AnonML: Locally private machine learning over a network of peers

Bennett Cyphers
LIDS, MIT
Cambridge, MA - 02139
bcyphers@mit.edu

Kalyan Veeramachaneni
LIDS, MIT
Cambridge, MA - 02139
kalyanv@mit.edu

***Abstract*—We present AnonML, a system for privacy-preserving model generation over a network of peers. Our goal is to allow a group of users to combine enough data to generate useful machine learning models without revealing private information. In our setting, each peer has a single row of featurized data according to a shared schema, and an aggregator would like to train a binary classification model on the union of all peers' data. Our system horizontally and vertically partitions the set of all peers' data and assembles a differentially-private histogram for each partition. An ensemble classifier can then be trained on the set of noisy partitions. AnonML can be used with or without differentially private data perturbation. Without perturbation, the resulting classifiers achieve performance competitive with centrally-generated models. With local differential privacy, a strong theoretical guarantee, AnonML is capable of producing useful models for practical prediction problems.**

I. Introduction

In the past two decades, the amount of user data collected by smartphones and internet services has skyrocketed. At the same time, advances in machine learning and data processing technology have increased the range of uses for all kinds of personal data. In particular, classifiers are used for everything from targeted advertising to media recommendations to medical diagnoses. Every machine learning application is only as good as the data used to train it, and many classifiers must be trained using detailed, sensitive information. This has led to a central tension between privacy and utility.

For many applications, one party must aggregate data from thousands of people in one place in order to train a classifier. As a result, a dominant paradigm for machine learning today involves a central data-holding authority that has extensive, exclusive access to data from a large number of users. A few large authorities – corporations, universities, and government institutions – control the collection and storage of vast amounts of personal data. These organizations are able to build whatever models they want with the data they have, and to use those models however they like. As machine learning continues to gain popularity, the number of entities who seek to collect sensitive user data will continue to grow.

Unfortunately, this paradigm limits who can use machine learning and what they can use it for, and it tends to disenfranchise the subjects of the data. Users who don't want to share data with one of these organizations may not be able to access all the machine-learning powered products and services they'd like. Entities without access to a large data-gathering apparatus – e.g. researchers, small companies, and normal people – have a much harder time gathering enough data to train certain kinds of models. And often, once shared, a user's data escapes their control.

We attempt to address these issues with AnonML, a system that enables privacy-preserving machine learning by an untrusted aggregator. In our setting, each of a group of peers controls a set of rich personal data. The peers agree to share small, anonymous packets of plain-text information with the aggregator. Our goal is to allow the aggregator to collect enough data from the group to generate a useful machine learning model without compromising the peers' privacy.

In this paper, we approach challenges in three domains:

1) **Trust and anonymity:** How can a peer share data with an aggregator without revealing his identity? How can an aggregator verify that the data she receives is coming from *some* peer in the network without linking the data to a specific peer, while preventing duplicate senders?
2) **Privacy:** Given that a peer can send data securely and anonymously, how can he guarantee that the contents of his data do not reveal sensitive information? Using *local differential privacy* as our metric, how can we maximize the utility of data releases while maintaining a high level of privacy?
3) **Learning:** Given a set of noisy, anonymous data points, how can an aggregator generate a useful model? What kinds of datasets lend themselves to useful machine learning with our system?

This paper is organized as follows: Section II summarizes related work. Section III describes a real-world scenario as a motivating example. Section IV gives an overview of the AnonML system. Section V sketches out AnonML's network communication backbone. Section VI outlines the data collection process and our model building process. Section VII discusses private data sharing techniques. Section VIII presents our experimental design and the results we achieved. Section IX concludes and suggests future work.

II. Related work

A great deal of recent work has focused on privacy preserving machine learning or secure multiparty machine learning. In this section we give a brief overview of related literature.

A. Data privacy

There is a rich tradition of research into privacy-preserving data collection and processing techniques. Random Response, first invented by Warner in 1965 [1] and improved by Greenberg et al. shortly after [2], involves having individuals answer questions incorrectly with some fixed probability. Post-randomization (PRAM) refers to a similar technique applied in a centralized setting, after unperturbed data has been collected [3]. The effects of PRAM have been studied in a number of data-analysis related contexts, including the utility of PRAM-perturbed joint type estimates [4] and the potential for training logistic regressions on data after PRAM [5].

Since its introduction in 2006, *differential privacy* has become a *de facto* standard for privacy-preserving data release [6], [7]. In general, differential privacy can be assessed in two settings: *global*, where a trusted central authority collects real data, then releases it in a privacy-preserving manner, and *local*, where there is no such authority, and each user releases a privatized version of their own data. We are concerned with the local setting.

One related line of work is private multiparty machine learning, where multiple data-holding parties would like to build a model without directly sharing data [8], [9], [10]. Those authors consider a setting in which each party possesses a *local classifier* based on private data. Many parties can then combine their local classifiers into a more powerful ensemble using locally-private releases. We want to accommodate the case in which a user can generate features, but does not have enough information to train a classifier on their own.

Our system uses locally-private histogram estimation for data sharing. RAPPOR, by Erlingsson et al, addresses a version of this problem [11], and our data sharing mechanism is based on their proposal.

B. Cryptography

Some recent work has focused on machine learning over encrypted data, either with fully homomorphic encryption (FHE) or secure multiparty computation (MPC) [12]. Others have attempted to use multiparty computation with differential private mechanisms to produce globally-private answers to queries in the absence of a trusted central authority. An early proposed system by Dwork [13] uses MPC and distributed Laplacian noise generation to allow a group of users to answer counting problems about their personal data with differential privacy.

These solutions are powerful, but they tend to have high communication and computation costs. As technology improves and computation and bandwidth continue to get cheaper, FHE- and MPC-based solutions may become more viable. However, we believe that the best way to approach a practical solution in 2017 is by sharing and computing on plain-text data.

III. Motivating Example

AnonML is designed as a solution to the general problem of privacy-preserving classifier generation without a trusted authority. AnonML can be used in a variety of different scenarios, some of which we discuss in section IX-B. Here, we describe a single potential use case as a motivating example.

Consider a Massive Online Open Course (MOOC): an online course with thousands of students, such as one offered by EdX or Coursera. Each student watches lectures online, completes homework assignments, and takes tests. As students interact with the courseware, their computers collect and store thousands of data points [14].

In a MOOC ecosystem, there are multiple stakeholders – students, instructors, universities, and platform providers. Each stakeholder may be interested in using student data to generate predictive models or classifiers, but the stakeholders are likely to have different goals. Instructors may want to know who is likely to drop out in the coming weeks; students may want to predict their chances of earning a passing grade. The platform provider may want to predict which courses will interest a particular student. And university policy makers might be interested in which attributes predict student retention and success.

To achieve their goals, each stakeholder needs access to some portion of the data, and the data generators (students) may trust different stakeholders to different degrees. In a traditional MOOC scenario, one authority, likely the course administrator, has access to the entire dataset. She can choose to share data with other stakeholders, like instructors or university researchers, or generate models on their behalf. Although students generated the data, they relinquish control over who can access it and what is done with it.

With AnonML, students are able to control their own data to a much greater degree. Different stakeholders can query students for different types of data depending on their needs, and students can grant each query independently. Since data is only ever released privately, AnonML makes it possible for any student in the group to become an aggregator themselves: imagine a machine learning MOOC in which students train models on each other's data. For course administrators and universities, AnonML minimizes risk and reduces overhead. If models can be learned on private data, there is less need to store sensitive personally-identifying information, and less risk of a damaging data breach.

IV. AnonML Overview

With AnonML, an *aggregator* collects data from a large group of data-holding *peers* in order to construct a classifier. The data shared by the peers is anonymous and differentially private, and the aggregator does not need to be trusted. The set of peers in the group is known to every peer in the group as well as the aggregator. After the group has been formed, no information to identify any individual within the group should be associated with any of the data points shared with the aggregator.

At a high level, AnonML works as follows:

1) **Proposal**: An aggregator proposes a classification problem and describes a label and a set of features. All the peers who agree to take part register their identities with the aggregator. The aggregator partitions the set of peers and requests a set of feature subsets from each partition.

2) **Local processing**: Taking direction from the aggregator, each peer computes a set of features and a label on their own data. Each peer assembles a set of *partial feature vectors* corresponding to the requested feature subsets.
3) **Perturbation**: Each peer perturbs each one of their partial feature vectors using a probabilistic, differentially private algorithm.
4) **Sharing**: For each partial feature vector, each peer generates a data packet consisting of their vector and a verification token for that vector. The peer shares each packet with the aggregator over a separate anonymous network connection.
5) **Verification**: The aggregator uses each data packet's token to verify that it was sent by a member of the group, and that no peer sent duplicate data.
6) **Learning**: The aggregator trains a classification model on each partition of the collected data. The models are combined into an ensemble based on cross-validation scores.

AnonML can be thought of as two separate, complementary data-sharing techniques. First is a system for anonymous data sharing among a number of trusted peers, as executed in steps 1, 4, and 5 above and described in section V of this paper. This system assumes nothing about the data being shared, and does not provide differential privacy on its own.

The second is a differentially private method for releasing feature vectors, as executed in step 3 and described in section VII. The differential privacy of the system does not depend on anonymous communication: AnonML's theoretical guarantees are the same regardless of how peers choose to share their data with the aggregator. However, we argue that the two techniques – anonymous communication and differentially-private perturbation – complement each other, and using both in combination serves to reduce the risk of privacy breaches more than either one could on its own.

Finally, in step 6, the aggregator learns a model. In section VI-B we present a simple method by which an aggregator can use data shared with AnonML to train an ensemble classifier. In section VIII we test our method on real-world datasets and describe our results.

V. Anonymous data exchange

In AnonML, when an aggregator receives a message, she must be able to verify that the message is indeed from one of the peers in the network, but must be unable to further narrow down the set of peers it came from. These are two separate requirements. The first, *anonymity*, ensures that the aggregator cannot tell from whom in the group a particular message was sent. The second, *verifiable membership*, ensures that the aggregator can verify that each message was sent by a legitimate peer.

A. Anonymous Routing

For AnonML to satisfy anonymity, any peer in the network must be able to send network packets to any other without leaking any identifiable information to the receiving party. To achieve this, AnonML peers send traffic through an onion routing network such as Tor [15]. Each peer generates a new identity for each packet so that consecutive packets cannot be linked to the same peer. In our experiments, we used Tor itself as the network layer, but this may not be ideal for larger-scale implementations of AnonML. Tor is not designed for clients to create a new circuit for each packet, as AnonML requires, as doing so can place a heavy load on Tor's network. In addition, Tor can be vulnerable to traffic frequency analysis by adversaries with powerful monitoring abilities [16]. These concerns can be mitigated with a custom implementation of an onion routing network in which each peer acts as a relay. We omit the details of such a system for the sake of brevity.

B. Anonymous verification

Anonymous network communication presents a problem for the aggregator: if she does not know where a given feature vector comes from, how can she be sure the sender is part of the trusted group of peers? Additionally, how can she be certain the same peer is not sending duplicate packets? To address these issues, AnonML peers must send each message to the aggregator with a *verification token* attached. The token must prove that the source of the message is someone from the group without revealing additional information, and it must ensure that the same peer cannot send duplicate messages to skew the aggregator's results.

In a recent paper, *Anonize* [17], Hohenberger et al. propose a system for solving the closely-related *anonymous survey response* problem. Anonize allows an authority to select an ad-hoc group of users and create a "survey" where each user can anonymously submit exactly one response. An Anonize *registration authority* (RA) issues a "master user token" to each user in the group. The *survey authority* (SA) then publishes a unique survey ID. Each user with a master token can use the survey ID to generate a non-interactive zero knowledge proof (NIZK) which proves they have been authenticated by the RA and commits them to their answer. This is the verification token. Each authenticated user can only generate one token for each survey ID, and anyone can verify their token. It is computationally hard to determine which authenticated user generated a particular token, and computationally hard to forge an illegitimate token. In the context of AnonML, the aggregator acts as both a RA and a SA, and create a new "survey" for each feature vector that they request.

VI. Learning

Here we describe the structure of the dataset which an aggregator collects, and how that data can be used to generate a classifier. We use the following terminology:

– An *entity* is an abstract object which the model will attempt to classify.
– A *feature* is a value computed from the data that quantifies behavior/property of an instance of an entity.
– The *label* is a special feature: the value which the model will be trained to predict.
– A *discriminatory model* is a mathematical function which accepts as its input a set of features pertaining to an instance of the entity, and produces as its output a prediction for the value of that instance's label. When used in this context, it is also referred to as a classifier.

Let's return to our MOOC use case for an example. Suppose the administrator of a class wants to train a model on last semester's students which will predict, half-way through next semester's class, whether each student will eventually pass or fail. In this case, the *entity* is a student. Some *features* may be variables like "age," "hours spent watching lecture videos," and "average homework grade," measured at the mid-semester point. The *label* is a boolean variable which represents whether or not a student passed the class. The resulting *discriminatory model* will accept as input features about a student's performance and produce as output a predicted label of "PASS" or "FAIL."

A. Data collection

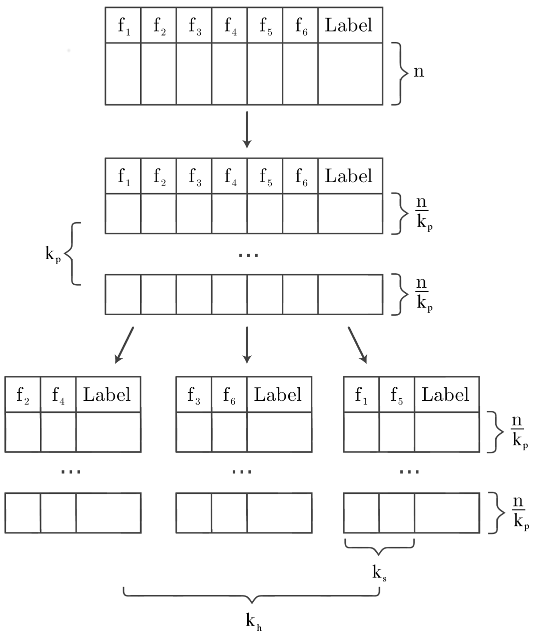


Fig. 1: Partitioning the dataset. In this example, there are $k_f = 6$ features total and n peers. The aggregator splits the group into k_p horizontal partitions, and then requests k_h subsets of k_s features each from each peer.

Suppose that:

– There are n data-holding peers in the network.
– Each peer i, $1 \le i \le n$ has a set of data pertaining to at least one instance of an entity.
– Each peer is able to compute k_f features on his/her data, $x^i_{1...k_f}$, as well as a class label l.

Without loss of generality, we will assume that each peer computes a full set of features corresponding to a single entity. The total set of data present in the network can be thought of as an $n \times k_f$ matrix, with each row corresponding to one instance and each column corresponding to one feature. First, the aggregator splits the group of peers into k_p equally-sized subgroups, which we'll call *peer partitions*, using a shared source of randomness. These are horizontial partitions of the dataset. The aggregator publishes the list of peers (by public key or other identifier) who belong to each partition.

The aggregator then requests a list of *feature subsets* from each peer partition, k_h subsets with k_s features each. Feature subsets must not overlap within a partition, so $k_h \cdot k_s \le k_f$. Each peer in each partition sends a separate data packet containing a *partial feature vector* for each feature subset requested of them. These data form vertical partitions. The purpose of creating peer partitions is to allow the aggregator to request different feature subsets from each one, thereby capturing more information about joint feature distributions in the dataset. In total, the aggregator collects $k_p \cdot k_h$ partitions. Each vertical partition contains k_s features and one label for each of $\frac{n}{k_p}$ entities. The partitioning process is visualized in figure 1.

B. Building a model

The aggregator's goal is to use the noisy partitions to learn a discriminatory model that maps a feature space to a label:

$$l \leftarrow g(x_{1...k_f}) \tag{1}$$

where k_f is the total number of features, $x_{1...k_f}$ are feature values, $g(.)$ is the model, and l is the label we want to infer. Without loss of generality, we will consider the case where the label is a binary variable, $l \in \{0, 1\}$.

Once the aggregator has collected all $n_{parts} = k_p \cdot k_h$ partitions of the dataset, she learns a classifier on each one, $g_i(.)$ for $1 \le i \le n_{parts}$. Each *partition classifier* accepts as input a subset of all features, $x_{sub(i)}$, and outputs a single label prediction in $\{0, 1\}$. In order to combine the classifiers into an ensemble, the aggregator cross-validates each partition's classifier on the noisy data to obtain a performance score, s_i, such as ROC/AUC or f1. Then the discriminatory function is

$$\begin{aligned} score(x_{1...m}) = &\sum_{i=1}^{n_{parts}} s_i g_i(x_{sub(i)}) \\ &- \sum_{i=1}^{n_{parts}} s_i(1 - g_i(x_{sub(i)})) \end{aligned}$$

$$g(x_{1...m}) = \begin{cases} 1, & \text{if } score(x_{1...m}) \ge 0 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

This method is similar to the feature subspace method, first described by Ho [18]. We tested different types of partition classifiers and different cross validation metrics, and we describe the results in section VIII-D.

VII. Data Privacy

Thus far, we have shown how AnonML peers exchange data with the aggregator such that the provenance of their network packets is hidden. However, a secure, anonymous data exchange protocol does not prevent disclosure caused by the content of the data. For example, suppose a student from our MOOC example shares a feature packet including their grade on homework 4 (feature), their zip code (feature), and their final grade (label). If the aggregator has auxiliary information about the student – perhaps they've alluded to their grade or their location on a public forum in the past – they may be able to uniquely identify the student's data packet, thereby learning sensitive information (their final grade). In such a situation, it doesn't matter whether the aggregator cannot connect the packet to their IP address or public key: the student's privacy is compromised anyway.

To account for these privacy violations, our system gives participants the option to obscure their data with perturbation, giving them strong theoretical protection with *differential privacy*.

Developed in a series of papers by Cynthia Dwork et al., [19], [20], differential privacy is a measure of the privacy-preserving quality of a system which releases information about a data set. It has been extremely influential in the past decade, and has come to dominate both theoretical and practical discussions on privacy. Differential privacy describes the way a system's output responds to small changes in the underlying data. Intuitively, no one person's presence or absence should affect the system's behavior more than a trivial amount. If a user is deciding whether to allow their data to be used in a differentially private release, they should have confidence that the algorithm will likely produce the same output whether they do or not.

More formally: Suppose we have an algorithm, A, which operates on a data set, D, and produces output according to some probability distribution. Now, suppose we have two datasets, D_1 and D_2, which differ by a single element d. Differential privacy quantifies amount that such a change can affect the probability distribution of the algorithm's output. A is said to be ϵ-differentially private if, for every D_1, D_2 and every set of possible outputs $S \subseteq range(A)$:

$$\frac{P[A(D_1) \in S]}{P[A(D_2) \in S]} \leq e^{\epsilon} \tag{3}$$

It suffices to show that (3) holds for every S with a single element. In other words, the probability that $A(D_1)$ produces any output y must be close (within a fixed multiplicative factor, e^{ϵ}) to the probability that $A(D_2)$ produces the same output. We will use ϵ-differential privacy to quantify the privacy loss incurred by our method of data perturbation and table release.

Differential privacy can be assessed in two settings:

- *Global:* A trusted data analyst has access to data from many different people. The analyst applies the algorithm A to the full dataset and releases aggregate statistics or full databases.
- *Local:* Each person only has access to their own data, and there is no trusted central authority. Each person perturbs and releases their own data separately.

In our setting, each peer must publish their own data as a single point, so AnonML provides local differential privacy. This is achieved as follows. First, peers preprocess their partial feature vectors [1], convert them into local, bit-string "histograms." Next, peers perturb their histograms and share them with the aggregator. Then the aggregator combines noisy histogram data and estimates the joint distribution of feature vectors in the group. Finally, the aggregator samples synthetic training data from this joint distribution and proceeds to build a model.

[1]A partial feature vector includes a peer's label value and a subset of their feature values.

A. Feature preprocessing

The aggregator's goal is to estimate the joint distribution of feature-label vectors in each vertical partition of the dataset using differentially private queries to each peer. We can map this task to the problem of locally-private histogram estimation.

Histogram estimation assumes that each member of a group has a single value from a finite domain of possible categorical values. The estimator must use differentially-private queries to estimate the frequency of each value in the group. The aggregator uses histogram estimation to approximate the distribution over each vertical partition in the dataset. This method requires that all features and labels be discrete values, so that each partial feature vector can be mapped to a single categorical value. As we will show, it is desirable for the domain of the histogram to be small, so features should be of low cardinality.

First, continuous numeric features are mapped to low-cardinality ordinal features. Then, to reduce error, high-cardinality categorical features may be mapped to lower-cardinality features via grouping. Finally, each partial feature vector is mapped to a single categorical value. For example, a vector containing three binary features and a binary label can be mapped to a bit string, $c \in \{0,1\}^4$.

B. Feature binning

Many features in real datasets are continuous or integer variables. In order to share them as private histograms, it's first necessary to reduce them to low-cardinality discrete values. We do so via *binning*: mapping continuous and ordinal variables to a smaller, discrete domain using threshold values. For example, a "test score" feature between 0 and 100 could be mapped to bins of $[0,20), [20,40), ..., [80,100)$.

As we will show in section VII-D, the expected error of histogram estimation is very sensitive to the histogram's cardinality. Therefore, to minimize expected error, we want to reduce feature cardinality as much as possible. For this paper we reduced all ordinal features to binary variables. We chose to bin variables around their global median, reasoning that without prior information about feature-label joint distributions, we should aim to maximize feature parity. In other words, each feature is mapped to bins of $[min, median)$ and $[median, max)$. We devised a privacy preserving median estimation technique, which we describe next.

Privacy preserving median estimation: We use a simple binary search with privacy-preserving queries to estimate the median for continuous-valued features. Our method requires an educated guess about the minimum and maximum values in the distribution. Luckily, such information is often available in practice: for example, our "test score" feature must be between 0 and 100. The algorithm involves splitting the group of peers into k partitions, then querying each one in sequence to obtain progressively more accurate estimates for the median. The full method that the aggregator uses is described in algorithm 1.

On the peers' end, the remote procedure call IsGreater returns a perturbed bit, b, which indicates whether the peer has a value greater than the proposed median estimate. IsGreater uses random response with privacy parameter ϵ_e, and algorithm 1 involves a single differentially private query to each peer in the group. This algorithm can be generalized to estimate the

Algorithm 1 Estimate the median of a numeric feature.

```
Require: $\Pi$ is a set of random partitions of all peers
Require: $(min, max)$ defines range of possible values
Require: $\epsilon$ is the privacy parameter
Require: MarginOfError is a method which finds the
    expected error of a noisy estimate (equation 4)

  function EstimateMedian($\Pi$, min, max, $\epsilon$)
      $e \leftarrow \frac{max-min}{2}$                    ▷ Estimated median
      step $\leftarrow \frac{max-min}{2}$
      for $\pi \in \Pi$ do
          $n \leftarrow 0$
          for $p \in \pi$ do                  ▷ Each peer in partition
              $n \leftarrow n + p$.IsGreater($e$)
          end for
          err $\leftarrow$ MarginOfError($\frac{n}{|\pi|}$, $\epsilon$)
          step $\leftarrow$ step / 2
          if $\frac{n}{|\pi|}$ - err > 0.5 then
              $e \leftarrow e$ + step
          else if $\frac{n}{|\pi|}$ + err < 0.5 then
              $e \leftarrow e$ - step
          end if
      end for
      return e
  end function
```

distribution for an arbitrary number of bins, although we do not address that problem here.

Once the aggregator has computed an estimate of the *median* for each feature, she shares the estimates with the peers. Each peer then generates a new, discrete feature vector based on the *median* values published by the aggregator.

C. Locally-private histogram release

Once peers have finished processing their feature vectors, the aggregator estimates the distribution over each feature partition in a differentially private way. We can map this task to the problem of locally-private histogram estimation.

Suppose the aggregator is trying to estimate the distribution of feature vectors in a vertical partition with domain $\mathcal{C}$. The distribution can be represented by a histogram of length $m := |\mathcal{C}|$, where the value in position i represents the number of peers who have the feature vector represented by the value $c_i \in \mathcal{C}$. Each peer shares a single "local histogram," a bit string of length m indicating whether or not they have each c_i. Without perturbation, each peer would share a bit string with exactly one bit set to 1. Here we explore the ways a peer can perturb its bit string to satisfy differential privacy.

Random response: A simple way to achieve local differential privacy is via *random response*. Each peer reports their real value, $x = c_i$, with some probability p. With probability $1-p$ they choose a value from the rest of the domain, $\mathcal{C} \backslash \{c_i\}$, and report that instead. Basic random response achieves local ϵ-differential privacy with $\epsilon = \ln(m \cdot \frac{p}{1-p})$ [21]. In the histogram setting, the peer first generates a perturbed categorical value $x' \in \mathcal{C}$ according to the method described. They then send a perturbed bit string, $B' \in \{0,1\}^m$, in which the bit corresponding to the value x' is 1 and all other bits are 0.

RAPPOR – Bitwise perturbation: RAPPOR is another set of methods for bitwise perturbation [11]. Using basic, one-time RAPPOR, each peer sends a length-m bit string in which each bit is perturbed independently. If B is a peer's local histogram, each bit $B_i \in B$ is reported as B'_i according to

$$B'_i = \begin{cases} B_i, & \text{with probability } p \\ 1 - B_i, & \text{with probability } 1-p \end{cases}$$

where p is a tunable privacy parameter. This technique is essentially the concatenation of m binary random responses, with each one indicating whether the peer has a specific value. Basic one-time RAPPOR achieves ϵ-differential privacy with $\epsilon = \ln(\frac{p^2}{(1-p)^2})$.

(p, q)**-perturbation**: We propose a slight generalization of the one-time RAPPOR algorithm which treats 0 and 1 bits asymmetrically. Specifically, if the peer's true bit is a 1, the perturbed bit is set to a 1 with probability p. If the true bit is a 0, the perturbed bit is set to 1 with probability q; $q < p$. If $q = 1 - p$, our method is equivalent to one-time RAPPOR.

Theorem 1: The bit-string perturbation method described above is $\ln(\frac{p(1-q)}{(1-p)q})$-differentially private.

Theorem 1 is proven in the appendix. As we will show, adding an extra degree of freedom to the bit-string perturbation allows us to achieve slightly better expected error, especially when cardinality is high or privacy requirements are low.

D. Histogram estimation and error minimization

Each one of the perturbation methods described above yields a noisy histogram, each "bar" of which is an approximate count of the number of peers in the group who have a specific categorical value. The noisy counts collected by the aggregator are skewed away from the actual counts in the dataset, so some post-processing is necessary to achieve a better set of estimates. Once the aggregator has computed an estimate of the dataset's histogram, she generates a synthetic set of training data by sampling from the histogram and converting categorical values back into feature-label vectors. These data are fed into the learning algorithm described in section VI-B.

Here we describe the histogram estimation process, and discuss how to minimize the expected error of the final histogram. Let n be the total number of peers in the group, and n_i be the number of peers who have a particular value $c_i \in \mathcal{C}$. Let $\tilde{n_i}$ be the noisy count collected by the aggregator. The aggregator can achieve a better estimate of n_i, which we'll call $\dot{n_i}$, with the following:

$$\dot{n_i} = \frac{\tilde{n_i} - qn}{p - q}$$

The expected value of $\dot{n_i}$ is the real value: $\mathbf{E}[\dot{n_i}] = n_i$.

1) Expected error: In order to compare perturbation techniques, we will look at the expected error of each. We are particularly interested in the accuracy of the type estimate, T_X: the $l1$-normalized histogram which describes the relative frequency of each value. We will attempt to minimize the expected $l2$-norm error of the noisy type estimate, $\dot{T}_X$.

Theorem 2: The expected type estimate error for (p,q)-perturbation is given by:

$$E\|\dot{T}_X - T_X\|_2 = \frac{\sqrt{(m-1)q(1-q)+p(1-p)}}{(p-q)\sqrt{n}} \quad (4)$$

Theorem 2 is proven in the appendix. We can use equation 4 to compute the expected error of one-time RAPPOR by substituting $1-p$ for q. Likewise, we can compute the expected error of random response by substituting $\frac{1-p}{m-1}$ for q.

2) Minimizing error with respect to ϵ: Let $f_X(m,n,p,q) := E\|\dot{T}_X - T_X\|_2$ be the error function. In general, m and n are determined by the structure of the problem. The privacy parameter ϵ is a function of p and q, as shown in 1, so if m, n, ϵ are fixed, q is determined by p. Therefore, we are interested in minimizing f_X with respect to q.

The univariate $f_X(p)$ is convex on $p \in (0,1)$, so error is minimized where $\frac{d}{dp}f_X = 0$. Let $\lambda := e^\epsilon$. The error is minimized at:

$$p_{min} = \frac{1}{\lambda^2 - 1}\Big(\lambda^2 + m\lambda - \lambda - \sqrt{(m-1)(\lambda^3+\lambda)+((m-1)^2+1)\lambda^2}\Big) \quad (5)$$

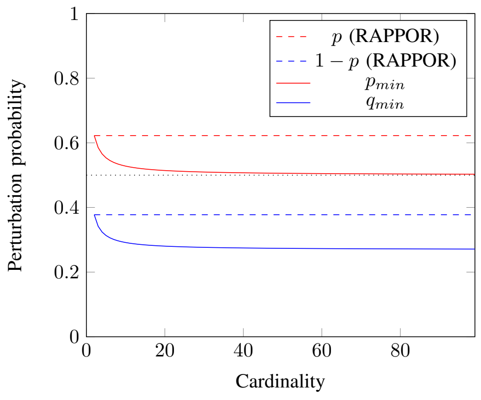


Fig. 2: Optimal p and q as a function of variable cardinality. p_{min} and q_{min} are the values for p and q which minimize expected error for a given cardinality m. As m grows, p_{min} approaches $\frac{1}{2}$. The dashed lines show the equivalent perturbation probabilities under basic one-time RAPPOR. Here, we have fixed $\epsilon = 1$.

Figure 2 shows the relationship between m and p_{min}, q_{min} for fixed ϵ. At $m = 2$, when the feature is boolean, p_{min} is equal to the p such that $p = 1-q$. This is the value for p used by basic one-time RAPPOR. As the cardinality m grows, the error-minimizing p approaches $\frac{1}{2}$.

3) Comparing perturbation techniques: We have described three different techniques for achieving local differential privacy: random response, one-time RAPPOR, and (p,q)-perturbation, a generalization of RAPPOR. We're interested in determining which method will minimize expected error for a given set of parameters.

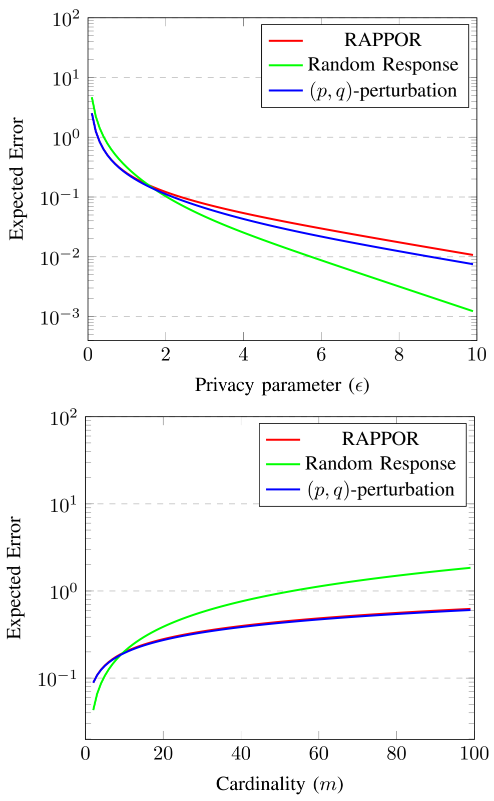


Fig. 3: Expected type estimate error as a function of ϵ and m at $n = 10,000$ peers. On top, $m = 16$; on the bottom, $\epsilon = 1$.

Figure 3 shows the $l2$ error incurred by each technique for various values of ϵ and m. In our setting, (p,q)-perturbation is a strict, if slight, improvement over one-time RAPPOR. The difference in error only becomes relevant at high values of ϵ. For low m and high ϵ, random response is the most effective perturbation method. For low ϵ and high cardinality, (p,q)-perturbation is optimal.

Since expected error can be computed with a simple equation, AnonML peers can determine the error-minimizing method of perturbation before each data exchange.

E. Privacy Budget

One of the earliest results in the field of differential privacy was that if the same data are released privately multiple times, the epsilons "add up." More formally, if there are k releases of the same dataset under differentially private mechanisms with parameters $\epsilon_1, ..., \epsilon_k$, the dataset is protected by $(\sum_{i=1}^{k} \epsilon_i)$-differentially privacy [7].

Under our system, each peer releases multiple private bit strings for histogram estimation. Each bit string release contains information about several features. Even if the feature subsets are mutually independent, the release of k_h bit strings involves k_h separate releases of the label value. If each histogram is released with ϵ_h-differential privacy, the system achieves $(k_h\epsilon_h)$-privacy with respect to the label.

Median estimation requires another set of private queries, each of which asks for a single binary attribute of a single feature. If we assume the features are independently distributed, the privacy cost is accrued by the privacy budget for each feature, which is separate from the privacy budget for the label.

Let ϵ_h^j be the privacy parameter for the j^{th} feature subset historgram release, and let $\epsilon_h(i)$ be the parameter for the feature subset containing feature i. Let ϵ_e^i be the parameter for the median estimate of feature i. Then the total privacy budget for a single feature is $\epsilon_f^i = \epsilon_h(i) + \epsilon_e^i$, and the budget for the label is $\epsilon_l = \sum_{j=1}^{k_h} \epsilon_h^j$.

If a peer has k_f features and releases private histograms for k_h feature subsets, the total privacy budget must be

$$\epsilon_T = \max\left(\max_{1\leq i\leq k_f}(\epsilon_f^i), \epsilon_l\right) \tag{6}$$

VIII. Results

We tested AnonML model generation on a number of datasets to assess its performance in real-world scenarios. Using what we consider to be reasonable values for ϵ, it was possible to generate useful, performant models on some problems. From this, we conclude that there may be a great deal of problems which can be solved with our system. However, further research should investigate what kinds of problems are amenable to locally-private learning.

A. Datasets

We primarily tested and evaluated our system with two datasets: a set of MOOC user data from the EdX platform and a US census release. Both datasets comprise simple numeric or categorical values and both have a binary label.

1) EdX: predicting dropout: We had access to raw data from a number of popular 2012/2013 MIT EdX online classes [14] collected with the MOOCDB system [22]. The data comprises rich, fine-grained logs of every student's interactions with the courseware, including clicks, problem submissions, interactions with lecture videos, forum posts, wiki edits, etc.

Our classification problem is the following: given a student's data up to and including week i of a class (and the fact that they were still enrolled at week i), predict whether the student would drop out before week j. The examples in this paper use $i = 6$ and $j = 10$. We processed rich log data into a set of 13 features, which are all numeric, continuous, and independent.

2) Census: predicting salary: We used the "Adult Data Set" of census data from the UCI machine learning repository [23]. The dataset contains 15 features on 48,842 individuals, including education level, age, sex, and marital status. The task is to predict whether a given person makes more than \$50,000 per year. There are six numeric features, including age and capital gains, and eight categorical features, including marital status and occupation area. We performed manual bucketing on some categorical features in order to reduce cardinality (e.g. by reducing the "nation of origin" feature into just "US" and "Non-US" categories), and removed two redundant or unnecessary features (e.g. "education level" and "years of education" carried the same information).

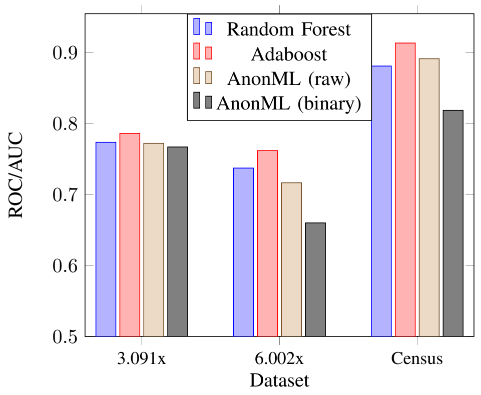


Fig. 4: Comparison of AnonML with other ensemble learning methods on unperturbed data. We used five peer partitions and as many non-overlapping feature subsets as possible.

B. Comparison with traditional methods

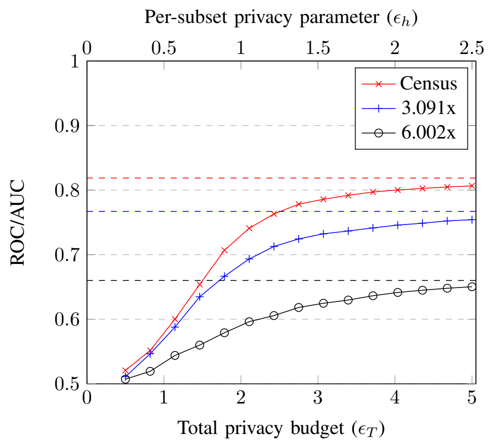
First we tested our model-generation technique on raw, unperturbed feature data. Many of the features are high-cardinality categorical or continuous variables. We tested both with and without binary feature binning. This mode does not offer any privacy guarantees, but gives a ceiling for attainable performance with privacy, and demonstrates that our vertical-partitioning method is sound. The results are in figure 4.

C. Performance with privacy

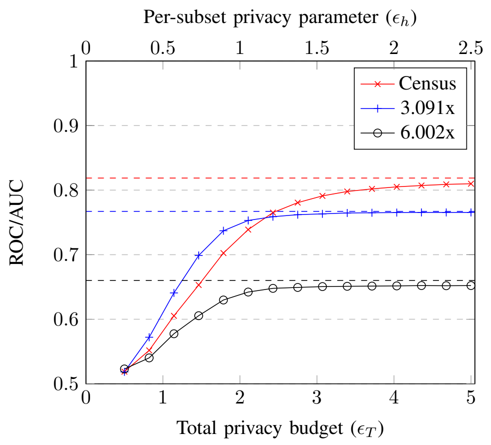
Next, we tested the problems with privacy-preserving binning and data release. For all tests, continuous features were binned into binary features using median estimation, and AUC scores are an average of 500 trials.

Figure 5 shows AnonML's performance on various problems under different privacy budgets. The dashed lines indicate the performance of our model without any perturbation. In those tests, all features were binned around their true medians, and all histograms were known precisely. In 5a, tests were performed with two sets of three random feature on each of five peer partitions, while in 5b, each peer was queried for two "sets" of one feature each – peers returned just a single feature and label at a time. The privacy parameter for subset perturbation (ϵ_h) is shown on the top axis, and the total privacy budget (ϵ_T) is shown on the bottom. In all tests, the privacy parameter for median estimation, ϵ_e, was equal to ϵ_h.

For all datasets, utility approached the optimum asymptotically as a function of ϵ_T. Sacrificing privacy increased performance, although the returns diminished as ϵ_T grew. For both EdX datasets, the single-feature models approached optimal performance much more quickly as a function of ϵ than the three-feature models. However, on the census dataset, models with three features per subset performed just as well as those with one feature per subset, indicating that joint feature distributions were more important than in the other datasets. Overall, we found AnonML's tradeoffs between utility and privacy to be reasonable and encouraging.



(a) Subset size 3; 2 subsets/partition; 5 partitions



(b) Subset size 1; 2 subsets/partition; 20 partitions

Fig. 5: Model performance as a function of ϵ_T on different datasets. All experiments are mean values from 500 trials. Dotted lines represent performance without any perturbation on each dataset. Note that the performance curves for the EdX datasets, 3.091x and 6.002x, tend towards their asymptotes much more quickly in 5b than in 5a. The curve for the Census dataset is nearly identical in both.

D. Optimal partition classifiers

In our tests, we found decision trees to be effective partition classifiers in low-privacy and high-cardinality settings, and logistic regressions to be more effective with low-cardinality variables or with lots of perturbation. Additionally, we found f1 score to be the most effective metric for scoring classifiers in order to optimize for the resulting ensemble's f1 and ROC/AUC.

E. Tuning parameters

AnonML has several parameters which can be tuned by the aggregator to maximize performance.

– *Partitions* (k_p): Splitting the dataset into more horizontal partitions means more classifiers can be trained and tested with a single query to each peer. The trade-off is that fewer peers' data is used to generate each classifier, which means higher expected histogram error.
– *Subset size* (k_s): Larger feature subsets capture more information about joint feature distributions, but cause more noise to be added to each histogram for fixed ϵ.
– *Subsets per partition* (k_h): Requesting more feature subsets from each peer captures more information. For a fixed ϵ_T, more subsets per peer means lower ϵ_h for each subset. Some peers may prefer to keep features more private in this way.

Figure 6 shows the results of experiments with varying subset sizes, and fig. 7 shows how model performance responds to different partition sizes with a fixed subset size. All experiments in figures 6 and 7 were performed on the census dataset using logistic regression as the partition classifier. Our experiments showed that using multiple partitions (when subset size is greater than 1) offered a significant boost to performance, with somewhere between $k_p = 8$ and $k_p = 16$ appearing to be the optimal value for subset size 3.

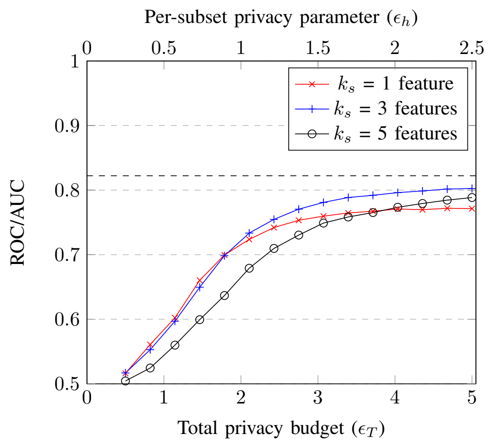
Subset size also had noticeable effect on performance. In general, for the higher-privacy (lower ϵ) domain, using smaller feature subsets yielded consistently better performance. With less privacy, subset sizes of two and three became very slightly preferable. Overall, we found that the best choice was usually a subset size of one – that is, each (feature, label) pair being perturbed and sent independently. This was especially true for the EdX datasets.

IX. Discussion

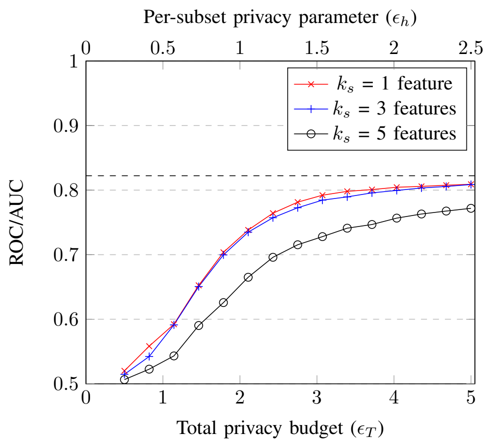
We have made two main contributions in this paper. First, we have explored the extent to which it is possible to build useful discriminatory models with locally-private data. Second, we have presented a practical, end-to-end system that enables such learning with practical and theoretical privacy guarantees. In our tests, we have shown that AnonML can perform well on real-world prediction problems while satisfying local differential privacy.

One interesting result of our tests is that our technique of median estimation and binary binning performed quite well on some datasets. The features in the EdX datasets all began as continuous values, so binary binning destroyed most of the information available. However, as shown in figure 4, the performance of the unperturbed 3.091x model after binning was nearly identical to the AnonML model trained on unperturbed continuous variables. The 6.002x and census datasets suffered more significant performance dropoffs from binning. We remain optimistic about the possibilities of learning with extreme cardinality reduction.

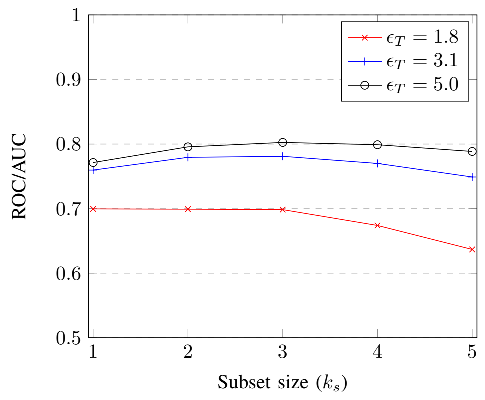
Another result is that small feature subsets consistently performed better than large ones. In other words, with privacy a factor, learning with the joint distribution of many features often hurt performance more than it helped. This is understandable: the expected error of the type estimate grows exponentially in subset size.



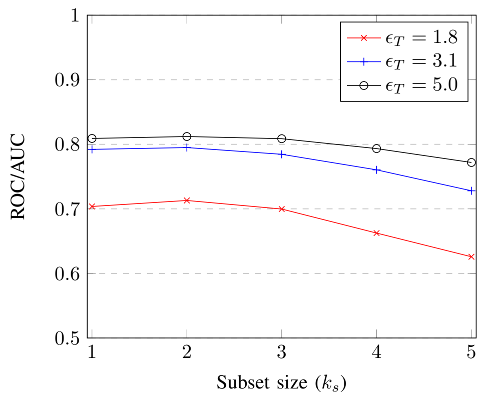
(a) Performance as a function of ϵ with 5 partitions



(b) Performance as a function of ϵ with 20 partitions



(c) Performance as a function of k_s with 5 partitions



(d) Performance as a function of k_s with 20 partitions

Fig. 6: Model performance response to the number of features per subset on the census dataset. 6a and 6c used 5 peer partitions while 6b and 6d used 20 peer partitions. All ROC/AUC values are the mean of 400 trials. The dotted line represents performance without any perturbation. With 5 partitions, overall peak performance was achieved at a subset size of 3, and with 20, performance was slightly better at a subset size of 2 in the high-privacy domain.

For the datasets we tested, it was more important to keep expected error low than to learn joint feature distributions. However, we note that our experiments used a new set of *random* feature subsets for each test. With larger feature subset sizes, the variance of the performance of each model was much greater, and some sets of feature subsets were much more performant than others. This suggests that skilled data scientists may be able to specify feature subsets which can outperform single-feature models in spite of the greater error.

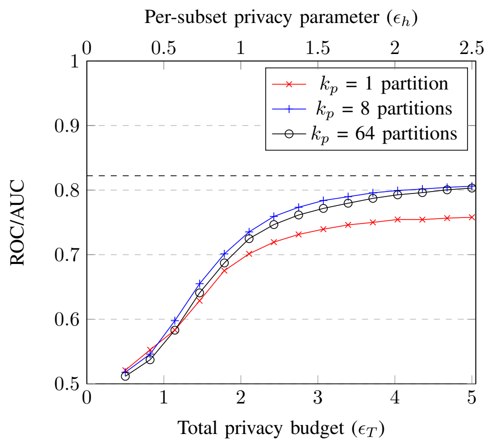
A. *Differential privacy and anonymity*

Our system comprises two privacy-preserving techniques. Anonymous networking, described in section V, ensures that the aggregator cannot link any two feature packets to the same peer. Differentially private data sharing, described in section VII, gives theoretical guarantees about the privacy of data in the whole system. Neither one, alone, can promise perfect privacy. Anonymous routing hides the network provenance of data, but it does not protect users from revealing sensitive information in the data itself. Differential privacy bounds the information an aggregator can learn from any one data release; however, if an aggregator knows a private release came from a particular peer, she can update her prior beliefs about the peer in a potentially meaningful way. With AnonML, a nosy aggregator faces two complementary barriers. Differential privacy ensures she cannot learn anything concrete about any peer or the group as a whole, and anonymous networking ensures that she cannot link any data release to any peer with certainty. Together, these techniques allow peers to share private, sensitive data with confidence.

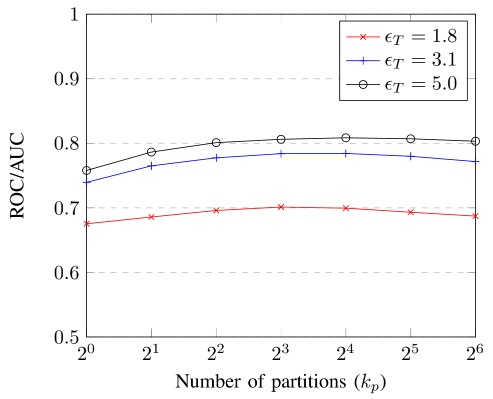
B. *Applications*

Our system is suitable for a variety of learning problems and scenarios where people do not want to trust a central authority. Here, we list a few potential use cases for AnonML:

1) A corporation or institution wants to provide a service which uses a classifier learned from its users' sensitive personal data. The data may be excessively revealing or highly regulated, so the organization does not want the liability of storing it on their servers. The organization only collects and uses sensitive data via AnonML.



(a) Performance as a function of ϵ



(b) Performance as a function of k_p

Fig. 7: Model performance response to the number of horizontal partitions, k_p, with 3 features per subset. The dotted line represents performance without perturbation. All ROC/AUC values are the mean of 400 trials. For this trial, peak performance was achieved between 8 and 16 partitions.

2) Many people use a particular piece of software such as a ride-sharing service. A central authority collects their data but does not allow others to access it. A group of users decide they want to use their own data to train a machine learning model that the authority does not provide for them, for example, to predict when ride prices will surge. One user acts as an aggregator and the rest act as peers and share sanitized data.
3) A new startup is trying to break into an industry with an established incumbent. The startup wants to gather data about when and why the incumbent's users leave their service. They offer a small amount of compensation for each of the incumbent's users to share their usage data in a privacy-preserving way.

C. Future work

The problem of private histogram estimation is relatively well-studied, but its applications to machine learning are not. A more thorough investigation into how best to use locally-private histogram releases to build machine learning models might incorporate heavy-hitter estimation [24]. In addition, $l2$-norm error may not be the best utility function our actual needs (i.e. machine learning performance) – a rigorous theoretical analysis akin to [21] would be desirable.

We used simple logistic regression and decision trees as the basis for AnonML's ensemble classifiers. It may be possible to tune these mechanisms, or find different ones, in order to better learn on noisy histograms. Our tactic of median estimation for feature binning worked surprisingly well, but it may not work for all problems; we recommend investigating higher-cardinality private binning and metrics other than median. To create ensembles, we used cross-validation scores on the noisy partitions. It may be possible to get more accurate scores, and better classifier performance, with another set of privacy-preserving queries to the data holders.

Appendix

Theorem 1: (p, q)-perturbation $\ln(\frac{p(1-q)}{(1-p)q})$-differentially private.

Proof: Let $\mathcal{B}$ be the universe of possible real bit strings: strings of length m in which one bit is set to 1 and the rest are 0. Let $\mathcal{B}'$ be the universe of possible perturbed bit strings, $\{0,1\}^m$. Let $b \in \mathcal{B}$ be a real bit string, and let $b' \in \mathcal{B}'$ be a set of perturbed bits.

$$P(B' = b'|B = b) = \prod_{j\in\{1\ldots m\}} P(B'_j = b'_j | B_j = b_j)$$

Since b_i is 1 and all other real bits are 0, we have

$$\begin{aligned}P(B' = b'|B = b) =& p^{b'_i}(1-p)^{1-b'_i} \times \\ & \prod_{j\in\{1\ldots m\}\setminus\{i\}} q^{b'_j}(1-q)^{1-b'_j}\end{aligned}$$

We need to show that, given two peers with any two real bit strings b and $b*$, respectively, the probability that either one will generate the perturbed string b' is similar. Let i be the index of the 1 bit in b, and let j be the index of the 1 bit in $b*$. Formally,

$$\begin{aligned}e^\epsilon \geq & \max_{b'\in\mathcal{B}';b,b*\in\mathcal{B}} \frac{P(B' = b'|B = b)}{P(B' = b'|B = b*)} \\ =& \frac{p^{b'_i}(1-p)^{1-b'_i}\prod_{k\in\{1\ldots m\}\setminus\{i\}} q^{b'_k}(1-q)^{1-b'_k}}{p^{b'_j}(1-p)^{1-b'_j}\prod_{k\in\{1\ldots m\}\setminus\{j\}} q^{b'_k}(1-q)^{1-b'_k}}\end{aligned}$$

Because the strings b and $b*$ differ in at most two spots, bits i and j, this can be simplified to

$$e^\epsilon \geq \frac{p^{b'_i}(1-p)^{1-b'_i} q^{b'_j}(1-q)^{1-b'_j}}{p^{b'_j}(1-p)^{1-b'_j} q^{b'_i}(1-q)^{1-b'_i}}$$

This expression is maximized when $i \neq j$ and $b'_i \neq b'_j$. In that case,

$$\epsilon = \ln \frac{p(1-q)}{(1-p)q}$$

■

Theorem 2: The expected error for (p, q)-perturbation is given by:

$$E\|\dot{T}_X - T_X\|_2 = \frac{\sqrt{(m-1)q(1-q) + p(1-p)}}{(p-q)\sqrt{n}} \quad (7)$$

Proof: Let n_i be the number of peers who have a value c_i, which means $n - n_i$ peers do not. Let $\tilde{X}_i$ be a random variable representing the number of c_i reported (before the normalization step). We can think of $\tilde{X}_i$ as being drawn from a combination of binomial distributions, $\tilde{X}_i \sim B(n_i, p) + B(n - n_i, 1-q)$. The variance of $\tilde{X}_i$ is

$$\mathrm{Var}[\tilde{X}_i] = n_i p(1-p) + (n - n_i)(1-q)q$$

After collecting the initial noisy count, $\tilde{n}_i \sim \tilde{X}_i$, the maximum likelihood estimation is applied to compute the final estimate, $\dot{X}_i$. The expected value of $\dot{X}_i$ is the true value, n_i, and its variance is

$$\mathrm{Var}[\dot{X}_i] = \frac{n_i p(1-p) + (n-n_i)(1-q)q}{(p-q)^2}$$

We're interested in a *type estimate* T_X for X: the portion of the population that has each $c_i \in \mathcal{C}$. This will be a vector which sums to 1, and our estimate can be computed as $\dot{T}_X = \frac{\dot{X}}{|\dot{X}|}$. The expected Euclidean distance between our estimate and the unperturbed T_X is

$$\begin{aligned}
E\|\dot{T}_X - T_X\|_2 &= \frac{1}{n}\sqrt{\sum_{i=1}^{m} \mathrm{Var}[\dot{X}_i]} \\
&= \frac{1}{n(p-q)}\sqrt{\sum_{i=1}^{m} n_i p(1-p) + (n-n_i)q(1-q)} \\
&= \frac{1}{n(p-q)}\sqrt{mnq(1-q) + n(p(1-p) - q(1-q))} \\
&= \frac{1}{\sqrt{n}(p-q)}\sqrt{(m-1)q(1-q) + p(1-p)}
\end{aligned}$$

■

References

[1] S. L. Warner, "Randomized response: A survey technique for eliminating evasive answer bias," *Journal of the American Statistical Association*, vol. 60, no. 309, pp. 63–69, 1965. [Online]. Available: http://www.jstor.org/stable/2283137

[2] B. G. Greenberg, A.-L. A. Abul-Ela, W. R. Simmons, and D. G. Horvitz, "The unrelated question randomized response model: Theoretical framework," *Journal of the American Statistical Association*, vol. 64, no. 326, pp. 520–539, 1969.

[3] J. M. Gouweleeuw, P. Kooiman, and P. de Wolf, "Post randomisation for statistical disclosure control: Theory and implementation," *Journal of official Statistics*, vol. 14, no. 4, p. 463, 1998.

[4] B.-R. Lin, Y. Wang, and S. Rane, "On the benefits of sampling in privacy preserving statistical analysis on distributed databases," *arXiv preprint arXiv:1304.4613*, 2013.

[5] Y. M. J. Woo and A. B. Slavković, "Logistic regression with variables subject to post randomization method," in *International Conference on Privacy in Statistical Databases*. Springer, 2012, pp. 116–130.

[6] C. Dwork, "Differential privacy: A survey of results," in *International Conference on Theory and Applications of Models of Computation*. Springer, 2008, pp. 1–19.

[7] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[8] M. Pathak, S. Rane, and B. Raj, "Multiparty differential privacy via aggregation of locally trained classifiers," in *Advances in Neural Information Processing Systems*, 2010, pp. 1876–1884.

[9] J. Hamm, A. C. Champion, G. Chen, M. Belkin, and D. Xuan, "Crowd-ml: A privacy-preserving learning framework for a crowd of smart devices," in *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*. IEEE, 2015, pp. 11–20.

[10] J. Hamm, P. Cao, and M. Belkin, "Learning privately from multiparty data," *arXiv preprint arXiv:1602.03552*, 2016.

[11] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 2014, pp. 1054–1067.

[12] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data." 2015.

[13] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2006, pp. 486–503.

[14] A. D. Ho, I. Chuang, J. Reich, C. A. Coleman, J. Whitehill, C. G. Northcutt, J. J. Williams, J. D. Hansen, G. Lopez, and R. Petersen, "Harvardx and mitx: Two years of open online courses fall 2012-summer 2014," *Available at SSRN 2586847*, 2015.

[15] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

[16] N. Hopper, E. Y. Vasserman, and E. Chan-Tin, "How much anonymity does network latency leak?" *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 2, p. 13, 2010.

[17] S. Hohenberger, S. Myers, R. Pass *et al.*, "Anonize: A large-scale anonymous survey system," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 375–389.

[18] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, Aug. 1998. [Online]. Available: http://dx.doi.org/10.1109/34.709601

[19] C. Dwork and K. Nissim, "Privacy-preserving datamining on vertically partitioned databases," in *Annual International Cryptology Conference*. Springer, 2004, pp. 528–544.

[20] C. Dwork, "Differential privacy," in *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, vol. 4052. Venice, Italy: Springer Verlag, July 2006, pp. 1–12. [Online]. Available: https://www.microsoft.com/en-us/research/publication/differential-privacy/

[21] P. Kairouz, S. Oh, and P. Viswanath, "Extremal mechanisms for local differential privacy," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2879–2887. [Online]. Available: http://papers.nips.cc/paper/5392-extremal-mechanisms-for-local-differential-privacy.pdf

[22] K. Veeramachaneni, S. Halawa, F. Dernoncourt, U.-M. O'Reilly, C. Taylor, and C. Do, "Moocdb: Developing standards and systems to support mooc data science," *arXiv preprint arXiv:1406.2015*, 2014.

[23] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[24] R. Bassily and A. Smith, "Local, private, efficient protocols for succinct histograms," in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*. ACM, 2015, pp. 127–135.