c values, where the error is higher, and the latter is better for smaller c values, where the error is lower. Also note that $1 : c$ results in a significantly larger standard deviation. For these reasons, we interpret the results as supporting the recommendation of using 1:$c^{2/3}$ budget allocation.

Results in the Non-interactive Setting. Figure 5.13, 5.14, 5.15 and, 5.16 reports the results for the algorithms that can be applied in the noninteractive setting. We observe that EM clearly performs better than SVT-ReTr-1:c23, which performs essentially the same as SVT-S-1:c23, which is the best algorithm for the interactive case, and is already much better than SVT algorithms used in the literature. For example, for the AOL dataset with $c = 150$, EM's SER is 0.15, while SVT-S with $1 : c^{2/3}$ allocation has SER of 0.59, and SVT-S with $1 : 1$ allocation has SER of 0.99.

It is interesting to see that increasing the threshold can significantly improve the accuracy of SVT with Retraversal. However, the best threshold increment value depends on the dataset and the number of items to be selected. For example, 5D works well for Zipf, and for Kosarak and AOL when c is large, but works not as well for BMS and for Kosarak and AOL when c is small. Since it is unclear how to select the best threshold increment value, and even with the best threshold increment, SVT-ReTr performs no better than EM, our experiments suggest that usage of SVT should be replaced by EM in the non-interactive setting.

Table 5.3, 5.4, 5.5,5.6 gives the numerical values for SER.



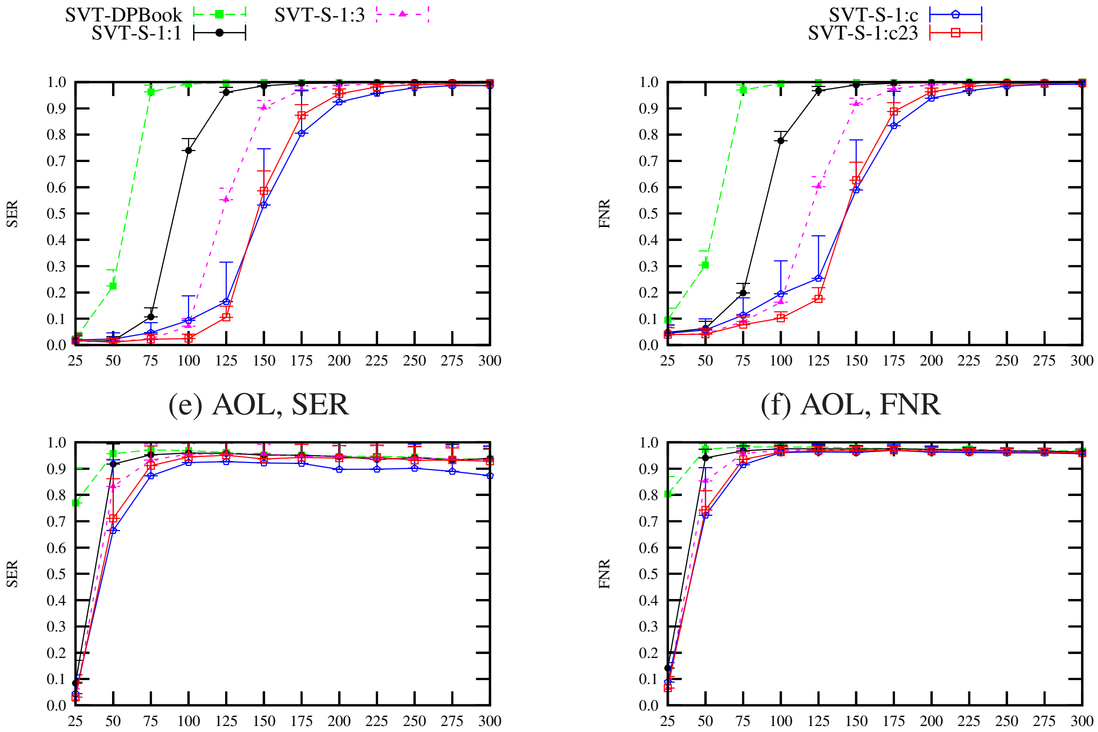
(a) BMS-POS, SER

(b) BMS-POS, FNR

(c) Kosarak, SER

(d) Kosarak, FNR

Figure 5.9.: Comparison of interactive approaches: SVT-DPBook and SVT-S with different budget allocation, $\epsilon = 0.1$, BMS-POS and Kosarak datasets. x-axis: top-c



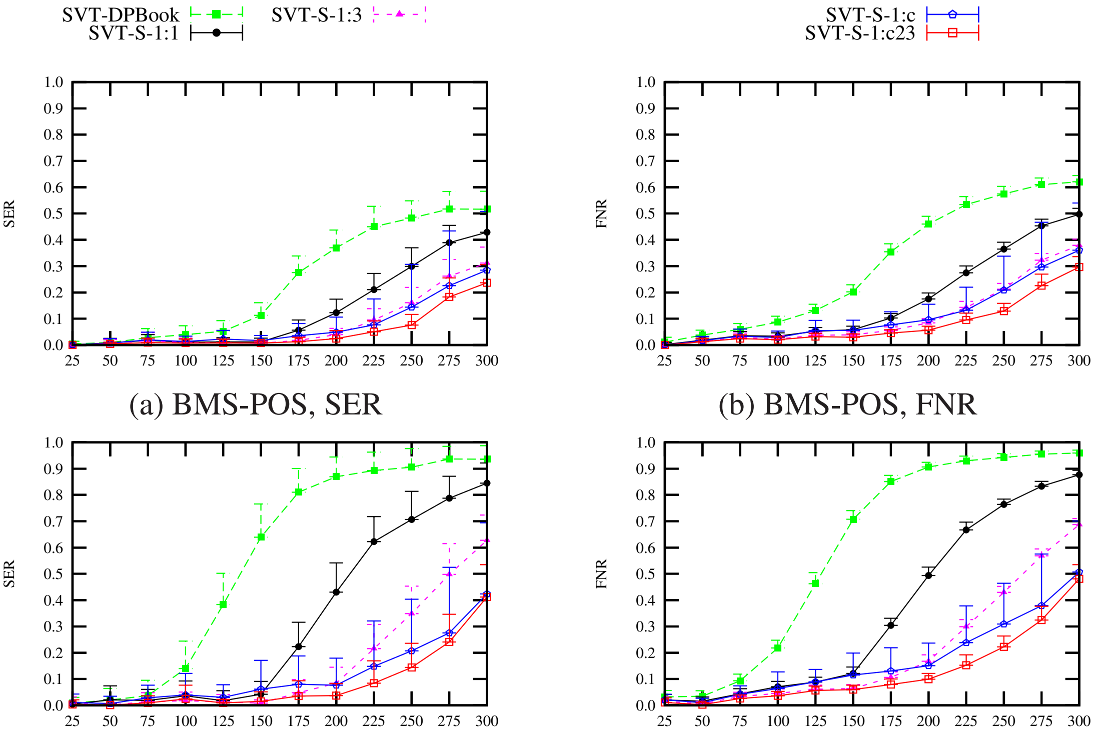
(e) AOL, SER

(f) AOL, FNR

(g) Zipf Synthe, SER

(h) Zipf Synthe, FNR

Figure 5.10.: Comparison of interactive approaches: SVT-DPBook and SVT-S with different budget allocation, $\epsilon = 0.1$, AOL and Zipf datasets. x-axis: top-c



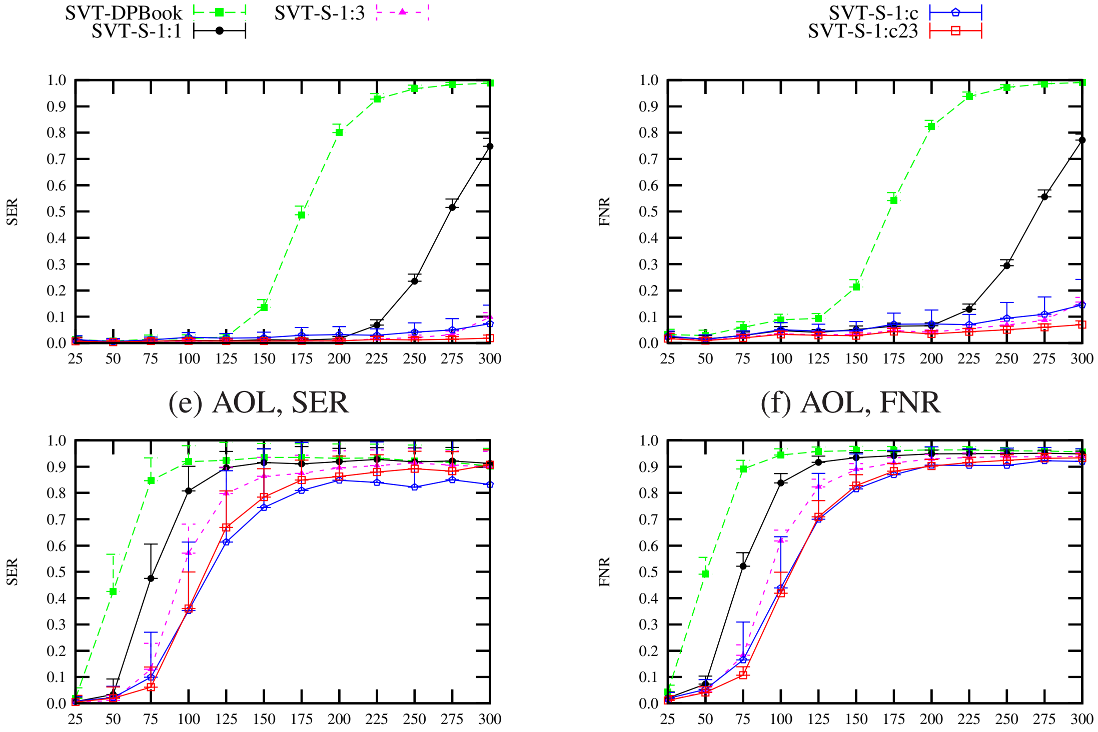
(a) BMS-POS, SER

(b) BMS-POS, FNR

(c) Kosarak, SER

(d) Kosarak, FNR

Figure 5.11.: Comparison of interactive approaches: SVT-DPBook and SVT-S with different budget allocation, $\epsilon = 0.5$, BMS-POS and Kosarak datasets. x-axis: top-c



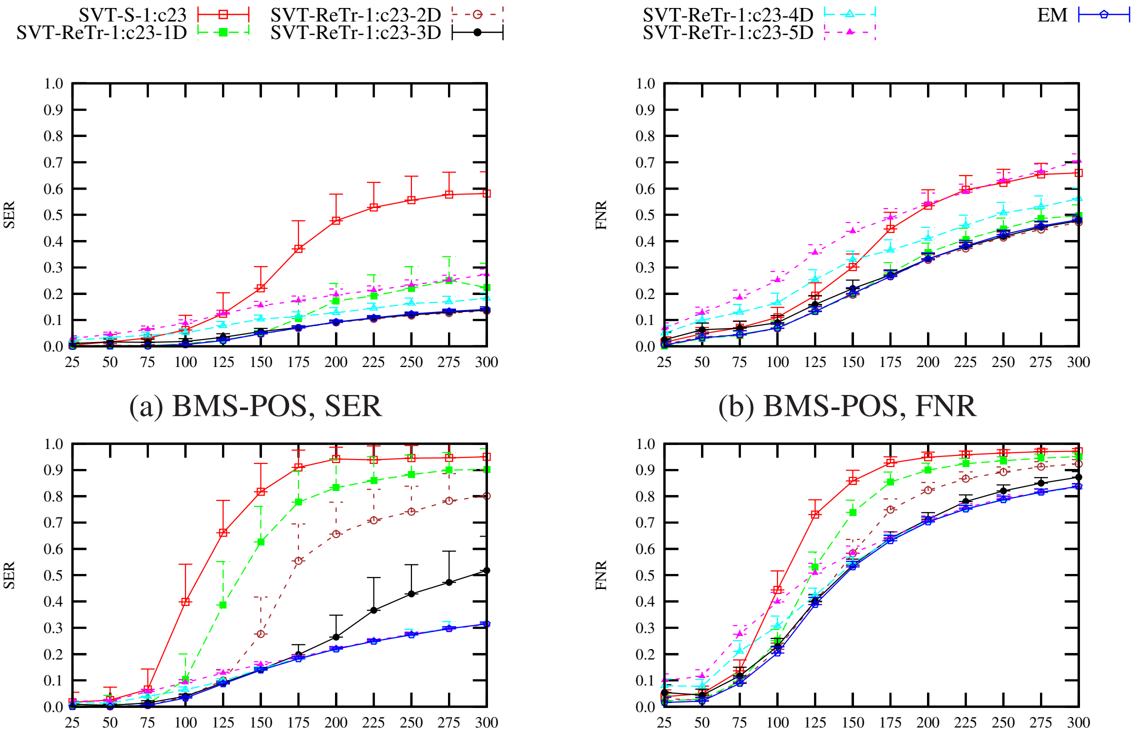
(e) AOL, SER

(f) AOL, FNR

(g) Zipf Synthe, SER

(h) Zipf Synthe, FNR

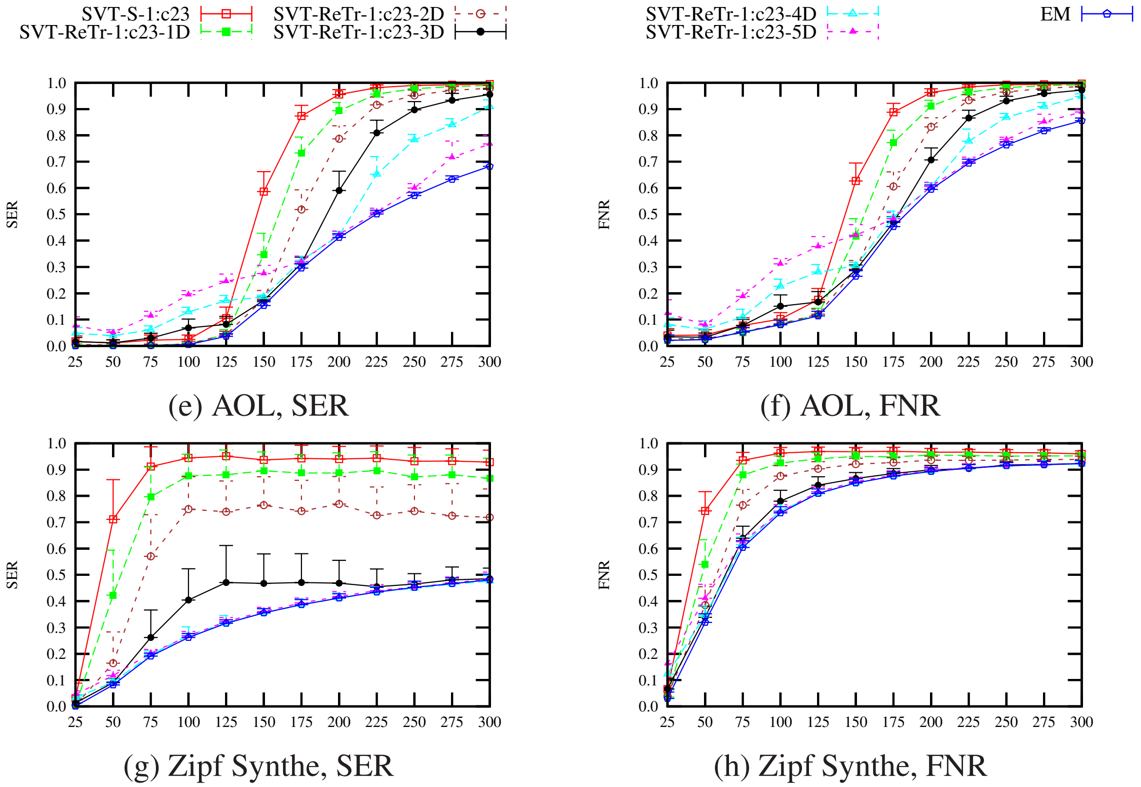
Figure 5.12.: Comparison of interactive approaches: SVT-DPBook and SVT-S with different budget allocation, $\epsilon = 0.5$, AOL and Zipf datasets. x-axis: top-c



(a) BMS-POS, SER

(b) BMS-POS, FNR

Figure 5.13.: Comparison of non-interactive approaches: EM and SVT-ReTr with different thresholds, $\epsilon = 0.1$, BMS-POS and Kosarak datasets. x-axis: top-c



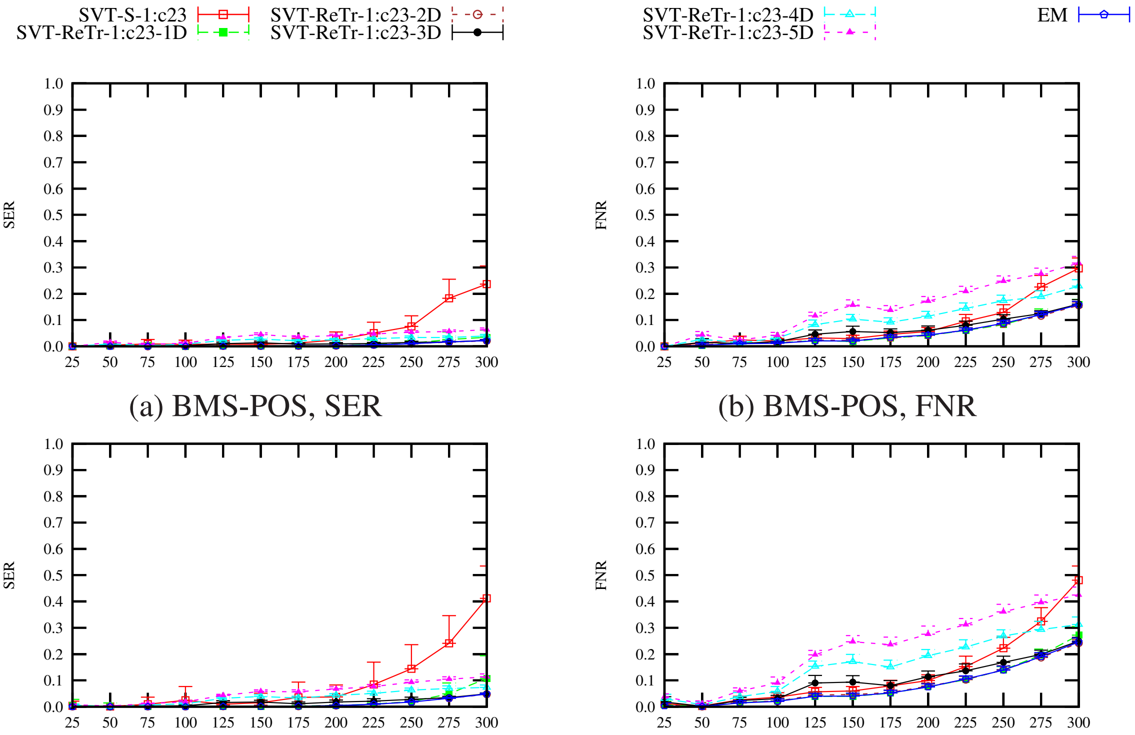
(e) AOL, SER

(f) AOL, FNR

(g) Zipf Synthe, SER

(h) Zipf Synthe, FNR

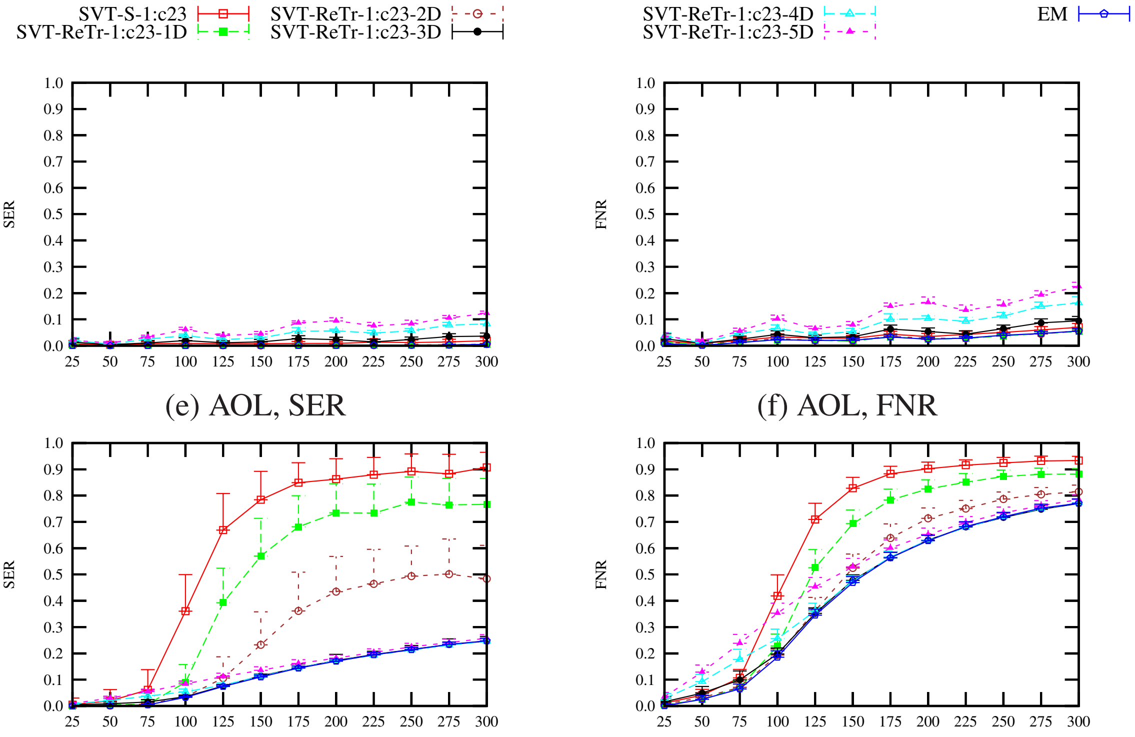
Figure 5.14.: Comparison of non-interactive approaches: EM and SVT-ReTr with different thresholds, $\epsilon = 0.1$, AOL and Zipf datasets. x-axis: top-c



(a) BMS-POS, SER

(b) BMS-POS, FNR

Figure 5.15.: Comparison of non-interactive approaches: EM and SVT-ReTr with different thresholds, $\epsilon = 0.5$, BMS-POS and Kosarak datasets. x-axis: top-c



(e) AOL, SER

(f) AOL, FNR

(g) Zipf Synthe, SER

(h) Zipf Synthe, FNR

Figure 5.16.: Comparison of non-interactive approaches: EM and SVT-ReTr with different thresholds, $\epsilon = 0.5$, AOL and Zipf datasets. x-axis: top-c

Table 5.3.: Comparison of SVT-DPBook, SVT-S, SVT-ReTr and EM on selecting top-c queries in terms of SER when $\epsilon = 0.1$ on datasets BMS-POS and Kosarak. For each row, the best SER value in the non-interactive setting is marked by italics and the best SER value in the interactive setting is marked by boldface. Each cell gives the average value of SER with standard deviation.

Dataset Budget	c	Interactive Setting					Non-interactive Setting			
		SVT-DPBook	SVT-S				SVT-ReTr-1 : $c^{2/3}$			EM
			1 : 1	1 : 3	1 : c	1 : $c^{2/3}$	1D	2D	3D	
BMS-POS $\epsilon = 0.1$	50	$0.072_{0.051}$	$0.017_{0.027}$	$\mathbf{0.016_{0.025}}$	$0.030_{0.040}$	$0.017_{0.029}$	$\mathit{0.001_{0.001}}$	$0.004_{0.007}$	$0.017_{0.010}$	$\mathit{0.001_{0.001}}$
	100	$0.561_{0.085}$	$0.255_{0.078}$	$0.096_{0.047}$	$0.101_{0.097}$	$\mathbf{0.063_{0.055}}$	$0.010_{0.019}$	$\mathit{0.007_{0.003}}$	$0.018_{0.011}$	$\mathit{0.007_{0.003}}$
	150	$0.669_{0.073}$	$0.505_{0.079}$	$0.341_{0.080}$	$0.289_{0.221}$	$\mathbf{0.220_{0.083}}$	$0.049_{0.012}$	$\mathit{0.047_{0.007}}$	$0.056_{0.013}$	$0.048_{0.006}$
	200	$0.682_{0.071}$	$0.628_{0.069}$	$0.562_{0.070}$	$0.508_{0.223}$	$\mathbf{0.478_{0.101}}$	$0.173_{0.067}$	$\mathit{0.091_{0.007}}$	$0.092_{0.007}$	$0.092_{0.008}$
	300	$0.621_{0.069}$	$0.631_{0.065}$	$0.607_{0.064}$	$\mathbf{0.535_{0.225}}$	$0.581_{0.083}$	$0.224_{0.092}$	$\mathit{0.134_{0.008}}$	$0.138_{0.007}$	$0.141_{0.009}$
Kosarak $\epsilon = 0.1$	50	$0.705_{0.142}$	$0.047_{0.061}$	$0.032_{0.062}$	$0.033_{0.063}$	$\mathbf{0.025_{0.049}}$	$0.009_{0.034}$	$0.001_{0.001}$	$0.005_{0.005}$	$\mathit{0.000_{0.000}}$
	100	$0.968_{0.043}$	$0.896_{0.080}$	$0.640_{0.142}$	$\mathbf{0.373_{0.247}}$	$0.398_{0.144}$	$0.103_{0.098}$	$0.038_{0.009}$	$0.040_{0.008}$	$\mathit{0.033_{0.005}}$
	150	$0.972_{0.037}$	$0.956_{0.046}$	$0.904_{0.075}$	$\mathbf{0.772_{0.208}}$	$0.818_{0.108}$	$0.627_{0.134}$	$0.276_{0.141}$	$0.141_{0.008}$	$\mathit{0.139_{0.007}}$
	200	$0.974_{0.037}$	$0.967_{0.044}$	$0.960_{0.034}$	$\mathbf{0.865_{0.177}}$	$0.942_{0.045}$	$0.833_{0.107}$	$0.656_{0.121}$	$0.265_{0.083}$	$\mathit{0.219_{0.008}}$
	300	$0.971_{0.040}$	$0.971_{0.034}$	$0.961_{0.039}$	$\mathbf{0.945_{0.065}}$	$0.950_{0.052}$	$0.902_{0.079}$	$0.801_{0.099}$	$0.518_{0.130}$	$\mathit{0.315_{0.007}}$

Table 5.4.: Comparison of SVT-DPBook, SVT-S, SVT-ReTr and EM on selecting top-c queries in terms of SER when $\epsilon = 0.1$ on datasets AOL and Zipf. For each row, the best SER value in the non-interactive setting is marked by italics and the best SER value in the interactive setting is marked by boldface. Each cell gives the average value of SER with standard deviation.

Dataset Budget	c	Interactive Setting					Non-interactive Setting			
		SVT-	SVT-S				SVT-ReTr-1 : $c^{2/3}$			EM
		DPBook	$1:1$	$1:3$	$1:c$	$1:c^{2/3}$	1D	2D	3D	
AOL $\epsilon = 0.1$	50	$0.225_{0.062}$	$0.016_{0.016}$	$0.013_{0.015}$	$0.023_{0.023}$	$\mathbf{0.011_{0.013}}$	$0.002_{0.005}$	$0.003_{0.007}$	$0.012_{0.011}$	$\mathit{0.001_{0.001}}$
	100	$0.993_{0.008}$	$0.739_{0.046}$	$0.072_{0.029}$	$0.093_{0.093}$	$\mathbf{0.025_{0.017}}$	$0.007_{0.006}$	$0.007_{0.010}$	$0.068_{0.034}$	$\mathit{0.005_{0.002}}$
	150	$0.998_{0.003}$	$0.986_{0.009}$	$0.902_{0.028}$	$\mathbf{0.532_{0.215}}$	$0.587_{0.075}$	$0.346_{0.082}$	$0.178_{0.033}$	$0.173_{0.014}$	$\mathit{0.153_{0.015}}$
	200	$0.999_{0.002}$	$0.996_{0.005}$	$0.987_{0.009}$	$\mathbf{0.924_{0.090}}$	$0.955_{0.019}$	$0.894_{0.031}$	$0.787_{0.050}$	$0.591_{0.073}$	$\mathit{0.412_{0.014}}$
	300	$0.999_{0.002}$	$0.998_{0.003}$	$0.997_{0.003}$	$\mathbf{0.986_{0.033}}$	$0.995_{0.005}$	$0.990_{0.006}$	$0.979_{0.011}$	$0.955_{0.020}$	$\mathit{0.683_{0.014}}$
Zipf $\epsilon = 0.1$	50	$0.957_{0.054}$	$0.917_{0.076}$	$0.832_{0.103}$	$0.665_{0.269}$	$0.711_{0.151}$	$0.423_{0.171}$	$0.164_{0.118}$	$0.090_{0.016}$	$\mathit{0.082_{0.011}}$
	100	$0.969_{0.029}$	$0.960_{0.053}$	$0.955_{0.041}$	$0.924_{0.116}$	$0.944_{0.057}$	$0.876_{0.083}$	$0.750_{0.123}$	$0.404_{0.118}$	$\mathit{0.262_{0.015}}$
	150	$0.952_{0.053}$	$0.952_{0.050}$	$0.958_{0.035}$	$0.922_{0.094}$	$0.936_{0.067}$	$0.896_{0.072}$	$0.765_{0.108}$	$0.468_{0.112}$	$\mathit{0.355_{0.017}}$
	200	$0.947_{0.049}$	$0.946_{0.052}$	$0.948_{0.038}$	$0.898_{0.128}$	$0.940_{0.047}$	$0.887_{0.077}$	$0.769_{0.105}$	$0.468_{0.087}$	$\mathit{0.412_{0.016}}$
	300	$0.937_{0.048}$	$0.939_{0.038}$	$0.933_{0.050}$	$0.874_{0.113}$	$0.928_{0.045}$	$0.867_{0.077}$	$0.719_{0.108}$	$0.485_{0.040}$	$\mathit{0.481_{0.020}}$

Table 5.5.: Comparison of SVT-DPBook, SVT-S, SVT-ReTr and EM on selecting top-c queries in terms of SER when $\epsilon = 0.5$ on datasets BMS-POS and Kosarak. For each row, the best SER value in the non-interactive setting is marked by italics and the best SER value in the interactive setting is marked by boldface. Each cell gives the average value of SER with standard deviation.

Dataset Budget	c	Interactive Setting					Non-interactive Setting			
		SVT-DPBook	SVT-S				SVT-ReTr-1 : $c^{2/3}$			EM
			1 : 1	1 : 3	1 : c	1 : $c^{2/3}$	1D	2D	3D	
BMS-POS $\epsilon = 0.5$	50	$0.009_{0.011}$	$0.008_{0.017}$	$0.006_{0.016}$	$0.007_{0.017}$	$\mathbf{0.004_{0.006}}$	$\mathit{0.000_{0.002}}$	$0.002_{0.003}$	$0.005_{0.005}$	$\mathit{0.000_{0.000}}$
	100	$0.040_{0.034}$	$0.011_{0.012}$	$0.009_{0.017}$	$0.013_{0.021}$	$\mathbf{0.008_{0.016}}$	$\mathit{0.000_{0.001}}$	$0.000_{0.001}$	$0.003_{0.003}$	$\mathit{0.000_{0.000}}$
	150	$0.113_{0.047}$	$0.012_{0.014}$	$\mathbf{0.007_{0.005}}$	$0.017_{0.019}$	$0.008_{0.008}$	$\mathit{0.001_{0.001}}$	$0.002_{0.002}$	$0.014_{0.006}$	$\mathit{0.001_{0.000}}$
	200	$0.369_{0.067}$	$0.123_{0.051}$	$0.040_{0.023}$	$0.048_{0.057}$	$\mathbf{0.024_{0.031}}$	$\mathit{0.003_{0.001}}$	$\mathit{0.003_{0.001}}$	$0.009_{0.005}$	$\mathit{0.003_{0.001}}$
	300	$0.516_{0.068}$	$0.429_{0.068}$	$0.312_{0.061}$	$0.284_{0.224}$	$\mathbf{0.237_{0.069}}$	$0.034_{0.028}$	$\mathit{0.021_{0.002}}$	$0.022_{0.003}$	$0.022_{0.002}$
Kosarak $\epsilon = 0.5$	50	$0.020_{0.044}$	$0.021_{0.053}$	$0.002_{0.005}$	$0.006_{0.028}$	$\mathbf{0.001_{0.003}}$	$0.002_{0.012}$	$\mathit{0.000_{0.001}}$	$\mathit{0.000_{0.001}}$	$\mathit{0.000_{0.000}}$
	100	$0.140_{0.104}$	$0.036_{0.056}$	$\mathbf{0.018_{0.032}}$	$0.039_{0.082}$	$0.025_{0.052}$	$0.002_{0.018}$	$\mathit{0.000_{0.001}}$	$0.004_{0.003}$	$\mathit{0.000_{0.000}}$
	150	$0.640_{0.126}$	$0.042_{0.050}$	$\mathbf{0.008_{0.013}}$	$0.062_{0.109}$	$0.015_{0.036}$	$\mathit{0.001_{0.000}}$	$0.003_{0.003}$	$0.018_{0.007}$	$\mathit{0.001_{0.000}}$
	200	$0.869_{0.075}$	$0.430_{0.112}$	$0.085_{0.061}$	$0.077_{0.102}$	$\mathbf{0.037_{0.045}}$	$\mathit{0.004_{0.003}}$	$\mathit{0.004_{0.002}}$	$0.018_{0.007}$	$\mathit{0.004_{0.001}}$
	300	$0.936_{0.051}$	$0.845_{0.077}$	$0.628_{0.095}$	$0.423_{0.270}$	$\mathbf{0.412_{0.123}}$	$0.108_{0.086}$	$\mathit{0.047_{0.004}}$	$0.049_{0.003}$	$0.048_{0.004}$

Table 5.6.: Comparison of SVT-DPBook, SVT-S, SVT-ReTr and EM on selecting top-c queries in terms of SER when $\epsilon = 0.5$ on datasets AOL and Zipf. For each row, the best SER value in the non-interactive setting is marked by italics and the best SER value in the interactive setting is marked by boldface. Each cell gives the average value of SER with standard deviation.

Dataset Budget	c	Interactive Setting					Non-interactive Setting			
		SVT-DPBook	SVT-S				SVT-ReTr-1 : $c^{2/3}$			EM
			1 : 1	1 : 3	1 : c	1 : $c^{2/3}$	1D	2D	3D	
AOL $\epsilon = 0.5$	50	$0.008_{0.010}$	$0.005_{0.009}$	$0.005_{0.009}$	$0.006_{0.011}$	$\mathbf{0.004_{0.005}}$	$0.001_{0.004}$	$0.002_{0.004}$	$0.004_{0.006}$	$\mathit{0.000_{0.000}}$
	100	$0.017_{0.015}$	$0.010_{0.009}$	$\mathbf{0.007_{0.007}}$	$0.022_{0.018}$	$0.009_{0.009}$	$\mathit{0.000_{0.001}}$	$0.003_{0.005}$	$0.020_{0.010}$	$\mathit{0.000_{0.000}}$
	150	$0.136_{0.029}$	$0.013_{0.010}$	$0.008_{0.008}$	$0.020_{0.021}$	$\mathbf{0.007_{0.007}}$	$\mathit{0.000_{0.000}}$	$0.001_{0.002}$	$0.014_{0.009}$	$\mathit{0.000_{0.000}}$
	200	$0.800_{0.032}$	$0.016_{0.009}$	$\mathbf{0.008_{0.005}}$	$0.032_{0.031}$	$\mathbf{0.008_{0.007}}$	$\mathit{0.001_{0.000}}$	$0.003_{0.004}$	$0.023_{0.010}$	$\mathit{0.001_{0.000}}$
	300	$0.988_{0.007}$	$0.748_{0.031}$	$0.098_{0.017}$	$0.074_{0.070}$	$\mathbf{0.018_{0.012}}$	$\mathit{0.005_{0.002}}$	$0.006_{0.003}$	$0.036_{0.012}$	$\mathit{0.005_{0.001}}$
Zipf $\epsilon = 0.5$	50	$0.426_{0.142}$	$0.034_{0.059}$	$0.011_{0.019}$	$0.022_{0.042}$	$0.020_{0.043}$	$0.004_{0.032}$	$0.002_{0.003}$	$0.009_{0.007}$	$\mathit{0.000_{0.000}}$
	100	$0.919_{0.061}$	$0.808_{0.093}$	$0.571_{0.111}$	$0.353_{0.260}$	$0.361_{0.138}$	$0.090_{0.068}$	$0.035_{0.006}$	$0.036_{0.006}$	$\mathit{0.033_{0.005}}$
	150	$0.935_{0.053}$	$0.916_{0.052}$	$0.864_{0.072}$	$0.745_{0.224}$	$0.784_{0.108}$	$0.570_{0.143}$	$0.232_{0.126}$	$0.114_{0.007}$	$\mathit{0.111_{0.007}}$
	200	$0.932_{0.054}$	$0.919_{0.050}$	$0.896_{0.065}$	$0.848_{0.156}$	$0.863_{0.077}$	$0.734_{0.111}$	$0.435_{0.134}$	$0.173_{0.024}$	$\mathit{0.170_{0.009}}$
	300	$0.903_{0.067}$	$0.912_{0.052}$	$0.899_{0.059}$	$0.832_{0.178}$	$0.907_{0.058}$	$0.767_{0.098}$	$0.484_{0.127}$	$\mathit{0.247_{0.018}}$	$\mathit{0.247_{0.011}}$

5.6 Related Work

SVT was introduced by Dwork et al. [23], and improved by Roth and Roughgarden [19] and by Hardt and Rothblum [17]. These usages are in an interactive setting. An early description of SVT as a stand-alone technique appeared in Roth's 2011 lecture notes [73], which is Alg. 19 in this chapter, and is in fact ∞-DP. The algorithms in [17, 19] also has another difference, as discussed in Section 5.2.4. Another version of SVT appeared in the 2014 book [76], which is Alg. 18. This version is used in some papers, e.g., [72]. We show that it is possible to add less noise and obtain higher accuracy for the same privacy parameter.

Lee and Clifton [69] used a variant of SVT (see Algorithm 20) to find itemsets whose support is above the threshold. Stoddard et al. [70] proposed another variant (see Algorithm 21) for private feature selection for classification to pick out the set of features with scores greater than the perturbed threshold. Chen et al. [71] employed yet another variant of SVT (see Algorithm 22) to return attribute pairs with mutual information greater than the corresponding noisy threshold. These usages are not private. Some of these errors were pointed in [74], in which a generalized private threshold testing algorithm (GPTT) that attempts to model the SVT variants in [69–71] was introduced. The authors showed that GPTT did not satisfy ϵ'-DP for any finite ϵ'. But there is an error in the proof, as shown in Section 5.2.3. Independent from our work, Zhang et al. [75] presented two proofs that the variant of SVT violates DP without discussing the cause of the errors. Also presented in [75] is a special case of our proposed Alg. 1 for counting queries. To our knowledge, the general version of our improved SVT (Alg. 17 and Alg. 23), the techniques of optimizing budget allocation, the technique of using re-traversal to improve SVT, and the comparison of SVT and EM are new in our work.

6. SUMMARY

In this dissertation, we considered the problem of differentially private data publishing for data analysis.

First, we have introduced PrivPfC, a novel framework for publishing data for classification under differential privacy. As a core part of PrivPfC, we have introduced a novel quality function that enables the selection of a good "grid" for publishing noisy histograms. We have also introduced a new technique for privately selecting of most relevant features for classification, which enables PrivPfC to scale to higher-dimension datasets. We have conducted extensive experiments on four real datasets, and the results show that our approach greatly outperforms several other state-of-the-art methods for private data publishing as well as private classification.

Second, we have improved the state-of-the-art on differentially private k-means clustering in several ways. We have introduced the EUGkM approach and improved the DPLloyd method based on a systemized error analysis. Extensive analysis and experimental comparison improves the understanding of the effectiveness of algorithms for private k-means clustering. We have introduced the novel concept of hybrid approach to differentially private data analysis, which is so far the best approach to k-means clustering, and may prove useful in other analysis tasks as well.

Third, we have introduced a new version of SVT that provides better utility. We also introduce an effective technique to improve the performance of SVT by optimizing the distribution of privacy budget. These enhancements achieve better utility than the state of the art SVT and can be applied to improve utility in the interactive setting. We have also explained the misunderstandings and errors in a number of papers that use or analyze SVT; and believe that these will help clarify the misunderstandings regarding SVT and help avoid similar errors in the future. We have also shown that in the non-interactive setting, EM should be preferred over SVT.

REFERENCES

REFERENCES

[1] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[2] M. Barbaro and J. Tom Zeller, "A face is exposed for AOL searcher no. 4417749," New York Times, August 2006.

[3] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, 2008, pp. 111–125.

[4] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig, "Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays," *PLoS Genet*, vol. 4, no. 8, p. e1000167, 2008.

[5] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proceedings of the 3rd Conference on Theory of Cryptography*, 2006, pp. 265–284.

[6] C. Dwork, "Differential privacy," in *Proceedings of the 33rd International Conference on Automata, Languages and Programming*, 2006, pp. 1–12.

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

[8] K. Chaudhuri and C. Monteleoni, "Privacy-preserving logistic regression," in *Advances in Neural Information Processing Systems*, 2008, pp. 289–296.

[9] A. Friedman and A. Schuster, "Data mining with differential privacy," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 493–502.

[10] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *Journal of Machine Learning Research*, vol. 12, pp. 1069–1109, 2011.

[11] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett, "Functional mechanism: Regression analysis under differential privacy," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1364–1375, 2012.

[12] J. Zhang, X. Xiao, Y. Yang, Z. Zhang, and M. Winslett, "Privgene: Differentially private model fitting using genetic algorithms," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 665–676.

[13] D. Kifer and A. Machanavajjhala, “No free lunch in data privacy,” in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, 2011, pp. 193–204.

[14] F. McSherry and K. Talwar, “Mechanism design via differential privacy,” in *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, 2007, pp. 94–103.

[15] S. L. Warner, “Randomized response: A survey technique for eliminating evasive answer bias,” *Journal of the American Statistical Association*, vol. 60, no. 309, pp. 63–69, 1965.

[16] A. Blum, C. Dwork, F. McSherry, and K. Nissim, “Practical privacy: The SuLQ framework,” in *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2005, pp. 128–138.

[17] M. Hardt and G. N. Rothblum, “A multiplicative weights mechanism for privacy-preserving data analysis,” in *Proceedings of the IEEE 51st Annual Symposium on Foundations of Computer Science*, 2010, pp. 61–70.

[18] M. Hardt and K. Talwar, “On the geometry of differential privacy,” in *Proceedings of the 42nd ACM Symposium on Theory of Computing*, 2010, pp. 705–714.

[19] A. Roth and T. Roughgarden, “Interactive privacy via the median mechanism,” in *Proceedings of the 42nd ACM symposium on Theory of computing*, 2010, pp. 765–774.

[20] I. Dinur and K. Nissim, “Revealing information while preserving privacy,” in *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2003, pp. 202–210.

[21] C. Dwork, F. McSherry, and K. Talwar, “The price of privacy and the limits of LP decoding,” in *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, 2007, pp. 85–94.

[22] C. Dwork and S. Yekhanin, “New efficient attacks on statistical disclosure control mechanisms,” in *Proceedings of the 28th Annual Conference on Cryptology: Advances in Cryptology*, 2008, pp. 469–480.

[23] C. Dwork, M. Naor, O. Reingold, G. N. Rothblum, and S. Vadhan, “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*, 2009, pp. 381–390.

[24] S. P. Lloyd, “Least squares quantization in PCM,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–136, 1982.

[25] K. Nissim, S. Raskhodnikova, and A. Smith, “Smooth sensitivity and sampling in private data analysis,” in *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, 2007, pp. 75–84.

[26] B. I. P. Rubinstein, P. L. Bartlett, L. Huang, and N. Taf, “Protecting respondent’s privacy in microdata release,” *Journal of Privacy and Confidentiality*, vol. 4, no. 1, pp. 65–100, 2012.

[27] F. McSherry, “Privacy integrated queries (PINQ) infrastructure,” 2009, https://www.microsoft.com/en-us/download/details.aspx?id=52363.

[28] J. M. Peña, J. A. Lozano, and P. Larrañaga, “An empirical comparison of four initialization methods for the k-means algorithm,” *Pattern Recognition Letters*, vol. 20, no. 10, pp. 1027–1040, 1999.

[29] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. E. Culler, “GUPT: Privacy preserving data analysis made easy,” in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 349–360.

[30] C. Dwork, “A firm foundation for private data analysis,” *Communications of the ACM*, vol. 54, no. 1, pp. 86–95, 2011.

[31] W. H. Qardaji, W. Yang, and N. Li, “Differentially private grids for geospatial data,” in *Proceedings of the 29th IEEE International Conference on Data Engineering*, 2013, pp. 757–768.

[32] D. Su, J. Cao, N. Li, E. Bertino, and H. Jin, “Differentially private k-means clustering,” in *Proceedings of the 6th ACM CODASPY Conference on Data and Application Security and Privacy*, 2016, pp. 26–37.

[33] J. Lei, “Differentially private m-estimators,” in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, 2011, pp. 361–369.

[34] N. Mohammed, R. Chen, B. C. M. Fung, and P. S. Yu, “Differentially private data release for data mining,” in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011, pp. 493–501.

[35] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, “Privacy in pharmacogenetics: An end-to-end case study of personalized Warfarin dosing,” in *Proceedings of the 23rd USENIX Security Symposium*, 2014, pp. 17–32.

[36] M. Fredrikson, S. Jha, and T. Ristenpart, “Model inversion attacks that exploit confidence information and basic countermeasures,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1322–1333.

[37] G. Jagannathan, K. Pillaipakkamnatt, and R. N. Wright, “A practical differentially private random decision tree classifier,” *Transactions on Data Privacy*, vol. 5, pp. 273–295, 2012.

[38] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao, “Privbayes: Private data release via Bayesian networks,” in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014, pp. 1423–1434.

[39] S. A. Vinterbo, “Differentially private projected histograms: Construction and use for prediction,” in *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases*, 2012, pp. 19–34.

[40] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang, “Principled evaluation of differentially private algorithms using DPBench,” in *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*, 2016, pp. 139–154.

[41] W. Qardaji, W. Yang, and N. Li, “Understanding hierarchical methods for differentially private histograms,” *Proceedings of the VLDB Endowment*, vol. 6, no. 14, pp. 1954–1965, 2013.

[42] R. J. Bayardo and R. Agrawal, “Data privacy through optimal k-anonymization,” in *Proceedings of the 21st International Conference on Data Engineering*, 2005, pp. 217–228.

[43] K. LeFevre, D. DeWitt, and R. Ramakrishnan, “Incognito: Efficient full-domain k-anonymity,” in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 2005, pp. 49–60.

[44] ——, “Mondrian multidimensional k-anonymity,” in *Proceedings of the 22nd International Conference on Data Engineering*, 2006, pp. 25–35.

[45] R. C.-W. Wong, A. W.-C. Fu, K. Wang, and J. Pei, “Minimality attack in privacy preserving data publishing,” in *Proceedings of the 33rd International Conference on Very Large Data Bases*, 2007, pp. 543–554.

[46] G. Cormode, D. Srivastava, N. Li, and T. Li, “Minimizing minimality and maximizing utility: Analyzing method-based attacks on anonymized data,” *PVLDB*, vol. 3, no. 1-2, pp. 1045–1056, Sep. 2010.

[47] V. S. Iyengar, “Transforming data to satisfy privacy constraints,” in *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002, pp. 279–288.

[48] B. C. M. Fung, K. Wang, and P. S. Yu, “Top-down specialization for information and privacy preservation,” in *Proceedings of the 21st International Conference on Data Engineering*, 2005, pp. 205–216.

[49] L. Devroye, L. Györfi, and G. Lugosi, *A probabilistic theory of pattern recognition*. Springer, 1996.

[50] S. Kotz, T. Kozubowski, and K. Podgorski, *The Laplace Distribution and generalizations: A revisit with applications to communications, economics, engineering, and finance*. Springer, 2001.

[51] W. Qardaji, W. Yang, and N. Li, “Differentially private grids for geospatial data,” in *Proceedings of the 2013 IEEE International Conference on Data Engineering*, 2013, pp. 757–768.

[52] A. Asuncion and D. Newman, “UCI machine learning repository,” 2010, http://archive.ics.uci.edu/ml/.

[53] S. Ruggles, J. T. Alexander, K. Genadek, R. Goeken, M. B. Schroeder, and M. Sobek, “Integrated Public Use Microdata Series: Version 5.0 [machine-readable database],” https://www.ipums.org/.

[54] T. Therneau, B. Atkinson, and B. Ripley, “Rpart: Recursive partitioning and regression trees,” http://cran.r-project.org/web/packages/rpart/index.html.

[55] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[56] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[57] Y. Yang and J. O. Pedersen, “A comparative study on feature selection in text categorization,” in *Proceedings of the 14th International Conference on Machine Learning*, 1997, pp. 412–420.

[58] X. Geng, T.-Y. Liu, T. Qin, and H. Li, “Feature selection for ranking,” in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007, pp. 407–414.

[59] A. Smith, “Privacy-preserving statistical estimation with optimal convergence rates,” in *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, 2011, pp. 813–822.

[60] P. Mohan, “GUPT: A platform for privacy-preserving data mining,” 2012, https://github.com/prashmohan/GUPT.

[61] P. Fränti, “Clustering datasets,” http://cs.joensuu.fi/sipu/datasets/.

[62] U. S. Census, “Topologically integrated geographic encoding and referencing,” http://www.census.gov/geo/maps-data/data/tiger.html.

[63] W. Qiu, “ClusterGeneration: Random cluster generation (with specified degree of separation),” http://cran.r-project.org/web/packages/clusterGeneration/index.html.

[64] Scipy.org, “Scientific computing tools for Python,” http://scipy.org/.

[65] K. Kummamuru and M. N. Murty, “Genetic k-means algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 29, no. 3, pp. 433–439, 1999.

[66] S. Ray and R. H. Turi, “Determination of number of clusters in k-means clustering and application in colour image segmentation,” in *Proceedings of the 4th International Conference on Advances in Pattern Recognition and Digital Techniques*, 1999, pp. 137–143.

[67] R. Tibshirani, G. Walther, and T. Hastie, “Estimating the number of clusters in a data set via the gap statistic,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 2, pp. 411–423, 2001.

[68] A. Gupta, A. Roth, and J. Ullman, “Iterative constructions and private data release,” in *Proceedings of the 9th International Conference on Theory of Cryptography*, 2012, pp. 339–356.

[69] J. Lee and C. W. Clifton, “Top-k frequent itemsets via differentially private FP-trees,” in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 931–940.

[70] B. Stoddard, Y. Chen, and A. Machanavajjhala, “Differentially private algorithms for empirical machine learning,” *CoRR*, vol. abs/1411.5428, 2014.

[71] R. Chen, Q. Xiao, Y. Zhang, and J. Xu, “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*, 2015, pp. 129–138.

[72] R. Shokri and V. Shmatikov, “Privacy-preserving deep learning,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1310–1321.

[73] A. Roth, “The sparse vector technique,” 2011, lecture notes for “ The Algorithmic Foundations of Data Privacy”.

[74] Y. Chen and A. Machanavajjhala, “On the privacy properties of variants on the sparse vector technique,” *CoRR*, vol. abs/1508.07306, 2015.

[75] J. Zhang, X. Xiao, and X. Xie, “Privtree: A differentially private algorithm for hierarchical decompositions,” in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 155–170.

[76] C. Dwork and A. Roth, “The algorithmic foundations of differential privacy,” *Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2013.

[77] C. Dwork, G. N. Rothblum, and S. Vadhan, “Boosting and differential privacy,” in *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, 2010, pp. 51–60.

VITA

VITA

Dong Su was born and raised in Tianjin, China. He gradudated from Tianjin University with a Bachelor of Engineering in software engineering in 2005 and gradudated from University of Chinese Academy of Sciences in 2010 with a Master of Engineering in computer science. Dong entered Purdue University in the Fall of 2010, and worked under the supervision of Dr. Ninghui Li in the Department of Computer Science. Dong's graduate work and research was in the area of differentially private data publishing. He received his Master of Science in computer science in December of 2015, and his Ph.D. in computer science in December of 2016.