PAC-Private Algorithms

Mayuri Sridhar
MIT CSAIL
Cambridge, MA, 02139
Email: mayuri@mit.edu

Hanshen Xiao
Purdue University/NVIDIA Research
West Lafayette, Indiana, 47907
Email: hsxiao.purdue@gmail.com

Srinivas Devadas
MIT CSAIL
Cambridge, MA, 02139
Email: devadas@mit.edu

Abstract—**Provable privacy typically requires involved analysis and is often associated with unacceptable accuracy loss. While many empirical verification or approximation methods, such as Membership Inference Attacks (MIA) and Differential Privacy Auditing (DPA), have been proposed, these do not offer rigorous privacy guarantees. In this paper, we apply recently-proposed Probably Approximately Correct (PAC) Privacy to give formal, mechanized, simulation-based proofs for a range of practical, black-box algorithms: K-Means, Support Vector Machines (SVM), Principal Component Analysis (PCA) and Random Forests. To provide these proofs, we present a new simulation algorithm that efficiently determines *anisotropic* noise perturbation required for any given level of privacy. We provide a proof of correctness for this algorithm and demonstrate that anisotropic noise has substantive benefits over isotropic noise.**

Stable algorithms are easier to privatize, and we demonstrate privacy amplification resulting from introducing regularization in these algorithms; meaningful privacy guarantees are obtained with small losses in accuracy. We propose new techniques in order to reduce instability in algorithmic output and convert intractable geometric stability verification into efficient deterministic stability verification. Thorough experiments are included, and we validate our provable adversarial inference hardness against state-of-the-art empirical attacks.

1. Introduction

The expansion of data collection and increasing complexity of data processing are happening at unprecedented rates. Concerns on information leakage are receiving increasing attention, while privacy preservation is simultaneously challenged by fast-paced and sophisticated advancements. Efficient and widely-applicable risk quantification has become a fundamental and urgent problem in privacy research. Most existing provable privacy analyses of data processing require strong assumptions. For example, Differential Privacy (DP) [1] requires bounded sensitivity [1], which can only be tightly computed in a few simple applications such as aggregation or linear queries. Further, Maximal Leakage (MaxL) [3] requires a white-box analysis of the likelihood functions, which is often complex. Moreover, to ensure these *input-independent* indistinguishability guarantees, artificial modifications are typically required to decompose most algorithms into multiple simpler and analyzable components, such as mean estimation or majority voting to enable tractable analysis; Differentially-Private Stochastic Gradient Descent (DP-SGD) [4] and PATE [5] are representative examples. Unfortunately, artificial modifications usually come with limits on algorithms and data structures, and often with a significant compromise on utility.

As a consequence, the lack of powerful risk quantification tools heavily restricts the study and design of defensive methods for leakage control, as the privacy implications of many operations are not well-understood. Even for perturbation, the most popular and straightforward privacy-preserving technique, the minimal noise to produce required security parameters, largely remains open for most practical algorithms. In addition, the definition of sensitive information varies across different processing tasks and different individual preferences. For example, for image data, people may worry about whether the adversary can reconstruct sensitive face features; for health data, the privacy objective can be the relationship between certain associations between patients and diseases; and in anonymous communication, identities are sensitive. Universal risk quantification is thus highly desirable to capture diverse and customized concerns.

Besides provable analyses, there is also a long line of works focusing on empirical defenses against adversarial inference. Privacy verification has been extensively studied, in particular, for membership inference attacks (MIA) [6], [7], [8]. For example, many operations such as regularization [6], [9], data augmentation [10], [11], and model compression [12] are empirically shown to resist certain kinds of attacks. However, qualitative analysis for those strategies is challenging and largely remains open, especially in involved data processing algorithms. Though carefully-designed empirical simulations can provide meaningful approximations of privacy risks with respect to *specific* adversarial strategies, a rigorous proof is desired to show worst-case guarantees against *arbitrary* adversaries. Closing this gap remains a key and open problem in security and privacy research.

One recent effort to technically address the risk quantification for black-box data processing is PAC Privacy [13]. From a statistical inference perspective, [13] develops a

1. In the context of DP, sensitivity captures the worst-case influence of an individual on the output, which is in general NP-hard to compute [2].

new framework to semantically interpret privacy risk as concrete inference hardness for a computationally-unbounded adversary to recover sensitive information satisfying a certain criterion, which can be arbitrarily selected. A set of new tools are also established in [13] to provably convert the objective inference hardness into simulatable quantities, which enables high-confidence estimation from end-to-end black-box simulations to provide a privacy proof. However, as a theoretical solution to conduct privacy analysis for a black-box processing, there are two important aspects of PAC Privacy which have not been systematically explored. First, how can we *efficiently* determine the (near-)optimal *anisotropic* noise[2] to add to each exposed output, and provide an associated privacy proof? Second, how can we *stabilize* a black-box data processing algorithm to provably produce a stronger privacy guarantee or a sharpened utility-privacy tradeoff?

In this paper, we contribute an initial comprehensive study to answer these questions, as summarized below.

1) **Novel algorithm for efficient simulation proofs**: We present an algorithm in Section 4 that adds anisotropic noise and is more computationally efficient than the algorithm (Algorithm 1) in [13] which requires running Singular Value Decomposition (SVD) on the *entire* output dimension which can be prohibitively expensive. Efficiency is further enhanced through faster convergence; our algorithm only needs to accurately estimate variance of output along each direction, as opposed to converging on a covariance matrix as in [13]. We prove the correctness of our algorithm.
2) **Efficient privatization for black-box algorithms**: We implement PAC-private versions of several classic algorithms. We provide noise estimates and utility tradeoffs, demonstrating that these algorithms can generally achieve meaningful privacy with a small losses in utility. We show that adding anisotropic noise has significant utility benefits over adding isotropic noise using l_2-norm estimation.
3) **Sharpening privacy-utility tradeoffs**: We first characterize the root of instability in these classic algorithms, and separate them into two large categories – superficial and intrinsic. Then, we provide novel canonicalization techniques to improve upon superficial instability, while exploring classic and novel techniques, based on regularization and data augmentation techniques, to improve intrinsic instability. In particular, we show how the use of a random unitary matrix in Principal Component Analysis (PCA) can essentially eliminate superficial instability.
4) **Empirical verification for end-to-end privacy**: Finally, we provide experimental support, based on simulated attacks to validate our privacy guarantees. We convert the theoretical mutual information guarantees into posterior guarantees and demonstrate that our privatized algorithms more than satisfy these guarantees against state-of-the-art attacks.

2. Noise varying across output dimensions.

2. Background

We first introduce the PAC Privacy model to describe information leakage and privacy risk in general. Let X denote the sensitive input, which is randomly generated from a (possibly black-box) distribution D, and $\mathcal{M}$ denote a (possibly black-box) processing mechanism, where the output, $\mathcal{M}(X)$, is released and observed by an adversary. We challenge the adversary as to whether they can return some estimation $\tilde{X}$ satisfying a certain criterion, which can be described by some indicator function ρ. Such an inference challenge can be used to capture arbitrary privacy concerns and customized leakage control that a user is comfortable with. For example, to capture a membership inference attack [6], ρ can be selected as $\rho(\tilde{X}, X) = 1$ if $\tilde{X}$ predicts the membership of some particular datapoint u_0 correctly in X; ρ may also capture data reconstruction [14], [15] where we may set $\rho(\tilde{X}, X) = 1$ iff $\|X - \tilde{X}\|_2 \leq 1$, i.e., the adversary can recover the input with error in l_2-norm smaller than 1. For side-channel attacks on a cryptographic protocol [16], where X corresponds to the secret key, ρ can capture the colliding bits between X and $\tilde{X}$.

Now, given the data entropy, determined by D, and the objective inference task, we can define the optimal *a priori* success rate $(1-\delta_o^\rho)$ that an adversary can return a satisfied estimation before they observe the release $\mathcal{M}(X)$, i.e.,

$$\delta_o^\rho = \inf_{\tilde{X}_o} \Pr_{X\sim\mathsf{D}}(\rho(\tilde{X}_o, X) \neq 1).$$

Similarly, we can define the posterior success rate $(1-\delta)$ to capture the probability for an adversary to return a satisfied estimation after observing the release. With the above preparation, we can now formally define PAC Privacy.

Definition 1 ($(\delta, \rho, \mathsf{D})$ PAC Privacy [13])**.** *For a processing function $\mathcal{M} : \mathcal{X}^* \to \mathcal{O}$, some data distribution D, and an inference criterion function $\rho(\cdot,\cdot)$, we say $\mathcal{M}$ satisfies $(\delta, \rho, \mathsf{D})$-PAC Privacy if the following experiment is impossible:*

A user generates data X from distribution D and sends $\mathcal{M}(X)$ to an informed adversary. The adversary who knows D and $\mathcal{M}$ is asked to return an estimation $\hat{X}$ on X such that with probability at least $(1-\delta)$, $\rho(\hat{X}, X) = 1$.

Equivalently, $\mathcal{M}$ can be defined as $(\Delta_f\delta, \rho, \mathsf{D})$ PAC-advantage private if the posterior advantage measured in f-divergence satisfies

$$\Delta_f\delta = \mathcal{D}_f(\mathbf{1}_\delta \| \mathbf{1}_{\delta_o^\rho}) = \delta_o^\rho f(\frac{\delta}{\delta_o^\rho}) + (1-\delta_o^\rho) f(\frac{1-\delta}{1-\delta_o^\rho}),$$

where $(1-\delta_o^\rho)$ represents the optimal prior success rate,

$$\delta_o^\rho = \inf_{X' \in \mathcal{X}^*} \Pr_{X\sim\mathsf{D}}(\rho(X', X) \neq 1),$$

and $\mathbf{1}_\delta$ and $\mathbf{1}_{\delta_o^\rho}$ represent two Bernoulli distributions of parameters δ and δ_o^ρ, respectively. Here, $\mathcal{D}_f$ is some f-divergence.

In [13], $\mathcal{D}_f$ is selected to be the KL-divergence and it is shown that,

$$\Delta_{KL}\delta = \mathcal{D}_{KL}(\mathbf{1}_\delta \| \mathbf{1}_{\delta_o^\rho}) \leq \mathsf{MI}\big(X; \mathcal{M}(X)\big), \tag{1}$$

where $\mathsf{MI}(\cdot,\cdot)$ represents *mutual information* and $\mathcal{D}_{KL}(\mathbf{1}_\delta \| \mathbf{1}_{\delta_o^\rho}) = \delta \ln(\frac{\delta}{\delta_o^\rho}) + (1-\delta)\ln(\frac{1-\delta}{1-\delta_o^\rho})$.

We now define the standard Membership Inference Attack (MIA) [6], formalized to match PAC Privacy below.

Definition 2 (Membership Inference Attack)**.** *Given a finite data pool $\mathsf{U} = \{u_1, u_2, \cdots, u_N\}$ and some processing mechanism $\mathcal{M}$, X is an n-subset of U randomly selected. An informed adversary is asked to return an n-subset $\hat{X}$ as the membership estimation of X after observing $\mathcal{M}(X)$. We say $\mathcal{M}$ is resistant to $(1-\delta_i)$ individual membership inference for the i-th datapoint u_i, if for an arbitrary adversary, $\Pr_{X \leftarrow \mathsf{U}, \hat{X} \leftarrow \mathcal{M}(X)}(\mathbf{1}_{u_i \in X} = \mathbf{1}_{u_i \in \hat{X}}) \leq 1 - \delta_i$. Here, $\mathbf{1}_{u_i \in X}$ ($\mathbf{1}_{u_i \in \hat{X}}$) is an indicator which equals 1 if u_i is in X ($\hat{X}$).*

In this paper, we will use PAC Privacy to provably and automatically measure the privacy risk. We will also qualitatively (and occasionally quantitatively) compare our results to prior work with Differential Privacy (DP); its formal definition is presented below.

Definition 3 ($(\epsilon, \bar{\delta})$ Differential Privacy [1])**.** *Given a data universe $\mathcal{X}^*$, we say that two datasets $\mathcal{S}, \mathcal{S}' \subseteq \mathcal{X}^*$ are adjacent, denoted as $\mathcal{S} \sim \mathcal{S}'$, if $\mathcal{S} = \mathcal{S}' \cup s$ or $\mathcal{S}' = \mathcal{S} \cup s$ for some additional datapoint s. A randomized processing function $\mathcal{M}$ is said to be $(\epsilon, \bar{\delta})$-differentially-private (DP) if for any pair of adjacent datasets $\mathcal{S}, \mathcal{S}'$ and any set o in the output space $\mathcal{O}$ of $\mathcal{M}$, it holds that $\Pr(\mathcal{M}(\mathcal{S}) \in o) \leq e^\epsilon \cdot \Pr(\mathcal{M}(\mathcal{S}') \in o) + \bar{\delta}$.*

We can interpret DP in a context of the posterior success rate for successful membership inference. In the same setup of Definition 2, if $n = \frac{N}{2}$ [3], i.e., each datapoint is included in X with probability 1/2, and $\mathcal{M}$ satisfies $(\epsilon, \bar{\delta})$-DP, then by [17], [18], the posterior success rate $(1-\delta_i)$ is upper bounded by

$$1 - \delta_i \leq 1 - \frac{1-\bar{\delta}}{1+e^\epsilon}. \qquad (2)$$

3. Automatic Privatization

3.1. A Template for Provable Privacy

In this section, we present a formal template for privatizing black-box algorithms using PAC Privacy. The key steps of this technique are summarized in Figure 1.

In particular, we consider any black-box algorithm $\mathcal{M}$. The goal of our template is to release $Y = \mathcal{M}(X)$ for a **secret** input X_j. We want to bound the *posterior* advantage the adversary gains upon observing $Y_j = \mathcal{M}(X_j)$ by adding noise to Y_j. Y_j is an arbitrary learned statistic about the input X_j that is **exposed** to the adversary after appropriate noise is added. We denote X_{train} as the complete training dataset which X_j is sampled from. We denote

$$r := \frac{|X_j|}{|X_{train}|}$$

3. For the general case, one can perform similar reasoning by solving a constrained linear program with respect to Type I and Type II errors as described by Eqn. (1) in [17].

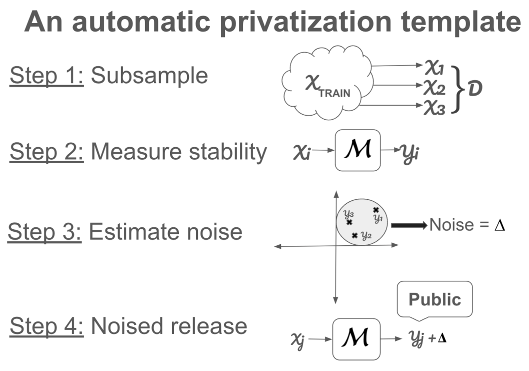

Figure 1. A simple 4-step process to automatically privatize a black-box algorithm $\mathcal{M}$. We first measure the stability of $\mathcal{M}$ by computing $Y_i = \mathcal{M}(X_i)$ on varying subsets of data X_i. We then use the variance of the output distribution Y_i to estimate the required noise necessary to privatize $\mathcal{M}$. Finally, we release a noisy version of the learned vector on a new, random subset X_j.

as our subsampling rate; that is, r is the fraction of the data that will be used to learn our released output Y_j. For our experiments, we choose $r = 0.5$, or 50%. Let D be the uniformly random distribution over all possible $r|X_{train}|$-sized subsets of X_{train}; as such, X_j is sampled from D.

We represent each $x \in X_j$ as a vector, with its l_2-norm bounded by a known constant. In order to provide a private representation of $Y = \mathcal{M}(X)$, we follow the framework of PAC Privacy [13] to determine the minimal noise we must add to the vector Y. We aim to minimize the noise to maximize *utility* of the output vector Y. Following the steps described in Figure 1, we can use the stability of $\mathcal{M}$ on distinct X_i's (which are drawn from the same *distribution* D as X_j) to determine the required noise to provide privacy for $\mathcal{M}(X_j)$.

To do this, we first compute $\mathcal{M}$ on distinct subsampled datasets $X_1 \dots X_m$, which are independent and identically distributed subsets of X_{train}, drawn from the same distribution as X_j, which means $|X_i| = |X_j|$. We denote m as the *round complexity* of the noise estimation process. The value of m is induced by the required precision of our convergence guarantee. This computation produces output vectors $Y_1 \dots Y_m$. We describe this in further detail in Section 4. We can then use the variance of the Y_i's (along with appropriate security parameters) to estimate the minimum noise required to add to the output of $\mathcal{M}(X_j)$, in order to provide a meaningful bound on the mutual information, which in turn bounds the posterior advantage. An illustrative instantiation of this procedure on the K-Means algorithm is provided in Figure 2.

In our analysis, we make the conservative assumption that the adversary knows D, meaning the adversary knows X_{train} and the sampling strategy, which is not typically true in the real world. Importantly, the computed posterior advantage will hold for this adversary or a weaker one with only partial or no knowledge of X_{train}. Note that the **sampled X_j is hidden from the adversary**, and therefore the participation of a particular data element in X_j is unknown to the adversary.

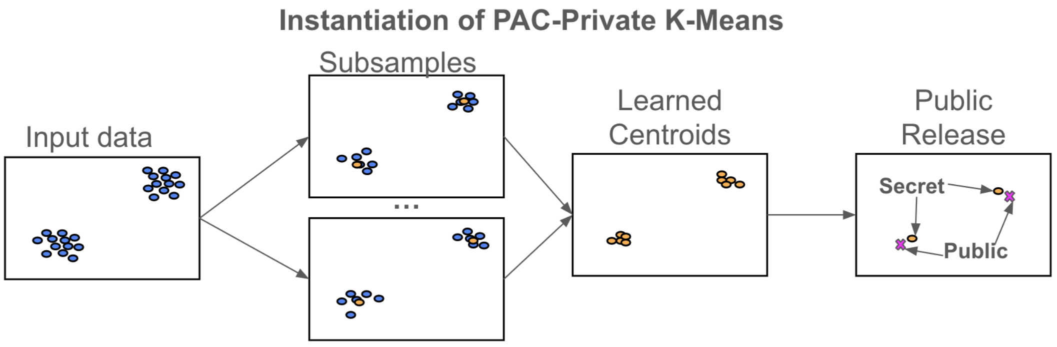

Figure 2. To instantiate PAC-Private K-Means, we first start with our input dataset, X_{train}, of size n. We approximate the distribution D by drawing m samples $X_1 \dots X_m$, where each X_i is a random subset of X_{train}, with $|X_i| = n/2$. We then compute the learned centroids of each X_i. The variance among these centroids determines the noise required to privatize our algorithm. When the centroids are close together, the algorithm is stable and thus, the required noise to privatize it is small. Finally, we generate a new, random subset X_j and compute the corresponding centroids; X_j is the *secret* set that the adversary wants to discover. We add the required noise and publicly release the perturbed centroids in the last step. The noise guarantees that the adversary has a bounded advantage in identifying whether any individual datapoint was used to construct the final released centroids.

The posterior advantage holds for an *arbitrary* inference task ρ on the input dataset X. The classic membership attack by [6] defines a specific ρ, where the goal is to determine whether a fixed sample x was included in X; that is, $\rho(\bar{X}, X) = 1$ if $\bar{X}$ correctly predicts the membership of x. Other attacks may include reconstruction attacks (recovering X), norm estimation, or others, e.g., [14], [15].

We make a few key observations about this template:

1) $\mathcal{M}$ is treated as a black-box. The magnitude of the added noise depends solely on the output distribution of Y_i's. This allows us to generate a privacy template for complex *black-box* algorithms in an instance-specific (i.e., specific to X_{train}) manner.
2) We observe that the magnitude of added noise only impacts the *posterior* advantage of the inference task. We make *no* assumptions on the specific inference task of the adversary; rather, PAC privacy allows us to bound the mutual information between the output Y and the secret input X, bounding the maximal posterior advantage. We further discuss the relationship between mutual information and the posterior advantage for specific membership inference attacks in Section 8.
3) In order to meaningfully measure the (co)variance across Y_i's, the outputs on varying inputs X_i must lie on the same output space. In particular, we must *canonicalize* our outputs. That is, if we assume Y_i is a learned vector, it must remain in the same order and even simpler, the same length. For certain tasks, like regression, this appears obvious, since a learned weight vector has fixed dimension and order. However, for unsupervised learning tasks (e.g., clustering), or certain classification algorithms, this becomes non-trivial.

We can then use Equation (1) to convert the mutual information guarantee to the maximal posterior advantage for an arbitrary inference task. We can expand Equation (1) below as

$$p_o \ln\left(\frac{p_o}{p}\right) + (1-p_o)\ln\left(\frac{1-p_o}{1-p}\right) \le \mathsf{MI}(X_i; Y_i) \quad (3)$$

where p is the prior success rate and p_o is the posterior success rate. Table 1 provides the theoretical maximal posterior success rates for two different prior success rates of 50% and 1%. The prior success rate p for a subsampling rate r equals $\max(r, 1-r)$ for an *individual* membership inference task; we choose $r = 0.5$ to minimize p to 50%.[4] However, p can be much lower for a generalized membership inference task for the same r (e.g., 1%) (cf. Appendix C).

Mutual Information	Posterior Success Rate (p_o)	
	Prior p = 50%	Prior p = 1%
1/128	56.241%	2.477%
1/64	58.815%	3.213%
1/32	62.434%	4.364%
1/16	67.490%	6.200%
1/8	74.464%	9.171%
1/4	83.789%	14.057%
1/2	95.181%	22.177%
1	100%	35.729%
2	100%	58.103%
4	100%	92.582%

TABLE 1. MUTUAL INFORMATION CAN BE RELATED TO THE THEORETICAL MAXIMAL POSTERIOR SUCCESS RATE FOR DIFFERENT PRIOR SUCCESS RATES USING EQUATION (3).

We can use Equation (2) to interpret $(\epsilon, \bar{\delta})$-DP parameters as posterior success rates. For example, a (0.36,0)-DP ((2.98,0)-DP) corresponds to a posterior success rate of 58.815% (95.181%) for a prior of 50%. This is useful in calibrating mutual information in Table 1 with a DP ϵ.

3.2. Privacy vs. Utility

In this section, we discuss techniques to *reduce the instability* in the outputs of our algorithms. That is, we first classify varying causes of instability in the output distributions for a black-box algorithm $\mathcal{M}$:

1) **Intrinsic instability:** We denote an algorithm's intrinsic instability as instability that cannot be reduced without *semantically* changing the output of the algorithm.
2) **Superficial instability:** We denote an algorithm's superficial instability as an instability in the output that

4. The prior success rate of positive identification of individual membership equals r.

does *not* reflect a semantic difference in the output. This can be addressed by *canonicalizing* our outputs, by representing them in a consistent manner.

In this work, we explore techniques to reduce both types of instability in a set of widely-used algorithms.

We first consider a simple example of superficial instability in unsupervised learning algorithms. In general, unsupervised learning algorithms provide a mechanism for *clustering*. However, by definition, these clusters do not have labels. Thus, an algorithm could return the same set of clusters in any order; while the ordered vector appears very different, the true result is semantically the same. In this case, canonicalizing the output is near-trivial; we can simply assign arbitrary labels to each cluster and choose labels to minimize the variance across Y_i's.

We now consider an example of intrinsic instability. In this, we consider the random forest algorithm [19]. The goal of this algorithm is to classify different classes within a dataset, by constructing several decision trees. Each decision tree chooses a subset of features to train on; then, each level of the tree splits the dataset into subsets in order to minimize entropy or Gini impurity [19]. These algorithms are known to be unstable, since small changes to the input dataset can lead to significant changes in the threshold values. In Section 5, we discuss how we modify this algorithm to provide meaningful guarantees in our framework.

Finally, we note that in the classic non-private setting for these algorithms, stability is useful primarily as a proxy for understanding the generalizability of these algorithms. However, in our setting, stability directly affects the utility of the privatized algorithm, since it is inversely correlated with the total added noise. This implies that efficiently privatizing these algorithms involves an inner optimization problem, similar to the hyperparameter search typically done using cross-validation. We discuss heuristic strategies for this search and empirical results in Section 6.

4. Efficiently Computing Anisotropic Noise

In this section, we formally describe a "best of both worlds" algorithm that is as efficient as the isotropic noise addition algorithm of [13] while computing anisotropic noise that minimally affects utility. We then prove that the noise mechanism satisfies the mutual information guarantees.

4.1. Noise Determination and Guarantees

The algorithm is described in full in Algorithm 1. We denote n as the number of input elements, A as a unitary projection matrix, τ as the precision required for convergence, f_τ as a function measuring whether our estimator for the variance has converged, and d as the output dimension. For any matrix X, we use the notation X_k to denote the k'th column and $X[i][j]$ to represent the element at row i and column j. For a vector v, we denote $v[i]$ as the i'th element. In this algorithm, we compute a matrix $\boldsymbol{G}$, with dimension $m \times d$, and $\boldsymbol{\sigma_m}[k] := Var(G_k)$ represents our estimate for the variance of $\mathcal{M}(X)$ along direction A_k. We choose m such that m is large enough that our convergence criterion, represented by f_τ is satisfied. In particular, for our experiments, we choose f_τ to be the maximal element-wise difference between $\boldsymbol{\sigma_t}$ and $\boldsymbol{\sigma_{t-1}}$ with $\tau = 10^{-6}$, where $\boldsymbol{\sigma_t}$ represents the empirical estimate of the variance in the directions A_i for $i \in [1, d]$ at round t. After computing $\Sigma_\mathcal{B}$, we add Gaussian noise $\mathcal{B} \sim \mathcal{N}(\mathbf{0}, \Sigma_\mathcal{B} A^T)$ to each element of the output $\mathcal{M}(X)$. For our experiments, we choose $A = \mathbf{I}_d$.

Algorithm 1 Anisotropic Noise Determination of Deterministic Mechanism $\mathcal{M}$

```
Input: The input distribution D represented by $X_{train}$ and a sampling strategy, $\tau$ as the precision required for convergence, $f_\tau$ as the convergence function, deterministic mechanism $\mathcal{M} : \mathcal{X}^n \to \mathcal{Y}^d$, mutual information requirement $\beta$, $d \times d$ unitary projection matrix $A$.
1) $m := 1$, $\boldsymbol{\sigma_0} :=$ null, $\boldsymbol{G} :=$ null.
2) while $m \le 2 \parallel f_\tau(\boldsymbol{\sigma_{m-1}}, \boldsymbol{\sigma_m}) \ge \tau$:
    a) Draw $X_m \sim$ D.
    b) $y_m := \mathcal{M}(X_m)$.
    c) Compute $\boldsymbol{g_m} = [y_m \cdot A_1, y_m \cdot A_2, \ldots, y_m \cdot A_d]$.
    d) Row append $\boldsymbol{g_m}$ to $\boldsymbol{G}$:
           $G_m^T := \boldsymbol{g_m}$.
    e) Compute the vector $\boldsymbol{\sigma_m}$ where $\boldsymbol{\sigma_m}$ is a vector of length $d$ and $\boldsymbol{\sigma_m}[k]$ is the empirical variance of $G_k$.
    f) $m := m + 1$.
3) Calculate the required noise in each direction $i$ as
       $e_i := \frac{\sqrt{\boldsymbol{\sigma_m}[i]} \sum_{j=1}^{d} \sqrt{\boldsymbol{\sigma_m}[j]}}{2\beta}$ for $i \in [1, d]$.
4) Return a diagonal matrix $\Sigma_\mathcal{B}$, where $\Sigma_\mathcal{B}[i][i] = e_i$.
```

Theorem 1. *For an arbitrary deterministic mechanism $\mathcal{M}$, a public unitary matrix A, and Gaussian noise of the form $\mathcal{B} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_\mathcal{B} A^T)$, where $\sigma_i = Var(\mathcal{M}(X) \cdot A_i)$ and Σ_B is a diagonal matrix with entries*

$$e_i := \frac{\sqrt{\sigma_i} \sum_{j=1}^{d} \sqrt{\sigma_j}}{2\beta},$$

the output $M(X) + \mathcal{B}$ satisfies

$$\mathsf{MI}(X; \mathcal{M}(X) + \mathcal{B}) \le \beta.$$

Proof. We first recall Theorem 3 of [13]; this theorem states that

$$\mathsf{MI}(X; \mathcal{M}(X) + \mathcal{B}) \le \frac{1}{2} \ln \det(\mathbf{I}_d + \Sigma_{\mathcal{M}(X)} \Sigma_\mathcal{B}^{-1}).$$

We then note that

$$\mathsf{MI}(X; \mathcal{M}(X) + \mathcal{B}) = \mathsf{MI}(X; \mathcal{M}(X)A + \mathcal{B}A),$$

since A is unitary and public.

By Hadamard's inequality, since $\Sigma_{\mathcal{M}(X)A}$ is positive semi-definite,

$$\det(\Sigma_{\mathcal{M}(X)A}) \leq \det(\operatorname{diag}(\Sigma_{\mathcal{M}(X)A})),$$

where $\operatorname{diag}(\Sigma_{\mathcal{M}(X)A})$ is the diagonal matrix with the i'th element σ_i. By construction, $\mathcal{B}A$ has variance $\Sigma_{\mathcal{B}}$, which is a diagonal matrix with elements e_i. Thus,

$$\begin{aligned}
\mathsf{MI}(X;\mathcal{M}(X)A+\mathcal{B}A) &\leq \frac{1}{2}\ln\det(\mathbf{I}_d+\Sigma_{\mathcal{M}(X)A}\Sigma_{\mathcal{B}}^{-1}) \\
&\leq \frac{1}{2}\ln\det(\mathbf{I}_d+\operatorname{diag}(\Sigma_{\mathcal{M}(X)A})\Sigma_{\mathcal{B}}^{-1}) \\
&= \frac{1}{2}\ln\prod_i(1+\sigma_i\frac{2\beta}{\sqrt{\sigma_i}\sum_j\sqrt{\sigma_j}}) \\
&= \frac{1}{2}\sum_i\ln(1+\frac{2\beta\sqrt{\sigma_i}}{\sum_j\sqrt{\sigma_j}}) \\
&\leq \frac{1}{2}\sum_i\frac{2\beta\sqrt{\sigma_i}}{\sum_j\sqrt{\sigma_j}} \\
&\leq \beta,
\end{aligned}$$

where the fifth inequality uses the fact that $\ln(1+x)\leq x$. $\square$

Theorem 1 shows that if Algorithm 1 computes the exact variance of $\mathcal{M}(X)$ over the directions A_i for $i\in[1,d]$, we can privatize any black-box mechanism $\mathcal{M}$. The primary advantage of Algorithm 1 is that it avoids building the covariance matrix and subsequent SVD (as in Algorithm 1 of [13]), while determining sufficient anisotropic noise for privacy. The optimal A for minimal noise can be determined by estimating the covariance matrix and using SVD. However, we note that Algorithm 1 computes an *empirical* estimate on the variance, rather than the true variance. In practice, using $A=\boldsymbol{I}_d$ allows us to estimate $\boldsymbol{\sigma}$ for $\tau=10^{-6}$ with reasonably small m (cf. Section 7) for most algorithms. However, unstable algorithms (e.g., decision trees) on large datasets are very computationally expensive even in this model. We therefore discuss an alternative, more efficient approach where we compute the *exact* variance with a small change in how the distribution D is constructed — see Appendix B.

We can further provide tighter bounds on the required noise when considering specific inference tasks, e.g., individual membership inference attacks (cf. Definition 2). We use these individual privacy guarantees to provide a quantitative comparison between PAC and DP for mean estimation. A complete description of the modifications required to Algorithm 1 for individual privacy is provided in Appendix A.

5. Algorithms

In this section, we discuss several classic algorithms and the required modifications to automatically privatize them.

For all the algorithms, we first normalize our data and separate it into a training dataset and a test dataset. We then measure "accuracy" (also referred to as utility) on the test dataset. All randomized algorithms are run with a fixed seed.

5.1. Clustering: K-Means

The K-Means clustering algorithm, originally developed by Lloyd in 1982, aims to partition an input set X into K non-overlapping subsets or *clusters* [20], [21]. Each subset $i\in[1,K]$ is defined by its centroid, μ_i. The objective is to minimize the sum-of-squares within each cluster, over all the clusters, i.e.,

$$\arg\min_{\mu}\sum_{i=0}^{n}\min_{\mu_j}\left\|x_i-\mu_j\right\|^2.$$

That is, the classic algorithm outputs a list of centroids corresponding to each cluster. We observe that K-Means requires minimal changes to fit into our PAC privacy framework. The x_i's are the **secret** input, and the learned centroids μ_i's are the **exposed** output.

In order to canonicalize the output, we simply order these centroids by inferring appropriate cluster labels. For supervised learning, we do this by choosing the label that is best associated with each cluster. We then measure test accuracy by comparing the inferred cluster label with the true class label on the test dataset.

The K-Means algorithm is not inherently designed for unbalanced datasets. To improve stability and generalization, we explore oversampling techniques such as SMOTE to automatically balance the classes [22].

5.2. Classification: SVM

Consider the multiclass support vector machine algorithm [23], [24]. The linear support vector machine problem solves the following optimization problem:

$$\min_{w,b}\quad \frac{1}{2}w^Tw+C\sum_{i=1}^{n}\max(0,1-y_i(w^Tx_i+b)). \tag{4}$$

Here, the x_i's are the features, and the y_i's are the labels; these both correspond to the **secret** inputs. The learned weight vector $w,\ b$ is the **exposed** output.

We use the regularization weight C to trade off between the hinge loss and the norm of the learned weight vector. Without modification, the standard value of C used is 1. To accommodate multiclass strategies, we consider a one-versus-rest classification strategy. That is, we train K classifiers for K different classes [25].

After the weight vector has been trained, we can use it to compute a "per-class" score for a new point x_i. The predicted label $\hat{y}_i$ is the class with the highest score. Similar to K-Means, we measure the accuracy of the test dataset by computing the class label predicted by SVM to the true label.

We note that this algorithm may or may not have a lot of superficial instability. That is, the learned weight vectors are inherently ordered by the labels of their corresponding classes; it thus requires almost no modification to fit into the PAC privacy framework. However, there may be several near-optimal solutions with no obvious ordering when regularization is not applied appropriately. Strong regularization (low values of C) can reduce the algorithm's intrinsic instability, though it may come with a utility tradeoff.

5.3. Dimensionality Reduction: PCA

Consider the classic dimensionality reduction algorithm, principal component analysis (PCA) [26]. PCA is used to decompose a multivariate dataset into orthogonal components that explain the most variance.

Unlike the other algorithms considered, PCA is not independently used for a regression or classification task; rather, it is typically a subroutine. We consider an initial data matrix $X \in \mathbb{R}^{m \times d}$, with m samples of dimension d, where $X \subset X_{train}$ is **secret**. We then reduce the dimensionality of X to be in $\mathbb{R}^{m \times d'}$ using PCA; that is, we compute the top d' principal components and denote them as a matrix $S \in \mathbb{R}^{d' \times d}$. S is the **exposed** output.

We observe that PCA has significant superficial instability. In particular, we consider the subspace defined by the basis vectors $[0, 1]$ and $[1, 0]$; this subspace is $\mathbb{R}^2$. However, there are an infinite number of basis vectors with the same span; in fact, any two linearly-independent vectors span $\mathbb{R}^2$. This implies that two calls to the PCA algorithm can return the *same* subspace, represented by *significantly* different basis vectors.

In order to canonicalize the basis vectors, we consider two instances of the PCA algorithm and denote the returned basis vectors as S_1 and S_2, where $S_i \in \mathbb{R}^{d' \times d}$ for $i = 1, 2$. We observe that we can choose a unitary matrix M and compute MS_2 as an equivalent representation of the basis chosen by S_2. The goal is now to choose M in order to minimize the distances between S_1 and S_2; we use this formulation and the properties of singular value decomposition (SVD) to compute the optimal M. We note that the SVD decomposition is unique up to the sign of the right and left singular vectors.

Consider the following optimization problem:

$$\min_{M; M^T M = I} \quad \|A - MB\|_F^2 ,$$

where A, B are matrices of basis vectors, with dimensionality $d' \times d$ and M is any unitary matrix. We first observe that this models our PCA problem *exactly*; that is, PCA returns to us a set of d' basis vectors with dimensionality d. We can freely optimize over the matrix M as long as it remains unitary, since M is simply a linear map.

Claim 1. *The optimal choice for M is of the form $M = C[0 : d', 0 : d']$, where*

$$C = V_A V_B^T ,$$

and V_A, V_B are the right singular vectors of A and B, respectively.

Proof. We prove this directly from the optimization problem. That is, M is chosen to minimize

$$\min_{M; M^T M = I} \quad \|A - MB\|_F^2 ,$$

for fixed matrices A and B of dimension $d' \times d$. We suppress the unitary requirement on M for succinctness in the remainder of this argument. We first denote the SVDs of A and B as $U_A \Sigma_A V_A$ and $U_B \Sigma_B V_B$, respectively. We observe that $U_A \in \mathbb{R}^{d' \times d'}$, $\Sigma_A \in \mathbb{R}^{d' \times d}$, $V_A \in \mathbb{R}^{d \times d}$. We further note that Σ_A has d' real values on the diagonal and the remaining entries are 0, since the underlying rank of A is assumed to be d'. We can thus denote Σ_A^{-1} as the inverse of Σ_A where $\Sigma_A^{-1} \Sigma_A = [I_{d'} | 0]$. The same constraints follow for B.

We then observe that

$$\begin{aligned}
&\min_{M} \quad \|A - MB\|_F^2 \\
&= \min_{M} \quad \|U_A \Sigma_A V_A - M U_B \Sigma_B V_B\| \\
&= \min_{M} \quad \left\|\Sigma_A^{-1} U_A^T U_A \Sigma_A V_A - \Sigma_A^{-1} U_A^T M U_B \Sigma_B V_B\right\| \\
&= \min_{M'} \quad \left\|[I_{d'}|0] - M'[I_{d'}|0] V_B V_A^T\right\| .
\end{aligned}$$

We note that the fourth equality switches from optimizing over M to optimizing over a different matrix M' after factoring out the relevant components of the SVD of B; however, since M and M' are both free variables, this does not affect correctness. The optimal solution for this is $M = V_A V_B^T$, truncated to the first d' rows and columns. □

By canonicalizing our output, we reduce the superficial instability; "nearby" subspaces are represented by "nearby" basis vectors. Without the appropriate canonicalization, the PCA algorithm is very unstable and difficult to privatize.

For a given input X_{test}, after running PCA, the projection of X_{test} is computed as $X_{test} S^T$, representing the best projection of X_{test} into the learned rank-d' subspace. We can then "restore" X_{test} into the original rank-d subspace by computing $X_{test} S^T S$, also known as the PCA inverse transform; we denote this matrix as X' and calculate the restoration error as

$$\text{Restoration error (RE)} := \frac{\|X' - X_{test}\|}{\|X_{test}\|}. \qquad (5)$$

We use the restoration error as a proxy of our accuracy metric for other downstream tasks, since low RE would imply high success rate on any secondary task.

5.4. Boosting: Random Forest

As mentioned in Section 3, random forest algorithms typically involve both superficial *and* intrinsic instability, making them an interesting case study for our template.

We first describe the classic random forest algorithm [19] and then describe our modifications for canonicalization. The classic random forest algorithm is an ensemble learning technique which combines several weak classifiers (decision trees) to make an ensemble model which performs better than any of the individual trees. Typically, these decision trees are trained on subsets of the provided dataset and the final classification is the plurality vote of the individual trees. For each tree, the algorithm chooses a feature (or subset of features) to split on. Then, the "value" to split on is chosen to minimize a metric – in our case, we use the metric of weighted entropy. The provided dataset is the secret input, and the learned trees are the exposed output: a number of trees with corresponding structures and weights.

In our setting, we require that the trees all have the same *structure*. To simplify this, we ensure that all the trees are complete and split on the same order of features. Thus, our random forest algorithm has two hyperparameters: the number of trees (a classic requirement for an ensemble model) and the *ordered list* of the features to split on for each tree, denoted here as L. The structure of a tree is fully determined from the ordered set of features; that is, if there are d features, then the tree will have exactly 2^d leaf nodes. Each node at level i will split on feature $L[i]$; the exact value of the split threshold is determined by computing the minimum weighted entropy across all possible values of the feature $L[i]$.

In particular, each possible "split" on the feature $L[i]$ at value v splits the dataset into two sets $S_r(v)$ (containing elements where feature $L[i]$ has value $\geq v$) and $S_l(v)$ (containing elements where feature $L[i]$ has value $< v$). We can calculate the entropy of each split as

$$H(S) = \sum_j -p_j \log p_j,$$

where p_j is the empirical probability of element j (the frequency of item j in S divided by $|S|$). The total entropy of a split can be calculated as

$$H_v := |S_l(v)| H(S_l(v)) + |S_r(v)| H(S_r(v)).$$

We then choose the split that minimizes the weighted entropy.

We consider the choice of ordered features akin to early work in bagging schemes [27], where subsets of features were chosen for each tree. We pass in all the data to each decision tree and output a simple majority vote of the trees as the final decision of our random forest. In this setting, the features x_i and the labels y_i for our training data represent our **secret** input. The coefficients of the learned trees represent the **exposed** output. As with the prior algorithms, after the trees are exposed, we can measure the test accuracy by comparing the learned classification of a test data point to its true label.

We note that classic regularization schemes on decision trees (or random forests) focus on pruning the depth of the tree or allowing the tree to split on a maximum number of features at each level [19]. Neither of these are consistent with our framework. In particular, the former does not allow for an efficient canonicalization since the trees will have different structures. The latter is irrelevant for us, since our trees split on a single feature at each level.

We use two techniques intended to increase the stability of the random forest algorithm, following the form of regularization and data augmentation defenses suggested in [9] and [10]. First, we define a *data augmentation* defense. That is, we first discretize the possible split values of each level. Thus, the possible split values of a feature $L[i]$ are in the range $[0, 1]$, evenly divided into $1/p$ segments of length p, where p is a tunable hyperparameter (typically 0.01). Then, rather than just calculating the entropy of the split, we calculate our final split value as

$$v := \arg\min_v \quad (1 - w_1)H_v + w_1(H_{v-p} + H_{v+p}).$$

We denote the tuple (p, w_1) as our augmentation regularization parameter. If the entropies of the neighbors ($v - p$ and $v + p$) are also low, then this suggests that the split value v is robust to small amounts of perturbation. Increasing the weight w_1 forces the algorithm to choose a split that is more robust.

Second, we add l_1 *regularization*, which adds a penalty of the form $|w_2 v|$ for any split value v. This follows the classic l_1 regularization scheme, where we encourage sparsity in the learned split vector. This is especially important in our setting; intuitively, we only want the complete tree to learn a non-degenerate split if the change in entropy is *significant* and thus, the learned split is stable. Thus, our overall regularization parameter is of the form (p, w_1, w_2).

6. Experiments

6.1. Datasets

Iris dataset: The Iris dataset is available in the UC Irvine Machine Learning Repository [28]. It is a classic dataset used in machine learning for supervised and unsupervised learning tasks. The goal is to classify three class of irises; there are 50 instances of each class and 4 features; its small size makes privatization difficult. We use 100 datapoints as our training set and 50 as the test dataset.

Rice dataset: The Rice dataset is available in the UC Irvine Machine Learning Repository [29]. It contains 3,810 instances of rice, from two distinct species: Osmancik and Cammeo; the goal is to classify the species of rice. Each example contains 7 features such as area, perimeter and eccentricity. We use 70% of the dataset for training and the remaining 30% as the test dataset.

Dry Bean dataset: The Dry Bean dataset is available in the UC Irvine Machine Learning Repository. This dataset contains seven different types of dry beans; there are 13,611 instances of data with 16 features each [30]. Example features include area, perimeter and eccentricity. We use 70% of the dataset for training and the remaining 30% as the test dataset.

CIFAR-10 dataset: Finally, we consider the CIFAR-10 dataset [31]. This dataset consists of 60,000 images across 10 classes. The classes represent varying objects (e.g., "cat" or "deer") and there are 6,000 images per class. Each image is represented as a length-3072 vector. We use 50,000 images as the training dataset and the remaining 10,000 as the test dataset. We only use CIFAR-10 for the PCA algorithm; images are not particularly appropriate for K-Means, SVM, and Random Forest.

6.2. Experimental Design

For each of our experiments, we follow the template from Figure 1. We first choose our required privacy guarantee, represented by an upper bound on the mutual information (MI) between the input and output to our algorithm, $\mathcal{M}$. In our experiments, we vary MI between $\frac{1}{128} = 2^{-7}$ and $4 = 2^2$. We then estimate the stability of $\mathcal{M}$, on the training data X_{train}. To do this, we repeatedly randomly sample $X_i \subset X_{train}$ where each X_i from $i = 1 \ldots m$ satisfies $|X_i| := 0.5|X_{train}|$. We denote m as the round complexity of the algorithm and we increment m until our estimator satisfies our precision requirements, as shown in Algorithm 1. We then compute the stability of $\mathcal{M}$ as a function of the variance of $\mathcal{M}(X_i)$ over all subsets $X_1 \ldots X_m$. We use this to compute the noise required to privatize $\mathcal{M}$, which we denote as $\Delta(\mathcal{M}, \mathsf{MI})$.

For all of our algorithms (Mean, K-Means, SVM, PCA, and Random Forest), we implement the noise estimation algorithm of Section 4 to determine additive noise.

We measure the *utility* of the baseline and both the isotropic and anisotropic privatized versions of $\mathcal{M}$. In particular, we first run $\mathcal{M}$ on the entire X_{train} and calculate the accuracy of $\mathcal{M}(X_{train})$ on the *test* dataset X_{test}. This provides our accuracy metric for the baseline non-private algorithm; we denote this as the "baseline accuracy" of $\mathcal{M}$. Then, we construct two privatized algorithms by, respectively, adding the required anisotropic noise and isotropic Gaussian noise to each element of the trained vector $\mathcal{M}(X_j)$. The required noise is sampled from a Gaussian with zero mean and variance determined by Algorithm 1. This creates two privatized trained vectors, $\mathcal{M}_P(X_j)$ (anisotropic noise) and $\mathcal{M}_Q(X_j)$ (isotropic noise); we then compute the accuracy of $\mathcal{M}_P(X_j)$ and $\mathcal{M}_Q(X_j)$ on X_{test}; these are, respectively, the anisotropic and isotropic "privatized accuracy" of $\mathcal{M}$ for a single *trial*. Our results are averaged over 1000 trials for each setting.

We now provide results across varying datasets and algorithms. All code used is provided at https://github.com/mayuri95/pac_algs.

6.3. Warmup: Estimating the Mean

Mean estimation is simple enough that we can provide a quantitative, head-to-head comparison between PAC and DP, since DP does not require significant changes beyond l_2-norm clipping for bounded sensitivity. For our experiments, we do a search to find the optimal clipping threshold to minimize the overall distance between the privatized mean estimate and the true mean. For a given clipping threshold C and dataset size n, the global sensitivity for the mean estimate is C/n. The required noise to provide an ϵ-DP guarantee is then a Laplacian with scale $C/(n\epsilon)$ [32].

In contrast to DP, which can use the entire dataset, PAC requires an input distribution, which we derive from subsampling. PAC does not require clipping. We chose the subsampling rate $r = 0.5$ to minimize prior success rate for an individual membership inference attack. (Any $0 < r < 1$ can be used, with different privacy-utility tradeoffs; we do not explore those here.)

To make a meaningful comparison, we compare DP and PAC *fixing* the posterior success rate. A given posterior success rate for membership inference can be translated to a particular ϵ-DP guarantee using Equation (2). Similarly, mutual information bounds and posterior success rates are related by Equation (3) (also see Table 1). We can therefore compare the expected l_2 distance between DP and PAC estimated means and the true means for the same success rates in Table 2.

As discussed in Section 4, we compare DP with PAC for both individual privacy and arbitrary inference tasks (denoted as global privacy). For individual privacy, we observe that most of the PAC error is due to subsampling and there is little noise addition required for all values of MI, as shown in the small gap between the l_2 distance after subsampling and after subsampling *and* anisotropic privatization in Table 2. The overall error between DP and PAC Individual Privacy is comparable for this task. The difference between isotropic and anisotropic noise with PAC Individual Privacy is negligible.

We observe that for PAC Global Privacy, the error for small values of MI is larger than its corresponding DP error. Isotropic noise with PAC Global Privacy increases the l_2-norm distance after privatization by up to $1.35\times$. It is important to note that DP and PAC privacy guarantees are not equivalent in the semantic sense; in particular, PAC Global Privacy provides a posterior bound for *arbitrary* inference tasks, e.g., the generalized membership attack discussed in Appendix C.

6.4. K-Means

As previously discussed, we expect K-Means to be easily compatible with the PAC Privacy framework; results are provided in Figure 3.

We observe that the baseline accuracy on our test set is above 90% for the Iris and Rice datasets. Even on small datasets like Iris, we observe a negligible gap between the baseline and privatized algorithms for $\mathsf{MI} \geq 2^{-4}$. On the Rice dataset, the centroids are quite stable and thus, we see no meaningful difference between privatized accuracy and the non-private baseline.

In the Dry Bean dataset, the underlying baseline accuracy is quite low ($\approx 70\%$). The anistropic privatized accuracy nearly matches it at $\mathsf{MI} \geq 1$, but the isotropic accuracy remains significantly worse. As discussed in Section 5, we posit that the low baseline accuracy for Dry Bean is due to the class imbalance. Thus, we explore using oversampling techniques, as seen in Figure 4.

We observe negligible differences due to oversampling for the Iris and Rice datasets, since the original datasets do not have a class imbalance. However, in the Dry Bean dataset, we first observe a significant increase ($\approx 15\%$) in the baseline accuracy. Moreover, we observe that the oversampling *also* increases the stability of the algorithm. In particular, we observe negligible utility differences between the anisotropic

Dataset	**Metric**	$\epsilon = 1.64$; $1-\delta = 0.84$; MI $= 1/4$	$\epsilon = 0.73$; $1-\delta = 0.67$; MI $= 1/16$	$\epsilon = 0.36$; $1-\delta = 0.59$; MI $= 1/64$
Iris	Differential Privacy	(0.004, 0.024)	(0.012, 0.05)	(0.027, 0.098)
Iris	PAC Individual Privacy	(0.045, 0.045)	(0.045, 0.045)	(0.044, 0.048)
Iris	PAC Global Privacy	(0.044, 0.045)	(0.043, 0.059)	(0.045, 0.16)
Rice	Differential Privacy	(1.5×10^{-4}, 0.0017)	(5.0×10^{-4}, 0.0037)	(0.0017, 0.007)
Rice	PAC Individual Privacy	(0.0076, 0.0076)	(0.0079, 0.0079)	(0.0078, 0.0078)
Rice	PAC Global Privacy	(0.0078, 0.0078)	(0.0078, 0.0081)	(0.0079, 0.011)
Dry Bean	Differential Privacy	(2.9×10^{-4}, 0.001)	(2.9×10^{-4}, 0.002)	(9.3×10^{-4}, 0.004)
Dry Bean	PAC Individual Privacy	(0.0054, 0.0054)	(0.0054, 0.0054)	(0.0054, 0.0054)
Dry Bean	PAC Global Privacy	(0.0053, 0.0054)	(0.0055, 0.0056)	(0.0054, 0.0069)

TABLE 2. QUANTITATIVE COMPARISON OF DP VS. PAC PRIVACY FOR PRIVATE MEAN ESTIMATION. WE PROVIDE PAC PRIVACY ESTIMATES FOR BOTH GLOBAL AND INDIVIDUAL GUARANTEES AS DISCUSSED IN SECTION 4. DP USES $\bar{\delta} = 0$ FOR VARYING POSTERIOR SUCCESS PROBABILITIES, $(1-\delta)$. DP CELLS PROVIDE l_2 DISTANCE AFTER CLIPPING AND AFTER CLIPPING AND PRIVATIZATION; PAC CELLS PROVIDE l_2 DISTANCE AFTER SUBSAMPLING AND AFTER SUBSAMPLING AND ANISTROPIC PRIVATIZATION USING ALGORITHM 1. ALL RESULTS ARE AVERAGED OVER 1000 TRIALS.

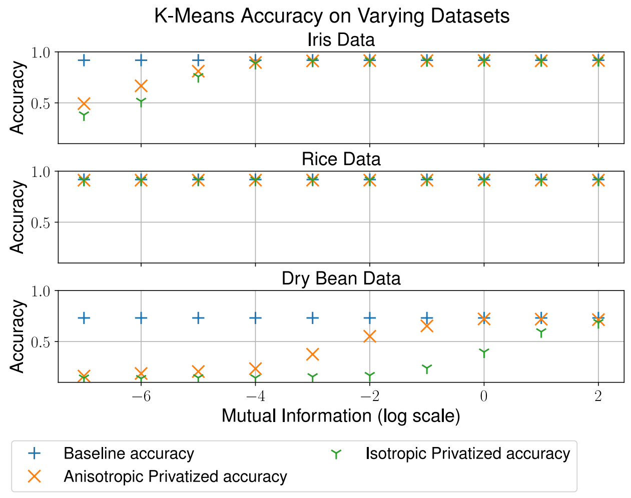

Figure 3. We plot the accuracy of the K-Means algorithm without privatization in blue. We then show the anisotropic privatization in orange and isotropic privatization in green. The accuracy is measured across mutual information varying from 2^{-7} to 2^{2}. As expected, we observe better utility using anisotropic noise across all datasets and mutual information values. The Rice dataset is the easiest to privatize, while the Dry Bean dataset is the hardest.

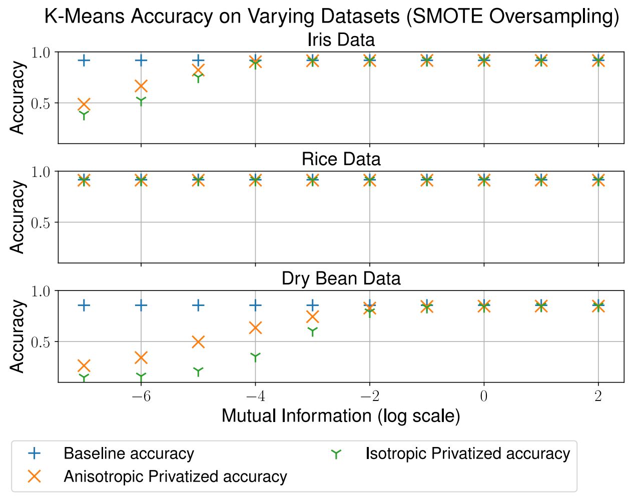

Figure 4. We observe a significant improvement in the Dry Bean baseline accuracy from $\approx 70\%$ to $\approx 85\%$. The algorithm also becomes easier to privatize.

privatized accuracy and the baseline for MI $\geq 2^{-2}$, with a privatized accuracy $> 80\%$. We consider this a crucial win-win situation, where stability techniques like oversampling can improve both privacy *and* utility.

6.5. Support Vector Machines (SVM)

Our initial results on SVM, without any additional regularization ($C = 1.0$), are summarized in Figure 5 [5].

We first consider the Iris dataset. For sufficiently large MI (> 1), the utility loss due to privatization is minimal. However, when we tighten the mutual information guarantee, the magnitude of required noise increases until the utility impact is quite severe – at an MI guarantee of 2^{-4}, the privatized algorithm has a utility $\approx 41\%$. The Dry Bean dataset is a more stark example of this phenomenon — the noise added is so large that the privatized utility does not achieve $> 50\%$ until MI > 2. While the gap is not large, the anisotropic utility is consistently better than the isotropic counterpart.

We observe a similar trend on the Rice dataset for low MI values. However, on the Rice dataset, the algorithm achieves stability at MI $\approx 2^{-1}$, where both baseline and privatized algorithms achieve accuracy of $> 90\%$.

There are many possible reasons for the difference in performance between the baseline and privatized algorithms. We consider two main cases, corresponding to superficial and intrinsic instability, respectively. We observe that many sources of superficial instability can be resolved by *regularization*. That is, regularization provides a technique to *order*

5. The Dry Bean SVM experiments are run to a precision of $\tau = 10^{-5}$ for computational efficiency.

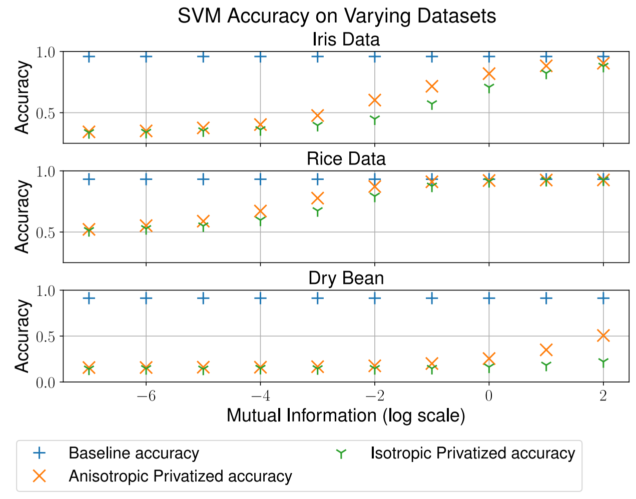

Figure 5. Without additional regularization, we observe that it is difficult to privatize the Iris dataset (significant utility loss for MI $\leq 2^0$) and nearly impossible to privatize the Dry Bean dataset. The Rice dataset is easier to privatize and shows minimal utility losses for MI $> 2^{-2}$.

multiple solutions which provide similar utility, by simply choosing the simplest one (lowest norm). However, increasing regularization too much can interfere with the baseline results – intuitively, we can prioritize simple solutions over those with higher utility. Thus, this cannot successfully resolve issues where the underlying algorithm is unstable due to *inherent* instability, without a significant utility impact.

We experiment with the stability of the SVM algorithm by increasing the regularization. We vary the regularization parameter C from Equation (4) and our results are in Figure 6.

We first consider the Iris dataset; the results here are plotted for $C = 0.05$. We first observe that the *non-private* version of the algorithm shows a decrease in accuracy, across all possible mutual information bounds – that is, the baseline accuracy drops from $> 90\%$ to $\approx 75\%$. This shows that the regularization is strong enough to overpower the loss in utility; that is, for the chosen value of C, the optimization problem prefers a stable low-norm solution more than the $\approx 15\%$ increase in accuracy. However, we observe that the *privatized* version of the algorithm shows a significant increase in accuracy, with minimal utility losses for MI as low as 2^{-4}. We observe similar results with the Dry Bean dataset, although stronger regularization is required. That is, we return to the privacy versus utility discussion, touched upon in Section 3. In the non-private setting, $C = 0.05$ for Iris represents regularization that is too strong since the error on the test dataset is *higher* than with $C = 1.0$.

We observe our best results on the Rice dataset. In this setting, we choose a regularization parameter of $C = 0.05$ – the baseline accuracy *increases* by 0.1% due to the regularization. Additionally, the stability of our algorithm is improved significantly and we achieve negligible utility loss in privatization. This suggests that even complex algorithms with sufficiently large and representative datasets can achieve stability with appropriate regularization techniques.

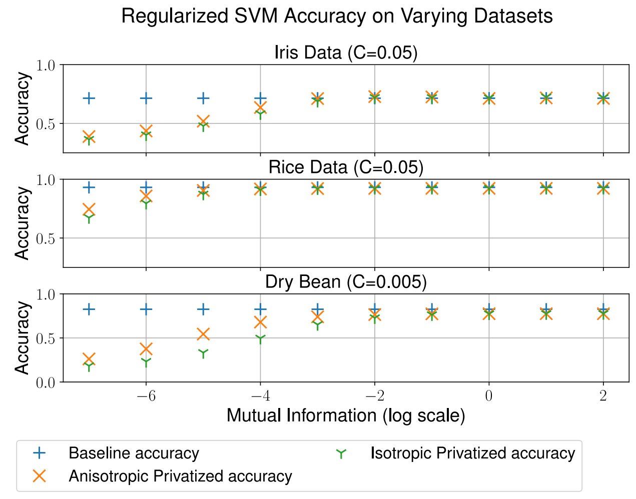

Figure 6. Regularization for SVM affects our datasets in significantly different ways. In the Iris dataset, the baseline algorithm suffers a significant utility loss due to the strong regularization (from $> 90\%$ to $\approx 75\%$). However, the gap between the privatized and baseline accuracy decreases, suggesting that strong regularization is optimal for tight mutual information guarantees. We observe similar results on the Dry Bean data set which achieves stability with small utility losses for MI $\geq 2^{-3}$ at $C = 0.005$. In the Rice dataset, regularization removes instability from the algorithm without any loss in baseline utility.

6.6. Principal Component Analysis

We then consider the principal component analysis algorithm for dimensionality reduction. For this algorithm, we evaluate its performance by measuring the distance between the reconstructed test matrix X' and the original test matrix X_{test} as defined in Equation (5). We first explore the underlying rank of the datasets.[6] These results are summarized in Figure 7.

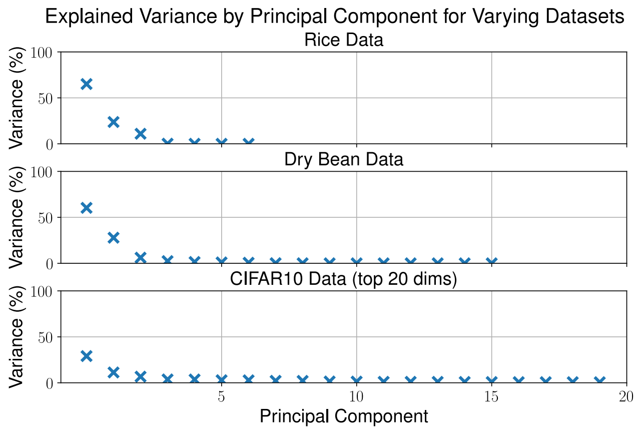

Figure 7. We measure the percentage of explained variance by the top principal components for each dataset. The Rice dataset has a total of 7 features, while the Dry Bean dataset has 16 features. The CIFAR-10 dataset has 3072 features — we only plot the explained variance for the top 20 dimensions, which account for 70% of the total variance.

6. We do not use the Iris dataset due to its small dimension.

In general, we expect that the PCA algorithm will be inherently unstable at tight MI guarantees when there are several principal components with similar "importance". That is, if there are two principal components that both explain $\approx 1\%$ of the underlying variance, we expect that either could be returned arbitrarily, even for extremely similar datasets. We first investigate $d = 1$ in Figure 8.

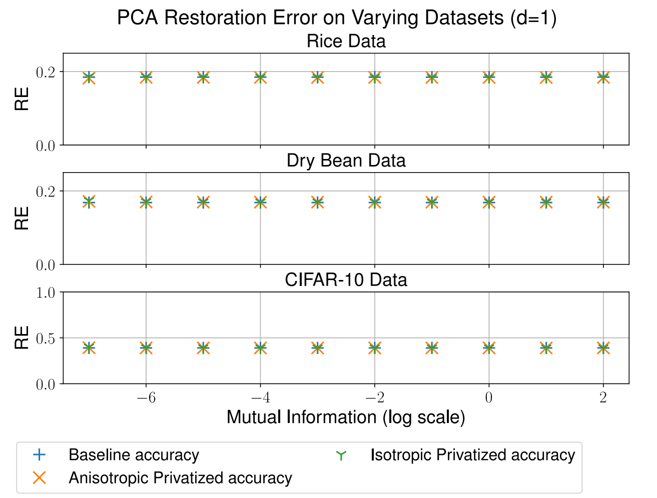

Figure 8. We observe that our algorithm is stable on all the datasets for all MI values. We observe a relatively low RE ($< 20\%$) for the Rice and Dry Bean datasets due to the significance of the top eigenvector. Meanwhile, CIFAR-10 has a larger RE ($\approx 40\%$), but similar stability guarantees.

As observed in Figure 7, most of the variance in the Rice and Dry Bean dataset are explained in the first component. Thus, we observe in Figure 8 that the restoration error is $< 20\%$ for all mutual information values and we can largely recover the original matrix. In contrast, less than 50% of the variance of the CIFAR-10 dataset is explained by the first component. Thus, this shows a much higher restoration error $\approx 40\%$. Across all the datasets, we observe negligible changes in RE for all mutual information values, indicating that the algorithm is stable in identifying the top eigenvector.

We then consider the same algorithm with higher dimensions, as seen in Figure 9. Here, we observe that all the datasets show a significant decrease in restoration error for the non-private baseline; this is expected since we are increasing the number of dimensions kept and thus, capturing more of the variance in the original matrix.

For the Rice dataset, we observe that the anisotropic noise achieves privatized RE $\leq 5\%$ at MI $\geq 2^{-2}$, which is a significant improvement over $d = 1$. The Dry Bean dataset shows similar results, achieving RE $\leq 5\%$ at MI $= 2^{-3}$. This suggests that we can privatize PCA on such large datasets with large enough dimensions to capture most of the variance. In both of these cases, we observe the benefit of anisotropic noise — the corresponding isotropic algorithm often has much worse results in higher dimensions.

The CIFAR-10 dataset, in contrast, can only be privatized for $d = 3$. In particular, the eigenvectors for $d > 3$ have similar "importance" and the stability of the algorithm drops significantly (the l_1 norm of the noise added from $d = 3$ to

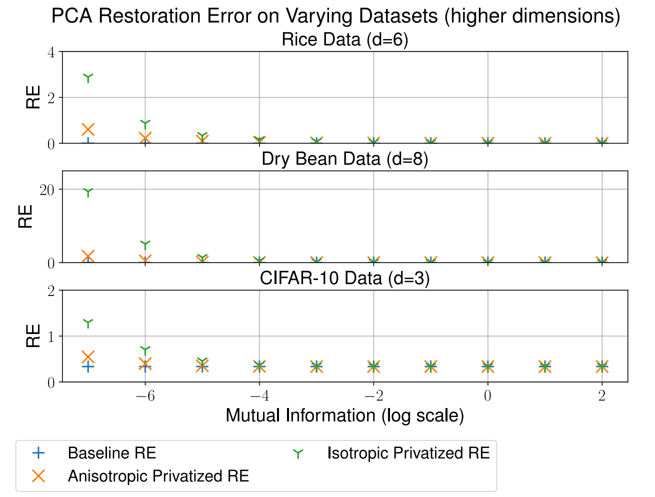

Figure 9. We run PCA with varying numbers of components (d in the plots) for the different datasets. With large d, the baseline restoration error drops to near zero for Rice and Dry Bean. This indicates that we can privatize these algorithms with negligible impact. In contrast, we choose $d = 3$ for CIFAR-10 in order to provide meaningful privatized utility; however, the baseline RE remains high due to the relatively low dimension. Increasing d further significantly reduces the stability, making the privatized algorithm unusable.

$d = 4$ increases by $100\times$). The anisotropic algorithm's RE for $d = 3$ varies from $\approx 55\%$ for MI $= 2^{-7}$ to $\approx 33\%$ for MI $= 2^2$, which is a small improvement over $d = 1$.

6.7. Random Forest

Finally, we consider the random forest algorithm. As discussed in Section 5, the random forest algorithm is known to be unstable and is quite difficult to adapt to our framework.

We first test the naïve algorithm with no additional regularization on the Iris and Rice datasets.[7]

Our results are summarized in Figure 10. We use a single tree for the Iris dataset with depth 3. We use 3 trees for the Rice dataset, with depth 3. In each iteration of the algorithm, the chosen features for each tree are randomly sampled and the variance across the threshold values is measured.

As expected, the privatized version of random forest without additional regularization shows significant instability for our datasets. Further investigation shows that there are several possible causes for the instability within a tree:

- When there are a small number of samples that are in a path, the optimal "threshold" value to split on is unstable. We resolve this by providing regularization penalties.
- The exact threshold value to split on can be noisy due to the exact set of points that are observed. To improve stability, we consider a fixed set of threshold values with finite precision.
- The threshold values are sometimes unstable due to the non-uniform spread of the feature values. To address this, we calculate a weighted average of the entropy.

7. For this algorithm, we do not use the Dry Bean dataset for computational efficiency. See Appendix B for results on this dataset.

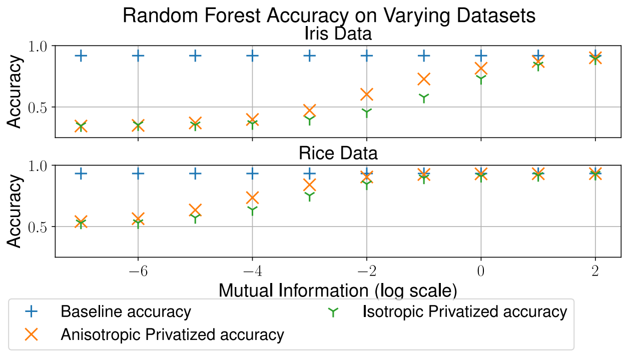

Figure 10. The naïve random forest algorithm shows significant instability on the Iris and Rice datasets. The Iris dataset achieves over 90% accuracy in the non-private case, but shows a dramatic loss in utility (down to $< 50\%$ for MI $< 2^{-2}$) after privatization. The Rice dataset shows better results, with $\approx 93\%$ accuracy in the baseline and $\approx 73\%$ accuracy after anisotropic privatization at MI $= 2^{-4}$.

We choose these three techniques in order to address the issue that the trees cannot be pruned while maintaining a canonical ordering that can be compared across iterations. We now experiment with adding regularization of the form (p, w_1, w_2) (cf. Section 5) for the Iris and Rice datasets.

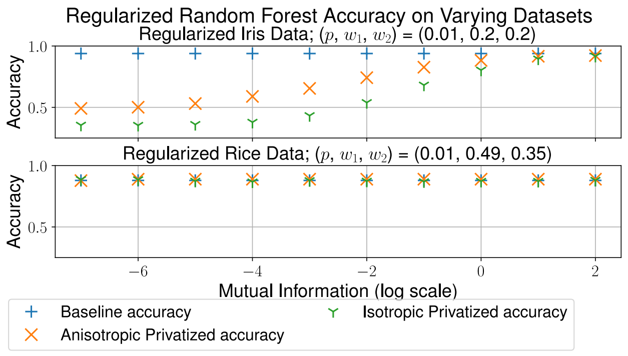

Figure 11. On the Iris dataset, we achieve over 70% privatized utility for MI $> 2^{-3}$. In the Rice dataset there is a much smaller utility loss in the baseline due to regularization (baseline accuracy is $\approx 87\%$). However, the privatized accuracy *increases* to achieve similar utility due to the improvements in stability.

We first consider the Iris dataset. As seen in Figure 11, we add significant regularization; however, we note that our baseline non-private accuracy actually *increases* by $\approx 2\%$ after regularization. Further, regularization improves the privatized algorithm's utility to $\geq 70\%$ for MI $> 2^{-3}$. We observe that the anisotropic noise provides a significant utility benefit over the isotropic setting.

We next analyze the Rice dataset. For this dataset, we suffer a small utility loss (93% to 87% baseline accuracy) due to our increased regularization. However, the improvement in stability is significant; there is thus a negligible loss in utility between the privatized and baseline algorithm after the increased regularization for all MI values. This again shows that unstable algorithms can be privatized with little utility cost when the dataset is sufficiently large and stable.

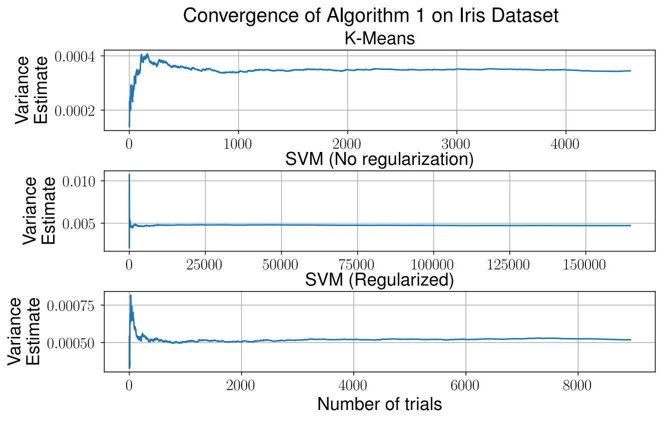

Figure 12. We choose our trial complexity m for Algorithm 1 by measuring the change in our variance estimate in each direction A_k. When all of the directions are stabilized within 10^{-6}, we return the current variance estimate.

7. Convergence of Algorithm 1

In this section, we discuss our empirical convergence guarantee. We observe that Theorem 1 provides a mutual information bound when the variances are estimated exactly. For practical guarantees, we choose trial complexity large enough such that each element of the variance estimate converges with very high precision ($\tau = 10^{-6}$). In particular, we run our noise estimation algorithm and estimate our variance vector after every 10 instances. The algorithm is considered to have converged when none of the estimates in our output vector have changed by more than τ. We choose τ sufficiently small such that the impact of adding noise in the order of τ is negligible. Results are shown in Figure 12.

Figure 12 provides the change in the first element of our variance vector for varying algorithms on the Iris dataset. We observe that the trial complexity varies across algorithms and datasets. Stable algorithms (e.g., SVM with strong regularization) converge more than $10\times$ faster than SVM without regularization.

Convergence on the complete covariance matrix would require an order of magnitude more trials. In Appendix B, we describe an alternate approach to true variance computation that can be more efficient than a convergence-based approach.

8. Empirical Privacy Estimation

Although PAC Privacy allows us to provably bound the posterior advantage for *any* attack, in this section, we focus on membership inference attacks for concreteness and validation. The objective of membership inference attacks (MIA) is defined in [6] as follows: given a machine learning model and a single datapoint x, the goal is to determine whether x was used to train the model.

For our purposes, we define our machine learning model as the trained output vector Y_i, representing some statistic about our input data $X_i \subset X_{train}$. In K-Means, this vector

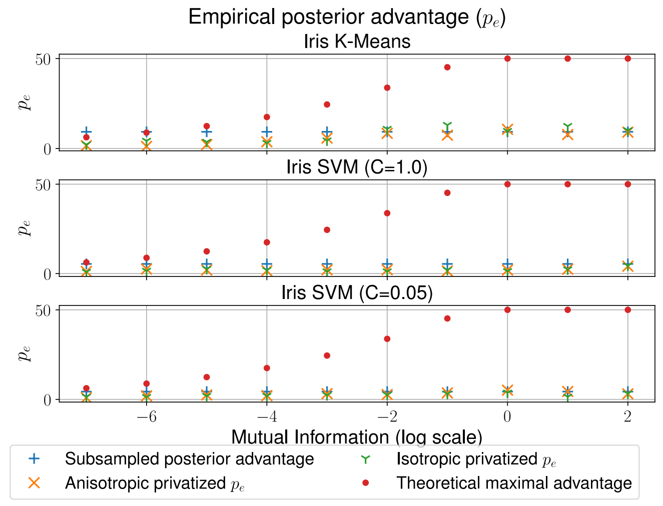

Figure 13. Empirical posterior advantage from LIRA over 1,000 trials. The empirical posterior advantage of the subsampled algorithms are all at most 11%. For the privatized algorithms, the empirical advantages are **always** below the theoretical posterior advantages of Table 1. The K-Means and naïve SVM algorithm show the largest average reduction in posterior advantage due to privatization of $\approx 3\%$ over all values of MI.

corresponds to the list of centroids; for SVM, the vector represents the list of hyperplanes separating the classes. We vary the mutual information bound and focus on the Iris dataset. We observe that our model is trained on a subset X_j where $|X_j| = 0.5(|X_{train}|)$ in all of our experiments. This indicates that for any particular datapoint x, the prior $\Pr[x \in X_j] = 0.5$. Table 1 gives a theoretical maximal posterior advantage, which can be compared to the empirical advantage observed.

We consider the Likelihood-Ratio Attack (LIRA) as described in [7] and adapt it to the K-Means and SVM algorithms; details are provided in Appendix D. Our results on the Iris dataset are summarized in Figure 13. PAC Privacy is necessarily conservative; in Figure 13, the empirical posterior advantages (denoted as p_e) for the privatized algorithms are significantly lower than the upper bounds given by Table 1 across all mutual information bounds.

We observe a decrease in the privatized posterior advantage across all the algorithms. In the K-Means algorithm, we observe the most significant change, from $\approx 9.61\%$ to 1.15% at MI $= 1/64$. In the SVM algorithms, the baseline advantage of the non-private algorithm is significantly lower at $\approx 5.4\%$ without regularization and $\approx 4.3\%$ at $C = 0.05$. For $C = 1.0$, we observe an average decrease of $\approx 3\%$ in p_e; for $C = 0.05$, the difference is $\approx 1.4\%$.

9. Related Work

Recent work in PAC Privacy addressed the formalization of privacy guarantees provided by heuristic encoding algorithms, and provided efficient estimation strategies for these specialized encoding techniques [33].

Quantitative comparisons to DP for generic mean estimation were provided in Section 6.3. For more complex algorithms, small ϵ-DP guarantees are harder to provide and often involve significant changes to algorithm implementation. Even with white-box changes, the resulting algorithm often requires a large dataset with small data dimension to provide meaningful utility guarantees. In contrast, PAC provides instance-specific guarantees with reasonable utility loss, even when subsampling small datasets of $\approx$ 1,000 datapoints with large output dimension. Given the substantial algorithmic differences between DP white-boxed algorithms and the PAC black-box approach, and the semantic difference in privacy guarantees, we restrict ourselves to qualitative comparisons between our PAC-privatized algorithms and state-of-the-art DP algorithms for the various problems.

We first consider K-Means. Early work developed DPLloyd [34], a DP version of Lloyd's algorithm for K-Means clustering. Intuitively, DPLloyd adds Laplacian noise each time the approximate centroids are computed. We observe that even for a simple algorithm, providing DP required significant changes to the algorithmic structure, e.g., fixing the number of iterations for convergence. Further, we observe that K-Means is *known* to be sensitive to the initial centroids chosen; thus, DPLloyd requires a new private initialization procedure as well.

We then consider SVM classifiers; there has been a long history of developing SVMs with DP guarantees [35], [36]. In general, these techniques use the SVM algorithm to compute the optimal weight vector and add appropriate noise to provide DP guarantees. However, as observed by [37], large training sets led to large weight vectors with increased noise. Moreover, often there were strong restrictions on the objective function (e.g., convexity) to enable tight bounds on the noise. [37] suggests a novel method in order to solve the dual problem of SVM, which approaches the non-private SVM accuracy for sufficiently large training sets. However, note that this is still a white-box mechanism to achieve privacy, i.e., Laplacian noise was added in *each* iteration and in each iteration, an inner loop is required to choose the pairs of dual variables to update.

Random forests with differential privacy have been explored less extensively. [38] suggests that differentially-private random forests can be constructed by allocating a privacy budget across trees, and then across levels of each tree. Each tree is "complete" when either all the features are used, the remaining samples all belong to the same class or when a maximum height is reached. [39] expands this work and constructs differentially-private median forests, which also improve the stability of the data structure. However, they still observe significant utility losses for sufficiently small ϵ.

Finally, we consider DP for identifying principal components. [40] constructs a near-optimal technique for identifying principal components while providing DP. Their technique requires datasets with a large number of datapoints, but the noise scales with the original dimension, making it impractical for datasets with large dimension, even if the true rank is constant. We further observe that their resulting utility guarantees are upon a secondary classification task, rather than the restoration of the original matrix task that we evaluate on. In practice, we expect the latter to be a stronger guarantee since a perfectly restored matrix would provide the best utility guarantee on *any* secondary task.

10. Conclusions

We have shown how PAC Privacy can be applied to privatize black-box algorithms by giving a template that can be applied to virtually any algorithm. Using Algorithm 1 to add anisotropic noise is critical to improving privacy-utility tradeoffs.

An exciting aspect of PAC Privacy that is demonstrated most clearly by our K-Means results is the potential win-win situation in the algorithm tradeoff space. Stability with respect to input changes is a desirable feature of algorithms, because stable algorithms generalize better to new inputs and have better worst cases. Concomitantly, stable algorithms require less additive noise on their outputs for privatization.

One aspect that is worth exploring in the future is the privacy-utility tradeoff at different subsampling rates. Additional future work includes using the compositional properties of mutual information [41] to tackle unstable algorithms such as Stochastic Gradient Descent (SGD). Algorithms can be broken down into phases, and noise is added at the end of each phase. It is conservatively assumed that the noisy outputs are exposed, similar to DP-SGD [4], although in PAC Privacy the phases can be hundreds of iterations and each phase can be treated as a black box.

11. Acknowledgements

We thank the anonymous reviewers for their detailed and constructive feedback. This work was supported in part by grants from Cisco Systems and Capital One. Mayuri Sridhar was supported by the NDSEG fellowship from the DoD and a MathWorks fellowship.

References

[1] C. Dwork, “Differential privacy,” in *International colloquium on automata, languages, and programming*. Springer, 2006, pp. 1–12.

[2] X. Xiao and Y. Tao, “Output perturbation with query relaxation,” *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 857–869, 2008.

[3] I. Issa, A. B. Wagner, and S. Kamath, “An operational approach to information leakage,” *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1625–1657, 2019.

[4] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep learning with differential privacy,” in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[5] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson, “Scalable private learning with pate,” *arXiv preprint arXiv:1802.08908*, 2018.

[6] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership inference attacks against machine learning models,” in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 3–18.

[7] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer, “Membership inference attacks from first principles,” in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1897–1914.

[8] H. Hu, Z. Salcic, L. Sun, G. Dobbie, P. S. Yu, and X. Zhang, “Membership inference attacks on machine learning: A survey,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–37, 2022.

[9] M. Nasr, R. Shokri, and A. Houmansadr, “Machine learning with membership privacy using adversarial regularization,” in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 634–646.

[10] Y. Kaya and T. Dumitras, “When does data augmentation help with membership inference attacks?” in *International conference on machine learning*. PMLR, 2021, pp. 5345–5355.

[11] Y. Yin, K. Chen, L. Shou, and G. Chen, “Defending privacy against more knowledgeable membership inference attackers,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2026–2036.

[12] Y. Wang, C. Wang, Z. Wang, S. Zhou, H. Liu, J. Bi, C. Ding, and S. Rajasekaran, “Against membership inference attack: Pruning is all you need,” in *International Joint Conference on Artificial Intelligence*, 2021.

[13] H. Xiao and S. Devadas, “Pac privacy: Automatic privacy measurement and control of data processing,” in *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, 2023, p. 611–644. [Online]. Available: https://doi.org/10.1007/978-3-031-38545-2_20

[14] B. Balle, G. Cherubin, and J. Hayes, “Reconstructing training data with informed adversaries,” in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1138–1156.

[15] J. Hayes, S. Mahloujifar, and B. Balle, “Bounding training data reconstruction in dp-sgd,” *arXiv preprint arXiv:2302.07225*, 2023.

[16] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa, “Strong and efficient cache {Side-Channel} protection using hardware transactional memory,” in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 217–233.

[17] P. Kairouz, S. Oh, and P. Viswanath, “The composition theorem for differential privacy,” in *International conference on machine learning*. PMLR, 2015, pp. 1376–1385.

[18] T. Humphries, S. Oya, L. Tulloch, M. Rafuse, I. Goldberg, U. Hengartner, and F. Kerschbaum, “Investigating membership inference attacks under data dependencies,” in *2023 IEEE 36th Computer Security Foundations Symposium (CSF)*, 2023, pp. 473–488.

[19] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://dx.doi.org/10.1023/A%3A1010933404324

[20] S. Lloyd, “Least squares quantization in pcm,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[21] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” vol. 8, 01 2007, pp. 1027–1035.

[22] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” *J. Artif. Int. Res.*, vol. 16, no. 1, p. 321–357, jun 2002.

[23] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “Liblinear: A library for large linear classification,” *Journal of Machine Learning Research*, vol. 9, no. 61, pp. 1871–1874, 2008. [Online]. Available: http://jmlr.org/papers/v9/fan08a.html

[24] C.-C. Chang and C.-J. Lin, “Libsvm: A library for support vector machines,” *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, may 2011. [Online]. Available: https://doi.org/10.1145/1961189.1961199

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[26] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. [Online]. Available: https://doi.org/10.1080/14786440109462720

[27] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[28] R. A. Fisher, "Iris," UCI Machine Learning Repository, 1988, DOI: https://doi.org/10.24432/C56C76.

[29] I. Cınar and M. Koklu, "Classification of rice varieties using artificial intelligence methods," *International Journal of Intelligent Systems and Applications in Engineering*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:208105752

[30] M. Koklu and I. A. Özkan, "Multiclass classification of dry beans using computer vision and machine learning techniques," *Comput. Electron. Agric.*, vol. 174, p. 105507, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219762890

[31] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[32] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3–4, p. 211–407, aug 2014. [Online]. Available: https://doi.org/10.1561/0400000042

[33] H. Xiao, G. E. Suh, and S. Devadas, "Formal Privacy Proof of Heuristic Encoding: The Possibility and Impossibility of Learnable Obfuscation," in *Computer and Communications Security Conference*, October 2024.

[34] D. Su, J. Cao, N. Li, E. Bertino, and H. Jin, "Differentially private k-means clustering," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 26–37. [Online]. Available: https://doi.org/10.1145/2857705.2857708

[35] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *J. Mach. Learn. Res.*, vol. 12, no. null, p. 1069–1109, jul 2011.

[36] B. I. P. Rubinstein, P. L. Bartlett, L. Huang, and N. Taft, "Learning in a large function space: Privacy-preserving mechanisms for svm learning," *Journal of Privacy and Confidentiality*, vol. 4, no. 1, Jul. 2012. [Online]. Available: https://journalprivacyconfidentiality.org/index.php/jpc/article/view/612

[37] Y. Zhang, Z. Hao, and S. Wang, "A differential privacy support vector machine classifier based on dual variable perturbation," *IEEE Access*, vol. 7, pp. 98 238–98 251, 2019.

[38] A. Patil and S. Singh, "Differential private random forest," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2014, pp. 2623–2630.

[39] S. Consul and S. A. Williamson, "Differentially private random forests for regression and classification," 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235365701

[40] K. Chaudhuri, A. Sarwate, and K. Sinha, "Near-optimal differentially private principal components," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/f770b62bc8f42a0b66751fe636fc6eb0-Paper.pdf

[41] H. Xiao, "Automated and Provable Privatization for Black-Box Processing," Ph.D. dissertation, Massachusetts Institute of Technology, August 2024.

[42] J. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Adv. Large Margin Classif.*, vol. 10, 06 2000.

Appendix A. Individual Privacy Guarantees

We can provide tighter bounds on the noise required to privatize algorithms when considering specific inference tasks.

In particular, for a d-dimensional mean estimation mechanism $\mathcal{M}$, we may decompose $\mathcal{M}$ as $\mathcal{M}_1, \cdots, \mathcal{M}_d$, where $\mathcal{M}_i$ is the i-th coordinate average of input X. We may follow the composition results (Theorem 7) of [13] to upper bound the mutual information of $\mathsf{MI}\big(X; \mathcal{M}(X)\big)$ by the sum of the KL divergence bound of each $\mathcal{M}_i(X)$.

In particular, for individual privacy where the adversary aims to infer whether a datapoint x^* is selected in the input set or not, rather than calculating the empirical variance over all possible subsets, we can simply compute the average expected distance between sets which contain the point x^* and sets which do not. Formally, as in [13], to calculate the privacy guarantee for an individual point x^*, we compute

$$\begin{aligned}
&\mathsf{MI}(x^*, \mathcal{M}(X)[i] + \mathcal{B}[i]) \\
&\le \mathbb{E}_{X\sim\bar{X}} \mathcal{D}_{KL}(\mathcal{M}(X)[i] + \mathcal{B}[i] \| \mathcal{M}(\bar{X}[i]) + \mathcal{B}[i]) \\
&\le \frac{\mathbb{E}_{X\sim\bar{X}}\left[\, \left\|\mathcal{M}(X)[i] - \mathcal{M}(\bar{X})[i]\right\|^2 \,\right]}{2e_i}.
\end{aligned}$$

where X and $\bar{X}$ are drawn from $\mathcal{D}$ and satisfy the constraint that $x^* \in X$ and $x^* \notin \bar{X}$. For the tightest guarantees on noise for a given X, we choose $\bar{X}$ and X as adjacent datasets.

Now, when we add independent Gaussian noises $\mathcal{B}_{[1:d]}$, in a form $\mathcal{N}(0, e_i)$, for $i = 1, 2, \cdots, d$, to each coordinate, to ensure that $\mathsf{MI}\big(X; \mathcal{M}(X) + \mathcal{B}\big)$ is upper bounded by β, it suffices to select $e_{[1:d]}$ such that

$$\sum_{i=1}^{d} \frac{\sigma_i}{2e_i} \le \beta,$$

where $\sigma_i := \mathbb{E}_{X\sim\bar{X}}\left[\, \left\|\mathcal{M}(X)[i] - \mathcal{M}(\bar{X})[i]\right\|^2 \,\right]$. This enables us to compute the anisotropic noise as in Theorem 1 where the optimal e_i is of the form

$$e_i = \frac{\sqrt{\sigma_i}\sum_{j=1}^{d}\sqrt{\sigma_j}}{2\beta}.$$

Finally, we note that we take the *maximum* e_i over all individual datapoints x^* to provide an individual membership guarantee for any chosen point.

Appendix B. An Alternate Approach to Computing Noise

The approach described in Section 4 uses empirical convergence guarantees to provide a tight variance estimate for $\mathcal{M}(X)$, rather than the high-probability guarantee of [13] which requires greater round complexity. In this section, we describe a different approach to constructing D, which provides computational efficiency *and* directly satisfies the requirements of Theorem 1 by providing a method to compute the true variance of $\mathcal{M}(X)$.

B.1. Constructing a Distribution D

First, consider sampling $X_1 \cdots X_m \subset X_{train}$, which are independent and identically distributed subsets of X_{train} with $|X_i| = |X_j|$ as before for a fixed choice of m. We then define $\mathcal{S}$ as the set $\{X_1 \cdots X_m\}$; throughout, we let D denote the uniform distribution over $\mathcal{S}$. Constructing $\mathcal{S}$ to be all possible $r|X_{train}|$ subsets of X_{train} matches the construction of D in Section 4. As before, our noisy release must be drawn from D. However, this choice of $\mathcal{S}$ is too large for the true variance to be exactly computed.

Varying choices of $\mathcal{S}$ affect the *prior* of our adversary. That is, if we choose $\mathcal{S}$ to be m uniformly random subsets of size $r|X_{train}|$, then a randomly chosen element x_i may occur in over 50% of these subsets; the number of subsets x_i occurs in would follow the binomial distribution of m tosses of a fair coin. This implies that certain membership attacks would have a prior over 50%; for large m, the prior will be overwhelmingly close to 50%.

However, the choice of $\mathcal{S}$ is free; that is, our privacy guarantees hold as long as the chosen X_j is drawn uniformly at random from the *same* set $\mathcal{S}$ that the X_i are subsampled from (and the prior is appropriately calculated). For instance, a simple instantiation of $\mathcal{S}$ for $r = 0.5$ is as follows:

1) Choose a random $0.5|X_{train}|$ subset of X_{train} that we denote as X_1.
2) Choose $X_2 := X_{train} \setminus X_1$.
3) $\mathcal{S} = \{X_1, X_2\}$, i.e., $m = 2$, and D corresponds to drawing randomly from $\mathcal{S}$.

For this choice of $\mathcal{S}$ and D, we observe that the prior for any individual datapoint is exactly 50% since X_1 and X_2 are constructed to be disjoint. Moreover, this set is of size 2 and computing the true variance is easy. The adversary, who is assumed to know D and $\mathcal{S}$, has a prior probability of $\frac{1}{m}$ of correctly guessing the chosen subset, and in this case it is $\frac{1}{2}$, which is the same as guessing whether an individual data element x_a is in the chosen subset or not. Knowing the chosen subset gives away the membership information of all data elements in X_{train}, regardless of the value of m. However, for the $m = 2$ case, knowing a *single* element that was chosen (or not) gives away the chosen subset and therefore the membership information of *all* data elements! The correlation between the memberships of the data elements decreases exponentially as m increases.

B.2. Noise Determination using True Variance

A slightly modified noise determination algorithm is presented in Algorithm 2, which takes as input the distribution D; in this model, this is represented exactly by the uniform distribution over the set $\mathcal{S}$. We denote $\mathcal{S}[k]$ as the k'th element of the set $\mathcal{S}$. All other terms are as before.

In practice, we construct $\mathcal{S}$ using $m = 1024$ possible subsets made up of 512 distinct pairs (X_i, X_{i+1}), $1 \leq i \leq 512$, where each pair is disjoint as described above. This has a prior for individual membership of exactly 50%, provides empirical stability in performance (cf. Section B.3), and has exponentially small correlation between data elements. This choice of $m = 1024$ allows us to efficiently privatize large datasets (e.g., Dry Bean with over 10,000 datapoints) even for complex algorithms like decision trees, which was intractable for the prior approach. For this value of $m = 1024$, the variance in performance across different $\mathcal{S}$ is small, and we are able to match the results in the main body of the paper for all algorithms and benchmarks. We provide further results for the Dry Bean dataset when privatizing the Random Forest algorithm in Figure 14.

Algorithm 2 Anisotropic Noise Determination of $\mathcal{M}$ Using True Variance

```
Input: The input distribution D represented by the set of subsets $\mathcal{S}$, deterministic mechanism $\mathcal{M} : \mathcal{X}^n \to \mathcal{Y}^d$, mutual information requirement $\beta$, $d \times d$ unitary projection matrix $A$.
1) $m := |\mathcal{S}|$, $\boldsymbol{G} := [0]_{m \times d}$
2) for $k = 1, 2, \ldots, m$:
    a) $X_k := \mathcal{S}[k]$.
    b) Compute $y_k = \mathcal{M}(X_k)$.
3) for each $k \in [1, \ldots m]$ and $i \in [1, \ldots d]$, set
       $G[k][i] := y_k \cdot A_i.$
4) Compute the variance vector $\boldsymbol{\sigma_m}$ where $\boldsymbol{\sigma_m}$ is a vector of length $d$ and $\boldsymbol{\sigma_m}[i]$ is the variance of $G_i$.
5) Calculate the required noise in each direction $i$ as
       $e_i := \frac{\sqrt{\boldsymbol{\sigma_m}[i]} \sum_{j=1}^{d} \sqrt{\boldsymbol{\sigma_m}[j]}}{2\beta}$ for $i \in [1, d]$.
6) Return a diagonal matrix $\Sigma_{\mathcal{B}}$, where $\Sigma_{\mathcal{B}}[i][i] = e_i$.
```

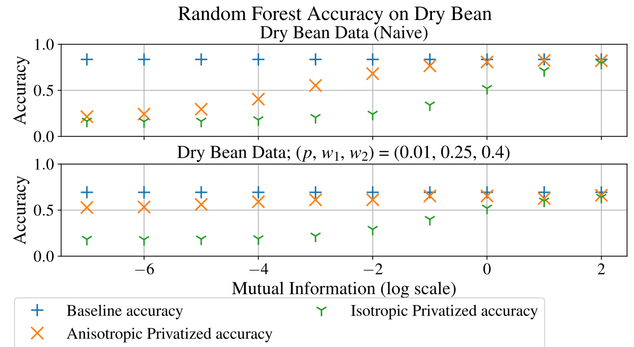

Figure 14. On the Dry Bean dataset, without regularization, we achieve 82% accuracy in the baseline, but significant loss in privatized utility for MI $< 2^{-1}$. With regularization, although the baseline utility is lowered to 70%, we find negligible losses in privatized utility for MI $\geq 2^{-3}$.

B.3. Tradeoffs in Choosing m

Smaller choices of m show a much larger variance in the noise estimates. For example, for $m = 2$, different random instantiations of the set $\mathcal{S}$ can produce very different Σ_B

from Algorithm 2. Increasing m leads to greater stability of Σ_B. Note that there are privacy guarantees even for $m = 2$; we can bound the posterior advantage of the adversary in guessing which subset was used. The prior and success rate of reconstruction attacks decrease as m increases.

To compute the variance of noise estimates, we fix $\mathsf{MI} = 0.5$ and consider the K-Means algorithm; we analyze the l_1 norm of diag(Σ_B), which represents the noise required to privatize this algorithm. At $m = 2$, over 100 instantiations, we observe that the norm of diag(Σ_B) varies from 0.01 to 0.27. In contrast, at $m = 128$, the norm of diag(Σ_B) varies from 0.10 to 0.20. This indicates that, at $m = 2$, the particular instantiation of S greatly affects the overall performance of the privatized algorithm. We observe stronger versions of this trend when we look at less stable algorithms. In particular, when we consider decision trees without any regularization, we observe that the norm of diag(Σ_B) varies from 0.06 to 1.16 at $m = 2$. At $m = 128$, the norm of diag(Σ_B) varies between 0.64 and 0.86.

Small m is computationally more efficient, but provides weaker privacy guarantees. It may require the addition of smaller or greater noise in different trials. We defer a fuller exploration of the tradeoffs between privacy, utility, computation cost and choice of m to future work.

Appendix C. A generalized membership attack

We observe that the prior guarantee of 0.5 is specific to membership attacks where we try to identify the membership of a *single, specific* datapoint x. Here, we consider a generalization of the membership attack where we aim to correctly identify the membership of *any* subset of points, where the subset has size at least k. Consider an attack that attempts to identify some fraction of bits of a secret key. In this setting, a secret key consisting of n bits is generated from a uniform random distribution and an adversarial task focuses on identifying at least k bits correctly. In this section, we primarily consider the *prior* for this attack. In practice, the adversary may be able to view some side channel, e.g., a timing, power, or cache side channel, and this additional exposure via some mechanism $\mathcal{M}$ will provide some posterior advantage that will correspond to the calculated MI. In this setting, the adversary *knows* the distribution that the secret key is drawn from, but it does not necessarily have a compact representation as a uniform distribution over a sufficiently small S.

Formally, the secret key is distributed uniformly at random over all 2^n possible bit vectors. That is, for any particular bit $i \in [1, n]$, we have a probability of 0.5 on guessing the bit correctly. Our inference task is to then construct a guess X', which is a bit vector of length n, such that at least k indices in X' are classified correctly. For any fixed size k, our prior probability becomes

$$p := 1 - \sum_{k'=0}^{k-1} \binom{n}{k'} \left(\frac{1}{2}\right)^n. \tag{6}$$

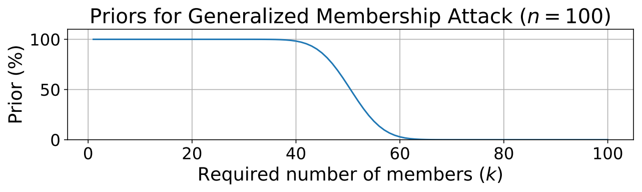

Figure 15. The prior from the generalized membership attack (Equation (6)) drops below 5% for $k \geq 59$. This indicates that loose MI guarantees can be meaningful for harder adversarial inference tasks.

In expectation, we guess $n/2$ bits correctly. However, as k increases beyond $n/2$, our prior success probability drops very quickly, as seen in Figure 15. When we consider $n = 100$ and $k = 63$, our prior probability drops below 1%, providing meaningful posterior bounds for large values of MI as discussed in Table 1. Thus, when we consider k non-negligibly larger than $n/2$, this allows for loose MI guarantees to still be meaningful. For example, a generalized membership attack with $n = 100$, $k = 70$ and a mutual information guarantee of 1, provides a $\leq 13.81\%$ posterior success guarantee, by solving Equation (3) given the appropriate prior.

Appendix D. Adapting LIRA to K-Means and SVM

The Likelihood-Ratio Attack by [7] exploits the idea that when a model is trained on a particular point x, its "confidence" on classifying x into a particular class or cluster will be higher than on a point it is not trained on. The original work by [7] exploits this by framing a membership attack as a hypothesis test. Let $\phi(\mathcal{M}(X_j), x)$ denote the confidence score of $\mathcal{M}(X_j)$ on x. In this work, they sample many subsets X_i and approximate the distribution of $\phi(\mathcal{M}(X_i), x)$, when $x \notin X_i$. For the attack, they then consider a varying set of thresholds t where the attack concludes $x \in X_j$ if and only if $\phi(\mathcal{M}(X_j), x) \geq t$. Each threshold t has corresponding true and false positive rates, and we simply consider the maximum accuracy over all thresholds t, which represents the maximum posterior advantage achievable by LIRA.

Unlike Carlini's work, we do not directly produce confidence values from our algorithms. We instead translate our output vectors to approximate confidence values. For the K-Means algorithm, we compute a confidence metric $\phi(\mathcal{M}(X_j), x) := 1 - d(x)$, where $d(x)$ represents the normalized distance to the cluster to which x is assigned. For SVM, we use Platt calibration to translate the distance from the point to the hyperplane into a confidence metric [42], i.e.,

$$\phi(\mathcal{M}(X_j), x, i) = \frac{1}{1 + \exp(-d(x, i))},$$

where $d(x, i)$ represents the distance between x and the hyperplane for class i. Then, $\phi(\mathcal{M}(X_j), x) = \max_i \phi(\mathcal{M}(X_j), x, i)$. We observe that the distribution of $\phi(\mathcal{M}(X_j), x)$ is not always Gaussian; we thus directly approximate its CDF through 1,000 trials.