DPSyn: Differentially Private Synthetic Data Publication

August 2, 2018

Ninghui Li, Zhikun Zhang, Tianhao Wang
Team DPSyn, Purdue University
{ninghui,zhan3072,tianhaowang}@purdue.edu

1 Introduction

In recent years, publishing high-dimensional datasets while preserving privacy becomes a prominent desire. However, due to the curse of dimensionality, this task is challenging. In this paper, we propose to develop DPSyn, an algorithm and open-source software for synthesize microdata while satisfying differential privacy. DPSyn builds on our previous work PriView (published at SIGMOD'14 under the title "PriView: practical differentially private release of marginal contingency tables"). The main idea is to generate a synthetic dataset that approximates many randomly-chosen marginal distributions of the input dataset.

1.1 Method Overview

Our technique deals with categorical datasets; and numerical attributes are first bucketized so that they become categorical values. Each marginal is specified by a subset of the attributes, and can be viewed as a projection from the full contingency table to those attributes. A marginal table consists of L cells; each cells contains a fraction number such that the sum of all numbers is 1. For example, a marginal table with 8 binary attributes has $L = 2^8 = 256$ cells, and fully captures the distribution on these 8 attributes. A dataset with 100 binary attributes has a huge number of (100 chooses 8, to be exact) 8-way marginals. Our approach is to randomly select m marginals of size L (e.g., $m = 50, L = 256$), compute these marginals on the input dataset in a way that satisfies differential privacy, and then synthesize a dataset based on these marginals. The rationale for our approach are as follows. Any analysis one may want to conduct on a dataset can be performed using the joint distribution of some subset of attributes. On most subsets of attributes, a synthesized dataset that simultaneously preserves many (from dozens to hundreds) randomly-chosen marginals would have a distribution close to that of the original dataset. Under differential privacy, one can answer counting queries with good accuracy; thus we utilize marginals (which are essentially counting queries) to extract information from the input dataset.

Given a dataset as input, the first step is to generate many randomly selected noisy marginals on the dataset. The algorithm decides the parameters m (number of marginals) and L (size of marginals). Analysis in the PriView paper showed that $L = 256$ is approximately optimal for binary attributes, we will study whether the same holds for general categorical attributes. The number m depends on the dataset size and the privacy parameter ϵ. The algorithm then computes these marginals of the input dataset, and finally adds Laplacian/Gaussian noises to them so that differential privacy is satisfied. The amount of noises depends on the privacy parameter ϵ, and the number of marginals, as publishing each marginal has a global sensitivity of 1.

In the second step, we use techniques developed in PriView to make all noisy marginals consistent with each other. The techniques presented in PriView were for binary attributes. We have already extended those techniques to categorical attributes. In PriView, it was shown that one can use these noisy marginals (called private views there) to reconstruct arbitrary marginals with high accuracy. This suggests that these noisy marginals captures a lot of information in the input dataset, and can be used for a broad range of data analysis tasks.

The third step is to generate a synthetic dataset given these private views. PriView only has techniques to reconstruct other query marginal. Here we propose to develop techniques to synthesize a dataset that approximates these views. This is where we expect to spend most of our efforts. We plan to investigate a few alternative methods. One method starts with a randomly generated dataset and gradually changes it to be consistent with the noisy marginals. Another is to use these noisy marginals to construct probabilistic graphical models of the dataset, and then synthesize data from these probabilistic models. The key challenge is efficiency. To be able to preserve information in datasets with dozens or more attributes, we expect to use dozens or more noisy marginals, each including $5-10$ attributes. We expect the implemented software tool can generate datasets with millions of records in this setting.

Clearly, the larger m (the more marginals), the more marginal information is preserved, and the better the utility. However, a larger m also means more noise for each marginal, when the total privacy budget is fixed. Fortunately, a larger dataset means that less privacy budget is needed for each individual marginal. With a dataset 10 times larger, one needs only 1/10 privacy budget to get marginal of the same accuracy. Furthermore, if we are willing to go beyond strict epsilon differential privacy, and accepts weaker notions such as (ϵ, δ) differential privacy, we can use more advanced composition theorems to our advantage. Essentially, if one spends 1/10 of original budget for one marginal, then one is able to publish a lot more than 10 times the original number of marginals. This means that our approach is very promising for large datasets.

1.2 Contribution

First, we improve PriView to handle the high-dimensional case and we extend PriView to handle the non-binary case. Second, contrary to existing methods, which can only handle specific tasks, our algorithm is generic, meaning that any task can be performed on the output of the algorithm. Third, we plan to open-source the code.

2 Preliminaries

2.1 Problem Definition

We assume that there are d attributes $\mathbb{A} = \{a_1, a_2, \ldots, a_d\}$. Each attribute a_i has c_i possible values. Wlog, we assume that the values for a_i are $[c_i] \coloneqq \{0, 1, \cdots, c_i - 1\}$. Each user has one value for each attribute. Thus user j's value is a d-dimensional vector, denoted by $v^j = \langle v_1^j, v_2^j, \ldots, v_d^j \rangle$ such that $v_i^j \in [c_i]$ for each i. The full domain for the users' values is given by $D = [c_1] \times [c_2] \times \cdots \times [c_d]$, in which $\times$ denotes cartesian product. The domain D has size $|D| = \prod_{i=1}^{d} c_i$.

Let us first consider the setting of answering marginal queries in the centralized setting, where the server has all users' data. For a population of n users, the *full contingency table* gives, for each value $v \in D$, the fraction of users having the value v. The full contingency table gives the joint distribution of all attributes in $\mathbb{A}$, and includes the complete information. We use F to denote the full contingency table, and call the fraction for each value $v \in D$ a cell in the full contingency table. When the domain size is very large, e.g., when there are many attributes, computing the full contingency table can be prohibitively expensive. Oftentimes, one is interested in the joint distribution of some subsets of the attributes. Given a set of attributes $A \subseteq \mathbb{A}$, we use $V_A = \{\langle v_1, v_2, \ldots, v_d \rangle : v_i \in [c_i] \text{ if } a_i \in A, \text{otherwise } v_i = *\}$ to denote the set of all possible values specified by A.

When given a set A of k attributes, the *k-way marginal over A*, denoted by M_A, gives the fraction of users having each value in V_A. We call the fraction for each value $v \in V_A$ a cell of the marginal table. M_A contains fewer cells than the full contingency table F. Each cell in M_A corresponds to many cells in F, which have the same indicating values on the attributes in A. M_A can be computed from F by summing up all corresponding cells in F. See Figure 1 for an example of the process. Note that when d is large, it is infeasible to compute F directly, and one needs to compute the marginals instead.

Given the marginals, we can generate the synthetic dataset that follows the marginal distribution. However, due to privacy concerns, it is not possible to compute the marginal tables directly. We propose to use Differential Privacy to quantify the information leakage.

	Gender	Age
v^1	male	teenager
v^2	female	teenager
v^3	female	adult
v^4	female	adult
$\cdots$	$\cdots$	$\cdots$
v^n	male	elderly

(a) Dataset.

v	$\mathsf{F}(v)$
⟨male, teenager⟩	0.20
⟨male, adult⟩	0.15
⟨male, elderly⟩	0.20
⟨female, teenager⟩	0.15
⟨female, adult⟩	0.20
⟨female, elderly⟩	0.10

(b) Full contingency table.

v	$\mathsf{M}_{\{\text{gender}\}}(v)$
⟨male,∗⟩	0.55
⟨female,∗⟩	0.45

(c) Marginal table for gender.

v	$\mathsf{M}_{\{\text{age}\}}(v)$
⟨∗,teenager⟩	0.35
⟨∗,adult ⟩	0.35
⟨∗,elderly⟩	0.30

(d) Marginal table for age.

Figure 1: Example of the dataset, the full contingency table, and the marginal tables.

2.2 Differential Privacy

Differential privacy [3] was proposed for the setting where there is a **trusted data curator**, who gathers data from individual users, processes the data in a way that satisfies DP, and then publishes the results. Intuitively, the DP notion requires that any single element in a dataset has only a limited impact on the output.

Definition 1 ((ϵ, δ)-Differential Privacy)**.** *An algorithm* $\mathbf{A}$ *satisfies* (ϵ, δ)*-differential privacy (*(ϵ, δ)*-DP), where* $\epsilon > 0, \delta \geq 0$*, if and only if for any datasets* D *and* D' *that* differ on one element*, we have*

$$\forall T \subseteq Range(\mathbf{A}) : \Pr[\mathbf{A}(D) \in T] \leq e^{\epsilon} \Pr[\mathbf{A}(D') \in T] + \delta, \tag{1}$$

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

Note that $(\epsilon, 0)$-DP is also called pure DP, which is a more strict definition. When t DP algorithms are applied, each satisfying ϵ-DP, in the pure DP setting, the total privacy budget would be $t \cdot \epsilon$; while in the non-pure setting, one can use advance composition stated below to compute the final privacy budget.

Theorem 1 (Advanced Composition in Adaptive Setting [4])**.** *If* $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_t$ *are* ϵ*-DP and* $\mathbf{A}(D) = \mathbf{A}_k(D, \mathbf{A}_{t-1}(D, \mathbf{A}_{t-2}(D, \ldots)))$*, then* $\mathbf{A}(D)$ *is* (ϵ', δ)*-DP for all* δ *and* $\epsilon' = \sqrt{2t \ln(1/\delta)}\epsilon + t(e^{\epsilon} - 1)\epsilon$*.*

Note that advanced composition is only a general bound, which is not very tight. To capture the privacy loss precisely, one can use a numerical method to calculate a tighter bound.

Numerically bounding privacy loss. In [10], the authors propose a numerical method to calculate the privacy loss on any r-fold differential privacy mechanism. The intuition is that, for the Gaussian/Laplace mechanism, since it follows a distribution, one can bound the privacy loss. For the composed mechanisms, the method compute privacy loss by enumerating the events and summarize the probabilities. As a result, the computed privacy loss is much smaller than that computed by advance composition theorem. Note that when pure DP is used, the computed privacy loss is the same as the sum of the privacy budget used in each step.

3 DPSyn

In this section, we describe our proposed method DPSyn for synthesizing dataset with differential privacy guarantee. Our method builds on the PriView method for publishing marginal tables [12], so we describe PriView first.

3.1 An Overview of PriView

The PriView method was designed for privately computing arbitrary k-way marginals for a dataset with d binary attributes. PriView publishes a synopsis of the dataset. Using the synopsis, it can reconstruct any k-way marginal. The synopsis takes the form of m size-ℓ marginals (the marginals contain ℓ attribute) that are called *views*. Note that the objective of PriView is to use the noisy views to answer arbitrary k-way marginals, but DPSyn aims to use the noisy views to synthesize a dataset that consistent with these views. Below we give an overview of the PriView method, using an example where there are $d = 8$ attributes $\{a_1, a_2, \cdots, a_8\}$, and we aim to answer all 3-way marginals. PriView has the following four steps.

3.1.1 Choose the Set of Views

The first step is to choose which marginals to include in the private synopsis as views. That is, one needs to choose m sets of attributes. PriView chooses these sets so that each size-2 (or size-3) marginal is covered by some view. For example, if aiming to cover all 2-way marginals, then one could choose the following $m = 6$ sets of attributes to construct views:

$$\begin{array}{lll} \{a_1, a_2, a_3, a_4\} & \{a_1, a_5, a_6, a_7\} & \{a_2, a_3, a_5, a_8\} \\ \{a_4, a_6, a_7, a_8\} & \{a_2, a_3, a_6, a_7\} & \{a_1, a_4, a_5, a_8\} \end{array}$$

Observe that any pair of two attributes are included in at least one set.

3.1.2 Generate Noisy Views

In this step, for each of the m attribute sets PriView constructs a noisy marginal over the attributes in the set, by adding Laplace noise $\mathsf{Lap}\left(\frac{m}{\epsilon}\right)$ to each cell in the marginal table. This is the only step that needs direct access to the dataset. After this step, the dataset is no longer accessed.

3.1.3 Consistency Step

Given these noisy marginals/views, some 3-way marginals can be directly computed. For example, to obtain the 3-way marginal for $\{a_1, a_2, a_3\}$, we can start from the view for $\{a_1, a_2, a_3, a_4\}$ and marginalizes out a_4. However, many 3-way marginal are not covered by any of the 6 views. For example, if we want to compute the marginal for $\{a_1, a_3, a_5\}$, we have to rely on partial information provided by the 6 views. We can compute the marginals for $\{a_1, a_3\}$, $\{a_1, a_5\}$, and $\{a_3, a_5\}$, and then combine them to construct an estimation for $\{a_1, a_3, a_5\}$.

Observe that $\{a_1, a_5\}$ can be computed both by using the view for $\{a_1, a_5, a_6, a_7\}$ and by using the view for $\{a_1, a_4, a_5, a_8\}$. Since independent noises are added to the two marginals, the two different ways to compute marginal for $\{a_1, a_5\}$ will have different results. In addition, the noisy marginals may contain negative values. PriView performs constrained inference on the noisy marginals to ensure that the marginals in the synopsis are all non-negative and mutually consistent.

3.1.4 Overall Consistency

We can conduct the following procedure to achieve overall consistency. First, enumerate all the subsets of $\mathbb{A}$. These subsets form a partial order under the subset relation, which can be organized as a topological graph. This topological graph starts from the empty set. Then, for each subset of A in the topological order, we ensure the consistency among the marginals that include this subset. It is shown in [12] that following the topological order, a later consistency step will not invalidate consistency established in previous steps.

Non-Negativity through Ripple. We propose to adopt the following "Ripple" non-negativity method, which turns negative counts into 0 while decreasing the counts for its neighbors to maintain the overall count constant. Specifically, given an k-way marginal table T_A, for any $v \in V_A$ with $\mathsf{T}_A(v) < -\theta$, we set the entry to 0 and subtract $|c|/h$ from each of its h neighboring cells, defined as the cells obtained by changing one of the attributes' category to other categories, and h is determined by the number of categories of each attribute in A. However, this procedure may make the count of other cells to be $c < -\theta$, the procedure iterates until no cell has count $c < -\theta$. As each iteration distributes a negative count into h neighbors, it

is guaranteed to terminate quickly. Applying the ripple non-negativity step to the marginals, however, may make them inconsistent. To resolve this problem, we run the consistency step after the non-negativity step several times.

3.2 Improvement over PriView

Utility optimization. PriView was proposed to handle datasets with limited columns, e.g., $d = 8$ to 16. When d grows larger, i.e., in the typical census dataset, d can be as large as 100, the number of marginals output by PriView grows accordingly, and the privacy budget allocated for each marginal is much less. Therefore, we propose to use approximate differential privacy and use the numerical approach to calculate the final privacy loss. Moreover, we propose to use Gaussian mechanism because it is shown to achieve better privacy-utility tradeoff in [10].

Extend to non-binary case. In the consistent step, PriView can only handle binary attributes. We should design technique to consist non-binary attributes. Notice that when different marginals have some attributes in common, those attributes are actually estimated multiple times. Utility will increase if these estimates are utilized together. Specifically, assume a set of attributes A is shared by s marginals, $A_1, A_2, \ldots, A_s$. That is, $A = A_1 \cap \ldots \cap A_s$. Now we can obtain s copies of T_A by summing from cells in each of the T_A's, i.e., $\mathsf{T}_{A_i}(v) = \sum_{v' \in V_{A_i}, v'_A = v_A} \mathsf{T}_{A_i}(v')$.

To obtain a better estimation of T_A, we use the weighted average of T_{A_i} for all marginal A_i. That is,

$$\mathsf{T}_A(v) = \sum_i w_i \cdot \mathsf{T}_{A_i}(v).$$

Since each T_{A_i} is unbiased, their average $\mathsf{T}_A(v)$ is also unbiased. To determine the distribution of the weights, the intuition is to put more weights to the more accurate estimations. Specifically, we minimize the variance of $\mathsf{T}_A(v)$, i.e., $\mathsf{Var}\left[\mathsf{T}_A(v)\right] = \sum_i w_i^2 \cdot \mathsf{Var}\left[\mathsf{T}_{A_i}(v)\right] = \sum_i w_i^2 \cdot C_i \cdot \mathsf{Var}_0$, where C_i is the number of cells from A_i that contribute to A, i.e., $C_i = |\{v' : v' \in V_{A_i}, v'_A = v_A\}|$, and Var_0 is the basic variance for estimating a single cell (we assume each marginal has a similar amount of users, but the analysis can be easily changed to different number of users). Formally, we have the following problem:

$$\begin{array}{ll} \mathsf{minimize} & \sum_i w_i^2 \cdot C_i \\ \mathsf{subject\ to} & \sum_i w_i = 1 \end{array}$$

According to KKT condition [7, 6], we can derive the solution: Define $L = \sum_i w_i^2 \cdot C_i + \mu \cdot (\sum_i w_i - 1)$, by taking the partial derivative of L for each of w_i, we have $w_i = -\frac{\mu}{2C_i}$. The value of μ can be solved by the equation $\sum_i w_i = 1$. As a result, $\mu = -\frac{2}{\sum_i \frac{1}{C_i}}$, and $w_i = \frac{\frac{1}{C_i}}{\sum_i \frac{1}{C_i}}$. Therefore, the optimal weighted average is

$$\mathsf{T}_A(v) = \frac{\sum_i \frac{1}{C_i} \cdot \mathsf{T}_{A_i}(v)}{\sum_i \frac{1}{C_i}}$$

Once the accurate T_A is obtained, all T_{A_i}'s can be updated. For any marginal A_i, we update all $v' \in V_{A_i}$ using the result of v where $v \in VA$ and $v'_A = v_A$. Specifically,

$$\mathsf{T}_{A_i}(v') \leftarrow \mathsf{T}_{A_i}(v') + \frac{1}{C_i}\left(\mathsf{T}_A(v) - \mathsf{T}_{A_i}(v)\right)$$

3.3 Generate Synthetic Dataset

Given the consistent noisy marginals, the goal is to generate a synthetic dataset S that has a similar distribution with the original dataset D. Specifically, for any set of attribute A, we want $\mathsf{T}_A \sim \mathsf{M}_A$, where M_A is the marginal of S, and T_A is the noisy marginal of D. Note that this step is a post-processing step, which takes the noisy marginals only.

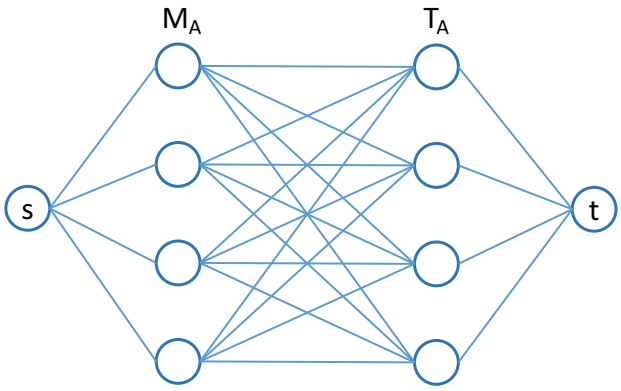
We propose two approaches for doing this, i.e., MCF and STS. MCF first initializes S by randomly generate n records; then all noisy marginals are traversed to make sure S follows the same marginals with

the noisy marginals obtained from D. STS works more like a generative model, which samples records that follow the desired distribution, and thus no update is needed.

MCF: update through Min-Cost Flow. Given the dataset S, for each noisy marginal T_A, we update S so that $\mathsf{T}_A \sim \mathsf{M}_A$. Note that during the process of update, it is possible that the consistency of previous step is invalidated by following step. We propose to model the update procedure as the minimum cost flow problem. The result of the min-cost flow problem will give the update operations that make the minimal changes to S. The intuition is that, by changing the dataset in the minimal way, the inconsistency effect is minimized.

To determine the minimal changes on A, a flow graph is constructed as Figure 2. In Figure 2, nodes s and t represent the source node and the sink node. Flows departing from s represent values in M_A, and flows arriving at t represent values in T_A. The edge capacity between M_A and T_A is 1. The edge cost is 0 for the horizontal edges, and is 1 for all other edges. The objective is to determine all the flows from M_A to T_A to minimize the total cost. Denote $c(u, v)$ and $f(u, v)$ as the cost and flow on edge (u, v), we formalize the minimum cost flow problem as the following linear programming problem:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{m_a \in \mathsf{M}_A, t_a \in \mathsf{T}_A} c(m_a, t_a) \cdot f(m_a, t_a) \\
\text{subject to} \quad & \forall_{m_a \in \mathsf{M}_A} f(s, m_a) = \textstyle\sum_{t_a \in \mathsf{T}_A} f(m_a, t_a), \\
& \forall_{t_a \in \mathsf{T}_A} f(t_a, t) = \textstyle\sum_{m_a \in \mathsf{M}_A} f(m_a, t_a), \\
& \forall_{m_a \in \mathsf{M}_A} \forall_{t_a \in \mathsf{T}_A} 0 \le f(m_a, t_a) \le 1,
\end{aligned}
$$



(a)

Figure 2: Illustration of minimum cost flow problem.

The above optimization problem can be solved by off-the-shelf solvers. Note that one still needs to repeat the process for all the noisy marginals multiple times until the amount of changes is small. This indicates S converges to be consistent with all the noisy marginals.

STS: Sample records from Topologically Sorted marginals. STS first topologically sorts the marginals, so that the first marginal is randomly chosen, and then each following marginal contains as few as new attributes as possible. For example, if the first marginal is $A_1 = \{a_1, a_2, a_3, a_4\}$, we choose $A_2 = \{a_1, a_2, a_3, a_5\}$ over $A_3 = \{a_1, a_2, a_5, a_6\}$ because A_2 contains only one new attribute a_5 but A_3 contains two a_5, a_6. If there is a tie, a random marginal is chosen. If there is a marginal that contains no new attribute, it is discarded. We iteratively choose a set of marginal so that all attributes $\mathbb{A}$ are covered. Notice that the topologically sorted marginals does not necessarily contain all marginals in $\mathbf{A}$.

Given the sorted marginals, records are generated one by one. Specifically, for the first marginal A, take the attributes combination with largest count $v = \arg\max_v \mathsf{T}_A(v)$, assign v to the attributes A of the data record. With the data record partially set, the new attributes in the following marginal is set to the combination with largest count. That is, for the new marginal A', find and assign $v' = \arg\max_{v'} \mathsf{T}_{A'}(v')$, where v' must agree with the existing instantiated value.

When the data record is fully generated, the counts of corresponding combinations in all marginals are reduced by 1. Following the same procedure, we can generate n records until the counts for all marginal are equal to 0. Finally, to eliminate the impact of the first randomly selected marginal, we repeat the process several times.

The following example illustrates the generation procedure. Given four binary attributes $\{a_1, a_2, a_3, a_4\}$ and three topologically sorted marginals $A_1 = \{a_1, a_2\}, A_2 = \{a_2, a_3\}, A_3 = \{a_3, a_4\}$. The original marginal tables are shown in Figure 3a. Observe that the first marginal A_1 contains two attributes $\{a_1, a_2\}$, and the combination with largest count is $\{1, 0\}$. The second marginal A_2 provides new attribute a_3. Given $a_2 = 0$, $A_2 = \{0, 1\}$ has the largest count (over $\{0, 0\}$). Similarly, given $a_3 = 1$, $A_3 = \{1, 1\}$ have the largest count. With above calculation, one can generate the first record $\{1, 0, 1, 1\}$, and reduce the corresponding counts by 1, see Figure 3b. Then, we can generate the other records following the same procedure. If the count of several combinations are the same, we randomly choose one of them.

	$\{0,0\}$	$\{0,1\}$	$\{1,0\}$	$\{1,1\}$
$A_1 = \{a_1, a_2\}$	3	2	4	1
$A_2 = \{a_2, a_3\}$	2	3	3	2
$A_3 = \{a_3, a_4\}$	4	3	1	2

(a) Original marginal table.

	$\{0,0\}$	$\{0,1\}$	$\{1,0\}$	$\{1,1\}$
$A_1 = \{a_1, a_2\}$	3	2	3	1
$A_2 = \{a_2, a_3\}$	2	2	3	2
$A_3 = \{a_3, a_4\}$	4	3	1	1

(b) Marginal table when the first record is generated.

Figure 3: Marginal tables.

4 Discussion

4.1 Utility

We optimize utility by taking advantage of the most advanced composition theorem. That is, given the privacy budget ϵ and δ, and the dataset column count d, we optimize the number of marginals ℓ and the size of each marginal w, so that the overall expected error is minimized. Specifically, for any combination of ℓ and w, we first calculate the privacy budget ϵ', δ' allocated for each marginal, using the numerical method proposed in [10]. We then calculate the expected utility, elaborated below. Finally, the ℓ and w with the minimal privacy loss is chosen as the configuration of DPSyn.

To calculate the expected utility, we compute the amount of noise on average. Specifically, for any k-way marginal (k is an integer between 3 to 8; and we found that the influence of k is not significant), we compare the squared difference between the ground truth and the result value for each cell, and take average. Since Laplace/Gaussian noise is added, the variance is given. Specifically, when ϵ' and δ' are given, we can calculate the among of noise added, and the derive the variance.

4.2 Range of Application

Ideal Case. DPSyn is designed for working with publishing information about Census. That is, DPSyn can handle tens of attributes (even more than one hundred if the attributes are binary). The method works best when the analysis task needs to evaluate all attributes.

Basically, any problems that involves inspecting the data carefully can be handled well. Specifically, identifying the relationship among attributes; causal inference; abnormal detection. Since a synthetic dataset is generated, we expect to see roughly the same performance. But if a task only needs several specific attributes, DPSyn will not perform as well as a ad-hoc methods, because intuitively DPSyn spreads privacy budget among all the attributes, instead of concentrating the budget on the several important ones.

Cases that cannot be handled by DPSyn. DPSyn works under the condition that noisy marginals can be generated. So when the noisy marginals cannot be obtained from the dataset, the method will not work well. It is possible though to modify the dataset or the method to handle the specific problems. Below we summarize the cases where our method cannot work, and propose tweaks for handling them. Note that these cases are equally hard for most other DP-based methods.

First, when data can be updated. Like many other privacy-preserving algorithms, DPSyn currently cannot handle the case when the data is constantly updated, because each publishing requires some privacy budget. It is possible though, to run DPSyn multiple times with splited privacy budget. However, when the data is constantly updated, DPSyn does not work. When the update is deterministic, on the other hand, it is possible to extend DPSyn to include some update rules, so that the update actions can be modeled.

Second, when some attribute is non-categorical. For example, numerical values should be bucketized to discret values in order for DPSyn to process. But one can only define buckets based on the sematics of the attribute; it is unclear what is the optimal bucketization for the data. Even worse, DPSyn cannot handle content data where the domain is very large or unbounded, such as text data, audio/video data.

Third, when the dataset is not in a relational database. Examples include graph models and streaming models. The original notion of differential privacy was proposed to handle the counting queries in the relational databases. Over the years, different notions have been proposed to handle different data types, e.g., location indistinguishability. But since DPSyn is built on differential privacy, it will have difficulty handling data types other than relational database. Adopting other privacy notions will be an interesting future direction.

4.3 Complexity

In the first step, the method scans the dataset to construct the marginals. The computation grows linear with the size of the dataset and the number of marginals, respectively. After that, injecting noise is fast, but making the marginals consistent requires more time, since it is basically an optimization problem. It is not clear what is the theoretical running time, but with a limit on the running time, the optimization can always terminate soon enough and output reasonable result. Finally, the synthesize step is linear in the size of the target dataset and the number of noisy marginals.

5 Evaluation Plan

We plan to evaluate the performance of the synthesized dataset by answering the marginal queries and training machine learning models (*e.g.*, regression/classification/clustering).

Metrics. Sum of Squared Error (SSE) is the metrics we use to measure accuracy of the marginal information. That is, we compute the ground truth and calculate the sum of squared difference in each cell. For the machine learning model, we use appropriate metrics such as mis-classification rate.

Environment. All algorithms will be implemented in Python 3.5 and all the experiments will be conducted on a PC with Intel Core i7-4790 3.60GHz and 16GB memory.

Datasets. We will run experiments on the following four datasets.

- POS [19]: A dataset containing merchant transactions of half a million users.
- Kosarak [1]: A dataset of click streams on a Hungarian website that contains around one million users.
- Adult [2]: A dataset from the UCI machine learning repository. After removing missing values, the dataset contains around 50 thousands records. The numerical attributes are bucketized into categorical attributes.
- US [13]: A dataset from the *Integrated Public Use Microdata Series* (IPUMS). It has around 300 millions records of the United States census in 2016. Each record contains over 100 attributes.

The first two are transactional datasets where each record contains some items. We treat each item as a binary attribute. Thus these two datasets are binary. When running experiments with k binary attributes, we pre-process a dataset to include only the top d most frequent items. The later two are non-binary datasets, i.e., each attribute contains more than two categories.

6 Related Work

Differential privacy has been the *de facto* notion for protecting privacy. Many DP algorithms have been proposed (see [4, 16] for theoretical treatments and [9] in a more practical perspective). Recently, Uber has deployed a system enforcing DP during SQL queries [5], Google also proposed several works that combine DP with machine learning, e.g., [11].

There are existing work on the specific task of regression/classification/clustering. For example, we have proposed [15] and [14] for classification and clustering, respectively. Compared with the ad-hoc methods, our approach is more general, but may not achieve the optimal utility for the specific regression/classification/clustering task. Compared to other data publication approaches, our method preserves more information and thus achieves better utility. Below we focus on the comparison with the general methods.

To publish synthetic datasets, a commonly used approach is to train a generative model satisfying differential privacy, and use the generative model to generate a synthetic dataset. For example, PrivBayes [17] publishes a noisy Bayesian network that approximates the data distribution by several low-dimensional marginals (histograms). PrivBayes determines the structure of a Bayesian network by first randomly selecting (using Exponential Mechanism) an attribute as the first node, and then iteratively selecting another attribute to create a new node with up to k nodes already created as the new node's parent nodes. After the structure is determined, PrivBayes perturbs the marginals needed for computing the conditional distributions of the data. This method, however, needs to call Exponential Mechanism many times, making utility bad when the privacy budget is limited; and the overall utility is sensitive to the quality of the first selected node.

Besides the Bayesian model, other models can also be trained. For example, there are methods that assume the dataset is sampled from some statistical distributions (e.g., mixture models). The methods first learn parameters from the dataset, then sample the synthetic dataset from the statistical models. To the best of our knowledge, DPSynthesizer [8] is the only existing work that satisfies differential privacy. The main limitation of these methods is that only a restricted set of knowledge can be conveyed by the models, and oftentimes the dataset cannot be modeled perfectly. For example, although the height distribution is expected to follow normal distribution, the reality may be different; and these differences will be the most interesting findings.

There are also work that uses deep neural networks. For example, DP-GAN [18] trains a deep generative adversarial network (GAN) with differential privacy, by injecting random noise in the optimization procedure (e.g., stochastic gradient descent). However, the purpose of GAN is to generate data records that look authentic, instead of looking similar to the original distribution, thus the GAN model cannot be used in the problem.

References

[1] Frequent itemset mining dataset repository. `http://fimi.ua.ac.be/data/`.

[2] A. Asuncion and D. Newman. UCI machine learning repository, 2010.

[3] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[4] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[5] N. Johnson, J. P. Near, and D. Song. Practical differential privacy for sql queries using elastic sensitivity. *arXiv preprint arXiv:1706.09479*, 2017.

[6] W. Karush. Minima of functions of several variables with inequalities as side constraints. *M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago*, 1939.

[7] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Traces and emergence of nonlinear programming*, pages 247–258. Springer, 2014.

[8] H. Li, L. Xiong, L. Zhang, and X. Jiang. Dpsynthesizer: differentially private data synthesizer for privacy preserving data sharing. *Proceedings of the VLDB Endowment*, 7(13):1677–1680, 2014.

[9] N. Li, M. Lyu, D. Su, and W. Yang. *Differential Privacy: From Theory to Practice*. Synthesis Lectures on Information Security, Privacy, and Trust. Morgan Claypool, 2016.

[10] S. Meiser and E. Mohammadi. Privacy buckets: Upper and lower bounds for r-fold approximate differential privacy. 2018.

[11] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson. Scalable private learning with pate. In *ICLR*, 2018.

[12] W. Qardaji, W. Yang, and N. Li. Priview: practical differentially private release of marginal contingency tables. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1435–1446. ACM, 2014.

[13] 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], 2010.

[14] D. Su, J. Cao, N. Li, E. Bertino, M. Lyu, and H. Jin. Differentially private k-means clustering and a hybrid approach to private optimization. *ACM Transactions on Privacy and Security (TOPS)*, 20(4):16, 2017.

[15] D. Su, J. Cao, N. Li, and M. Lyu. Privpfc: differentially private data publication for classification. *The VLDB Journal—The International Journal on Very Large Data Bases*, 27(2):201–223, 2018.

[16] S. Vadhan. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*, pages 347–450. Springer, 2017.

[17] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):25, 2017.

[18] X. Zhang, S. Ji, and T. Wang. Differentially private releasing via deep generative model. *arXiv preprint arXiv:1801.01594*, 2018.

[19] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 401–406. ACM, 2001.