Differentially Private Simple Genetic Algorithms

by

Thomas Humphries

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Author's Declaration

Abstract

We study the differentially private (DP) selection problem, where the goal is to select an item from a set of candidates that approximately maximizes a given objective function. The most common solution to this problem is to use the exponential mechanism. The issue with this approach is that the exponential mechanism must compute the objective function for all possible candidates in the domain. For many real-world problems, the length of the domain is exponential, making this approach impractical. Genetic algorithms (GAs) use the principles of evolution in nature to efficiently search through large domains and find the best candidate. However, current work applying DP to GAs exhibits poor utility and the results are difficult to reproduce.

This work provides a new DP GA based on the popular simple genetic algorithm from the non-private literature. The biggest challenge is the number of selections made in the simple GA, each consuming a part of the privacy budget under DP. Our design reduces the number of selections and takes advantage of advanced composition techniques to overcome this challenge without impeding the heuristics that make the simple GA effective. We evaluate our solution over four different datasets using both convex and non-convex problems. The results demonstrate that our GA outperforms previous work in DP GAs as well as DP local search techniques. We further show that our DP GA offers increased utility across different datasets for efficiently scaling the exponential mechanism to large domains. Finally, we demonstrate that our general solution is competitive in utility or efficiency with state-of-the-art problem-specific solutions.

Acknowledgements

I want to thank my supervisor, Florian Kerschbaum, for all of his guidance with this project. His expertise, patience, and encouragement were above and beyond. I look forward to continuing to learn from him over the next few years. I would also like to thank Ashraf Abdelbar for his insightful discussions about genetic algorithms and for suggesting I study at Waterloo in the first place. The people in the CrySP lab have defiantly been the highlight of my time in Waterloo. The lab culture is second to none, and the friends I have made will surely last a lifetime. Specifically, I would like to thank Rasoul, Shannon, Simon, Matt, Miti, John, Bailey, Lindsey, Jason, Emily, and Masoumeh for being amazing friends and colleagues. They had my back through everything, and to say we had fun would be an understatement. This work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo. All experiments were conducted using RIPPLE machines.

Dedication

To my wife Shane, for your endless love, support, and encouragement.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

A fundamental problem in data science is selecting a candidate from a set of items that maximizes some objective function. For example, one might want to identify the most frequent conditions, choose the most representative features, or find the best ML model parameters for a given dataset. However, these datasets often contain sensitive information; thus, protecting the participants' privacy is crucial. An increasingly popular notion for protecting the privacy of individuals while allowing the computation of aggregate statistics is differential privacy (DP). Differential privacy guarantees that the output of an algorithm is approximately the same regardless of the participation of any single user. The intuitive guarantee, along with a tunable privacy parameter, has led to wide adoption of DP by organizations such as Google [21], Microsoft [11], Apple [10], and the U.S. Census Bureau [42].

We focus on the problem of differentially private selection, which involves selecting an item from a set of candidates that *approximately* maximizes a given objective function, *while preserving differential privacy.* The most popular mechanism for solving this problem is the exponential mechanism [44]. The exponential mechanism can be used as a standalone algorithm for heavy hitter or median [19, 41] calculations. It can also act as a building block for much more sophisticated algorithms such as principal component analysis [7], synthetic data generation [60], and empirical risk minimization [4]. The biggest weakness of the exponential mechanism is scalability. The exponential mechanism requires that the utility function is evaluated for all possible candidates in the domain. This results in exponential runtime for many problems such as ML model fitting or k-medians clustering. One possible solution is the sub-sampled exponential mechanism; however, this technique depends heavily on the utility of the sample [38]. Another solution specific to top-k counting queries avoids considering the whole domain but assumes the domain has been sorted and has a non-trivial chance of returning no solution [16].

Genetic algorithms (GAs) use the principles of evolution in nature to search through large domains and find an optimal candidate efficiently. Like the exponential mechanism, GAs require minimal assumptions about the objective function and can be applied to a wide range of problems. GAs also tend to be robust to noisy objective functions [56]. Despite these appealing properties, there have been no promising experimental results applying GAs to the DP selection problem. The only work in this space is a solution called PrivGene which presents the first DP GA [61]. In this work, Zhang et al. first introduce a private GA but move to a private evolutionary strategy (without crossover) as it shows better performance. All of the experiments were conducted using the evolutionary strategy which we call PrivEEM. Therefore, the performance of the GA was not evaluated. When evaluating PrivGene, both Su et al.'s [54] work and our own have not been able to reproduce similar results. Su et al. speculate that the poor performance is due to the destructive nature of the crossover operator and the large number of selections needed [54]. A contributing factor to the reproducibility problems is that the source code published with PrivGene does not match any of the algorithms in the paper. Instead, Zhang et al. included code for an algorithm called PrivLocal that follow-up work mistakenly evaluated as PrivGene [39]. To summarize, the current literature either does not evaluate a true DP GA [39, 61] or shows very poor performance [54].

In this thesis, we propose a new DP GA based on the classic simple GA as defined by Mitchell [46]. A naive approach would replace all selections in the simple GA with calls to the exponential mechanism. However, this would incur a prohibitively high cost in the privacy budget and reduce the selection pressure to such an extent that it thwarts the evolution of the population. Instead, we use the less popular truncation selection operator in conjunction with a DP selection mechanism to reduce the privacy budget while maintaining both selection pressure and diversity. We experimentally choose the DP mechanism and composition technique that yields the highest utility when paired with our simple GA. After designing our GA, we investigate the effect of DP on the hyperparameters of the simple GA and suggest good default values that generalize across the datasets and problems we considered.

We empirically evaluate the utility of our solution on two example problems: logistic regression (convex) and k-medians (non-convex). We conduct an extensive experimental evaluation against four categories of prior work: genetic algorithms and evolutionary strategies, local search techniques, efficient exponential mechanism alternatives, and problem-specific solutions. First, for genetic algorithms and evolutionary strategies, our DP simple GA not only outperforms PrivGene, but PrivEEM as well. For local search approaches, we show we can outperform PrivLocal in logistic regression, and the local search of Gupta et al. [29] for k-medians. We demonstrate that our DP GA offers an efficient exponential mechanism alternative for large domains providing a much more stable solution than the subsampled exponential mechanism. Finally, we show that our general-purpose solution is competitive in utility or efficiency when compared with state-of-the-art problem-specific solutions.

The remainder of the thesis is organized as follows: In Chapter 2 we introduce genetic algorithms and differential privacy. We define our problem and detail the relevant literature in Chapters 3 and 4. In Chapter 5 we design a non-private simple GA for our example problems and then discuss how we add privacy in Chapter 6. Chapter 7 tunes the hyperparameters and looks into alternate utility functions. Finally, Chapter 8 compares our solution with four categories of related work.

Chapter 2

Preliminaries

2.1 Genetic Algorithms

The Genetic Algorithm (GA) is a metaheuristic that mimics the evolutionary concept of natural selection [25, 28]. The idea was first introduced by John Holland in the 1960s and has since been developed extensively [30]. The foundations of the GA are very simple operations, that when applied carefully and repeatedly, enable an effective search through complex solution spaces. These simple operations are often probabilistic in nature but heavily guided by the utility of the solutions. The key ingredients required to apply a GA are an encoding of possible solutions, which we call chromosomes, and a utility function. The encoding is most commonly a binary string but can also be a real or multi-valued vector depending on what is most natural for the problem at hand. The utility function takes as input a chromosome (encoded solution vector) and returns a real number representing the effectiveness of the chromosome at solving the problem. To guide the search, a GA evaluates this utility function directly and does not require derivatives or any auxiliary knowledge. As such, there are minimal restrictions on the types of utility functions that can be considered.

Rather than starting from a single point, a GA begins with a population of randomly generated chromosomes and continues to modify or evolve this population over time. Upon this population, a series of simple operations that mimic those seen in evolution are applied. These operations are very probabilistic in nature to keep the population diversity and effectively explore the space. These operations simulate the ideas of mating and mutation in the real world. For instance, to mimic mating, the GA will take two high utility parent chromosomes and combine them to create offspring chromosome(s). For mutation, a chromosome undergoes a minor perturbation that may or may not increase its utility. The population is updated in this manner for multiple generations until some stopping criteria are met. This could either be that the optimal solution has been found, the population has converged, or a preset amount of time has passed.

2.1.1 The Simple GA

There are many variants of the GA; we focus on the simple GA as defined by Mitchell [46] for this work. The simple GA consists of four main operations: initialization, selection, crossover, and mutation. There are many possible versions of each of these operations. We will describe the most simple example of each. In Algorithm 1, we lay out the basic procedure of the simple GA and how each of the four main operations are invoked. As input, the algorithm takes a utility function and a set of four hyperparameters which we will describe as they are used.

The first step in the algorithm (line 1) is to initialize the population with N_p chromosomes. Most commonly, these chromosomes are sampled uniformly at random from the domain of possible chromosomes. Alternatively, if domain knowledge is available, one can employ more sophisticated techniques such as sampling from a particular distribution or inserting baseline solutions. Once the initial population is created, we enter the algorithm's main loop (line 2). Each iteration of this loop represents a generation. For simplicity, we assume a fixed number of generations (N_g). In each generation, the simple GA creates an entirely new population based on the current population. Once created, this new population will completely replace the current population for the next generation. We denote this new population $\mathcal{P}'$ and build this population using the while loop in line 4.

To create the new population, we repeatedly use the remaining three main operators: selection, crossover, and mutation. First, using the selection operator, we select two parent chromosomes p_1 and p_2 (line 5). The most common way to do this is Goldberg's roulette wheel, or fitness proportionate selection. Simply put, we randomly sample two parents weighted linearly by their utility. That is, we give each solution a slice of an imaginary roulette wheel proportionate to its utility. We then spin the wheel twice to choose the parents.

Next (line 6), we use the p_c hyperparameter to decide if we should breed these parents or simply give them a free ticket to the next generation. More precisely, we sample from a Bernoulli distribution centered at p_c (toss a biased coin) and, based on the outcome, either invoke the crossover operator (line 7) or simply return the parents themselves (line 9). The most simple crossover operator is the single-point crossover operator. For this approach, we choose a random point in the chromosome and create children by taking everything before that point from p_1 and everything after from p_2 and vice versa for the other child.

Once we have the set of children, $\mathcal{C}$, the mutation operator is applied to each child (line 10). The most basic mutation operator iterates through each dimension of the given chromosome and, with a very small probability p_m, replaces the value with a uniformly random one. Finally, after mutation, the children are added to the new population (line 11). This process (selection, crossover, and mutation) repeats until we have a new population $\mathcal{P}'$, which is the same size as the old one $\mathcal{P}$. This marks the end of a generation. As the final step in each generation, we kill off the current generation and replace it with the new one (line 12). After all N_p generations have passed, we are left with a population $\mathcal{P}$ containing some of the best chromosomes we have seen so far. Typically, the last step (line 13) is to greedily choose the best chromosome from that population to solve the problem.

Algorithm 1 The Simple GA

```
    Inputs: $u$: Utility function.
    $p_c$: Fixed probability we do crossover.
    $p_m$: Fixed probability we do mutation.
    $N_g$: Number of generations.
    $N_p$: Size of the population.
 1: $\mathcal{P} \leftarrow$ INITIALIZE($N_p$)
 2: for generation in range($N_g$) do
 3:     $\mathcal{P}' \leftarrow \emptyset$
 4:     while $|\mathcal{P}'| \leq N_p$ do
 5:         $p_1, p_2 \leftarrow$ SELECT($u(\mathcal{P})$)
 6:         if Bernoulli($p_c$) then
 7:             $\mathcal{C} \leftarrow$ CROSSOVER($p_1, p_2$)
 8:         else
 9:             $\mathcal{C} \leftarrow p_1, p_2$
10:         $\mathcal{C} \leftarrow$ MUTATE($\mathcal{C}, p_m$)
11:         $\mathcal{P}' \leftarrow \mathcal{C}$
12:     $\mathcal{P} \leftarrow \mathcal{P}'$
13: return $argmax(u(\mathcal{P}))$
```

2.2 Differential Privacy

An increasingly popular approach to utilizing private data while maintaining the privacy of individuals is differential privacy (DP) [17]. In recent years, many organizations such as Google [21], Microsoft [11], Apple [10], and the U.S. Census Bureau [42] have all adopted differential privacy. The mathematical definition of differential privacy guarantees that the information learned from a private database would be approximately the same with or without any single user's participation. Using this notion, data analysts can perform many useful queries on the data while giving an intuitive privacy guarantee to participants.

We work in what is called the central model of DP, where a user sends their data to a trusted curator who collects the data and runs certain private algorithms on the private data. That is, the curator is assumed to have access to the entire private dataset D in plaintext. We denote a general private algorithm as $\mathcal{M}$ that returns some aggregate statistic $\mathcal{M}(D) \in \mathcal{R}$. More formally, differential privacy can be defined as follows.

Definition 2.1 (Differential Privacy)**.** A randomized algorithm $\mathcal{M} : \mathcal{D} \mapsto \mathcal{R}$ is (ϵ, δ)-DP, if for any pair of neighbouring datasets $D, D' \in \mathcal{D}$, and for any $T \subseteq \mathcal{R}$ we have

$$\Pr[\mathcal{M}(D) \in T] \leq e^{\epsilon} \Pr[\mathcal{M}(D') \in T] + \delta. \tag{2.1}$$

The parameter ϵ captures how much sensitive information is leaked by the mechanism $\mathcal{M}$. The lower the value of ϵ, the better the privacy. Typically, values less than $\epsilon = 1$ are considered to provide high privacy. The parameter δ makes it easier to satisfy DP by allowing a small chance of failure in the guarantee. Thus, if $\delta \neq 0$, then we say that the mechanism provides approximate differential privacy. When $\delta = 0$ it satisfies pure differential privacy. It is typical to choose $\delta < 1/n$, where n is the number of elements in the dataset [19]. We say that two datasets are neighbouring if $|D| = |D'| = n$ and $|D \cap D'| = n - 1$. That is, we allow the replacement of a single data point. This is known as the bounded definition of differential privacy.

The most common way to make a function or statistic satisfy differential privacy is to bound the output of the function and add random noise. Specifically, we bound the maximum change in a functions output when changing the input from D to D'. More formally,

Definition 2.2 (Sensitivity)**.** Let $f : \mathcal{D} \mapsto R$. If m is a distance metric between elements of R then the m-sensitivity of f is

$$\Delta_m^{(f)} = \max_{(D,D')} m(f(D), f(D')), \tag{2.2}$$

where (D, D') are pairs of neighbouring datasets.

Typical examples of the distance metric m are the ℓ_1-norm and ℓ_2-norm. For our work, we assume ℓ_1-norm unless otherwise specified. The most common mechanism in differential privacy adds Laplace noise to a function based on the sensitivity and the privacy parameter ϵ. This mechanism is called the Laplace mechanism and is defined as follows.

Definition 2.3 (Laplace Mechanism)**.** Let $f : \mathcal{D} \mapsto R$. The Laplace mechanism is defined as

$$M(D) = f(D) + Lap(\Delta^{(f)}/\epsilon), \tag{2.3}$$

where $\Delta^{(f)}$ represents the ℓ_1 sensitivity of f.

Lemma 2.4 (Privacy of the Laplace Mechanism [19])**.** *The Laplace mechanism as defined above satisfies ϵ-DP*

The Laplace mechanism is particularly useful when R is continuous. For discrete spaces, the exponential mechanism is a more popular choice [44]. We will define this and other mechanisms in Section 6.2.

Finally, we discuss some useful properties of differential privacy. The first is composition. Specifically, if we apply a differentially private mechanism(s) sequentially, the resulting privacy parameter is simply the sum of the privacy parameter in each step. We call this the naive composition theorem.

Lemma 2.5 (Naive Composition [19])**.** *Let $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n)$ be a sequence of ϵ-DP mechanisms applied sequentially and adaptively. Then $\mathcal{M}$ is $n\epsilon$-DP.*

This is the most basic of the DP composition theorems. We will discuss more sophisticated compositions in Section 6.3. Another useful property of differential privacy is the post-processing lemma.

Lemma 2.6 (Post-Processing Lemma [19])**.** *If $\mathcal{M} : \mathcal{D} \mapsto R$, is a ϵ-DP mechanism, then $A \circ \mathcal{M}$ is ϵ-DP where A is an arbitrary function applied on the output of $\mathcal{M}$.*

This lemma states that once a DP result has been published, we can apply any additional processing we wish without affecting the privacy guarantee.

Chapter 3

Problem Statement

3.1 Defining DP Selection

In a selection problem, an analyst is assumed to have a large number of candidate items as input, from which they want to select the best (or the top-k). We define this notion of "best" using a utility function which is also given as an input to the problem. In the private version of this problem, the utility of the items depends on a dataset of sensitive information in some way. For instance, an analyst may want to know the most common level of education among a group of individuals. In this example, the candidate items are the various levels of education, and the utility function would be the count of individuals who obtained this level of education. What constitutes a candidate item can vary but is usually constructed from the sensitive data in some way, such as choosing a row of the database, training a machine learning model, or creating a synthetic database.

More formally, given a set of candidate items $\mathcal{R}$ and corresponding utility function $u : \mathcal{D} \times \mathcal{R} \to R$, the objective is to return a candidate item $r \in \mathcal{R}$ that approximately maximizes $u(D, r)$, whilst satisfying differential privacy. We define the *sensitivity* of this utility function as

Definition 3.1. The sensitivity of a utility function $u : \mathcal{D} \times \mathcal{R} \to R$ is defined as

$$\Delta^{(u)} = \max_{r \in \mathcal{R}} \max_{D, D' \in \mathcal{D}} |u(D, r) - u(D', r)| \tag{3.1}$$

where D, D' are neighbouring datasets.

The only assumptions we make about a given utility function are that the sensitivity in (3.1) is bounded.

The goal of this thesis is to provide a general purpose, optimization algorithm for solving DP selection problems that have large and complex candidate item spaces. We will evaluate this goal experimentally in terms of the utility of the solutions obtained on a variety of problems in Chapter 8. To satisfy our goal of privacy, a complete proof of privacy is included in Section 6.5.

3.2 Example Problem Definitions

Throughout the thesis, we will focus on two specific instances of DP selection with larger candidate item spaces. We consider the convex optimization problem of training a logistic regression model and the more difficult non-convex problem of k-medians.

3.2.1 Logistic Regression

Let $\mathcal{D}$ represent the data domain, where each entry x is a tuple of dimension d, with an integer label y. We consider a dataset $D \sim \mathcal{D}$. We study the task of creating a classifier $h()$ that given an input x can accurately predict the true label y. Specifically, we fit a single logistic regression model with a sigmoid activation function to the dataset D. The model has the following parameters to be learned: a d-dimensional vector of weights, α, and a bias β. Using these parameters, for a given tuple x, the model makes a classification using the following formula

$$h(x) = \frac{1}{1 + e^{-x \cdot \alpha - \beta}} \tag{3.2}$$

The goal in this problem is to find a set of parameters, (α, β) such that the following function is maximized

$$u(D, (\alpha, \beta)) = -\frac{1}{|D|} \sum_{x \in D} \mathbb{I}(\lfloor h(x) \rceil \neq y) \tag{3.3}$$

where $\mathbb{I}$ is an indicator function, $\lfloor h(x) \rceil$ represents rounding $h(x)$ to the nearest integer, and y represents the true label of a point x. This represents the standard zero one loss, i.e., the fraction of miss-classified instances in the training set.

3.2.2 k-Median / Facility Location

The k-median problem has many definitions in the literature as is often confused with the k-medoid and k-means problems. To avoid this confusion, we begin by describing similar problems. The k-means problem aims to find a set of k cluster centers (real-valued vectors) that minimize the Euclidean distance from all data points to their closest center. The k-medoids problem aims to find a set of k data points (from the dataset) or medoids that minimize the distance from all data points to their nearest medoids. The key difference between k-medoids and the k-means problem is that in k-medoids, all cluster centers must be present in the dataset rather than just arbitrary points in Euclidean space. The k-median problem has multiple definitions, but in general, the problem sits somewhere between k-medoids and k-means. In the privacy literature [29, 38], k-median refers to selecting the cluster centers from a finite domain set V. Almost unanimously, the k-median problem considers the ℓ_1 distance metric while the ℓ_2 metric is more common for k-means and k-medoid. Some works consider a variant of k-medians over the reals with the ℓ_1 distance metric. We focus on the former definition where we are given some finite domain V and a private set of demand points $D \subset V$.

Since we are working in the private setting, we generalize this definition to instead consider a private set of demand points and a public set of possible cluster centers. We remark that setting the public set to be the domain V gives the previous definition from the literature. The goal remains to find a set of k points from the public set such that the distance for a member of the private set to its nearest cluster in the public set is minimized. We let V represent the data domain, where each record is a tuple of dimension d. We consider a private dataset $D \subset V$ and a public set of potential medians (facilities) $P \subset V$ such that $P \cap D = \emptyset$.[1] The goal is to choose $M \subset P$ with $|M| = k$ such that the following function is maximized.

$$u(D, M) = -\sum_{x \in D} \min_{y \in M} dist(x, y) \tag{3.4}$$

where we assume $dist$ represents ℓ_1 distance.

[1]This can be relaxed when considering $P = V$

Chapter 4

Related Work

4.1 Differentially Private Selection

The most well-known solution for differentially private selection is the exponential mechanism [44]. The exponential mechanism assigns a probability to each candidate item based on its utility, scaled by the privacy budget and sensitivity. Under this regime, high utility solutions are exponentially more likely to be chosen, resulting in strong accuracy guarantees. Recently, McKenna and Sheldon introduced an improved version of the exponential mechanism called the permute and flip mechanism [43]. This algorithm always improves the utility of the exponential mechanism up to a factor of two. A common alternative to the exponential mechanism is the report noisy max mechanism by Dwork and Roth [19]. In this mechanism, we add Laplacian noise to the utility of each candidate before choosing the max. Recent work has shown that both the exponential mechanism and permute and flip are equivalent to the noisy max algorithm with Gumbel and exponential noise respectively [12, 16].

A major limitation of the exponential mechanism and report noisy max is the utility of every candidate must be computed and considered for selection. If the domain of candidate items is large, this results in prohibitively high computation complexity. One attempt to address this issue was the sub-sampled exponential mechanism, which applies the exponential mechanism to a random sample of the domain [38]. The limitation of this work is that high-quality solutions are often discarded in the sampling. In a similar light, Durfee and Rogers investigate solving the top-k selection problem for counting queries without considering the entire domain [16]. They introduce a plethora of novel techniques along the way, which we take advantage of in our work. The limitations of this work is that they assume the domain of candidates is sorted in order of utility, and there is a non-trivial chance that no solution is returned.

All the above work considers the non-interactive setting of differentially private selection. That is, we assume all candidates are present at the start of the decision process. An interesting problem is how to choose the best candidate in an online setting. The most common solution is to use the spare vector technique (SVT) introduced by Dwork et al. [18]. In this online setting, it is impossible to know when we have encountered the best solution. Instead, we assume there exists a publicly known threshold, and SVT returns the candidate(s) whose utility exceeds said threshold. The novelty of SVT is that the privacy budget depends only on the number of queries above the threshold. In this work, we focus solely on the non-interactive setting of differentially private selection.

4.2 Differentially Private Nature-Inspired Algorithms

Very few works have focused on nature-inspired algorithms under the constraints of differential privacy. Most similar to our work is PrivGene, a work by Zhang et al. [61]. This work was the first and only work to propose a differentially private genetic algorithm in the literature. We explain the algorithm in detail and compare our work to PrivGene in Section 8.1. Although no other works have studied differentially private genetic algorithms, others have evaluated PrivGene in follow-up work [39, 54]. Most notably, Su et al. [54] thoroughly evaluated PrivGene and pointed out several flaws in the algorithm that caused it to perform poorly. Specifically, they mention how crossover often does not result in reasonable solutions and that the number of selections requires too much privacy budget. We address these issues in our work, reducing the amount of crossover and optimizing the privacy budget consumption. Su et al. [54] were also the first to formalize PrivLocal, a local search algorithm whose source code was published with the PrivGene paper [61]. While this algorithm is not a genetic algorithm, it does yield impressive performance. We evaluate the performance of PrivLocal in Section 8.2.1.

While PrivGene has been the only work investigating differentially private genetic algorithms, other works have made use of swarm intelligence and evolutionary computation alongside differential privacy. Zorarpacı and Özel used the artificial bee colony algorithm and differential evolution to improve the classification accuracy of differentially private 1R classification [63]. However, they applied these techniques non-privately as a pre-processing step and did not attempt to make either of these nature-inspired techniques private. Liu and Hui study the use of the bat algorithm to provide a differentially private consensus protocol for multi-agent systems [40]. We instead focus on the problem of private selection in the central model of differential privacy.

4.3 Differentially Private Logistic Regression

Logistic regression is arguably one of the most commonly used examples of a convex optimization problem. Most works do not typically aim to solve the problem of differentially private logistic regression. Instead, they use it as one of the multiple examples to evaluate the effectiveness of their optimization algorithm. We discuss the three major categories of techniques in the differentially private optimization space: output, objective, and gradient perturbation. Chaudhuri et al. [6] first introduced output and objective perturbation as solutions for differentially private empirical risk minimization. They showed that objective perturbation was the most effective algorithm that they introduced. Following up on this work, Kifer et al. [36] provided an approximate-DP algorithm that improves the utility and applicability of Chaudhuri et al.'s work. While these algorithms are strong theoretically, they rely on finding the exact minimum to be private, limiting their effectiveness in practice [31].

A more popular approach is to use a private version of the gradient decent algorithm, which adds noise to the gradients to preserve privacy. Gradient perturbation was first suggested by Williams and McSherry [57] and later formally introduced by Song et al. [52]. Bassily et al. significantly developed this algorithm and provided optimal lower bounds for the problem [4]. Arguably the most significant development in gradient perturbation was the moments accountant of Abadi et al., which scaled gradient perturbation to practical deep neural networks [1].

Recent work by Iyengar et al. [31] introduces a new technique for differentially private convex optimization, related to objective perturbation, called approximate minima perturbation. Aside from the novel algorithm, one of the significant contributions of this work is an extensive performance benchmark of state-of-the-art algorithms from the three major areas of work in differentially private convex optimization (output, objective, and gradient perturbation). We use this publicly available benchmarking codebase to evaluate the performance of our techniques in Section 8.4.1.

In addition to the approaches mentioned above, we discuss some other notable works that have been used to solve logistic regression. The Frank-Wolfe algorithm [26] is a popular iterative first-order optimization algorithm that was made private by Talwar et al. in 2016 [55]. The functional mechanism, introduced by Zhang et al. [62], provides a different approach to objective perturbation utilizing the Laplace mechanism. This technique is limited to polynomial objective functions with bounded degree, and thus to solve logistic regression, they use a Taylor series approximation. Mohan et al. developed GUPT [47], a general-purpose DP algorithm based on the sample and aggregate framework [51]. The algorithm breaks a given problem into sub-problems which can be solved non-privately before taking a differentially private average of the results. We further evaluate the DP Frank Wolfe algorithm in section 8.4.1 using Iyengar et al.'s benchmark [31]. Zhang et al. [61] showed that PrivEEM consistently outperforms both GUPT and the functional mechanism, so we exclude these from our evaluation.

4.4 Differentially Private k-Median

We recall that we focus on the k-median problem over a finite domain as defined in Section 3.2. The first work to define the k-median problem in a private setting was Gupta et al. [29]. Gupta et al. presented a differentially private local search algorithm inspired by the non-private local search of Arya et al. [2]. To adapt this search to the private setting, Gupta et al. simply apply the exponential mechanism to decide which is the best swap in each iteration [29]. Following up on this work, Lantz et al. used the k-median problem to show the effectiveness of the sub-sampled exponential mechanism [38]. Recall that this mechanism randomly samples the set of possible solutions before applying the standard exponential mechanism. Lantz et al. show that their approach outperforms Gupta et al.'s in both time and utility [38].

Finally, the most recent work in this space is that of Jones et al. [32]. Jones et al. create a differentially private coreset (a small synthetic dataset optimized for clustering) and then apply any off-the-shelf non-private algorithm as post-processing [32]. This technique first uses the unweighted set cover algorithm of Gupta et al. [29] to select candidate cluster centers whose epsilon balls contain many private data points. They then map the private data points onto the candidate cluster centers using the Laplace mechanism. We evaluate all three of these mechanisms (Gupta, Lantz, and Jones) against our work in Chapter 8. Nguyen et al. [48] take a similar approach to Jones et al. [32] but instead focus on the k-means objective. The main differences between the two works are how they tailor the discretization and maximum coverage steps to the euclidean k-means problem.

A closely related problem, and a typical application of the k-median problem, is the facility location problem. In this problem, there is a set of clients (private data points) and facilities (candidate cluster centers). The goal is to choose a set of k facilities such that the distance from clients to their closest facility is minimized. Gupta et al. were once again the first to consider this problem in the private setting [29]. They first gave a lower bound on the approximation ratio of this problem under differential privacy. Motivated by this, they solve a more general version of the problem that outputs a superset of the facilities to be opened. Esencayi et al. also considered the more general problem definition of Gupta et al. and provided improved algorithms to further reduce the approximation ratio under differential privacy [22]. In this work, we focus exclusively on the vanilla k-median problem.

Multiple works have used the k-medians algorithm of Gupta et al. [29] as a building block for solving the euclidean k-means problem under differential privacy. For example, Balcan et al. [3] solves the euclidean k-means (and k-median) problem by first discretizing the euclidean space, then applying the techniques of Gupta et al. Kaplan and Stemmer [34] later improved on this work using locality-sensitive hashing to improve the discretization step. Both works focus on ways to apply Gupta et al.'s local search to the euclidean space, and thus we exclude them from our evaluation since our space is already discrete.

There are other notable works that solve variants of the k-median problem. For example, Cohen et al. study a simplified clustering problem, which they call k-tuples clustering, where the data is assumed to be distributed in k well separated clusters [8]. Feldman et al. [23] solves the differentially private corset problem in two-dimensional Euclidean space, which can be post-processed to get a solution for k-medians clustering. The coreset idea was later improved and applied to k-means problem [24]. Ghazi et al. [27] focus on the densest ball problem, which has many applications, including solving the k-means problem. For our evaluation, we focus on the literature that solves the vanilla k-median problem over a discrete solution space [29,32,38]. Applying our algorithm to other variations of the k-median problem mentioned above would be an interesting line of future work.

Chapter 5

Building a Simple Genetic Algorithm

In this chapter, we first outline how to solve our example problems defined in Section 3.2 using a simple GA. Then, we explain how we will empirically evaluate the effectiveness of the GA in solving these problems.

5.1 Encodings and Basic GA Operations

Encoding Solutions Recall that to apply a genetic algorithm to a given problem, we require two main things. First, a utility function, which we have defined in Section 3.2. Second, an encoding of potential solutions to the problem. For the problem of logistic regression, recall that a solution to this problem is a set of parameters (α, β). To encode them, we simply concatenate α and β to obtain a real valued vector θ of length $d + 1$. We restrict each entry $\theta_i \in [-1, 1]$. For the k-median problem, a solution is a representation of the set M of k unique data points representing the best medians. We encode this as a vector of length k where each entry is an integer representing the index of a tuple in P.

Initialization Operators Recall that this operator is used to create the initial populations that the GA will update over time. We mentioned that the most popular method is to randomly initialize the population unless one has problem-specific background information. For logistic regression, starting from a solution of all zero weights is common as this represents the center of the sigmoid activation function. However, initializing all chromosomes to zero vectors would not allow for much diversity in the early generations. As such, we generate 95% of the population uniformly at random (such that $\theta_i \in [-1, 1]$) and insert zero vectors as the remaining 5%. For the k-median problem, there is no well-known initialization technique that does not use the private data, and thus we use a uniformly random initialization.

Crossover Operators The crossover operator is used to combine two parent solutions into one or two child solutions. We use a standard crossover operator called uniform crossover (Algorithm 2), where each parameter value is equally likely to come from either parent. This can be thought of as a generalization of single-point crossover with the maximum number of crossover points. For the k-median problem, we note that crossover can introduce repeated entries in the chromosome, which is no longer a valid solution ($|c| \neq k$). We use a variant of uniform crossover to address this to ensure that the $|c| = k$, i.e., there are no repeated entries. Specifically, we take two parent solutions p_1, p_2 and obtain the set union of their entries, then uniformly sample a child $c \sim p_1 \cup p_2$ without replacement.

Algorithm 2 Uniform Crossover

```
1: function UNIFORM_CROSSOVER($p, q$)
2:     for $i$ in range($d + 1$) do
3:         if BERNOULLI(0.5) then
4:             $c_1[i] \leftarrow p[i]$, $c_2[i] \leftarrow q[i]$
5:         else
6:             $c_1[i] \leftarrow q[i]$, $c_2[i] \leftarrow p[i]$
7:     return $c_1, c_2$
```

Mutation Operators The mutation operator takes as input a single chromosome. It iterates over each dimension of the chromosome vector and with a certain probability p_m replaces this value. For logistic regression, we add Gaussian noise of scale 0.1 to the current value and truncate if the new value falls outside of $[-1, 1]$. For k-median, we replace the current index with the index of a new uniformly random tuple from P. If the randomly selected index already exists in the set M, we simply re-sample until $|c| = k$.

Default Hyperparameters For the initial experiments, we must choose a default set of hyperparameters. In the following chapters, we will tune these parameters for privacy and set better defaults. However, the following parameters given in Table 5.1 are a good starting point.

Parameter	Value
N_p	200
N_g	100
N_s	10
p_c	0.5
p_m	$\frac{1}{\lvert c \rvert+1}$

Table 5.1: Default hyperparameters where $|c|$ is the length of the chromosome.

5.2 Experiment Details

As is the case with deep neural networks, it is often difficult to prove approximation ratios for a GA. Thus, our main line of evaluating how well we solved a given problem is the empirical utility of the solutions. In this section, we lay out the experimental setup that will be used for the evaluation of various design choices as well as the comparison to related work in Chapter 8.

Datasets To instantiate our example problems, we use a series of datasets from the UCI machine learning repository [14]. The properties of these datasets are summarized in Table 5.2.

Dataset	Dimension (after encoding)	Number of Records
Adult	104	48842
Credit	24	30000
Spam	57	4601
Mushrooms	107	8214

Table 5.2: Dataset sizes.

We give a brief synopsis of each dataset.

- **Adult:**[1] The Adult dataset is a set of records extracted from the 1994 US Census. It contains attributes such as age, education, and occupation with the classification task to predict whether the individual made more or less than $\$50,000$ in salary.

[1]https://archive.ics.uci.edu/ml/datasets/adult

- **Credit** [59]:[2] The Credit dataset contains records of customers from a Taiwanese credit card company. It contains attributes such as credit limit, education, and payment history with the classification task to predict if the customer defaulted on their credit card payment this month.

- **Spam:**[3] The Spambase dataset is a collection of emails from the Hewlett-Packard company. It contains attributes such as the frequency of certain words and length of character sequences with the classification task to predict of the email was spam or not.

- **Mushrooms:**[4] The Mushroom dataset contains hypothetical samples of different types of mushroom. It contains attributes such as cap shape, odor, and habitat with the classification problem of predicting if the mushroom is poisonous or not.

We preprocess these datasets by replacing the missing values with the mean or mode of the column. All data is normalized using min-max scaling such that each column is in the range $[0, 1]$. Categorical attributes are converted to numerical ones using a one-hot or binary encoding. For the logistic regression problem, all datasets have a binary classification task. We split the datasets uniformly at random into a training and test set using an $80 : 20$ train to test ratio. For all evaluations, we will report the utility on the testing set. For k-median, we disregard the classification label and use the entire dataset as the private dataset. To create the public set, we sample additional points uniformly from the range of the attributes such that we achieve an $80 : 20$ private to public data ratio. We note that if we were to instead split the dataset into a public and private set, the public set would be similarly distributed, and thus one would not need the private set to solve the problem.

We consider a baseline and a non-private solution for each evaluation to give reference minimum and maximum performance. For logistic regression, the baseline is a solution of all zero vectors. Whereas for k-median, we sample 1000 solutions uniformly at random and plot the average utility. As non-private algorithms, we use Scikit-learn packages. For logistic regression, we use the default logistic regression model with the stochastic average gradient solver.[5] Since our definition of k-median is specific to the privacy literature, finding a non-private baseline is non-trivial. The problem is closer to the k-medoid problem than

[2] https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

[3] https://archive.ics.uci.edu/ml/datasets/spambase

[4] https://archive.ics.uci.edu/ml/datasets/mushroom

[5] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

the k-means problem as we are presented with a finite list of possible cluster centers. Hence, algorithms for the k-medoids problem can be most easily adapted to this setting. As a non-private baseline, we consider the partitioning around medoids (PAM) algorithm of Kaufman and Rousseeuw [35]. This algorithm is implemented in Scikit-learn.[6] We adapt this implementation so that it considers distances between a private and public set (rather than distances between points in the same set). The only modifications we made were to the distance matrix such that it is no longer square and changing the indexing appropriately. We make all source code, including this modification, publicly available[7].

[6]https://scikit-learn-extra.readthedocs.io/en/stable/generated/sklearn_extra.cluster.KMedoids.html

[7]https://git.uwaterloo.ca/t3humphr/dp-simple-ga

Chapter 6

Adding Privacy to the Simple Genetic Algorithm

In this chapter, we discuss how we modify the simple GA (described in Chapters 2 and 5) in order to make it private. Recall the simple GA is composed of four major operators: initialization, selection, crossover, and mutation. Out of all the operators, the only one that uses the utility of the chromosomes is the selection operator. Thus, the selection operator will be our main focus when making the GA private. A naive approach to making the selection operator private would be to simply swap the standard selection techniques (such as roulette wheel) with a DP alternative (such as the exponential mechanism). We begin by detailing the pitfalls of this naive approach and then present our solution.

When using roulette wheel, we sample a fresh chromosome every time we call the select operator. This high number of selections is in part responsible for why roulette wheel is so effective. A critical issue in genetic algorithms is premature convergence. This happens when the population is not diverse enough and converges to local optima before fully exploring the space. Making so many probabilistic selections gives the GA many opportunities to select a less than optimal chromosome and thus helps maintain the population's diversity. On average, the GA will still choose high utility chromosomes, which tends to increase the utility of the overall population.

Using an off-the-shelf roulette wheel selection leads to two major issues when applying DP. First, adding DP to this selection, inherently increases the chance of picking lower utility solutions as we do not want to depend too heavily on the utility function. This can result in too much exploration or diversity and not enough exploitation of good solutions to drive the population forwards. Second, when using differential privacy, each selection consumes part of the privacy budget. In the non-private setting, the simple GA involves selecting on the order of the population size multiplied by the number of generations. This results in a prohibitively high privacy cost which destroys the utility of the final result.

6.1 Proposed Solution

To overcome the drawbacks of a high number of selections (high privacy budget and too much diversity), we use a less common selection technique, truncation selection [50]. Truncation selection chooses a pool of the top-N_s chromosomes in each generation. To create the new population, it samples uniformly at random from this pool to obtain the parent chromosomes for crossover and mutation. Truncation selection is not popular in the non-private setting as it does not allow for population diversity. However, we have found that this is not a concern since we introduce additional diversity by using DP. Truncation selection is known to be a very efficient selection algorithm [9]. This is an appealing property when considering the composition of DP as we don't need to make and compose as many selections.

Starting with the simple GA using truncation selection, we make a few additional changes to simple GA to make it private. We detail our complete solution in Algorithm 3. We explain the key differences between our private algorithm and the standard simple GA (Algorithm 1). The additional inputs over the simple GA are the number of parent chromosomes for truncation selection N_s and privacy parameters ϵ, δ. Furthermore, our algorithm requires a DP selection mechanism `DPSelect` and a DP composition theorem `DPComp` (chosen in Section 6.4). The first difference in the algorithm is lines 2 and 3, where we compute the privacy budget for each selection using the chosen composition theorem. We conduct a binary search over ϵ_s with n compositions until we reach the desired total privacy budget ϵ. We use a binary search as most composition theorems do not have a closed-form inverse.

With the privacy budget computed, we choose the top N_s solutions, the first part of truncation selection, in lines 6 and 7. In line 8, we initialize the new population using these candidates. This technique is known as elitism [46], where we preserve the best candidates between generations. Elitism is commonly added to simple GAs to increase exploitation, which helps us offset the effects of DP on the GA. The rest of the algorithm proceeds exactly the same as the simple GA, until the last step in line 18. In the simple GA, we would simply return the candidate of maximum utility, whereas in our case, we must use a DP mechanism to return the noisy max. We note that in line 19 we expand the definition of the mutation function for completeness; however, this is the standard definition used in the non-private simple GA.

Algorithm 3 DPSGA

```
    Inputs: $u$: Utility Function.
    $p_c$: Fixed probability we do crossover.
    $p_m$: Fixed probability we do mutation.
    $N_g$: Number of generations.
    $N_p$: Size of the population.
    $N_s$: Number of chosen parents.
    DPComp, DPSelect: DP composition and selection mechanisms.
    $(\epsilon, \delta)$: Privacy Parameters.
 1: $\mathcal{P} \leftarrow$ INITIALIZE($N_p$)
 2: $n \leftarrow N_g * N_s + 1$
 3: $\epsilon_s \leftarrow$ BinarySearch(DPComp, $n, \epsilon, \delta$)
 4: for generation in range($N_g$) do
 5:     $\mathcal{B} \leftarrow \emptyset$
 6:     for selection in range($N_s$) do
 7:         $\mathcal{B} \leftarrow$ DPSelect($u(\mathcal{P} - \mathcal{B}), \epsilon_s$)
 8:     $\mathcal{P}' \leftarrow \mathcal{B}$
 9:     while $|\mathcal{P}'| \leq N_p$ do
10:         $p_1, p_2 \leftarrow$ UniformRandomSelect($\mathcal{B}$)
11:         if Bernoulli($p_c$) then
12:             $\mathcal{C} \leftarrow$ UniformCrossover($p_1, p_2$)
13:         else
14:             $\mathcal{C} \leftarrow p_1, p_2$
15:         $\mathcal{C} \leftarrow$ MutateAll($\mathcal{C}, p_m$)
16:         $\mathcal{P}' \leftarrow \mathcal{C}$
17:     $\mathcal{P} \leftarrow \mathcal{P}'$
18: return DPSelect($u(\mathcal{P}), \epsilon_s$)

19: function MutateAll($\mathcal{C}, p_m$)
20:     for $c$ in $\mathcal{C}$ do
21:         for $i$ in range($|c|$) do
22:             if Bernoulli($p_m$) then
23:                 $c[i] \leftarrow$ Mutate($c[i]$)
24:     return $\mathcal{C}$
```

The remaining sections will investigate the best ways to instantiate our algorithm using differentially private mechanisms. We will begin by considering the DP mechanisms we will use as building blocks in Section 6.2, followed by how to compose these mechanisms in Section 6.3. In Section 6.4 we conduct an empirical evaluation to choose the best combination of techniques to use for our DP simple GA. Finally, in Section 6.5 we prove the privacy of our simple GA under the leading composition techniques.

6.2 Different DP Selection Mechanisms

We have argued that truncation selection is a promising solution for making the simple GA private. In order to make truncation selection private, we need to choose the top-N_s chromosomes in a privacy-preserving manner. The top-k problem is a common problem in the DP literature [16], so there are several potential approaches to consider. For simplicity, we consider popular mechanisms that return a single element of maximal utility. More formally, each mechanism probabilistically maximizes a utility function, $u : \mathcal{D} \times \mathcal{R} \to \mathbb{R}$, that indicates how advantageous the pairing of given inputs and outputs $(D, r) \in \mathcal{D} \times \mathcal{R}$ is for the particular usage scenario. To select multiple elements, we will employ a peeling version of each mechanism in a similar style to Durfee and Rogers [16]. That is, we select the best candidate using the chosen mechanism then "peel off" (remove) this candidate from the pool before selecting again (Algorithm 3 lines 6-7).

We consider three state-of-the-art techniques in non-interactive DP selection: The exponential mechanism [44], report noisy max [19], and permute and flip [43].[1] We begin by defining each mechanism and the corresponding DP guarantees.

Exponential Mechanism The exponential mechanism, introduced by McSherry and Talwar, is the most common approach for selecting an outcome of maximal utility [44]. Under the exponential mechanism, the utility function exponentially affects the probability of selecting a given output; the higher the utility, the larger the chance of selection. More formally,

[1]We note that SVT is another common choice for DP selection in the interactive setting that uses a public threshold. However, it is not a natural choice since we are in the non-interactive setting and do not have an obvious threshold. Furthermore, Lyu et al. [41] show that the exponential mechanism outperforms SVT queries for top-k style queries in the non-interactive setting.

Definition 6.1 (Exponential Mechanism [44])**.** The *exponential mechanism* defines a probability distribution in which each output r, is sampled with the following probability:

$$Pr[r] = \frac{\exp\left(\frac{\epsilon u(D,r)}{2\Delta^{(u)}}\right)}{\sum\limits_{i \in \mathcal{R}} \exp\left(\frac{\epsilon u(D,i)}{2\Delta^{(u)}}\right)} \tag{6.1}$$

where $\Delta^{(u)}$ is the sensitivity of the utility function and ϵ is the differential privacy parameter.

Lemma 6.2 (Theorem 6 [44])**.** *The exponential mechanism guarantees ϵ-differential privacy*

Report Noisy Max Report noisy max is a simple algorithm originally proposed to return the largest counting query from a list [19]. We consider the more general version given in Algorithm 4 that maximizes a given utility function u. The mechanism simply computes the utility of each outcome in $\mathcal{R}$ and adds Laplace noise, then returns the outcome with the largest *noisy* utility.

Algorithm 4 Report Noisy Max [19]

```
1: for $r$ in $\mathcal{R}$ do
2:     $q_r \leftarrow u(D,r) + Lap(\frac{2\Delta^{(u)}}{\epsilon})$
3: return $argmax_r q_r$
```

Lemma 6.3 (Claim 3.9 [19])**.** *Report noisy max guarantees ϵ-differential privacy*

Permute and Flip Mechanism McKenna and Sheldon recently introduced a new mechanism for differentially private selection called the permute and flip mechanism [43]. We describe the algorithm in Algorithm 5. Intuitively, this mechanism is very similar to the exponential mechanism, with the key difference being the permutation (which mimics sampling without replacement). McKenna and Sheldon show that this new mechanism never has a worse utility than the exponential mechanism and can improve upon it by up to a factor of 2 [43, Theorem 2].

Algorithm 5 Permute and Flip [43]

```
1: $u_* \leftarrow \max_{r \in \mathcal{R}} u(D, r)$
2: for $r$ in RandomPermutation($\mathcal{R}$) do
3:     $p_r \leftarrow \exp\left(\frac{\epsilon}{2\Delta^{(u)}}(u(D, r) - u_*)\right)$
4:     if $Bernoulli(p_r)$ then
5:         return r
```

Lemma 6.4 (Theorem 1 [43])**.** *The permute and flip mechanism guarantees ϵ-differential privacy*

Implementation We note that both the exponential mechanism and permute and flip can be equivalently implemented using a noisy max algorithm. For the exponential mechanism, Durfee and Rogers proved that using noisy max with Gumbel noise is equivalent to the exponential (and peeling exponential) mechanism [16, Lemma 4.2]. Ding et al. recently proved that the permute and flip mechanism is equivalent to the noisy max with exponential noise [12, Theorem 5].[2] We implement both the exponential mechanism and permute and flip with their noisy max counterparts to improve efficiency as well as circumvent precision issues when the loss is very high. That is, we follow Algorithm 4 replacing the Laplace noise with the following distributions. For the exponential mechanism, we sample from a Gumbel distribution with the following PDF

$$p_{Gumbel}(z; b) = \frac{1}{b} \cdot \exp\left(-(z/b + e^{-z/b})\right). \tag{6.2}$$

We set $b = 2\Delta^{(u)}/\epsilon$ following Durfee and Rogers [16, Lemma 4.2]. For the permute and flip mechanism, we sample from an exponential distribution which has the following PDF

$$p_{Expo}(z; b) = b \cdot \exp(-bz). \tag{6.3}$$

We set $b = \epsilon/2\Delta^{(u)}$ following Ding et al. [12, Lemma 2].

6.3 Composition Theorems

Now we have defined the various mechanisms we will consider, we investigate the best way to compose these mechanisms in order to use them repeatedly. There are many ways

[2]It is currently an open question whether Gaussian noise could be used in a noisy max algorithm. However, lower bounds on the DP selection problem suggest the mechanisms we have presented are essentially optimal [12, 53]

to compose a differentially private mechanism under a variety of definitions of differential privacy. We recall that all the mechanisms we have considered satisfy the basic definition of pure differential privacy. Thus, we begin by considering various ways to compose general, pure, differentially private mechanisms.

In its most simple form, a composition theorem tells us how much privacy budget we will incur if we run a mechanism n times. The most basic (and an upper bound for all compositions) is naive composition as defined in Section 2.2. Each theorem below is significantly tighter than the naive composition theorem. That is, the overall epsilon will be less than if we were to simply sum the epsilon of each part. We consider the case of adaptive composition, which allows each run of the mechanism to depend arbitrarily on the previous run. We let ϵ_s denote the privacy budget of a single run of a specific mechanism. We consider the simplest case where ϵ_s is fixed across all n runs of the mechanism. For each composition theorem, we state the total epsilon ϵ required for all runs. The cost of this tighter composition is relaxing the overall privacy guarantee from pure to approximate DP. Thus, each composition theorem includes a δ which we fix as $1/|D|^{1.1}$ to satisfy the rule of thumb that delta should be less than $1/|D|$. We list (ϵ, δ)-DP guarantees of each composition technique below.

6.3.1 Composing Differentially Private Mechanisms

We consider two methods of composing an arbitrary DP mechanism, advanced composition [20, 33] and the moments accountant [1, 45].

Advanced Composition We begin with the most well known composition theorem (aside from naive composition) introduced by Dwork et al. [20].

Lemma 6.5 (Advanced Composition [20]). *The adaptive composition of a ϵ_s-DP mechanism under n-fold adaptive composition is (ϵ, δ)-DP with*

$$\epsilon = \epsilon_s\sqrt{2n \ln 1/\delta} + n\epsilon_s(e^{\epsilon_s} - 1) \tag{6.4}$$

Kairouz et al. improved this bound and proved the optimal advanced composition theorem for a general (ϵ, δ)-DP mechanism [33]. They remark that the optimal bound is hard to use in practice as it does not have a closed-form expression. Kairouz et al. then give a simplified theorem that provides a slightly looser bound with a closed-form expression described below.

Lemma 6.6 (Theorem 3.4 [33]). *The adaptive composition of a ϵ_s-DP mechanism under n-fold adaptive composition is (ϵ, δ)-DP with*

$$\epsilon = \min\left\{ n\epsilon_s, \tfrac{(e^{\epsilon_s}-1)\epsilon_s n}{(e^{\epsilon_s}+1)} + \epsilon_s\sqrt{2n\ln\left(e + \frac{\sqrt{n\epsilon_s^2}}{\delta}\right)}, \tfrac{(e^{\epsilon_s}-1)\epsilon_s n}{(e^{\epsilon_s}+1)} + \epsilon_s\sqrt{2n\ln\left(\tfrac{1}{\delta}\right)} \right\} \quad (6.5)$$

The Moments Accountant Concentrated differential privacy (CDP) and Renyi differential privacy (RDP) are popular privacy definitions used for the composition of DP mechanisms that take advantage of the Renyi divergence to give bounds [5, 45]. The two definitions are very similar in that they both bound the Renyi divergence; however, CDP bounds all moments and RDP is defined for a specific alpha. We focus on the RDP as it tends to give more accurate analysis [45]. Furthermore, composing mechanisms using RDP is equivalent to the moments' accountant of Abadi et al. [1]. To use RDP as a composition theorem, we must first provide a conversion from pure DP to RDP. We can then use the composition theorems of RDP to compose the mechanisms and finally convert back to approximate DP.

We begin by defining RDP. To define RDP we must first recall Renyi divergence.

Definition 6.7. Renyi Divergence [49] For two probability distributions P and Q defined over $\mathcal{R}$, the Renyi divergence of order $\alpha > 1$ is

$$D(P\|Q) = \frac{1}{1-\alpha}\ln \mathop{\mathbb{E}}_{x\sim Q}\left(\frac{P(x)}{Q(x)}\right)^{\alpha} \quad (6.6)$$

where $P(x)$ is the density of P at x.

Definition 6.8. (RDP [45]) An algorithm A is said to be (α, ϵ)-RDP if for all neighboring databases $D, D' \in \mathcal{D}$

$$D(A(D)\|A(D')) \leq \epsilon \quad (6.7)$$

The first step is to convert the arbitrary DP mechanism to the RDP definition. To do this we take advantage of an intermediary result from Bun and Steinke [5, Proposition 19]. Specifically,

Lemma 6.9 (From DP to RDP [5]). *If a mechanism satisfies ϵ_s-DP, it also satisfies $(\epsilon(\alpha), \alpha)$-RDP with*

$$\epsilon(\alpha) = \frac{1}{\alpha-1}\ln\left(\frac{\sinh(\alpha\epsilon_s) - \sinh((\alpha-1)\epsilon_s)}{\sinh(\epsilon_s)}\right) \quad (6.8)$$

After we have converted to RDP, we can take advantage of the following composition theorem (similar to naive composition for DP).

Proposition 6.10 (Proposition 1 [45])**.** Consider two mechanism F and G. If F is (α, ϵ_1)-RDP and G is (α, ϵ_2)-RDP then $F(G(x))$ is $(\alpha, \epsilon_1 + \epsilon_2)$-RDP

Finally, we need to convert the final result to approximate DP as follows.

Proposition 6.11 (Proposition 3 [45])**.** If a mechanism satisfies (α, ϵ)-RDP, then it is also $(\epsilon + \frac{\ln 1/\delta}{\alpha-1}, \delta)$-DP for any $\delta \in (0, 1)$.

A natural question to ask is how to set α? The answer is that we consider choosing α arbitrarily such that the overall ϵ is minimized. Mirnov showed that in practice, it suffices to consider a restricted set of alphas and compute an approximate minimum [45]. In our implementation, we take this approach, brute-forcing over a small set of alphas to find the empirical minimum. Putting all of these steps together, we obtain the following end-to-end composition theorem.

Lemma 6.12. *The adaptive composition of a ϵ_s-DP mechanism under n-fold adaptive composition is (ϵ, δ)-DP with*

$$\epsilon = \min_{\alpha} \left\{ \frac{n}{\alpha - 1} \ln \left(\frac{\sinh(\alpha\epsilon_s) - \sinh((\alpha-1)\epsilon_s)}{\sinh(\epsilon_s)} \right) + \frac{\ln 1/\delta}{\alpha - 1} \right\} \tag{6.9}$$

Proof. The result follows by first applying Lemma 6.9, then multiplying by n as per Proposition 6.10. Finally, we apply Proposition 6.11 and minimize over α. □

6.3.2 Bounded Range Composition

When considering a particular DP mechanism, one can often tighten the composition even further. This is typically done by creating a tighter privacy proof of a particular mechanism under an alternate privacy definition such as RDP, then taking advantage of the composition techniques of that definition. We are not aware of any work proving the privacy of our mechanisms under RDP or CDP (outside of the general formula stated in Section 6.3.1). However, the exponential mechanism, in particular, has a tighter analysis under what is called bounded range DP. Bounded range DP was first introduced by Durfee and Rogers in their work on top-k queries [16].

Definition 6.13 (Range-Bounded [16])**.** Given a mechanism $\mathcal{M}$ that takes a collection of records in $\mathcal{D}$ to some outcome set R, we say that $\mathcal{M}$ is ϵ-range-bounded (ϵ-BR) if for any $y, y' \in R$ and any neighboring databases D, D' we have

$$\frac{\Pr[\mathcal{M}(D) = y]}{\Pr[\mathcal{M}(D') = y]} \leq e^{\epsilon} \frac{\Pr[\mathcal{M}(D) = y']}{\Pr[\mathcal{M}(D') = y']}. \tag{6.10}$$

This definition says that in addition to the distribution of outputs being similar across neighbouring databases, the mechanism must also offer a similar distribution over the outputs themselves. Bounded Range is a general notion of privacy, but it is particularly useful for exponential mechanisms. All ϵ-DP mechanisms satisfy 2ϵ-BR; however, the exponential mechanism enjoys a tighter analysis given below.[3]

Lemma 6.14 (Lemma 4.3 [16])**.** *The exponential mechanism (Definition 6.1) is ϵ-BR.*

Using this more restrictive form of DP allows one to prove tighter composition bounds. Durfee and Rogers [16] showed this in their initial work, which was later tightened by Dong et al. [13]. Dong et al. proved an optimal bound for the composition of ϵ-BR mechanisms [13]. However, as was the case with advanced composition, this optimal bound did not have a closed form expression. We instead use a preliminary result with a closed form expression that was proven by computing the supremum of the KL divergence.

Lemma 6.15 (Proposition 4 [13])**.** *The adaptive composition of a ϵ_s-BR mechanism under n-fold adaptive composition is (ϵ, δ)-DP with*

$$\epsilon = \min \left\{ n\epsilon_s, n\left(\frac{\epsilon_s}{1 - e^{-\epsilon_s}} - 1 - \ln\left(\frac{\epsilon_s}{1 - e^{-\epsilon_s}}\right)\right) + \sqrt{\frac{n\epsilon_s^2}{2}\ln(1/\delta)} \right\}. \tag{6.11}$$

6.3.3 Evaluation of Composition Theorems

We have presented several composition theorems. However, it is not apparent which are best. In this section, we will attempt to narrow down the list of composition theorems. In Section 6.4 we will perform an empirical evaluation to further narrow down which DP mechanism and composition theorem is best. To narrow down the composition theorems, we will plot the per selection privacy budget ϵ_s for a given overall privacy budget ϵ and

[3]A natural question to ask is if the permute and flip mechanism also enjoys a similar analysis. A recent blog post by Durfee and Rogers shows that is likely not the case as the permute and flip mechanism has a lower bound close to 2ϵ-BR (the value that applies to all ϵ-DP algorithms) [15].

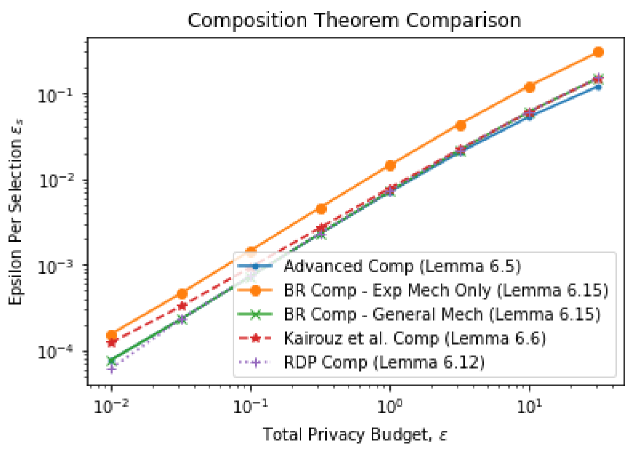


Figure 6.1: Comparison of the privacy budget available for each selection under various composition techniques.

a typical run of our simple GA. That is we let $n = 1000$ (100 generation each with 10 selections) and fix $\delta = 10^{-5}$. To compute the corresponding ϵ_s, we simply binary search, evaluating the equation given by each lemma and plot the results in Figure 6.1. We remark that a higher ϵ_s means more privacy budget is available at each step and thus a preferable composition theorem.

We observe BR composition on the exponential mechanism outperforms all other techniques. However, BR composition on a general mechanism (all ϵ-DP mechanisms are 2ϵ-BR) is comparable to RDP. In general, we see a lot of similarities between the other techniques, although the composition from Kairouz et al. offers slight improvement for smaller values of ϵ. Thus, we choose Lemma 6.15 for the exponential mechanism and Lemma 6.6 for all other mechanisms.

6.4 Empirically Choosing the Best Mechanism

In this section, we will evaluate the performance of our simple GA using the various mechanisms and composition theorems we have introduced. Although it is clear that bounded range composition is the tightest composition theorem, it is not clear which combination of composition theorem and DP mechanism will perform best. For example, the permute and flip mechanism is known to have better utility than the exponential mechanism but does not benefit from bounded range composition [15, 43]. Thus, we will evaluate the exponential, permute and flip, and Laplace noisy max mechanisms using the composition theorem of Kairouz et al. (Lemma 6.6). Additionally, we will consider the exponential mechanism under bounded range composition (Lemma 6.15). We study the performance over all datasets described in Section 5.2 and over both of our example problems described in Section 3.2. We use the set of default hyperparameters given in Table 5.1.

Figure 6.2 shows the results. On the y-axis, we have the performance (or utility) of the candidate returned by the GA (higher is better). We recall that the utility of logistic regression is measured on the test set. On the x-axis, we evaluate various levels of privacy ϵ (lower is better). We repeat each experiment 100 times and plot the mean and 95% confidence interval as the shaded region. We can see that using the exponential mechanism with bounded range composition consistently gives the best results. Furthermore, the composition theorem seems to be the dominating factor in performance. That is, all three mechanisms perform very similarly under the same composition theorem. However, when using bounded range composition, the exponential mechanism performs significantly better. We focus on this combination of the exponential mechanism and bounded range composition for the remainder of the thesis.

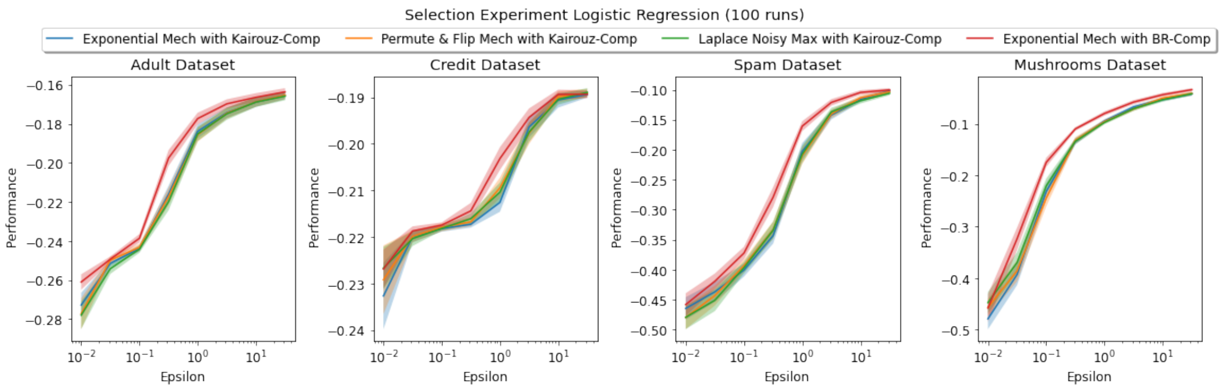
6.5 Proof of Privacy

Using our chosen DP mechanism and composition theorem, we prove the end-to-end privacy of our algorithm. We will focus on using the exponential mechanism and bounded range composition, although the proof is similar for different mechanisms. At a high level, proving the privacy of our simple GA consists of the following four parts.

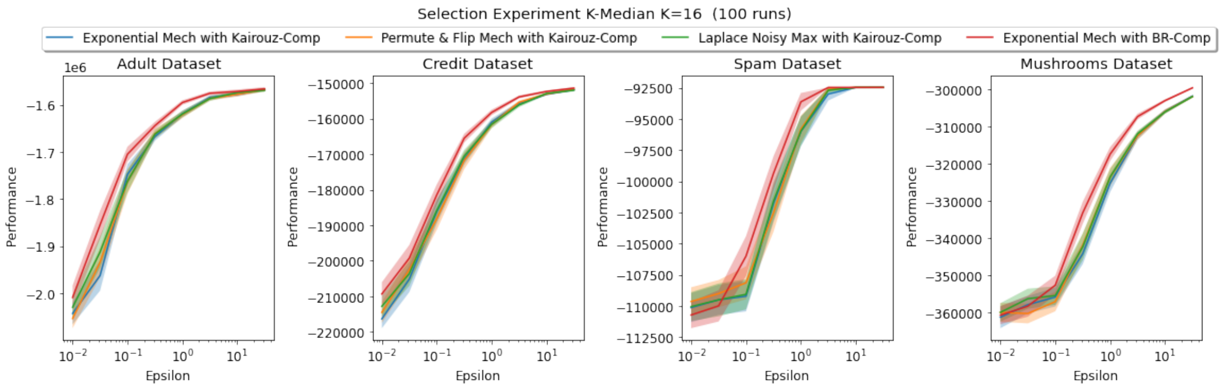
1. Bound the sensitivity of a given utility function.
2. Prove that each selection (Algorithm 3, line 7) is ϵ-BR (or ϵ-DP).
3. Prove that all other components of the simple GA incur no additional privacy cost.
4. Prove that the adaptive composition of all selections is (ϵ, δ)-DP.

We begin by bounding the sensitivity of our example problem utility functions under bounded-DP.

Lemma 6.16. *The sensitivity, $\Delta^{(u)}$ of the zero-one loss for logistic regression (given in (3.3)) is at most $\frac{1}{|D|}$*



(a)



(b)

Figure 6.2: An evaluation of the various selection techniques and composition theorems discussed in this section.

Proof. For a given vector θ changing D for D' changes at most one entry in the summation. This can change the value of the indicator function by at most one. Thus, the sensitivity is equivalent to the normalization term $\frac{1}{|D|}$. $\square$

Lemma 6.17. *The sensitivity, $\Delta^{(u)}$ of the k-median loss function (given in (3.4)) using the ℓ_1 distance metric is d*

Proof. For a given set M changing D for D' changes at most one entry in the summation. This can change the minimum of that entry by at most the sensitivity of $dist(x, y)$. Assuming we use the ℓ_1-norm and all records in V are normalized onto the interval $[0, 1]$, $dist(x, y) \leq d$.[4] $\square$

We state and prove the end-to-end privacy of our algorithm, assuming that we use the exponential mechanism and bounded range composition.

Theorem 6.18. *If we set the parameter of the exponential mechanism to $\frac{\epsilon_s u(D,r)}{2\Delta^{(u)}}$ as specified in Definition 6.1, Algorithm 3 is (ϵ, δ)-DP where*

$$\epsilon = \min\left\{ n\epsilon_s, n\left(\frac{\epsilon_s}{1-e^{-\epsilon_s}} - 1 - \ln\left(\frac{\epsilon_s}{1-e^{-\epsilon_s}}\right)\right) + \sqrt{\frac{n\epsilon_s^2}{2}\ln(1/\delta)} \right\}. \tag{6.12}$$

with $n = N_g * N_s + 1$.

Proof. We begin by showing that each selection is ϵ_s-BR. This follows from the definition of the exponential mechanism (Definition 6.1) and Lemma 6.14. Next, we discuss the various steps of the algorithm to show that the selection step is the only step that requires spending any privacy budget. We follow Algorithm 3 line by line. First, the random initialization (line 1) is entirely independent of the database, so it incurs no privacy cost. Then for a given generation, we first select the best parents (lines 6-7). This incurs a privacy cost of ϵ_s-BR and is performed $N_g * N_s$ times. The remaining steps in each generation perform post-processing on the output of this step $\mathcal{B}$ (Lemma 2.6). Specifically, line 10 uniformly selects from the pool, line 12 randomly combines the chosen solutions, and line 15 randomly changes the solutions created by crossover. None of these steps require the use of the utility function, and thus the dataset, in any way. Finally, after all generations have passed, we must choose the best solution from the final population (line 18). For this, we simply make one more selection, and thus we add this to our count n. Hence, the privacy

[4] We note that this is a rather pessimistic bound on the sensitivity that can be improved using domain information. We show one such example in Section 7.1.

budget consumption of our algorithm consists of $n = N_g * N_s + 1$ calls to exponential mechanism. Thus applying the adaptive composition theorem from Lemma 6.15 gives the desired result. $\square$

Chapter 7

Tuning a Genetic Algorithm for Privacy

In this chapter, we investigate how privacy affects the learning process of the GA. We begin by discussing how to reduce the sensitivity of the utility function to improve performance. Following this, we investigate how to set the various hyperparameters of the simple GA when using DP.

7.1 Remarks on Sensitivity

The amount of noise added in differential privacy is directly proportional to two parameters. The first is epsilon and the second is the sensitivity. We recall that the sensitivity of a function is a bound on how much the output can change when we add or remove a single record in our dataset. This gives us a measure of the worst possible difference in order to scale the noise to hide such a change appropriately. Intuitively, it makes sense to try to reduce the sensitivity of the main objective function as much as possible. For example, when training a logistic regression model, a possible loss function is cross-entropy loss. However, the cross-entropy loss has a sensitivity of $O(d)$ as we will see in Section 8.1. We instead used the zero-one loss as it has sensitivity one and measures the actual classification accuracy (which we would want to report regardless).

The easiest way to reduce the sensitivity is to find an alternative utility function, as we did with logistic regression. However, the k-median problem is not so simple. Recall that the classic k-median utility function has sensitivity d as this is the maximum distance between two points in the feature space. We could change the distance metric used; however, we found that lower sensitivity metrics such as the ℓ_∞ norm do not even loosely represent the actual loss function we are interested in reporting. A common approach in the private ML literature is to use clipping [1]. Clipping enforces a tighter bound on the function by truncating any large values to a constant.

When studying the distance matrix of our example datasets, we noticed that a distance of 0 and a distance of d do not occur. Further, the minimum distance is usually greater than $d/4$, and the maximum is less than $3d/4$. Therefore, we would not lose any utility information if we clip all values below $d/4$ and above $3d/4$. Furthermore, we found that preserving large distance values is much less important than the smaller ones. That is when a solution is terrible, we don't really care how terrible. Any poor utility value is enough for the GA to avoid it. Thus, we experimented with clipping all distances above the mean distance. For our datasets, we found a good estimator for this was $d/2$.

In Figure 7.1 we show the results of this experiment. Particularly we clip the data to be in the range $[d/4, d/2]$. This allows us to achieve a sensitivity of $d/4$ (the width of the new interval). We can see that, in general, this gives us a significant boost in utility across all datasets and values of epsilon. We note that this interval does not generalize to all datasets or distributions of public vs. private data. The credit dataset for higher epsilon values is one such example since the min is slightly higher than $d/4$. A simple approach to setting the clipping parameters, in such a case, would be first to compute the DP min and mean values of the distance matrix and then clip accordingly. However, in our case, $[d/4, d/2]$ is a reasonable choice without tailoring to each specific dataset. Thus, for the remainder of our experiments, we will apply this clipping to the distance matrix.

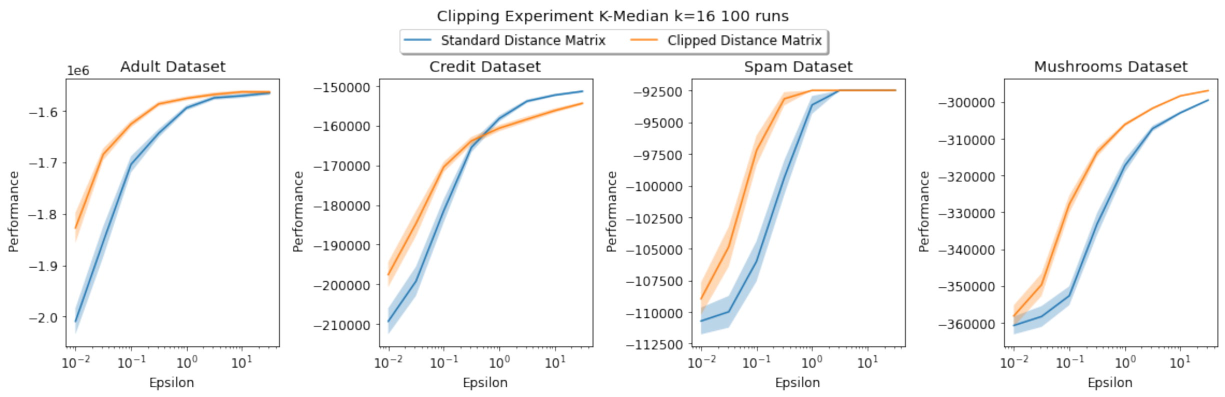
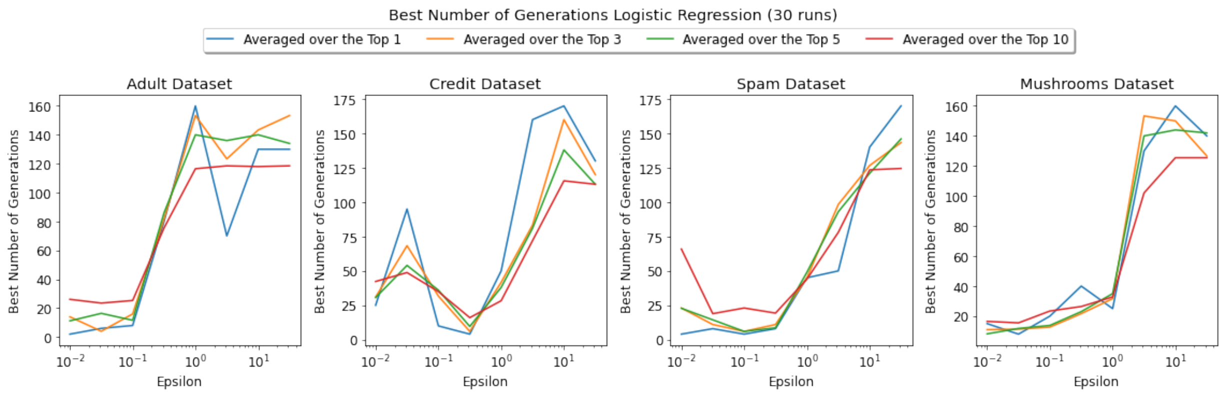
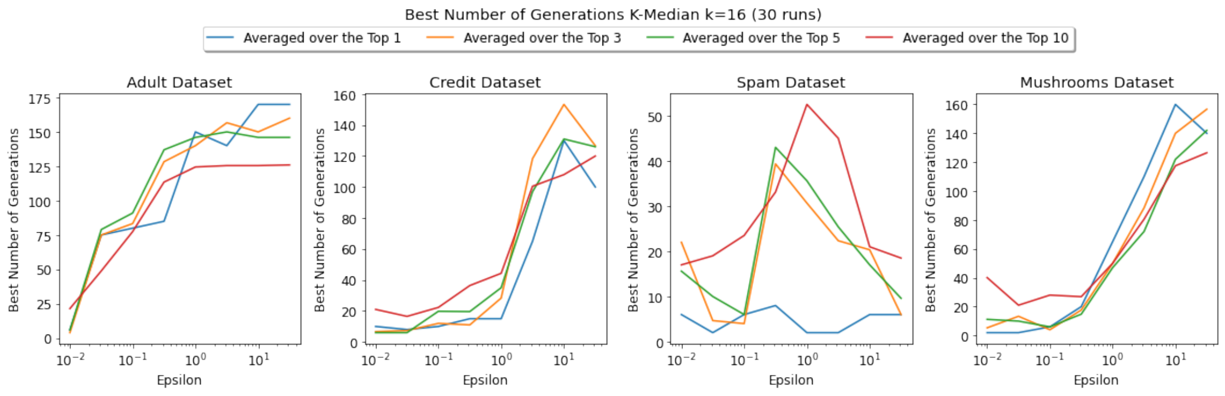


Figure 7.1: Evaluating the performance of distance matrix clipping.

7.2 The Curse of Iterations



(a)



(b)

Figure 7.2: The best setting for the number of generations parameter found from a grid search.

When training a GA with privacy, we have an interesting conflict. The more generations we conduct, the more likely we are to find a good solution. Conversely, the more generations, the less privacy budget (less accuracy) we have available for each of our selections. Thus, the N_g parameter is a highly influential parameter, and it is hard to choose its value under this trade-off. To better understand how to choose the number of generations, we conduct a preliminary grid search over different N_g and ϵ. Additionally, the results of this preliminary grid search will be used to narrow down the best values to try in our large grid search in Section 7.3. We consider the following set of 30 possible values for $N_g \in \{2, 4, 6, 8, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 110, 120, 130, 140, 150, 160, 170\}$. We run the experiment 30 times for each configuration and compute the mean utility for each parameter value.

First, we plot the value of N_g that gives the highest utility results in Figure 7.2. Specifically, we sort the results based on utility and plot the average of the top-k N_g values that gave the result. We see a general trend of a lower number of generations for lower values of epsilon. This is somewhat to be expected as there is simply not enough privacy budget to make a large number of non-trivial selections when epsilon is small. The spikes in the graphs can be explained by different parameters giving a similar utility. Even though a certain number of generations was best, other numbers of generations were close, so the randomness in the algorithm plays a large role in which values were chosen, leading to odd trends in the graphs. However, as we increase the number of results we average, we tend to see the curves smooth out. This also explains the odd behaviour in the Spam dataset for the k-median problem. We recall from Figure 7.1 that the utility converges around $\epsilon = 0.3$. After this point, the GA seems to converge regardless of the number of generations, leading to the odd trend we see here.

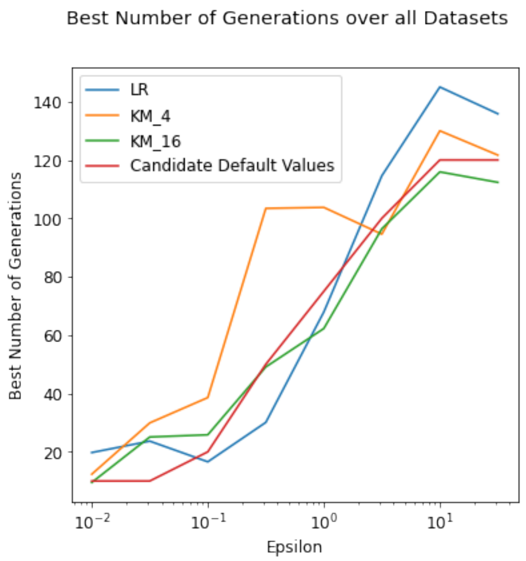
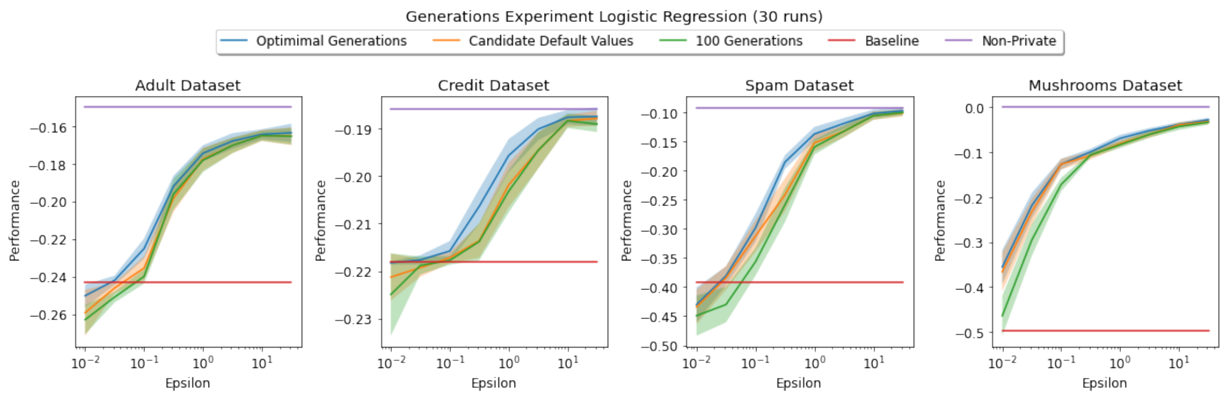


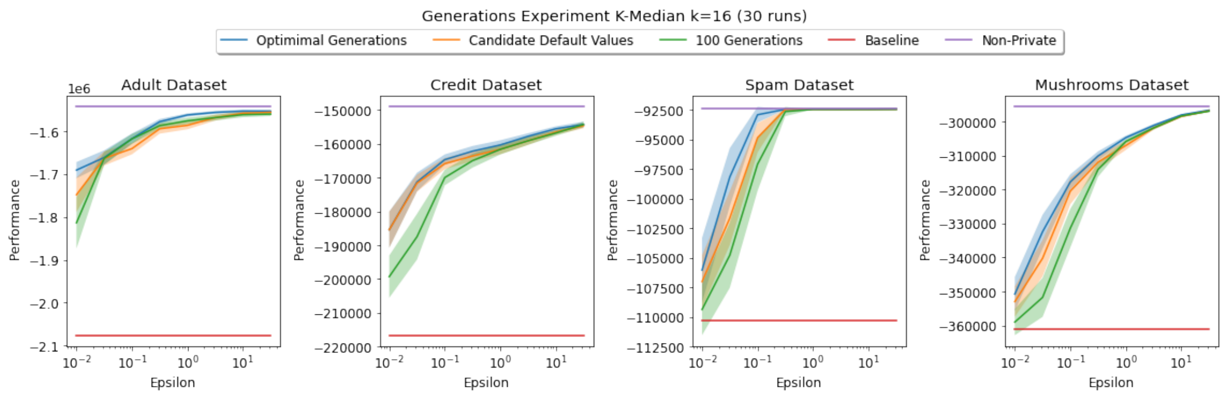
Figure 7.3: Best values of the number of generations parameter averaged of over all of our example datasets.

If possible, we would like to choose a good set of default values for N_g per ϵ across all problems and datasets. We see a general similarity between the values in Figure 7.2

suggesting a good set of defaults may exist. To test this, we first average the best generations across datasets. In Figure 7.3 we plot the average the top-3 line from the previous plots, across datasets. We see that each of the problems has a similar curve, and thus it is likely we can find a good default set. We choose a set of values that approximates these trends. Specifically, we choose N_g =$\{10, 10, 20, 50, 75, 100, 120, 120\}$ for $\epsilon \in \{10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 10^0, 10^{0.5}, 10^1, 10^{1.5}\}$ respectively. To evaluate this choice we plot the utility in Figure 7.4. We compare to the previous default of $N_g = 100$ and the optimal utility



(a)



(b)

Figure 7.4: Utility of the DP simple GA for various settings of the N_g parameter.

obtained over all generations parameters. We can see that our candidate default values almost always improve over 100 generations and are often much closer to the optimal values. Thus, we will maintain these as our default values for the rest of the experiments.

7.3 Effects of Hyperparameters

After narrowing down reasonable defaults for the generations parameter, we study the remaining hyperparameters and their interactions. We would like to see if any patterns exist with respect to epsilon as we did for the generations parameter. Further, we want to study the effect of hyperparameter tuning on the overall utility. To do this, we conduct a large-scale grid search of 2016 configurations of the 4 most influential parameters: N_g, N_s, p_c, p_m. We list the values we considered for each parameter in Table 7.3. The

Parameter	Values
N_g	$\{2, 10, 20, 50, 75, 100, 120\}$
N_s	$\{5, 10, 20\}$
p_c	$\{0.35, 0.5, 0.65\}$
p_m	$\{\frac{2}{\lvert c\rvert+1}, \frac{1.5}{\lvert c\rvert+1}, \frac{1}{\lvert c\rvert+1}, \frac{0.75}{\lvert c\rvert+1}\}$

Table 7.1: Values of each hyperparameter considered in the grid search.

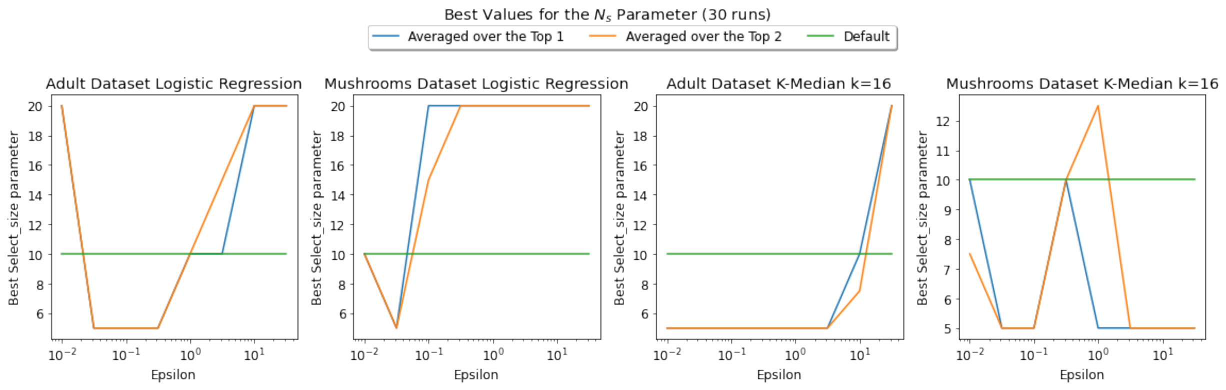
remaining hyperparameter N_p is fixed to its default value of 200. Each possible parameter configuration is run 30 times, and we consider the mean and 95% confidence intervals.

We begin with our first question: Do any patterns exist between the parameters and epsilon? To study this, we consider the values of each parameter that gave the highest utility (as we did in the previous section). Due to the low number of values for each parameter, we only consider the top one and two value averages. To reduce the number of plots, we consider only the Adult and Mushrooms dataset for each parameter (the results for the other datasets are similar). The results are given in Figure 7.5. We omit the generations parameter as we already studied it, and the results here are similar.

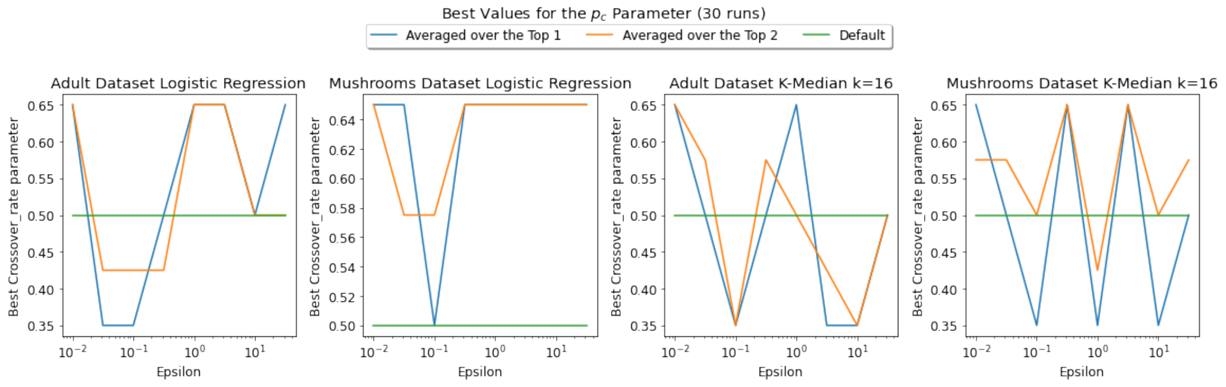
We begin with the number of selections. Similar to the generations parameter, we see an approximate trend of increase with epsilon. Intuitively, this is likely for the same reason as the generations parameter. There is simply not enough privacy budget at low epsilons to make more useful selections. We investigated setting default values in a similar fashion to the generations parameter. However, we found that the increase in performance was relatively small. Thus, we maintain the default value of 10.

The probability of crossover parameter does not have an obvious trend across datasets or problems. Thus, we also maintain the default value of 0.5. Finally, the probability of mutation parameter similarly has no obvious trend with epsilon. However, we generally see values higher than our default do perform better. We investigated increasing the mutation rate but once again found only a slight performance increase, so for simplicity, we maintain our default value of $\frac{1}{|c|+1}$.

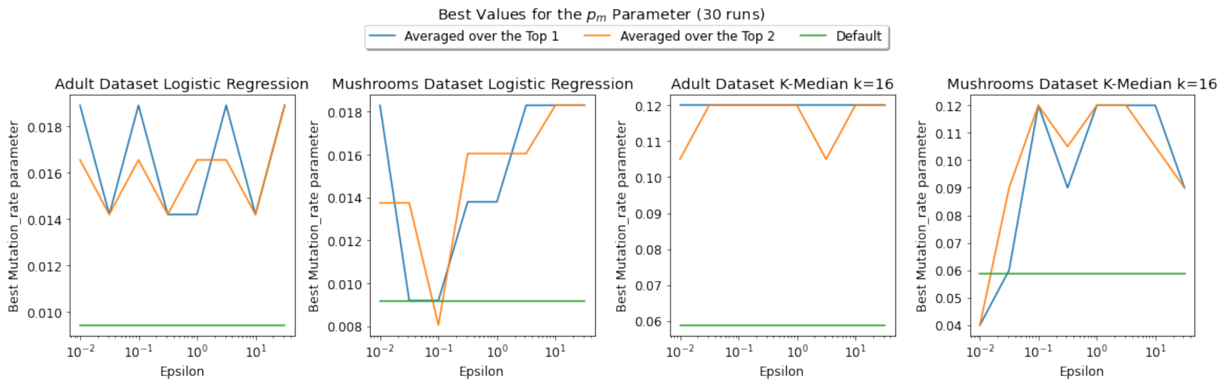
Our second question was in regard to the effect of hyperparameter tuning on overall performance. To study this, we plot the mean utility for our default parameters vs. the best results found in the grid search in Figure 7.6. We see that for logistic regression, there is a slight advantage to hyperparameter tuning, especially for higher values of epsilon. Whereas for k-median, the effect is more subtle and is most prominent for lower epsilon values. We conclude that hyperparameter tuning is helpful but not necessary for our algorithm on the use cases we have considered.



(a)



(b)



(c)

Figure 7.5: The best settings for the various hyperparameters found from the grid search.

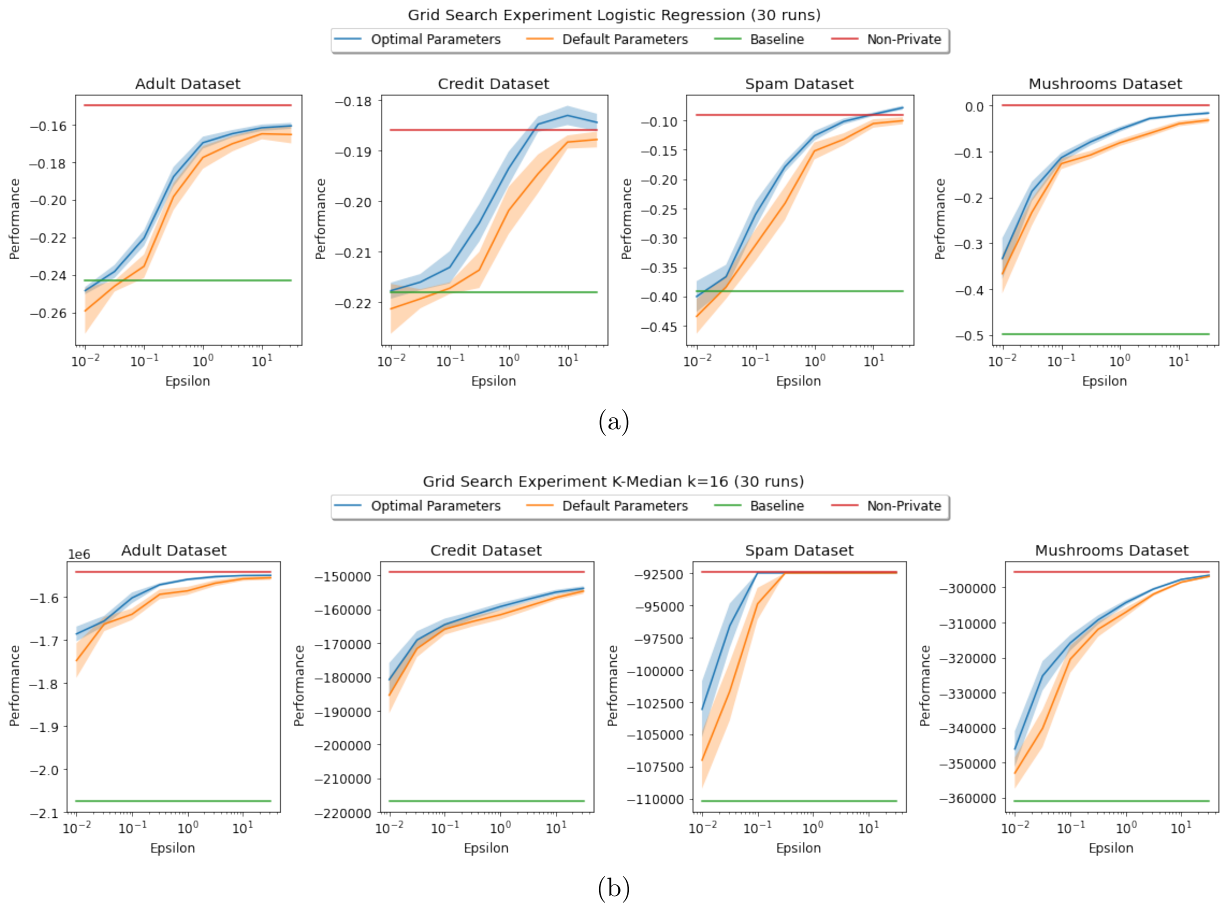


Figure 7.6: Utility of the DP simple GA for our default parameters vs. the optimal value from the grid search.

Chapter 8

Performance Evaluation

In this chapter, we compare the performance of our solution against both general and problem-specific related work.

8.1 PrivGene

As previously mentioned, the first private GA, PrivGene, was introduced by Zhang et al. [61]. In the paper, Zhang et al. introduce a preliminary private GA that they later adapt into an evolutionary strategy. We begin with the private GA (which we refer to as PrivGene) as it is most closely related to our work. We include the algorithm from the original paper using our notation in Algorithm 6. Line 1 initializes the population with 180 random parameter vectors as well as 20 vectors of all zeroes with a random positive or negative number in the last position (10 of each). Following this, in each generation, $N_s = 10$ of the fittest candidates are chosen using the exponential mechanism in line 3. This is done using the `DP_SELECT` function, which calls the exponential mechanism in a peeling manner.

The utility function used for logistic regression was log loss (or cross entropy loss). Defined as

$$u(D, (\alpha, \beta)) = -\frac{1}{|D|} \sum_{x \in D} y \log(h(x)) + (1 - y) \log(1 - h(x)) \tag{8.1}$$

where $h(x)$ is the prediction of the logistic regression model. Zhang et al. showed that when the dataset was normalized to $[-1, 1]^d$ that the following bound hold on the sensitivity.

Lemma 8.1 (Section 5.1 [61])**.** *The sensitivity of the log loss function is*

$$\frac{dR + 1}{|D|} \tag{8.2}$$

The privacy of the algorithm follows from the privacy of the exponential mechanism (Lemma 6.2) and naive composition (Lemma 2.5).

After the DP selection, the chosen pool of candidates undergoes crossover and mutation (lines 6-8) to create a new population with a few key differences to our algorithm. First, we note that the crossover and mutation operators are always invoked in PrivGene (corresponding to $p_c = 1$ and p_m being approximately one over the chromosome length). Second, PrivGene uses a single-point crossover operator, whereas we use uniform crossover. Finally, they use a very specific mutation operator that first selects a dimension of the parent solution then adds noise with absolute value at most 5% of the domain.[1] The amount of noise is reduced by 5% per generation for the remaining generations. The above process is repeated for $N_g = c \cdot \frac{|D|\epsilon}{N_s}$ where c is a constant that was experimentally set to 1.25×10^{-3} in the paper. We remark that PrivGene also uses truncation selection to make their GA private. However, our approach is different in almost every part of the algorithm. Specifically, we start from the simple GA [46], use less sensitive utility functions, use better crossover and mutation, and apply more advanced privacy analysis.

We implement PrivGene from scratch following the description above. Applying PrivGene to logistic regression is straightforward, as Zhang et al. also evaluated this use case. We assume a domain of $[-5, 5]$ for the model parameters and set the initial mutation noise to be 0.5 as described above. k-medians is more difficult as PrivGene only considered problems with real-valued candidate vectors. To run PrivGene on k-medians, we need to change the candidate representation, mutation, and crossover operations to a discrete version. For simplicity, we use our operators (initialization, crossover, and mutation), which we believe only improves PrivGene (if PrivGene's operators were converted to their discrete counterparts). We maintain all other hyperparameters of PrivGene for the k-median problem.

[1]The distribution of the noise was not specified in the paper, so we assume uniform noise

Algorithm 6 PrivGene

```
    Input: $D$, $u$, $\epsilon$: Dataset, utility function, and privacy budget.
    $N_s$=10, $N_p$=200, $N_g$: Size of selected set, population, and number of iterations.
    $\lambda = 0.5$, $\beta = 0.95$: mutation scale and decay rate.
    Output: $\omega$: best candidate chosen.
 1: $\mathcal{P} \leftarrow$ INITIALIZE_RANDOM($N_p$)
 2: for  $i = 1$ to $N_g - 1$ do
 3:     $\mathcal{S} \leftarrow$ DP_SELECT($N_s$, $\mathcal{P}$, $D$, $u$, $\epsilon/N_g$)
 4:     $\mathcal{P} \leftarrow \emptyset$
 5:     for $j = 1$ to $N_p/2$ do
 6:         $x_1, x_2 \leftarrow$ RANDOM_CHOICE($\mathcal{S}$)
 7:         $y_1, y_2 \leftarrow$ SINGLE_POINT_CROSSOVER($x_1, x_2$)
 8:         $z_1, z_2 \leftarrow$ MUTATE($y_1, \lambda$), MUTATE($y_2, \lambda$)
 9:         $\mathcal{P} \leftarrow z_1, z_2$
10:     $\lambda \leftarrow \lambda \cdot \beta$
11: $\{\omega\} \leftarrow$ DP_SELECT($1, \mathcal{P}, D, u, \epsilon/N_g$)
12: return $\omega$

13: function DP_SELECT($N_s$, $\mathcal{P}$, $D$, $u$, $\epsilon_s$)
14:     $\mathcal{S} \leftarrow \emptyset$
15:     for $i = 1$ to $N_s$ do
16:         $x \leftarrow$ EXPONETIAL_MECHANISM($u(\mathcal{P} - \mathcal{S}), \epsilon_s/N_s$)
17:         $\mathcal{S} \leftarrow x$
18:     return $\mathcal{S}$
```

To address the limitations of their DP GA, Zhang et al. also introduce another variant of PrivGene that is more of an evolutionary strategy than a GA. The main improvement comes from using a modified exponential mechanism which they call the enhanced exponential mechanism (EEM). We shall call this variant of PrivGene that uses EEM, PrivEEM. Zhang et al. show that EEM has a significant effect on performance outperforming the standard GA [61][Figure 6, 7]. The only difference between EEM and the standard exponential mechanism is in the sensitivity analysis. The rest of the mechanism is identical to Definition 6.1. EEM was designed by Zhang et al. for utility functions of the following form[2]

$$u(D,r) = h(r) + \sum_{x \in D} l(D,r) \tag{8.3}$$

First we recall the standard definition of sensitivity is

$$\Delta^{(u)} = \max_{r \in R} \max_{D,D'} |u(D,r) - u(D',r)|. \tag{8.4}$$

EEM generalizes this formula to the following

$$\Delta^{(u)} = \min \Bigg\{ \Delta_1 = \max_{r \in R} \max_{D,D'} |u(D,r) - u(D',r)|, \\ \Delta_2 = \max_{r,r' \in R} \max_{D} |u(D,r) - u(D,r')| \Bigg\} \tag{8.5}$$

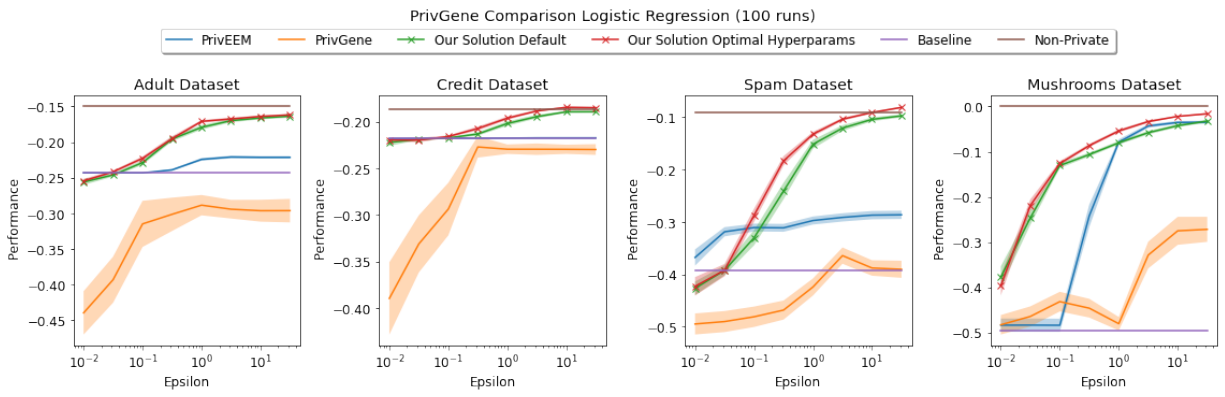
where Δ_1 is the standard definition of sensitivity in (8.4), and thus in the worst case EEM reduces to the standard exponential mechanism. Δ_2 is the novel part that bounds the distance between candidates $r \in R$. The intuition behind this mechanism is that overtime as an algorithm starts to converge, solutions will become increasingly similar and thus $\Delta_2 < \Delta_1$.

Lemma 8.2 (Theorem 1 [61]). *EEM with sensitivity defined in (8.5) is ϵ-DP.*

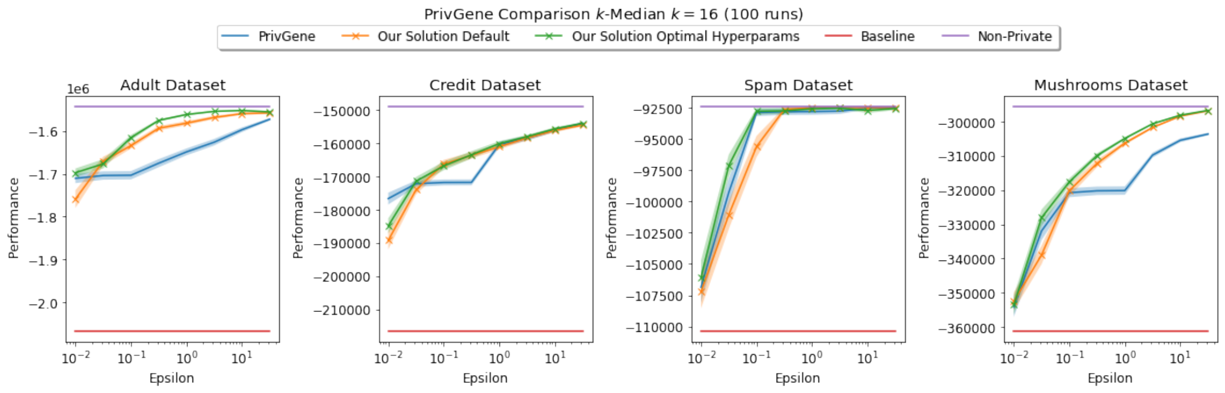
To use EEM with PrivGene, a modification must be made in order to reap the benefits of the tighter sensitivity analysis. Specifically, in PrivEEM , the chosen size (N_s) is reduced to 1. This removes the crossover operator completely and changes Algorithm 6 to an evolutionary strategy that only uses mutation. However, when the only operation conducted is a mutation on a single candidate, the population becomes bounded by Δ_2. For logistic regression, we use the bound proved by Zhang et al. that $\Delta_2 = 2\lambda$. However, for k-median, the uniform mutation operator has no such bound, and thus (similar to Zhang et al. [61][Section 5.3]) we only evaluate PrivEEM on logistic regression.

8.1.1 Comparison

Now we have defined both variants of PrivGene, we perform an empirical evaluation against our technique. We consider our solution under the default hyperparameters chosen in Chapter 7 as well as the optimal hyperparameters found from the grid search. We compare



(a)



(b)

Figure 8.1: Evaluation of our approach vs. PrivGene and PrivEEM [61].

this to the utility of PrivGene and PrivEEM in Figure 8.1. We repeat the experiment 100 times and plot the 95% confidence intervals as the shaded area. We can see that we consistently outperform PrivGene for logistic regression. For k-Median, we either tie or do better when $\epsilon > 0.1$. We recall that we use our own operators for PrivGene in the k-median problem, which explains why PrivGene performs better here than logistic regression. The performance we observe in logistic regression is in line with the results of Su et al. [54, Figure 6].

For PrivEEM , we either outperform or tie in logistic regression (recall we do not evaluate k-median). We note how the results of PrivEEM depend largely on the dataset,

[2]It should be noted that later Dong et al. defined a more general sensitivity analysis for the exponential mechanism [13], but our goal here is to reproduce the work of Zhang et al. [61].

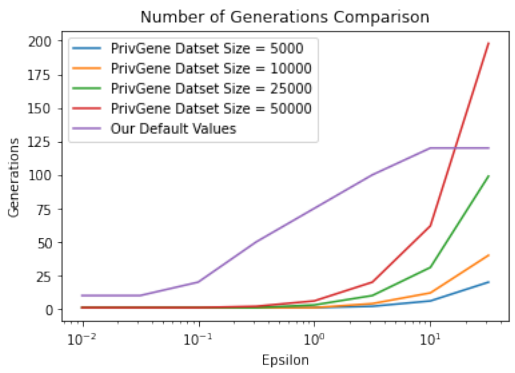


Figure 8.2: Comparing the number of generations for PrivGene and our approach.

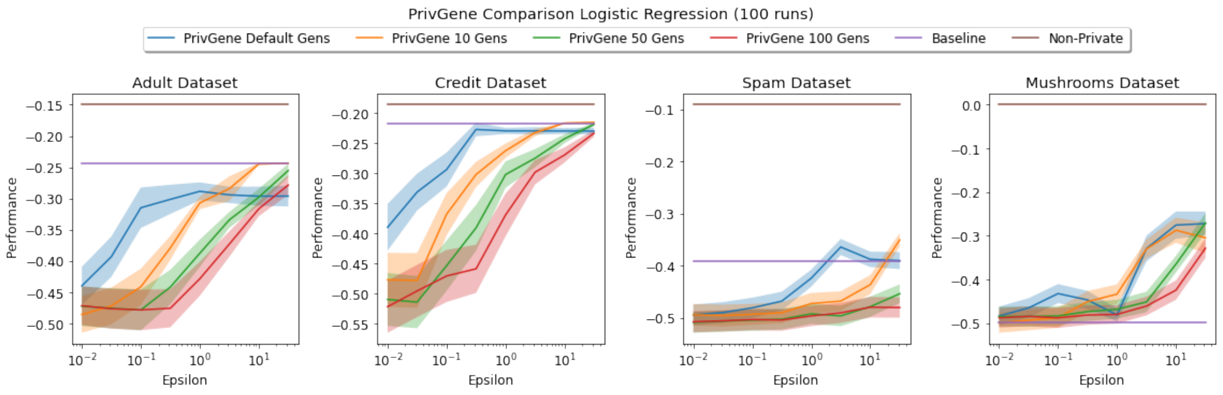
whereas our solution remains more stable. Furthermore, we always beat PrivEEM in the most important region of $\epsilon \in [0.1, 1]$

8.1.2 Generations Formula

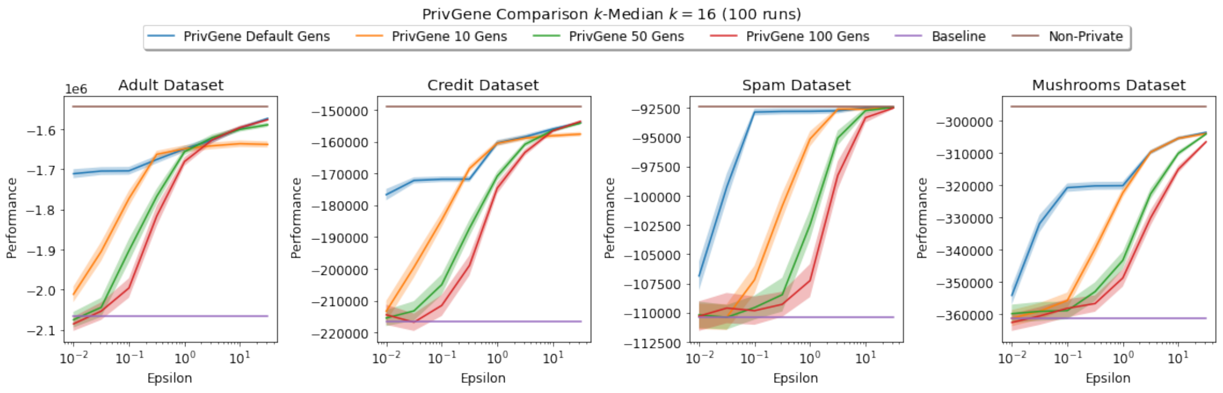
We noticed while running our experiments that the number of generations in PrivGene is quite low when following their formula. In Figure 8.2, we plot the value of this formula for various values of ϵ and $|D|$. For this plot, we set $N_s = 10$ and take the max of the formula and 1 (as anything below this is meaningless). We observe that for $\epsilon < 1$, the number of generations is strictly less than 10, and in fact, it is often 1. We believe these values are prohibitively low for a GA, especially in the important region of $\epsilon \in [0.1, 1]$. In comparison, our solution carries out between 20 and 75 generations in this region. We carry out an additional experiment where we override the number of generations with a constant (10, 50, and 100) to give PrivGene a fair evaluation. The results are given in Figure 8.3.

We can see that, in general, varying this parameter only makes PrivGene perform worse. We remark that the number of generations is one for low values of ϵ (approx. $\epsilon < 1$ for our datasets). In this case, PrivGene performs no evolutionary operations at all. Instead, they simply sample the best solution from the initial random population. That is, PrivGene reduces to the sub-sampled exponential mechanism (which we evaluate further in Section 8.3).

In conclusion, we see that our approach offers a significant and stable increase in utility over PrivGene (and PrivEEM) across all problems and datasets without sacrificing GA operations.



(a)



(b)

Figure 8.3: Evaluating the performance of PrivGene when varying the number of generations parameter.

8.2 Local Search Techniques

8.2.1 PrivLocal

Algorithm 7 PrivLocal from the source code of PrivGene [61] as defined by Su et al. [54]

```
    Input: $D$, $d$, $u$, $\epsilon$: Dataset, dimension, utility function, and privacy budget.
    $r$, $\omega_0$ : Number of iterations and initial solution
    $\lambda = 0.5$, $\beta = 0.95$: perturbation scale and decay rate.
    Output: $\omega$: best candidate chosen.
1:  $\omega \leftarrow \omega_0$
2:  for $i = 1$ to $r$ do
3:      $\mathcal{P} \leftarrow \emptyset$
4:      for $j = 1$ to $d$ do
5:          $x_+ \leftarrow \omega$
6:          $x_+[j] \leftarrow x_+[j] + \lambda$
7:          $x_- \leftarrow \omega$
8:          $x_-[j] \leftarrow x_-[j] - \lambda$
9:          $\mathcal{P} \leftarrow x_+, x_-$
10:     $F \leftarrow$EVALUATE_FITNESS($\mathcal{P}$, $D$, $u$)
11:     $\omega \leftarrow$ EXPONETIAL_MECHANSIM($F, \epsilon/r$)
12:     $\lambda \leftarrow \lambda \cdot \beta$
13: return $\omega$
```

Su et al. first observed that the source code accompanying PrivGene actually contains code for another algorithm that did not appear in the PrivGene paper [54]. We confirmed that the source code publicly available for PrivGene[3] actually implements a different algorithm named PrivLocal (and not PrivGene). Su et al. were the first to publish this algorithm in their work, and we follow their definition using our notation in Algorithm 7. PrivLocal is a type of local search technique that uses some of the ideas from the PrivGene algorithm. The algorithm starts with an initial solution (a vector of zeros) and makes iterative, greedy improvements to this solution. That is, in lines 4-9 a pool of candidates are created by iteratively increasing and decreasing the value of each entry. After all possible perturbations have been created, the exponential mechanism is used to select the best, which moves on

[3]https://sourceforge.net/p/privgene/

to be the seed for the next iteration. The number of iterations parameter follows the same formula as PrivGene. The values of λ and β are also the same as PrivGene; however, this value represents a constant search step rather than the mutation scale.

Another paper by Lee and Kifer [39] evaluated PrivGene and showed much better performance than our results or those of Su et al. [54]. However, upon inspection of their source code, we found that they actually evaluated PrivLocal. We use Lee and Kifer's python implementation [4] which follows the Matlab code of Zhang et al. exactly.

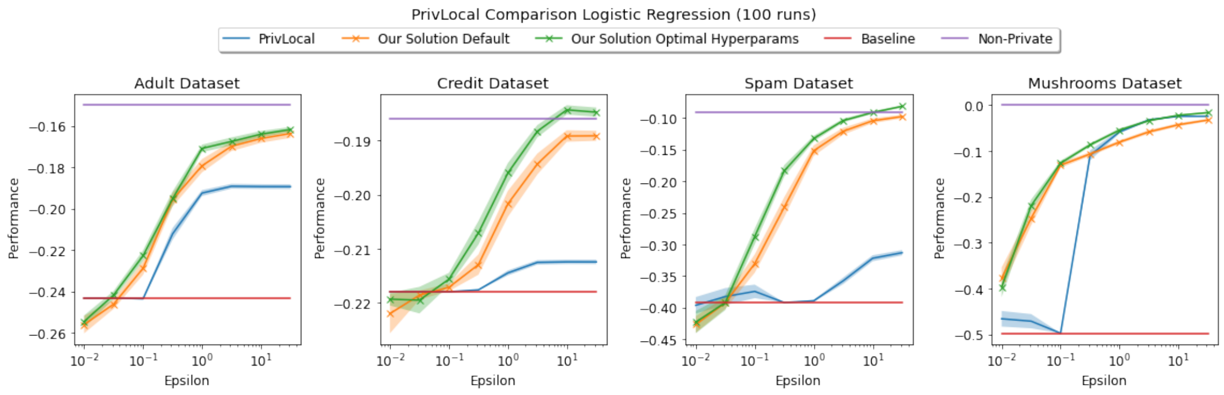


Figure 8.4: Evaluation of the PrivLocal [54, 61] algorithm against our approach.

Performance We compare our solution to PrivLocal in Figure 8.4. While PrivLocal gives much better performance than PrivGene, we once again observe very dataset-dependent performance. We significantly outperform PrivLocal on our first three test datasets over all values of ϵ. PrivLocal performs better on the Mushrooms dataset, slightly outperforming our default solution when $\epsilon > 0.3$. However, our optimized solution can consistently outperform PrivLocal. We conclude that our solution offers a much more stable solution that is a better choice than PrivLocal.

We note that Figure 8.4 highlights that our optimal hyperparameter solution outperforms the non-private baseline in the Credit and Spam datasets. We recall that we measure test set utility for the logistic regression problem, and thus even though the Scikit-learn model converged on the training set, our DP GA generalizes better to the test set.

[4]https://github.com/ppmlguy/DP-AGD

8.2.2 Gupta et al.'s k-Median

Since the k-median problem is discrete, PrivLocal is not a valid solution. However, the first work in private k-medians by Gupta et al. is also a local search algorithm [29]. The algorithm is based on the non-private search of Arya et al. [2], simply replacing each selection with the exponential mechanism. The complete algorithm is given in Algorithm 8. We state the algorithm in our more general notation; however, in the original algorithm $P = V$.

Algorithm 8 Gupta et al.'s k-median Algorithm [29]

```
    Input: $P$, $D$, $u$, $k$, $\epsilon$: Public Set, Private Dataset, utility function, number of medians, and privacy budget.
    Output: $M$: best set of medians.
1: $M_1 \leftarrow$ Random_Solution($k$)
2: $\epsilon_s \leftarrow \frac{\epsilon}{2\Delta^{(u)}(T+1)}$
3: for $i = 1$ to $T$ do
4:     Select $(x, y) \in M_i \times (P - M_i)$ with probability proportional to
       $\exp(-\epsilon_s u(M_i - \{x\} + \{y\}))$
5:     $M_{i+1} \leftarrow M_i - \{x\} + \{y\}$
6: Select $j$ from $\{1, 2, \ldots, T\}$ with probability proportional to $\exp(-\epsilon_s u(M_j))$
7: return $M_j$.
```

Performance We implemented this algorithm from scratch in order to compare it to our solution. We found that the algorithm is rather inefficient for large public set sizes due to the large number of swaps considered. Thus, we focus on our two smaller datasets Spam and Mushrooms for this evaluation. We give the results of our comparison for $k = 4$ and $k = 16$ in Figure 8.5. We can see that our solution drastically outperforms Gupta et al.'s solution for all datasets and all values of ϵ. We conclude that in both utility and runtime, our solution is superior to that of Gupta et al. [29].

8.3 Sub-Sampled Exponential Mechanism

While our evaluation considers two specific example problems, our solution is not problem-specific. Thus, we wish to evaluate against other general DP selection techniques. Very little related work solves the general problem of DP selection for such large search spaces

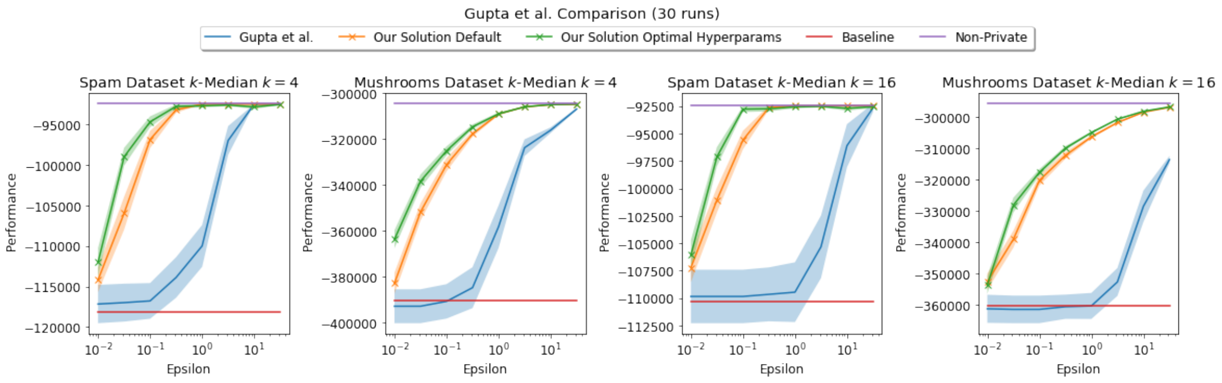
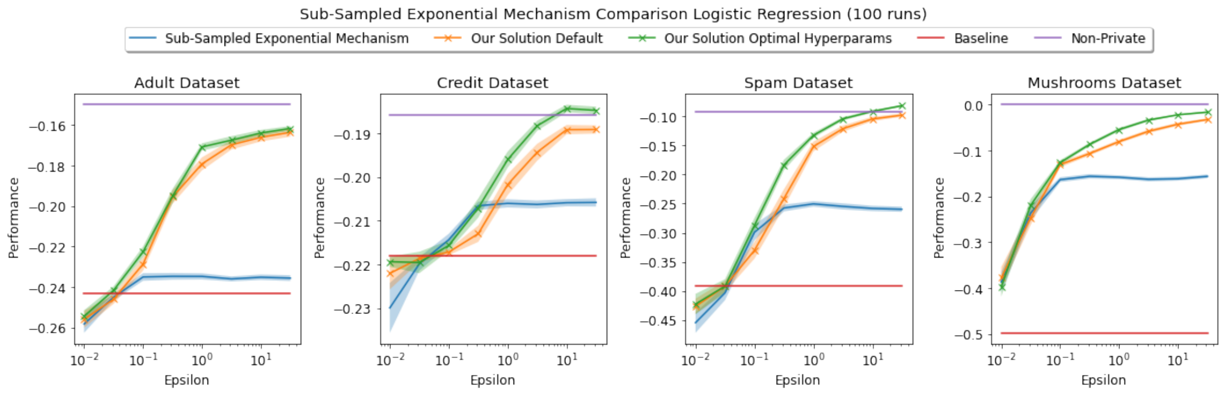


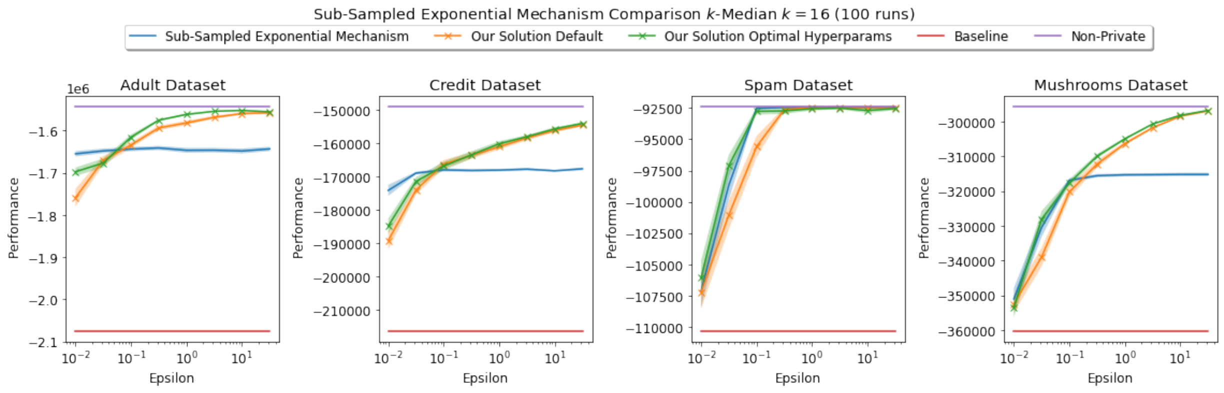
Figure 8.5: Comparison of our solution vs. the solution of Gupta et al. [29]

as those we consider. The one exception is the sub-sampled exponential mechanism from Lantz et al. [38]. This technique first takes a random sample from the space of possible solutions before applying the standard exponential mechanism. This circumvents the need to compute the utility of all possible solutions in a large space. However, this approach clearly relies on the random sample being representative of the greater population.

We evaluate the sub-sampled exponential mechanism on both logistic regression and the k-median problem. The original paper by Lantz et al. also considered the k-median problem and showed competitive performance compared to the techniques of Gupta et al. [38] (evaluated in Section 8.2.2). We choose the sample size to be 1000 and run the sub-sampled exponential mechanism 100 times using the same experimental setup as above. The results are given in Figure 8.6. We again see that this solution is dataset-dependent, as we predicted. Furthermore, the utility appears to plateau at a specific value much less than the non-private baseline (except k-median on Spam). Our default solution almost always outperforms the sub-sampled exponential mechanism when $\epsilon > 0.1$ (except Credit logistic regression). Our optimal parameter solution always outperforms the sub-sampled exponential mechanism when $\epsilon > 0.1$. This once again indicates our solution provides a much more stable solution that offers significantly better utility in the important region of $\epsilon \in [0.1, 1]$. Furthermore, this means our algorithm gives the most promising alternative to date for the exponential mechanism on large solution spaces.



(a)



(b)

Figure 8.6: Evaluating our solution against the sub-sampled exponential mechanism [38].

8.4 Problem Specific Techniques

In this section, we consider the state-of-the-art solutions specific to each of our example problems. The goal is to show that our general solution is comparable to these solutions.

8.4.1 Logistic Regression

As mentioned in Chapter 4, most works do not focus on the problem of differentially private logistic regression directly. Instead, they use it as one of the multiple examples to evaluate their techniques. As a result, many works consider the performance of DP logistic regression. Recent work by Iyengar et al. [31] conducts an extensive performance benchmark of state-of-the-art algorithms in differentially private convex optimization. We believe this benchmark well represents some of the best solutions in this field, and thus we use it for our evaluation. The benchmark includes the following solutions:

- **Approximate Minima Perturbation**: a new technique introduced by Iyengar et al. [31] closely related to objective perturbation.
- **P-SGD:** Private SGD based off of the work by Bassily et al. [4].
- **P-PSGD:** Private Perturbation-based SGD from the work of Wu et al. [58].
- **P-SCPSGD:** Private, strongly convex Perturbation-based SGD, another variant from Wu et al. [58].
- **DP Frank-Wolfe**: the DP version of the popular optimization algorithm from Talwar et al. [55].

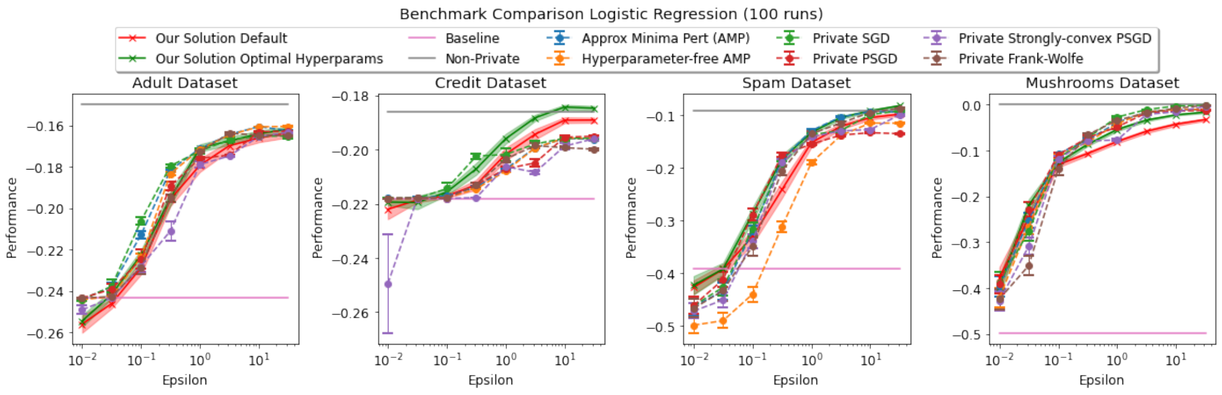
The code was made publicly available[5], which allows us to simply run their benchmark, unchanged, on our own datasets. We note that the code performs a grid search over the hyperparameters of the various algorithms and reports the best in a similar manner to our optimal hyperparameter solution.

The results are given in Figure 8.7. We adjusted the logs to obtain the 95% confidence intervals over 100 runs as we have done for all of our previous experiments. The first plot gives a comparison against all techniques, and the second shows only the best techniques for readability. We can see that our algorithm performs comparably to the other techniques with a strong performance on the Credit dataset in particular. In general, our optimal hyperparameter solution is very rarely outperformed, and if so, it is only by a small amount. We conclude that even against problem-specific solutions in convex optimization, our general technique provides a viable (and often the best) solution.

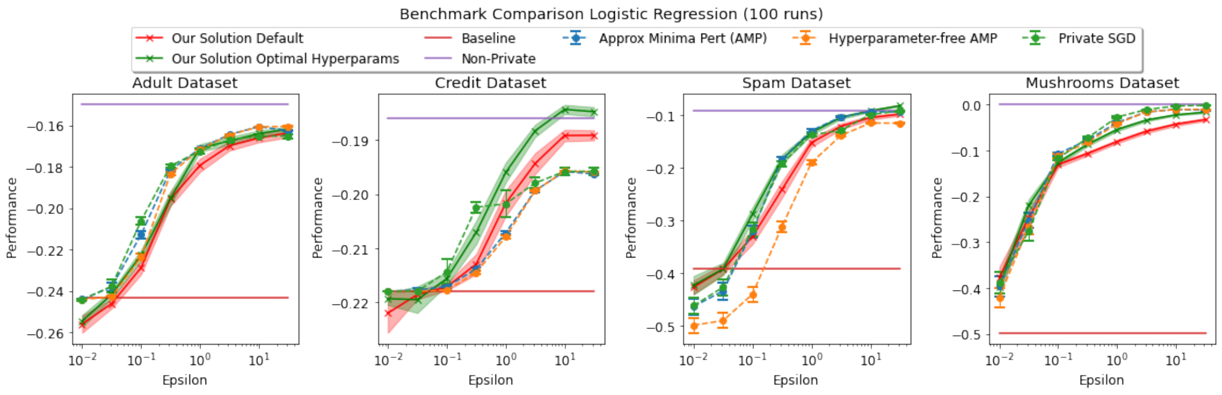
8.4.2 k-Median

The most recent work in this space is a solution by Jones et al. [32]. From a theoretical standpoint, this work is the state-of-the-art for the k-median problem, achieving smaller additive error than any other work. We conduct the first empirical evaluation of this technique to compare it with our solution.

[5]https://github.com/sunblaze-ucb/dpml-benchmark



(a)



(b)

Figure 8.7: Evaluation our solution using the benchmark of Iyengar et al. [31]

The algorithm first creates a small, weighted, synthetic dataset by choosing points from the public set that best represents the private set.[6] To do this, they first use the DP unweighted set cover algorithm of Gupta et al. [29] to choose public points in the densest areas of the private set. They map the private data points onto the candidate cluster centers using the Laplace mechanism to obtain the weights. The synthetic dataset (or coreset) is then released, and any off-the-shelf non-private algorithm is applied to obtain the final result. For specific details on the algorithm, we direct readers to the original paper [32]. We implement the algorithm from scratch in python and use our non-private

[6]The original work created the synthetic dataset from the domain. We once again adapt this to our more general setting and choose points from the public set. However, one could easily make the public set equal to the domain.

Scikit-learn baseline for the final step. The implementation can be found along with the rest of our code[7].

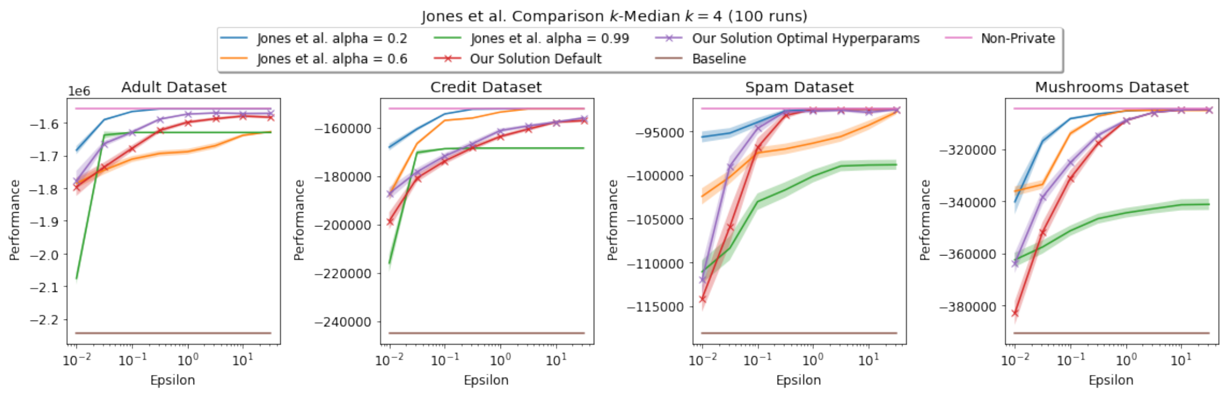
The algorithm has a single hyperparameter ϵ, which we will rename α to avoid confusion. Since the paper is theoretical, there is no information about how to set this parameter. This parameter controls the number of iterations and plays a major role in the accuracy and runtime of the protocol. We found that the lower the value, the better utility but also, the longer the runtime. We deduced that $\alpha \in (0, 1)$ otherwise, we would obtain invalid parameters (such as negative iterations). Thus, for our experiments we chose $\alpha \in \{0.2, 0.6, 0.99\}$. We did not choose $\alpha < 0.2$ as the runtime was too large, and the utility gains were insignificant.

We repeat the experiment for 30 runs and plot the 95% confidence intervals. The results are presented in Figure 8.8. We see that the hyperparameter α strongly influences the performance of Jones et al.'s algorithm. For higher values of α, our algorithm often outperforms Jones et al. significantly. However, for smaller values, where the runtime of Jones is significantly higher (approximately ten times our solution on Adult), the solution of Jones et al. is the best. We note that the computation complexity of Jones et al. is $O(k|P||D| \log_{1+\alpha}(|D|) \ln(1/\alpha))$ as opposed to our solution that is $O(kN_pN_g|D|)$ where α and $N_p * N_g$ are technically constants, but, we include them as they have a significant effect in practice. In particular, as α approaches zero, $\log_{1+\alpha}(|D|)$ and $\ln(1/\alpha)$ approach infinity. We observe that our algorithm has no dependence on the size of the public set and removes this additional log dependence on the private dataset. Since our solution still offers reasonable utility with a significant improvement in efficiency, we believe it is indeed competitive with this problem-dependent approach.

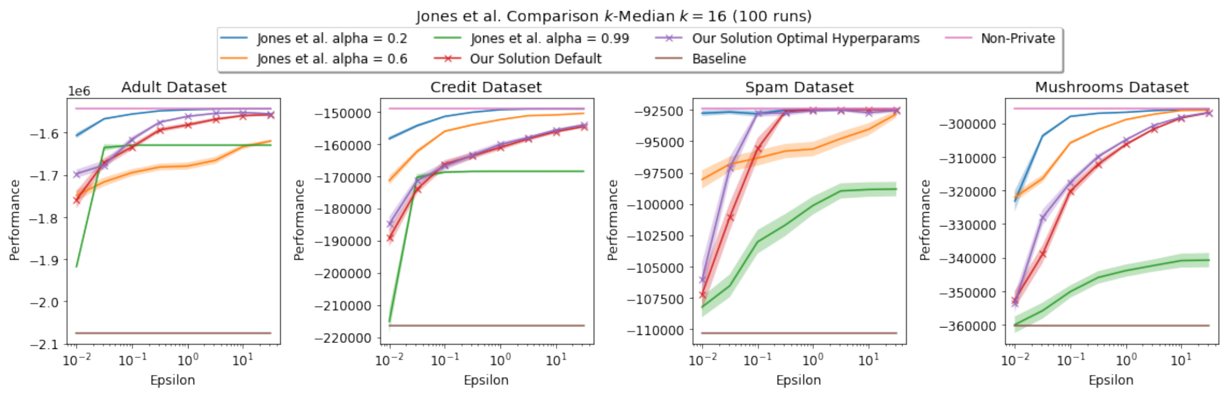
We remark that the performance of Jones et al.'s solution is entirely dependent on this specific problem definition. The solution takes advantage of the public vs. private data assumption (or the assumption that the domain is known). However, if we were to consider a variant of this problem where we wish to choose centers from the private set (the k-medoid problem), this technique would suffer a severe drop in utility. This is because the approach requires the publishing of the coreset. Satisfying DP when the centers are not public would require a significant amount of additional privacy budget.

In conclusion, our evaluation chapter has shown that our solution outperforms related work on DPGAs and related evolutionary and local search techniques. Furthermore, it offers the most stable and efficient alternative to the exponential mechanism for large solution spaces without requiring any additional assumptions on the utility function or problem specification. Our runtime scales polynomially with the size of the private dataset

[7] https://git.uwaterloo.ca/t3humphr/dp-simple-ga



(a)



(b)

Figure 8.8: Comparison of our solution to the work of Jones et al. [32]

and does not require iterating over the whole domain of solutions. Finally, our solution offers a general mechanism that is competitive with problem-specific solutions in the example problems we evaluated.

Chapter 9

Conclusion

As the public becomes more privacy savvy, we are starting to see an increased effort from organizations to use privacy-preserving mechanisms. A crucial trade-off in the adoption of these mechanisms is privacy vs. utility vs. efficiency. Our work provides a better compromise in terms of these objectives than both general and problem-specific related work, offering a solution to the DP selection problem with high utility while scaling efficiently to large domains. Our solution is not problem-specific and thus can replace existing solutions such as the sub-sampled exponential mechanism when the domain is large. Furthermore, our work shows that despite the current literature, GAs are quite robust to the effects of DP noise and should be considered in future work on DP selection.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 308–318, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[3] Maria-Florina Balcan, Travis Dick, Yingyu Liang, Wenlong Mou, and Hongyang Zhang. Differentially private clustering in high-dimensional euclidean spaces. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 322–331. JMLR.org, 2017.

[4] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 464–473, 2014.

[5] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Proceedings, Part I, of the 14th International Conference on Theory of Cryptography - Volume 9985*, page 635–658, Berlin, Heidelberg, 2016. Springer-Verlag.

[6] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(29):1069–1109, 2011.

[7] Kamalika Chaudhuri, Anand D Sarwate, and Kaushik Sinha. A near-optimal algorithm for differentially-private principal components. *Journal of Machine Learning Research*, 14, 2013.

[8] Edith Cohen, Haim Kaplan, Yishay Mansour, Uri Stemmer, and Eliad Tsfadia. Differentially-private clustering of easy instances. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2049–2059. PMLR, 18–24 Jul 2021.

[9] James F Crow and Motoo Kimura. Efficiency of truncation selection. *Proceedings of the National Academy of Sciences*, 76(1):396–399, 1979.

[10] Apple Differential Privacy Team. Learning with privacy at scale, December 2017.

[11] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3574–3583, Red Hook, NY, USA, 2017. Curran Associates Inc.

[12] Zeyu Ding, Daniel Kifer, Sayed M. Saghaian N. E., Thomas Steinke, Yuxin Wang, Yingtai Xiao, and Danfeng Zhang. The permute-and-flip mechanism is identical to report-noisy-max with exponential noise, 2021.

[13] Jinshuo Dong, David Durfee, and Ryan Rogers. Optimal differential privacy composition for exponential mechanisms. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2597–2606. PMLR, 13–18 Jul 2020.

[14] Dheeru Dua and Casey Graff. UCI machine learning repository, 2021.

[15] David Durfee and Ryan Rogers. One-shot dp top-k mechanisms. DifferentialPrivacy.org, 08 2021. https://differentialprivacy.org/one-shot-top-k/.

[16] David Durfee and Ryan M Rogers. Practical differentially private top-k selection with pay-what-you-get composition. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[17] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, page 265–284, Berlin, Heidelberg, 2006. Springer-Verlag.

[18] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil Vadhan. On the complexity of differentially private data release: Efficient algorithms and hardness results. New York, NY, USA, 2009. Association for Computing Machinery.

[19] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, August 2014.

[20] Cynthia Dwork, Guy N. Rothblum, and Salil Vadhan. Boosting and differential privacy. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, page 51–60, USA, 2010. IEEE Computer Society.

[21] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 1054–1067, New York, NY, USA, 2014. Association for Computing Machinery.

[22] Yunus Esencayi, Marco Gaboardi, Shi Li, and Di Wang. Facility location problem in differential privacy model revisited. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[23] Dan Feldman, Amos Fiat, Haim Kaplan, and Kobbi Nissim. Private coresets. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 361–370, New York, NY, USA, 2009. Association for Computing Machinery.

[24] Dan Feldman, Chongyuan Xiang, Ruihao Zhu, and Daniela Rus. Coresets for differentially private k-means clustering and applications to privacy in mobile sensor networks. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '17, page 3–15, New York, NY, USA, 2017. Association for Computing Machinery.

[25] David B Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*, volume 1. John Wiley & Sons, 2006.

[26] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[27] Badih Ghazi, Ravi Kumar, and Pasin Manurangsi. Differentially private clustering: Tight approximation ratios. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan,

and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4040–4054. Curran Associates, Inc., 2020.

[28] David E Goldberg and John Henry Holland. Genetic algorithms and machine learning. 1988.

[29] Anupam Gupta, Katrina Ligett, Frank McSherry, Aaron Roth, and Kunal Talwar. Differentially private combinatorial optimization. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, page 1106–1125, USA, 2010. Society for Industrial and Applied Mathematics.

[30] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[31] Roger Iyengar, Joseph P. Near, Dawn Song, Om Thakkar, Abhradeep Thakurta, and Lun Wang. Towards practical differentially private convex optimization. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 299–316, 2019.

[32] Matthew Jones, Huy L. Nguyen, and Thy D Nguyen. Differentially private clustering via maximum coverage. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):11555–11563, May 2021.

[33] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1376–1385, Lille, France, 07–09 Jul 2015. PMLR.

[34] Haim Kaplan and Uri Stemmer. Differentially private k-means with constant multiplicative error. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 5436–5446, Red Hook, NY, USA, 2018. Curran Associates Inc.

[35] Leonard Kaufman and Peter J. Rousseeuw. *Partitioning Around Medoids (Program PAM)*, chapter 2, pages 68–125. John Wiley & Sons, Ltd, 1990.

[36] Daniel Kifer, Adam Smith, and Abhradeep Thakurta. Private convex empirical risk minimization and high-dimensional regression. In Shie Mannor, Nathan Srebro, and Robert C. Williamson, editors, *Proceedings of the 25th Annual Conference on Learning Theory*, volume 23 of *Proceedings of Machine Learning Research*, pages 25.1–25.40, Edinburgh, Scotland, 25–27 Jun 2012. PMLR.

[37] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 202–207. AAAI Press, 1996.

[38] Eric Lantz, Kendrick Boyd, and David Page. Subsampled exponential mechanism: Differential privacy in large output spaces. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, AISec '15, page 25–33, New York, NY, USA, 2015. Association for Computing Machinery.

[39] Jaewoo Lee and Daniel Kifer. Concentrated differentially private gradient descent with adaptive per-iteration privacy budget. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, page 1656–1665, New York, NY, USA, 2018. Association for Computing Machinery.

[40] Qishuai Liu and Qing Hui. The bat-inspired consensus protocols with differential privacy. In *2018 IEEE 14th International Conference on Control and Automation (ICCA)*, pages 829–834, 2018.

[41] Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy. *Proc. VLDB Endow.*, 10(6):637–648, February 2017.

[42] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. Privacy: Theory meets practice on the map. In *2008 IEEE 24th international conference on data engineering*, pages 277–286, Cancun, Mexico, 2008. IEEE, IEEE.

[43] Ryan McKenna and Daniel R Sheldon. Permute-and-flip: A new mechanism for differentially private selection. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 193–203. Curran Associates, Inc., 2020.

[44] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, page 94–103, USA, 2007. IEEE Computer Society.

[45] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275, 2017.

[46] Melanie Mitchell. *An introduction to genetic algorithms.* MIT press, 1998.

[47] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. Gupt: Privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM*

SIGMOD International Conference on Management of Data, SIGMOD '12, page 349–360, New York, NY, USA, 2012. Association for Computing Machinery.

[48] Huy L. Nguyen, Anamay Chaturvedi, and Eric Z Xu. Differentially private k-means via exponential mechanism and max cover. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9101–9108, May 2021.

[49] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 547–561. University of California Press, 1961.

[50] Dirk Schlierkamp-Voosen and H Mühlenbein. Predictive models for the breeder genetic algorithm. *Evolutionary Computation*, 1(1):25–49, 1993.

[51] Adam Smith. Privacy-preserving statistical estimation with optimal convergence rates. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, page 813–822, New York, NY, USA, 2011. Association for Computing Machinery.

[52] Shuang Song, Kamalika Chaudhuri, and Anand D. Sarwate. Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 245–248, 2013.

[53] Thomas Steinke and Jonathan Ullman. Tight lower bounds for differentially private selection. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 552–563. IEEE, 2017.

[54] Dong Su, Jianneng Cao, Ninghui Li, and Min Lyu. Privpfc: Differentially private data publication for classification. *The VLDB Journal*, 27(2):201–223, April 2018.

[55] Kunal Talwar, Abhradeep Thakurta, and Li Zhang. Private empirical risk minimization beyond the worst case: The effect of the constraint set geometry, 2016.

[56] T.W. Then and E.K.P. Chong. Genetic algorithms in noisy environment. In *Proceedings of 1994 9th IEEE International Symposium on Intelligent Control*, pages 225–230, 1994.

[57] Oliver Williams and Frank Mcsherry. Probabilistic inference and differential privacy. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[58] Xi Wu, Fengan Li, Arun Kumar, Kamalika Chaudhuri, Somesh Jha, and Jeffrey Naughton. Bolt-on differential privacy for scalable stochastic gradient descent-based analytics. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1307–1322, New York, NY, USA, 2017. Association for Computing Machinery.

[59] I-Cheng Yeh and Che hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2, Part 1):2473–2480, 2009.

[60] Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Privbayes: Private data release via bayesian networks. *ACM Trans. Database Syst.*, 42(4), October 2017.

[61] Jun Zhang, Xiaokui Xiao, Yin Yang, Zhenjie Zhang, and Marianne Winslett. Privgene: Differentially private model fitting using genetic algorithms. New York, NY, USA, 2013. Association for Computing Machinery.

[62] Jun Zhang, Zhenjie Zhang, Xiaokui Xiao, Yin Yang, and Marianne Winslett. Functional mechanism: Regression analysis under differential privacy. *Proc. VLDB Endow.*, 5(11):1364–1375, July 2012.

[63] Ezgi Zorarpacı and Selma Ayşe Özel. Differentially private 1r classification algorithm using artificial bee colony and differential evolution. *Engineering Applications of Artificial Intelligence*, 94:103813, 2020.