A System for Privacy-Preserving Machine Learning on Personal Data

by

Bennett James Cyphers

S.B., C.S. M.I.T., 2016

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2017

Author .....
Department of Electrical Engineering and Computer Science
September 20, 2017

Certified by .....
Kalyan Veeramachaneni
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Christopher Terman
Chairman, Masters of Engineering Thesis Committee

A System for Privacy-Preserving Machine Learning on Personal Data

by

Bennett James Cyphers

Submitted to the Department of Electrical Engineering and Computer Science
on September 20, 2017, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science

Abstract

This thesis describes the design and implementation of a system which allows users to generate machine learning models with their own data while preserving privacy. We approach the problem in two steps. First, we present a framework with which a user can collate personal data from a variety of sources in order to generate machine learning models for problems of the user's choosing. Second, we describe AnonML, a system which allows a group of users to share data privately in order to build models for classification. We analyze AnonML under *differential privacy* and test its performance on real-world datasets. In tandem, these two systems will help democratize machine learning, allowing people to make the most of their own data without relying on trusted third parties.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Principal Research Scientist

Acknowledgments

First and foremost, I have to thank my family: my parents, Luke and Lisa, and my sister Lee; may I someday escape her shadow. Everything I accomplish is a testament to my upbringing—the values they instilled in me, the support they showed me, and the passions they encouraged me to follow.

I need to thank my advisor, Kalyan. When I first approached him, a year and a half ago, I didn't know what I wanted to study, and I was unsure whether research was right for me at all. He not only offered guidance, but inspiration, encouragement, and most generously, time. When I launched headlong into a project that was outside either of our comfort zones, he jumped in right after me. When I was woefully behind on my first paper, deadline looming, he pulled an all-nighter along side me. And towards the end of the year, when the project wore me down, he reminded me why I cared in the first place. He has taken every opportunity to go above and beyond, and I can't thank him enough.

Thank you to MIT and Course VI, for five stressful, exhilarating, formative years. Thank you to Anne Hunter and the undergraduate office, the heroes we need but don't deserve. Thank you to the undergraduate community: I've never met a more interesting, inspiring group of people. In particular, thanks to the residents of Senior Haus—for being yourselves, and for showing us that we could do it too. And thanks to all my friends on Burton Third. Without you, I might have had more sleep, but I would not have had such a home.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 2005, two years before the first iPhone was released, an influential paper by Eagle and Pentland declared that the preceding decade had been "[that] of the mobile phone" [18]. At that time, the world-wide web was eleven years old and still growing fast. Mobile sensors were getting smaller, cheaper, and more accessible by the year, and mobile phone sales had begun to outpace sales of personal computers. Their work highlighted the fact that phones were driving a new phenomenon: constant, passive data collection by millions of people around the world. They showed that the data gathered by hand-held electronics was enough to "recognize social patterns..., infer relationships, identify socially significant locations, and model organizational rhythms." Already, it was clear that these devices, and the information they generated, would bring about a new era of empirical analysis.

In the years that followed, the amount of data collected by electronic devices continued to grow in breadth and depth. In 2009, Pentland returned to the subject in an article for the World Economic Forum [46]. He described an emerging "global nervous system" of electronic sensors and telecommunication devices:

> It seems that the human race suddenly has the beginnings of a working nervous system. Like some world-spanning living organism, automobile traffic systems, security sensors, and especially mobile telephone networks are all becoming intelligent, reactive systems with sensors serving as their

eyes and ears.

Today, there is little doubt that the technological underpinnings for such a nervous system are in place. If 1995-2005 was the decade of the mobile phone, 2007-2017 has been the decade of the smartphone. There were over 3.5 billion mobile broadband subscriptions globally in 2016. This year, 44% of the world is expected to own a smart phone, and those numbers will continue to grow [19]. A typical smartphone is equipped with:

- high-speed internet via LTE and WiFi,
- an always-on GPS radio,
- a gyroscopic sensor and accelerometer for determining orientation and movement, and
- an HD video camera and microphone (or several).

Device owners carry this bundle of sensors with them wherever they go. In addition, most consumers rely on dozens of internet-connected services for entertainment, communication, and to help them manage their daily lives. The result is that, for hundreds of millions of people, the following data are nearly always being recorded *somewhere*:

- regular measurements of location;
- any purchases made through online marketplaces or in brick-and-mortar stores with a phone or credit card;
- any communications made over SMS, email, or messaging apps;
- any media consumed with an electronic device, including music, movies, e-books, and web pages;
- any queries to a search engine or a personal voice assistant.

The sum of this data is highly sensitive and powerfully revealing. Who owns this data, who can access it, and what they can do with it are some of the major questions for our time.

1.1 Machine learning

In this thesis, we focus on the domain of machine learning. Over the past two decades, machine learning tools have become more powerful, more accessible, and easier to use. It has been applied to the ever-growing troves of personal data collected by the "global nervous system" with great success.

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 sensitive personal data. These organizations are able to build whatever models they want with the data they have, and use those models however they like.

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 escape their control.

An unfortunate corollary to this paradigm is that every step of the machine learning pipeline, from data acquisition to model generation, is normally handled by the same authority. This means that a user must be willing to trust such an authority with large amounts of raw data if they want to contribute to training a model. As we will describe in chapter 3, only a small, processed subset of a user's data is necessary to train any given model. If users are able to perform data cleaning and feature processing on their data by themselves, they can contribute to machine

learning datasets without revealing as much information. And, as we will demonstrate, local preprocessing makes it possible for users to perform privacy-preserving transformations before sharing data.

We attempt to address these issues with two contributions. First is a framework that enables users to clean, aggregate, and learn with their own data locally. Second is AnonML, a system that lets users participate in privacy-preserving machine learning with an untrusted aggregator. AnonML builds on and fits into our framework. With AnonML, users of our framework can collaborate to train models of their choosing that they could not build alone, even if they do not trust one another.

This thesis is organized as follows: Chapter 2 summarizes related work. Chapter 3 describes the machine learning pipeline and introduces our framework. Chapter 4 introduces AnonML. Chapter 5 sketches out AnonML's network communication backbone. Chapter 6 describes AnonML's data collection setup and learning process. Chapter 7 discusses privacy-preserving data sharing techniques. Chapter 8 presents our experimental design and the results we achieved. Finally, chapter 9 draws conclusions and suggests future work.

Chapter 2

Related work

A great deal of recent work has focused on how to make machine learning more accessible and more private. In this section we give a brief overview of related literature.

2.1 Personal data processing

The idea of a *personal data store* has existed in academic literature for some time [3, 7, 42]. In recent years, new revelations about digital surveillance have brought the issues of privacy and data equity to the forefront [21], and researchers have pursued privacy-preserving personal data solutions with renewed zeal. OpenPDS is a proposed system that would enable users to store their personal data and metadata in a "black box," and selectively answer queries about the data based on a set of user-defined rules [11]. Another platform, Databox, acts as a hub for a variety of digital data sources and collates personal data to improve availability [24]. Unfortunately, attempts build systems that people actually use have largely failed to gain traction.

OpenPDS and others attempt to be general solutions: platforms which are both data- and application-agnostic. A developer can use OpenPDS to collect nearly any kind of personal data, and once the data is collected, OpenPDS's data-access interface supports nearly every type of application. However, there is a chicken-and-egg problem blocking such a system's actual adoption. App developers have little incentive to build around a new system without user demand, and consumers are unlikely to

demand software that uses a personal data store without seeing examples of it in the marketplace. In our opinion, the main obstacle to the widespread adoption of such a data store is a lack of concrete applications. Developers and consumers have proven unlikely to adopt platforms when they are not sure what they should use them for.

Our system targets a particular use case: machine learning. Rather than build a one-size-fits-all solution for personal data management, we have chosen to strike a balance between specificity and generality. As described in chapter 3, our contributions focus on two weak spots in the pipeline from data collection to machine learning model generation: data collation and private data sharing. We present a framework for merging personal data from multiple sources into one dataset and a simple API that developers can use to prepare that data for machine learning.

2.2 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 [52] and improved by Greenberg et al. shortly after [23], involves having individuals answer questions incorrectly with some fixed probability. This technique gives each respondent a kind of plausible deniability about their answers, and can be useful for sensitive questions like "Are you a member of the communist party?"

Post-randomization (PRAM) refers to a similar technique applied in a centralized setting, after unperturbed data has been collected [22]. The effects of PRAM have been studied in a number of data-analysis related contexts, including the utility of PRAM-perturbed joint type estimates [39] and the potential for training logistic regressions on data after PRAM [53]. These are relevant to us because our local perturbation method is based on random response, and the resulting datasets resemble those perturbed with PRAM.

2.2.1 Differential privacy

Since its introduction in 2006, *differential privacy* has become a *de facto* standard for privacy-preserving data release [14, 17]. 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.

A great deal of work has focused on how a trusted data-holding authority can generate or sanitize models for privacy-preserving release in the global setting [8, 45, 51]. A parallel line of work has focused on how an untrusted data scientist can perform analysis and build models over privacy-protected datasets, e.g. using a series of differentially-private queries to a trusted authority [31, 32].

We are interested in the case where users are not willing to share raw data with any trusted authority, or where a trusted authority (like a corporation) with whom users have shared data is not willing to offer any kind of access to third parties. Therefore we focus on 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 [44, 26, 25]. Authors consider a setting in which each party posesses a *local classifier* based on private data. A group of data-holding parties can then combine their local classifiers into a more powerful ensemble using local differential privacy. We want to accommodate the case in which a user can generate features, but does not have enough information to train a classifier locally.

Our system uses locally-private histogram estimation for data sharing. RAPPOR, by Erlingsson et al, addresses a version of this problem [20], and our data sharing mechanism is based on their proposal. In particular, we are interested in choosing private disclosure mechanisms that minimize expected type estimate error for a fixed set of parameters. In a similar vein, Kairouz et al proved the optimality of certain local perturbation methods with respect to any f-divergence utility function [33].

Their results, including that random response is optimal in the low-privacy regime, do not seem to map to our domain directly. So far, we have not applied the same theoretical rigor to optimizing histogram estimation as Kairouz, but doing so may be warranted in the future. Other work on private histogram estimation has focused on heavy-hitters [2]. We have considered this as a way to improve AnonML's models, but have not explored it yet.

2.2.2 Cryptography

Some recent work has focused on machine learning or general computation over encrypted data, either with secure multiparty computation (MPC) or fully homomorphic encryption (FHE) [6]. Recently, Google deployed a new system for assembling a deep learning model form thousands of locally-learned models while preserving privacy, which they call Federated Learning [41, 5]. On their own, these mechanisms do not provide guarantees about the privacy of their outputs.

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 [15] 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 have high communication and computation costs, or they are optimized for specific use cases. As technology improves and computation and bandwidth continue to get cheaper, FHE- and MPC-based solutions may become more viable for general machine learning. However, we believe that the best way to approach practical private classifier generation in 2017 is by sharing and computing on plain-text data.

Chapter 3

Democratizing Machine Learning

One of our primary goals in this thesis is to take steps towards *democratizing* machine learning: making it simpler, more accessible, and more equitable. Great leaps have been made towards these ends in the past few years, both in the literature and elsewhere. Thanks to open source libraries, open datasets, and excellent online courseware, it is easier for an amateur software developer to learn, practice, and deploy machine learning solutions than ever before.

However, it remains difficult for the same amateur to perform learning on the data that he has created. Although most Americans' lives generate vast amounts of data each day, even those who are technically proficient do not have an easy way of putting that data to good use. Our contribution in this chapter is a framework that aims to democratize machine learning for personal data.

First, we summarize the stages through which an aspiring data scientist must shepherd her data on its way to becoming a useful machine learning model. We define a *model* to be a function, $g : \mathcal{X} \to \mathcal{Y}$, which maps elements from a *feature space* $\mathcal{X}$ to a *label space* $\mathcal{Y}$. A model is trained by *fitting* it to a set of training data, comprising features X and corresponding labels Y. We will expand on this pipeline and describe how our proposed solution addresses each step in section 3.1.

1. **Data acquisition**: Raw data is collected, or retrieved from a data source, in a machine-readable format.

2. **Data processing**: Raw data is "cleaned" and converted to a format fit for feature engineering. Cleaning may include parsing keywords and numbers from raw text, removing redundant or useless information, and applying strong typing to individual values.

3. **Problem definition**: Once the available data has been assembled and organized, the data scientist defines a problem. In practical terms, this amounts to defining the *entity* for the problem, the label space, $\mathcal{Y}$, and a means of extracting a label, $y \in \mathcal{Y}$, from training data pertaining to a single entity.

4. **Feature engineering**: Features are values which the learned function g will use to predict the label – the inputs to g. Features may be raw values directly from the dataset, or may be computed via complex functions on the input data. Feature engineering defines the function's input space, $\mathcal{X}$, as well as a way to extract a feature vector $x \in \mathcal{X}$ from training or test data pertaining to a single entity.

5. **Learning**: Finally, a function that maps the feature space to the label space is "trained." This is usually achieved by passing a large set of (feature vector, label) tuples, called *training examples*, into a learning function. This function uses the training examples to iteratively update a set of parameters which define the discriminatory function g.

Most research has focused on optimizing step 5: learning. Traditionally, "machine learning research" has referred to this part of the problem, and there is far too much literature in the area to list here. Recently, significant work has been done towards automating step 4 by programmatically searching spaces of possible features for the best options [35]. Some research [48] has attempted to move towards automating step 3, but for the most part, problem definition remains the domain of human intuition. Steps 1 and 2, furthest removed from the model-building process, are not frequently thought of as machine learning problems. Though they are simple, repetitive tasks, data acquisition and cleaning are normally handled by ad hoc, single-purpose solutions.

In the next two sections, we examine each stage of the pipeline in more detail. We propose a framework with which amateur data scientists can transform personal data from its raw source to a machine learning model, and explain how the framework fits in to each stage of the pipeline. In section 9.3.1, we describe some potential uses for and expansions to our framework.

3.1 The machine learning pipeline

Structure	**Description**
Field	A single, measured value, and the most basic unit of data. A field has a static type, such as float, boolean, or categorical. After initialization, its value is immutable.
Event	A single unit of measured data. An event comprises a set of fields and an optional timestamp. An event can represent a single measurement from a sensor (like a thermometer reading), a static property (like street address), or a real-world event (like a click or a ride-sharing trip). If an event represents a static property not associated with a time, its timestamp is left as null.
Dataset	A set of events belonging to a single user in a specific time interval. A dataset can contain events from many different data sources. If a dataset contains data pertaining to a single training example, it is referred to as an *instance dataset*. Feature and label computation are executed directly on datasets.
Feature Vector	A set of fields which meaningfully describe a single instance. The fields are *features*, the inputs to the model function.
Label	A field corresponding to a single instance; the value which the model will try to predict.
Feature Matrix	A set of feature vectors and their corresponding labels. A feature matrix can be passed directly into a learning algorithm to train a model.

Table 3.1: Summary of data structures used by our framework.

In this section we examine each step in the machine learning pipeline and describe how our framework fits in. Table 3.1 describes the data structures used by the framework.

3.1.1 Data acquisition

The first step in the pipeline is to gather data from multiple sources so that they may be processed together. Logically, a *data source* is an app or service that collects data about a user and exposes it via some software interface. Some sources, like Google's Gmail and Maps, make it easy to download user data in a software-accessible format. Others, like Uber and Facebook, are less accommodating. For example, Uber does not have a web API, but it does have a ride history page that users can access on its website. In order to use uber as a data source, we found it necessary to use an html-parsing script ("web scraper").

Data collected from a source may be messy, redundant, or unformatted. These inconsistencies will be corrected in the next stage of the pipeline; the important thing is that data is all in one place.

3.1.2 Data processing

Each data source exports data in a different format, and data from different sources likely aren't compatible by default. However, all sources contain information about a single entity – the user. Our goal is to enable the user to leverage all of the information he or she has available for machine learning. In order to do so, data from multiple sources must be coerced into a single format and collated.

We present a simple, generic format for feature engineering. In our system, all data is converted into a series of time-stamped *events*. Each event contains one or more primitive values, called *fields*. Many events make up a *dataset*. Our framework's data structures are summarized in table 3.1.

Each data source must be coerced into the field-event-dataset format via an *importer*. The author of the importer for a particular data source must make decisions

about which data is relevant, which is not, and how to convert dense data like raw text and images into events and fields.

This is similar to the process every data scientist must go through when working with a new data source: raw data must be processed and simplified in order to prepare it for feature engineering. However, once an importer for a data source has been written once, it can be used by everyone with access to the same type of data. We imagine a large, community-generated library of importers, wherein popular apps and services will tend to receive more robust support. When a user installs a new app, she can install an importer for its data at the same time, and the app's data will be seamlessly integrated into her personal dataset.

3.1.3 Problem definition

Once raw data has been cleaned and processed into a dataset, it is time to define a problem. In our context, defining a machine learning problem amounts to defining an *entity* and a *label*. The *label* is a property or event that the resulting model will attempt to predict, and the *entity* is the person, event, or thing that the label describes.

First, the user needs to break her dataset into *training examples*. Each training example is a subset of the data which describes a single instance of the entity in question, and from which a single label can be derived. Training examples manifest as datasets which are subsets of the global dataset, or *instance datasets*.

Second, the user needs to compute a label for each training example. The label can be any value that is derived from a training example and is expressible as a single field. In our framework, the user executes this phase with an *instancer*. As described in table 3.2, the instancer accepts a dataset, and returns a list of labels and training example data-subsets.

We describe two examples of problem definitions below:

1. *Ride-share pricing*: Suppose a user wants to predict what the price of a ride-share trip from point A to point B will be at some time in the future, perhaps

to plan their commute. The user has access to data about several hundred rides she has taken in the past and what they cost. The entity in this case is a ride, and the label is the price.

2. *Farm equipment failure*: Suppose a farmer has an irrigation system set up on his land, with several sprinklers that are turned on and off depending on the weather. He has a meter on each one that logs how much water is flowing and when, and he can use the data to detect when pipes burst. He wants to predict when a pipe is likely burst so that he can perform maintenance before it happens. He may formulate a problem like, "Given historical data about a pipe, how likely is it to burst in the next week?"

   The entity is a pipe-week. The instancer in this case would create separate training examples for each pipe, for each week. The label is a number representing probability of failure.

In general, problems for which each user has access to lots of labeled training examples, like those above, are preferable. However, there are many problems for which a single user does not have access to enough data to train a model on their own. Many of these are problems in which the user is the entity, and so can only generate a single training example. For example, a person may want to predict whether he will pass a particular MOOC class, or how likely he is to secure a loan from a major bank. To model these problems, it is necessary for users to share data with each other. In section 4, we present a general solution for these cases.

3.1.4 Feature engineering

A *feature* is a value that a machine learning model accepts as input in order to predict its output: the label. Once a data scientist has defined a problem, they must define a set of features which they believe will lead to the best predictive power for their model. In our system, each feature is defined as a *feature function* which acts on a dataset and returns a single field. The dataset passed to the function must be associated

with a single instance of the problem's entity. The set of all features pertaining to an instance is a *feature vector*.

To define a set of features and a label for a problem, a data scientist using our system writes a *featurizer*. The featurizer accepts a list of training examples – (instance dataset, label) tuples – as input, and returns a list of feature vectors as output. In the notation used above, X comprises the feature vectors, and Y the list of labels. The full set of feature vectors and labels is called a *feature matrix*.

3.1.5 Learning

After raw data have been transformed into a feature matrix, learning can begin. The goal of learning is to transform a set of observations into a functional model for predicting unobserved traits or events. Machine learning makes the assumption that the values in X and Y are drawn from dependent distributions, $\mathcal{X}$ and $\mathcal{Y}$, such that knowing an instance's feature vector x allows one to reliably predict its label y.

At this point, the data scientist has a full set of observations drawn from $\mathcal{X}$ and $\mathcal{Y}$, and needs to find a function, g, that will most reliably map some $x \in \mathcal{X}$ to the correct $y \in \mathcal{Y}$ in the future.

There are myriad learning algorithms and techniques, too many to describe in detail here. At this point in the pipeline, most ambiguous choices have been made by a human, and it's possible measure a model's performance with simple, low-dimensional metrics. There are a number of tools [49, 4] which allow data scientists to select the best learning algorithm and hyperparameters for any well-defined problem with training data.

3.2 A framework for personal machine learning

We propose a framework to make the generation of predictive models as accessible as possible. Our framework assumes that, for most users, data collection is already taking place as a consequence of the dozens of apps and services that most consumers use. We also acknowledge and embrace the fact that open-source and free-to-use

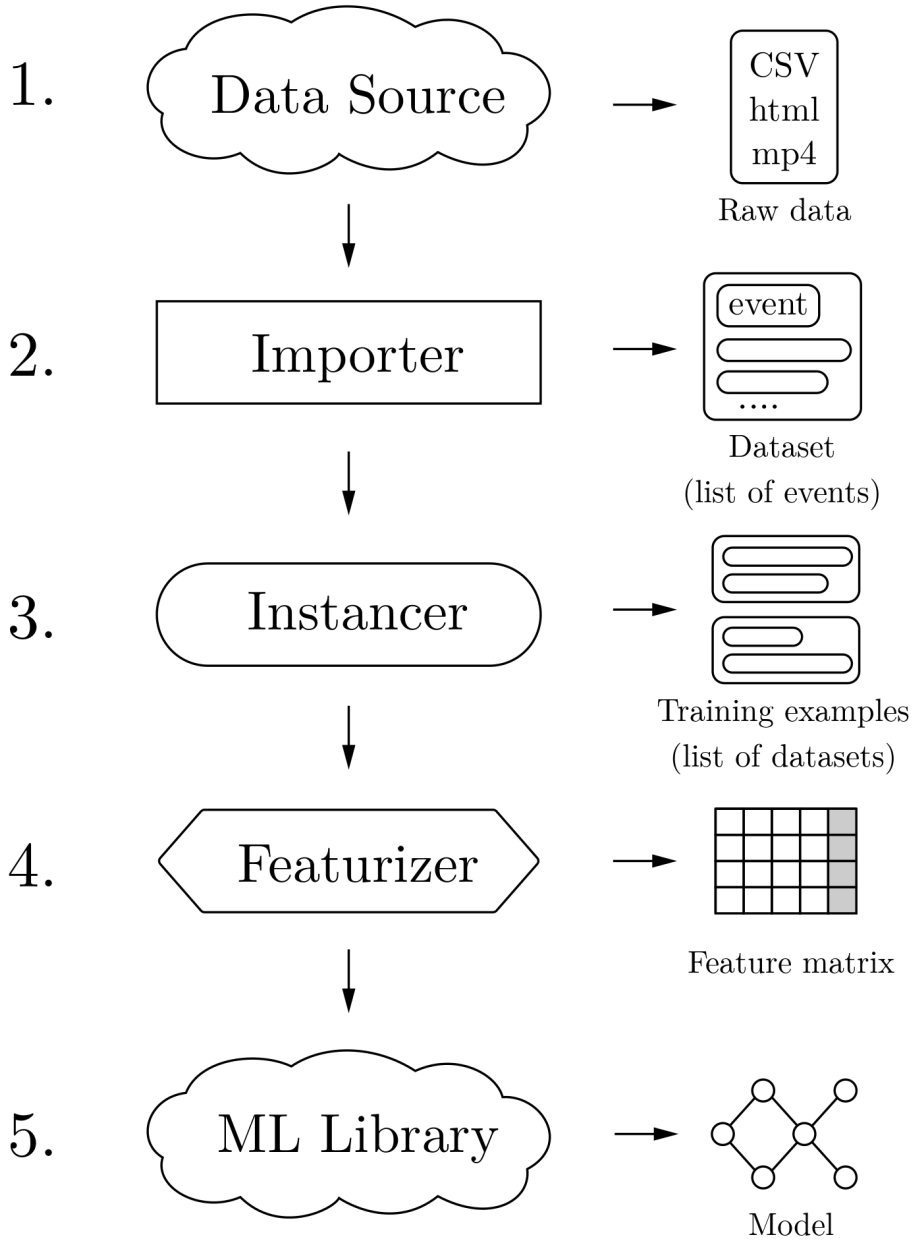


Figure 3-1: The machine learning pipeline, and how our framework interacts with it. Functional units are on the left hand side, and the data representations they produce are on the right.

machine learning libraries have become extremely accessible and powerful in recent years. We do not intend to replace tools such as `scikit-learn` or `tensorflow`; rather, we want to help users work with them. Our goal is to streamline the process of transforming data from whatever raw form they are collected in into a feature matrix, ready to be fed to a machine learning algorithm.

Our framework is built around three functional units, which handle stages 2-4 of the pipeline above. Table 3.2 summarizes the inputs and outputs of the units.

1. **Importer:** An importer is responsible for cleaning and processing raw data from a data source. Each distinct data source requires its own importer. An importer accepts raw data as input and converts it to a series of timestamped *events* (collectively, a *dataset*).

2. **Instancer:** An instancer defines a prediction problem. An instancer defines a function, `make_instances()`, which converts a dataset into a series of *training examples*.

3. **Featurizer:** A featurizer is a wrapper around a set of *feature functions*. Each feature function accepts a training example in the form of an event or dataset and computes a single feature. The featurizer outputs a *feature vector*, a list of features and a label, for each training example.

We start by observing that data collection and processing – stages 1 and 2 – are independent of the latter half of the pipeline. Machine learning problem formulations have a lot in common. If data can be coerced into the right intermediate representation, data scientists can build many different kinds of predictive models using the same set of processed data. And if developers are able to start with data in a standardized format, they can jump right in to building models, focusing their efforts on intuitive problem solving rather than writing boilerplate code.

We also note that most data sources are popular applications with widespread adoption. Software for collecting and processing data should only need to be written once per data source. In our framework, this task is handled by an importer. Around

Processor	Phase	Input	Output
Importer	2	Raw data from a data source, like html, csv, or mp4 files.	A list of timestamped events as a dataset.
Instancer	3	A dataset with all training data for a desired model.	A set of labeled training examples. Each training example is represented as a dataset (set of events) which pertains to a single instance of the entity.
Featurizer	4	A list of training examples.	A list of feature vectors. Each feature vector contains a set of features pertaining to a single instance.

Table 3.2: The inputs and outputs of functional units in our framework. The *Phase* column indicates at which phase of the machine learning pipeline each processor operates.

the world, thousands of data scientists have written thousands of single-purpose "importers" for the same few data sources; there is a great redundancy of work at this stage in the pipeline. Data representation, described in table 3.1, is at the core of our framework. By providing a standardized intermediate representation into which data can be processed, we allow users to share their solutions for processing popular data sources and, hopefully, to save great collective effort.

Chapter 4

AnonML

In chapter 3, we described a typical machine learning pipeline, as well as our framework for enabling learning with personal data. Some types of models can be trained with the data available to a single user. For many problems, it may be possible, or even desirable, to use data from only one person.

For other problems, data from multiple users are required. In particular, any problem for which the entity is the user, such that each person can only generate one training example, necessitates data sharing.

In this chapter, we introduce AnonML, a system which allows a group of peers to share featurized data with an untrusted aggregator privately and anonymously. AnonML extends the framework we presented in the previous chapter, and allows a single data scientist to direct stages 3-5 of the pipeline for an ad-hoc group of data holders.

4.1 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 [27].

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 interests. 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 what courses will interest a particular student. And university policy makers are interested in what attributes predict student retention and success.

The point is that each stakeholder needs access to different aspects of the data, but the data generators (students) may trust different stakeholders to different degrees. The central authority can generate models for herself and on behalf of others. She can share parts of the dataset with other stakeholders, like universities or instructors, but she is unlikely to share sensitive data with the students themselves.

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 dataset, they don't control who can access it or what is done with it.

With AnonML, students are able to control their own data and put it to good use. Since data are only ever released in a privacy-preserving manner, AnonML makes it feasible for any student to propose a model and collect data from the rest of the group. 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 lesser risk associated with a data breach.

4.2 System overview

AnonML enables an *aggregator* to collect data from a large group of data-holding *peers* in order to construct a classifier.

We begin by noting that, in order to build a model, an aggregator only needs

to collect featurized data. This means that raw data can be processed locally by the peers for stages 1-4 of the pipeline, and data only has to be shared for stage 5. AnonML extends our local-learning framework. Peers who want to take part in learning with AnonML are assumed to have a common set of data sources as well as importers for those sources. The aggregator sends the peers an instancer and a featurizer to use, and the peers compute feature vectors with their data on their own machines (locally). Peers then share a handful of features and their label, rather than any significant portion of their raw data, with the aggregator. The local learning pipeline is shown in figure 4-1, and the pipeline with AnonML is illustrated in figure 4-2.

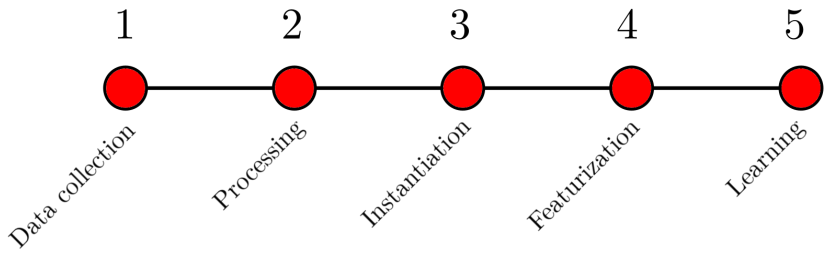


Figure 4-1: The basic machine learning pipeline. Data flow from left to right. Each circle represents a stage in the pipeline where data is transformed.

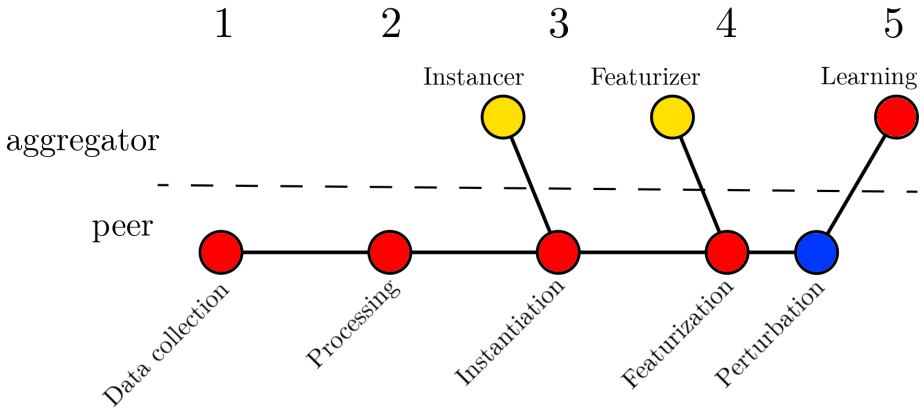


Figure 4-2: The machine learning pipeline with AnonML. Data flow from left to right. The aggregator dictates an instancer and a featurizer (yellow), which the peers use to transform their data (red) on their own machines. AnonML adds a new step, perturbation (blue), wherein peers privatize data before sharing with the aggregator. Finally, the aggregator performs learning on private feature vectors collected from the peers.

The data shared by the peers are 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 for the group to solve, along with an instancer and a featurizer. A group of peers who agree to take part register their identities with the aggregator.

2. **Local processing**: Each of the peers computes a feature vector and a label using the aggregator's instancer and featurizer with their own data.

3. **Perturbation**: Each peer vertically partitions their feature vector into a set of *partial feature vectors*. Each partial vector has a subset of the peer's features and their label. The peer then perturbs each partial vector using a probabilistic, differentially private algorithm.

4. **Sharing**: For each partial feature vector, each peer generates a data packet consisting of the 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 uses the set of perturbed data to generate synthetic training examples for each partition. She trains a classification model on each partition of the synthetic 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 chapter 5. 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 low-dimensional, discrete feature vectors, as executed in step 3 and described in chapter 7. 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, in chapter 9 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, we present a simple method by which an aggregator can use data shared with AnonML to train an ensemble classifier: step 6. In chapter 8 we test our method on real-world datasets, discuss performance with variable parameters, and describe our results.

Chapter 5

Anonymous data exchange

The networking aspect of AnonML allows peers in the group to communicate with the aggregator *pseudo-anonymously*. Here we describe the technical backbone of the AnonML peer-to-peer network.

In AnonML, when the 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 nobody can 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 from the group.

5.1 Anonymous Routing

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.

Anonymous network communication without a central authority was introduced by Chaum in 1981 [9], and the basic concept has been adopted into onion routing and TOR, the most popular anonymization network in the world today [47, 12]. Onion networks allow a client to establish an anonymous, two-way connection to any server at any time.

An onion network consists of a large group of peers, known as *relays*, who publish their public encryption keys. Using an onion network, a client may establish an anonymous connection, known as a "circuit," to any server via the network at any time; the client and server do not have to be relays in the network in order to use it. The server can respond to queries from the client without knowing anything about the client's identity. A simplified version of the protocol a client would follow to send an anonymous request to a server is described below.

1. The client chooses a sequence of k relays in the network, $\{r_1, ..., r_k\}$, identified by their IP addresses. This is the path the client's traffic will follow on its way to the server, known as a *circuit*.

2. The client encrypts the message it wants to send in the key of the server. It then concatenates the encrypted message, M_{enc}, with the address of its destination server, *dest*, in the public key of the last relay, P_k. We'll call this intermediate message $M_k \leftarrow Enc(P_k, M_enc|dest)$.

3. The client then encrypts M_k and the identity of the last relay, r_k, in the public key of r_{k-1}: $M_{k-1} \leftarrow Enc(r_{k-1}, M_k|r_k)$. They encrypt M_{k-1} in r_{k-2}'s key in the same way. This continues until the client has generated $M_1 \leftarrow Enc(r_1, M_2|r_2)$.

4. The client sends M_1 to r_1, who decrypts it and sends M_2 to r_2, and so on. Finally, relay r_k receives M_k, decrypts the original message M_{enc}, and sends it to *dest*.

At any given point in the circuit, each relay only knows about the relay just before it and the one just after it; it has no way of knowing where the packet originally came from or where it is eventually headed. As long as at least one of the relays in the circuit is honest, onion routing prevents any number of malicious relays from compromising a client's anonymity.

Some onion networks, including TOR, are vulnerable to traffic frequency analysis by adversaries with powerful monitoring abilities [10, 43, 30]. Traffic analysis attacks involve monitoring the amount and timing of network traffic at source and destination

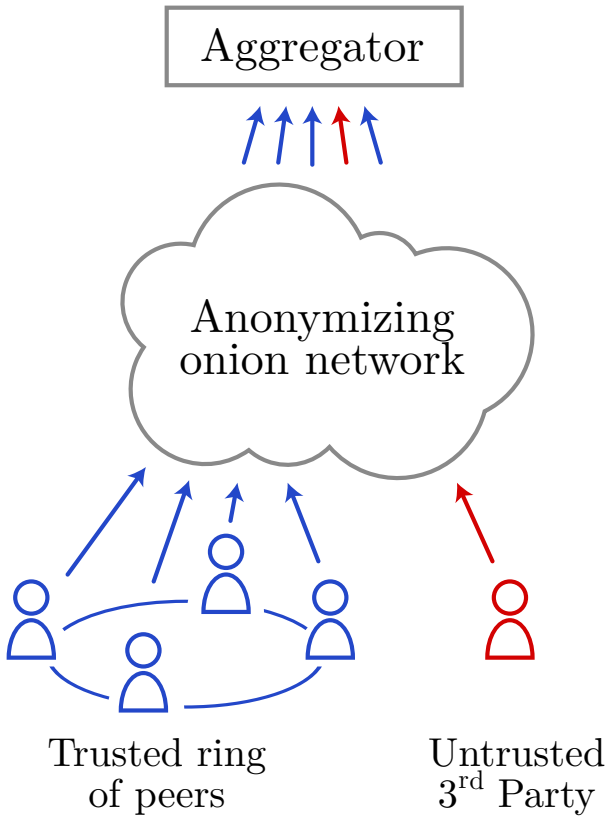


Figure 5-1: A group of semi-trusted peers form an anonymous network. Onion routing provides anonymity, and tokens or ring signatures allow verification. The aggregator can tell which packets come from within the group, but not whom they are from.

nodes. Traffic analysis attacks work despite robust encryption. In our model, a traffic analysis attack could involve an aggregator monitoring outgoing traffic from one peer in an attempt to correlate the it with the incoming data packets it receives.

Here we observe that natural constraints on AnonML network traffic allow us to provide more robust anonymity than standard onion networks can. Data flowing through the AnonML network has a regular structure and schedule, and all peers are responsible for sending data of the same structure to the aggregator at roughly the same time. Therefore, traffic analysis attacks can be mitigated with slight modifications to the standard onion routing scheme.

First, peers should pad their packets such that all packets of similar data type are

the same size. Second, packet delivery should be synchronized among peers. All peers should send similar data packets to the aggregator within the same predetermined time window. Finally, intermediate routing nodes should randomly stagger packet delivery within a second time window. Suppose all peers send a similar packet within a set time window, $(t_s, t_s + \delta)$. If at least one routing node in each cascade staggers packet forwarding such that every packet arrives at the aggregator during a second window, $(t_s + \delta, t_r)$, an adversary will be unable to correlate outgoing traffic from any peer with incoming traffic to the aggregator.

5.2 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, we require that AnonML peers 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* [29], 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. The full details of the Anonize protocol are too long to reproduce here, so we will give a high-level summary of its operation.

To administer a survey, Anonize requires two roles, a *registration authority* and a *survey authority*. The purpose of the roles is to separate the privilege of administering surveys from that of registering users. For the purposes of AnonML, the aggregator acts as both authorities at once.

To begin, the each peer registers with the registration authority, committing to a secret key with a public signature. The registration authority issues a *master user*

token to each authorized peer in the group, which will be used to verify responses later. The survey authority then publishes a set of *survey IDs*, one for each registered user.

Each peer in the group then uses his survey ID and his secret key as inputs to a deterministic hash function, and generates a single-use token, *tok*. The peer then generates a non-interactive zero knowledge proof, π, which proves that *tok* was generated by *some* valid survey ID with *some* validated user's secret key. The NIZK also commits to a specific message, preventing. This is the verification token.

Each user only has one validated secret key and is only issued one survey ID per survey. Since the hash function is deterministic, the user can only generate one valid token per survey. Additionally, anyone in the group can verify that a user's single-use token is valid. It is computationally hard to determine which authenticated user generated a particular token, and computationally hard to forge an illegitimate token.

The aggregator issues master user tokens to each peer in the group at the start of a round of model generation. The aggregator also issues one survey ID to each peer for each feature vector that she requests. In this way, peers are prevented from sending duplicate responses, and parties outside the group are prevented from sending data to the aggregator at all.

Chapter 6

Learning with AnonML

Here we describe the structure of the data that an aggregator collects, and how that data can be used to generate a classifier.

Terminology:

- An *entity* is the abstract object which the model will attempt to classify.
- A *feature* is a value that quantifies some property of an instance of an entity.
- The *label* is the value which the model will be trained to predict. Without loss of generality, we assume a binary label.
- 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."

Features, labels, and entities are defined by the aggregator. To define a new problem, the aggregator creates an instancer and a featurizer and shares both with the group. Each peer in the group must have data of the correct format to compute the aggregator's proposed features.

6.1 Data collection

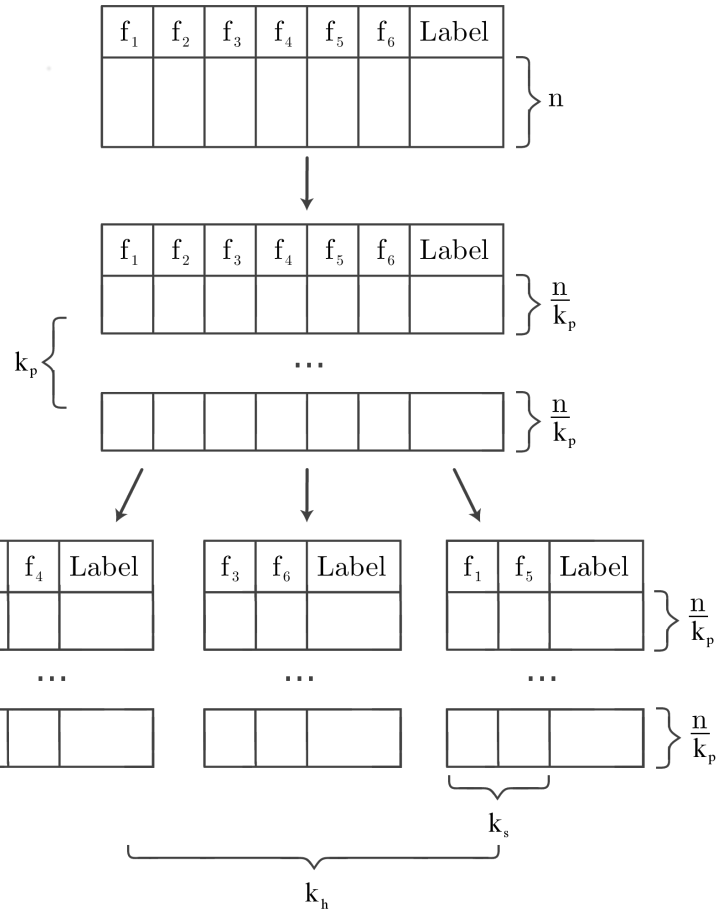


Figure 6-1: Partitioning the feature matrix. In this example, there are $m = 6$ features total and n peers. The aggregator splits the group into k_p horizontal partitions, and then requests k_h subsets of k_s features each from each peer. Each vertical partition includes the label.

Suppose that an aggregator has proposed a problem, and a group data-holding peers have agreed to take part. Each peer has processed their local dataset with the aggregator's proposed instancer and featurizer. Without loss of generality, assume each peer generates a full set of features corresponding to a single entity. In a real world scenario, each peer may possess data for one instance or many, and may have the data to generate all features for an instance or only a subset of all features.

Let n be the number of peers in the group and k_f the number of features held by each peer. 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 horizontal partitions of the dataset. The aggregator publishes the list of peers (by public key or other identifier) who belong to each partition. In order to ensure the peer partitions are not chosen maliciously, the aggregator can commit to a random seed and then use a shared or public source of randomness, e.g. the NIST randomness beacon [1], to generate random partitions.

The aggregator then requests a list of *feature subsets* from each peer partition. Feature subsets are rather self-explanatory: groups of features from the full set of k_f shared features that do not overlap. Each peer responds with a separate data packet for each feature subset requested of them, comprising their values for the requested features and their label. These data form vertical partitions.

The aggregator collects k_h feature subsets with k_s features each. 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 full matrix. 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 6-1.

6.2 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:

$$l \leftarrow g(x_{1...k_f}) \tag{6.1}$$

where m is the total number of features, $x_{1...k_f}$ are feature values, $g(.)$ is the model, and l is the binary label she wants to infer.

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 $1 \leq i \leq n_{parts}$. 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

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

$$g(x_{1...m}) = \begin{cases} 1, & \text{if } score(x_{1...m}) \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

This method is similar to the feature subspace method, first described by Ho [28]. We tested different types of partition classifiers and different cross validation metrics, and we describe the results in section 8.3.

Chapter 7

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., [16, 13], 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 $S \subseteq range(A)$:

$$\frac{P[A(D_1) \in S]}{P[A(D_2) \in S]} \leq e^{\epsilon} \tag{7.1}$$

It suffices to show that (7.1) 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

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

vectors in the group. Finally, the aggregator samples synthetic training data from this joint distribution and proceeds to build a model.

7.1 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, $c \in \{0, 1\}^4$.

7.2 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 7.3.4, 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.

7.2.1 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 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.

7.3 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

Algorithm 1 Estimate the median of a numeric feature.

Require: Π is a set of random partitions of all peers
Require: (min, max) defines range of possible values
Require: ϵ is the privacy parameter
Require: MarginOfError is a method which finds the expected error of a noisy estimate (equation 7.2)

```
function EstimateMedian(Π, min, max, ε)
    e ← (max−min)/2                                   ▷ Estimated median
    step ← (max−min)/2
    for π ∈ Π do
        n ← 0
        for p ∈ π do                                  ▷ Each peer in partition
            n ← n + p.IsGreater(e)
        end for
        err ← MarginOfError(n/|π|, ε)
        step ← step / 2
        if n/|π| - err > 0.5 then
            e ← e + step
        else if n/|π| + err < 0.5 then
            e ← e - step
        end if
    end for
    return e
end function
```

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.

7.3.1 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 $\epsilon = \ln(m \cdot \frac{p}{1-p})$ [33]. 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.

7.3.2 RAPPOR – Bitwise perturbation

RAPPOR is another set of methods for bitwise perturbation [20]. 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

$\epsilon = \ln(\frac{p^2}{(1-p)^2})$.

7.3.3 (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.

7.3.4 Error correction and 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 group's 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 6.2.

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: $\mathbf{E}[\dot{n_i}] = n_i$.

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:*

$$E\|\dot{T}_X - T_X\|_2 = \frac{\sqrt{(m-1)q(1-q) + p(1-p)}}{(p-q)\sqrt{n}} \tag{7.2}$$

Theorem 2 is proven in the appendix. We can use equation 7.2 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.

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:

$$\begin{aligned} p_{min} = &\frac{1}{\lambda^2 - 1}\Bigg(\lambda^2 + m\lambda - \lambda - \\ &\sqrt{(m-1)(\lambda^3 + \lambda) + ((m-1)^2 + 1)\lambda^2}\Bigg) \end{aligned} \tag{7.3}$$

Figure 7-1 shows the relationship between m and p_{min}, q_{min} for fixed ϵ. At $m = 2$,

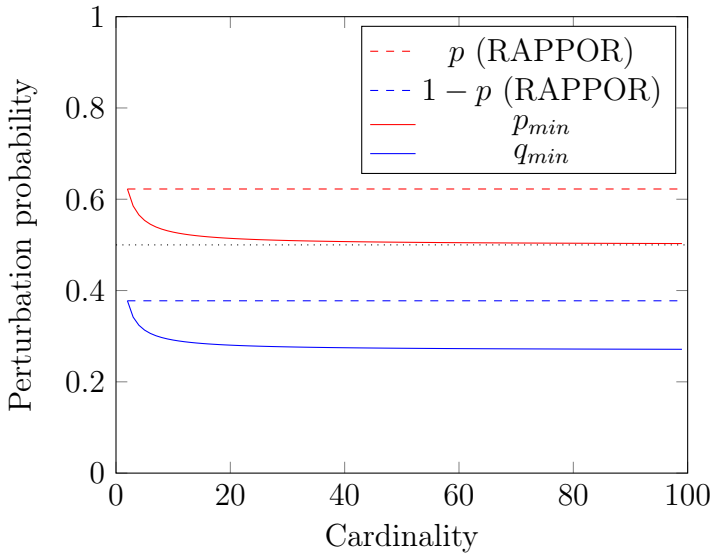


Figure 7-1: Optimal p and q as a function of variable cardinality. p_{min} and q_{min} are the values for p and q which minimize expected error for a given cardinality m. As m grows, p_{min} approaches $\frac{1}{2}$. The dashed lines show the equivalent perturbation probabilities under basic one-time RAPPOR. Here, we have fixed $\epsilon = 1$.

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 $\frac{1}{2}$.

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.

Although random response and (p, q)-perturbation are similar, they have somewhat unintuitive differences in privacy. For a given ϵ, a peer using random response is e^{ϵ} times more likely to report their real value than any other value, while a peer using bit vector perturbation is, at most, only $e^{\frac{\epsilon}{2}}$ times more likely.

Figure 7-2 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

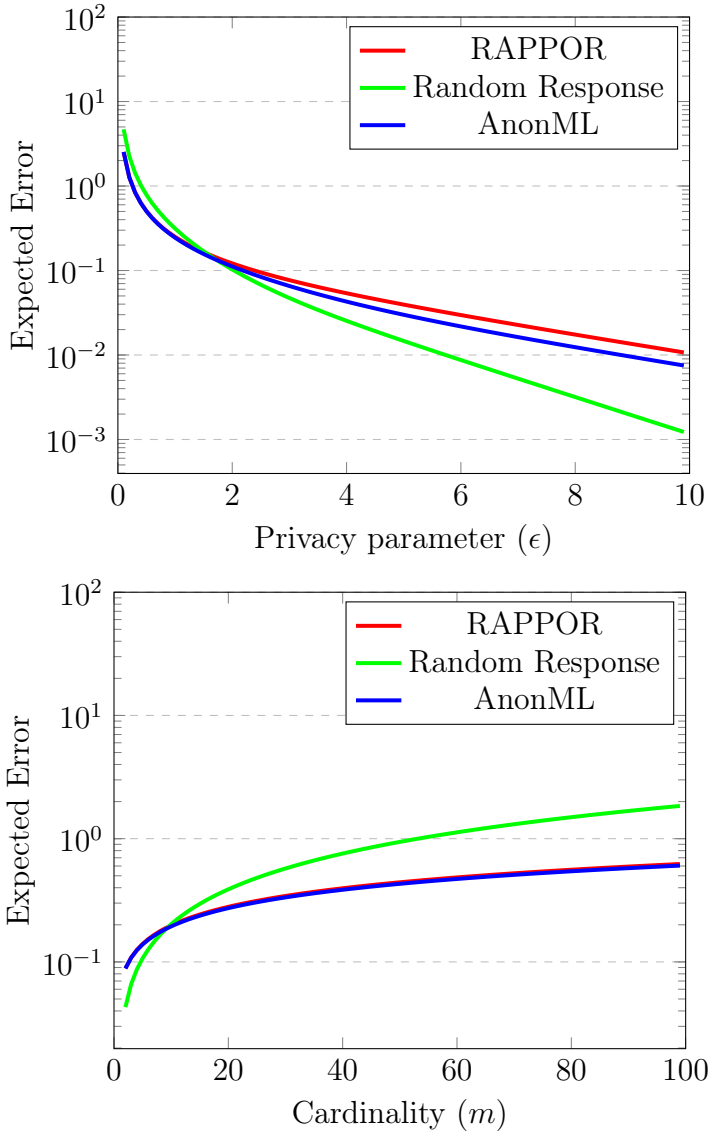


Figure 7-2: Expected type estimate error as a function of ϵ and m at $n = 10,000$ peers. On top, $m = 16$; on the bottom, $\epsilon = 1$.

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.

7.4 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 [17].

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

$$\epsilon_T = \max\left(\max_{1 \le i \le k_f} (\epsilon_f^i), \epsilon_l\right) \tag{7.4}$$

Chapter 8

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 relatively easy machine learning problems which can be solved with our system. However, further research should investigate what kinds of problems are amenable to locally-private learning.

8.1 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.

8.1.1 EdX: predicting dropout

We processed raw data from a number of popular 2012/2013 MIT EdX online classes [27] collected with the MOOCDB system [51]. The data includes 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.

8.1.2 Census: predicting salary

We used the "Adult Data Set" of census data from the UCI machine learning repository [38]. 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).

8.2 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 8-1. Our classifier performed nearly as well as other ensemble methods on some problems, but not as well on others. This establishes that our fundamental learning technique is sound. It also suggests that access to the joint distribution over the full set of features is unnecessary for some learning problems.

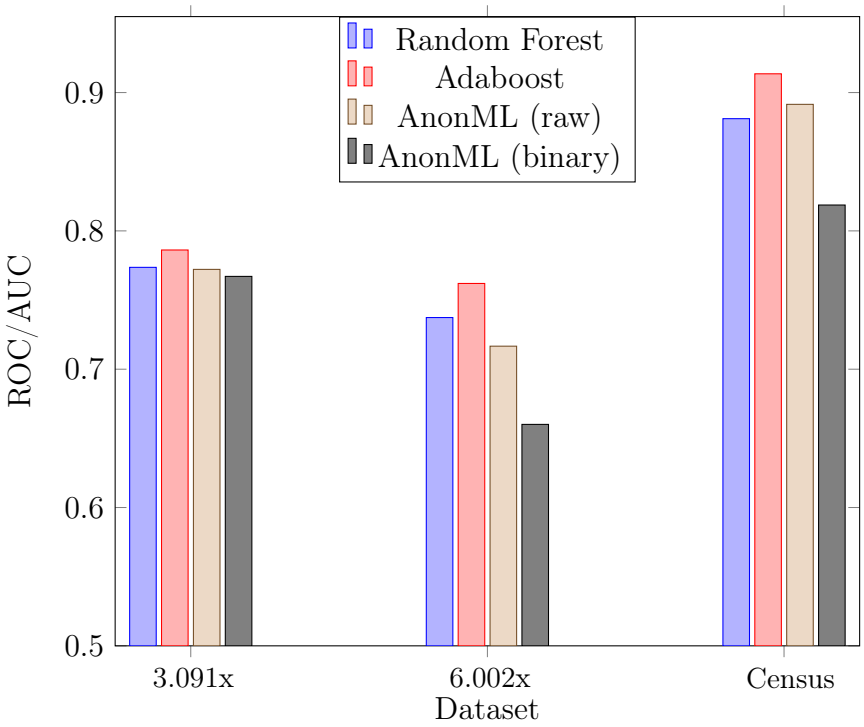


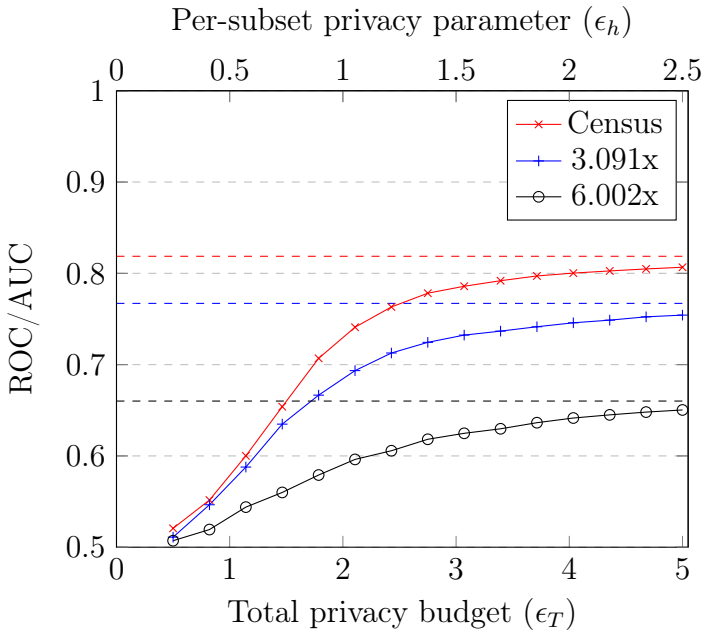
Figure 8-1: Comparison of AnonML with other ensemble learning methods on unperturbed data. These tests used five partitions, three features per subset, and as many non-overlapping feature subsets as possible.

8.2.1 Performance with privacy

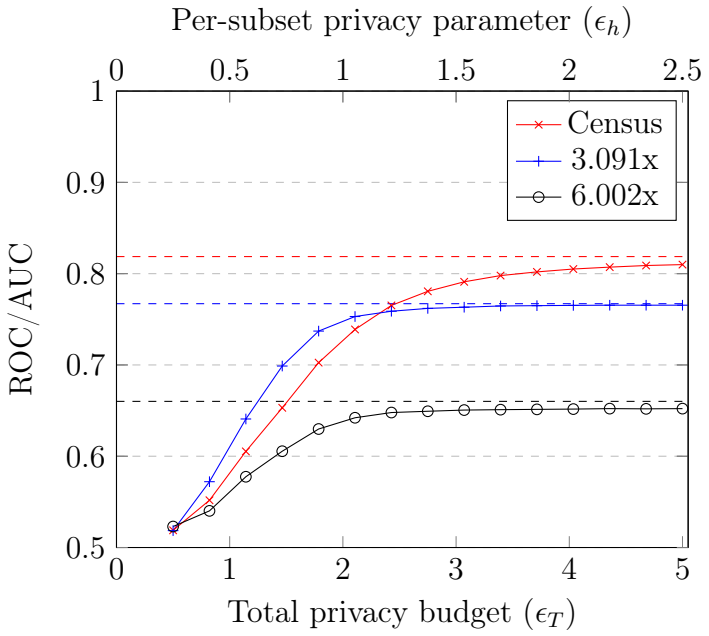
Next, we tested the problems with privacy-preserving binning and data release. For all tests, continuous features were binned into binary features using median estimation, and AUC scores are an average of 500 trials.

Figures 8-2 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 using exact median estimates, and all histograms were known precisely. In 8-2a, tests were performed with two sets of three random feature on each of five partitions, while in 8-2b, each peer was queried for two "sets" of one feature. 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. For both EdX datasets, the single-feature case was able to approach optimal



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



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

Figure 8-2: Performance as a function of ϵ_T on different datasets. All experiments are mean values from 500 trials. Note that the performance curves for the EdX datasets, 3.091x and 6.002x, tend towards their asymptotes much more quickly in 8-2b than in 8-2a. The curve for the Census dataset is nearly identical in both.

performance very quickly as a function of ϵ. However, on the census dataset, models with three features per subset outperformed single-feature models in lower privacy scenarios. Overall, we found AnonML's tradeoffs between utility and privacy to be reasonable and encouraging.

8.3 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.

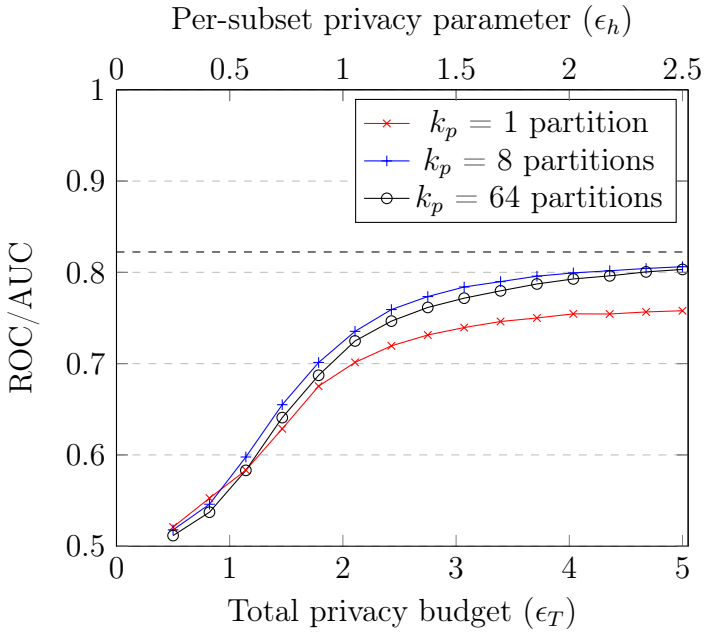
8.4 Tuning parameters

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

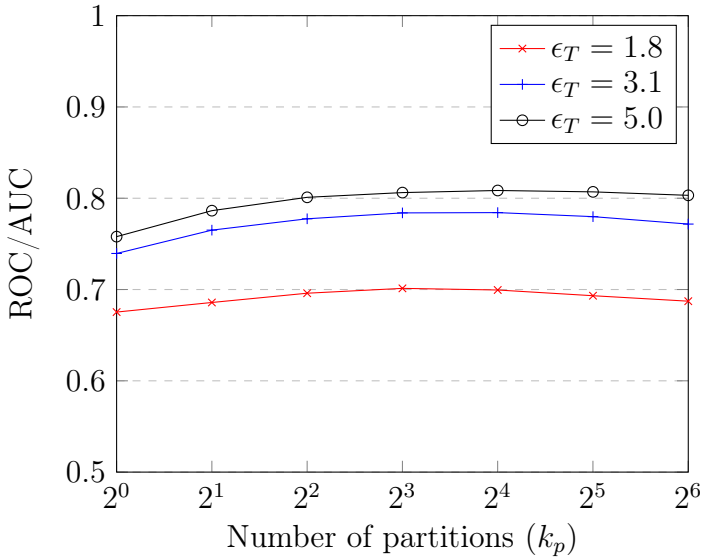
- *Partitions* (k_p): Splitting the dataset into more horizontal partitions means more classifiers can be trained and tested with a single query to each peer. The trade-off is that fewer peers' data is used to generate each classifier, which means higher expected histogram error.

- *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.

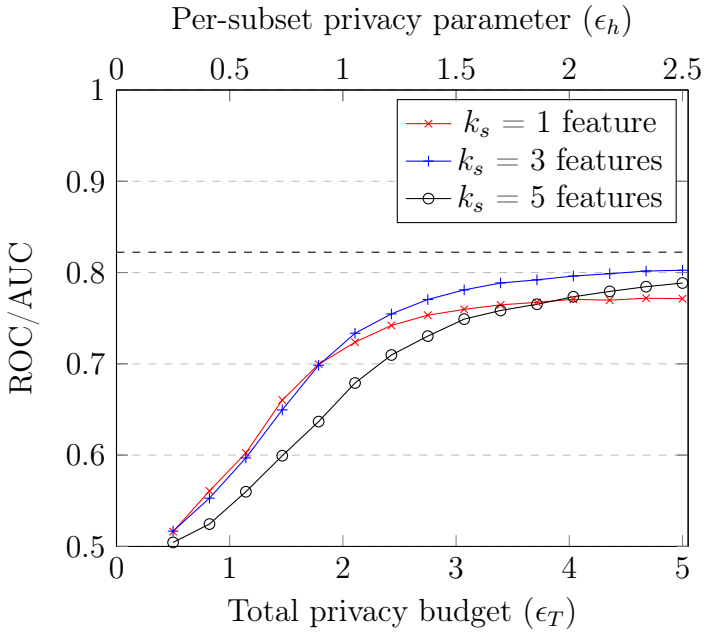


(a) Performance as a function of the privacy parameter ϵ with different partitioning factors.

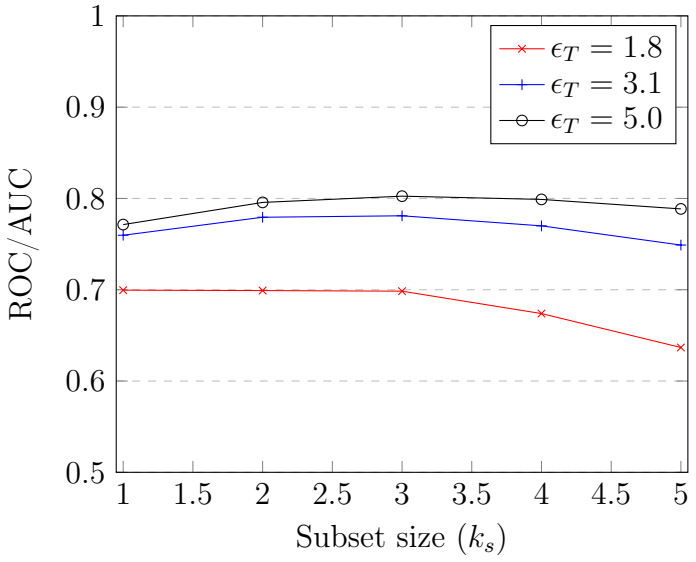


(b) Performance as a function of partition factor at different privacy budgets.

Figure 8-3: Model performance response to the number of horizontal partitions, k_p, with 3 features per subset. All ROC/AUC values are the mean of 400 trials. For this trial, peak performance was achieved between 8 and 16 partitions.

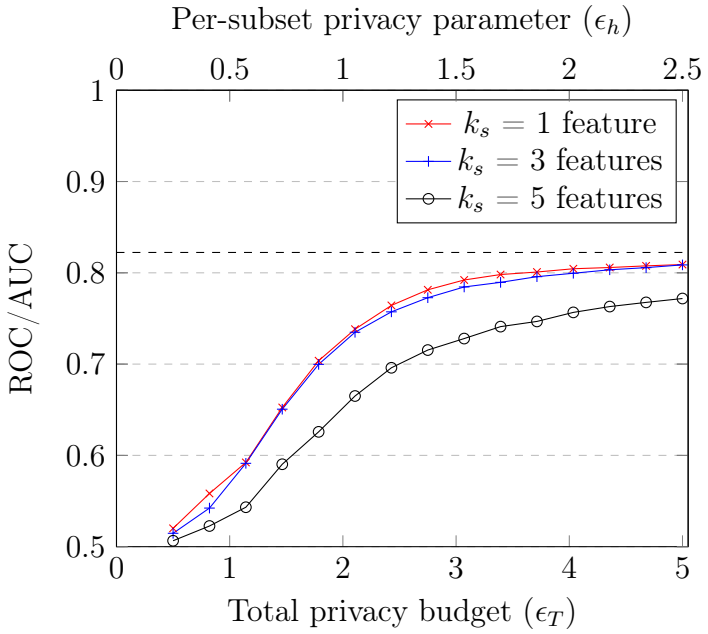


(a) Performance as a function of the privacy parameter ϵ at different subset sizes (k_s)

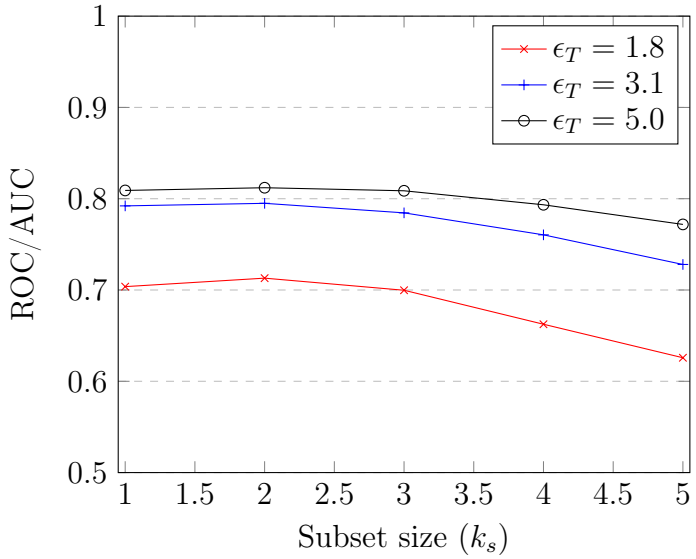


(b) Performance as a function of subset size k_s at different privacy budgets

Figure 8-4: Model performance response to the number of features per subset, k_s, with 5 horizontal partitions. All ROC/AUC values are the mean of 400 trials. For this trial, peak performance was achieved at $k_s = 3$ in the lower-privacy settings; in the high-privacy setting, k_s of 1 to 3 were nearly identically performant.



(a) Performance as a function of the privacy parameter ϵ at different subset sizes (k_s)



(b) Performance as a function of subset size k_s at different privacy budgets

Figure 8-5: Model performance response to the number of features per subset, k_s, with 20 horizontal partitions. All ROC/AUC values are the mean of 400 trials. For this trial, overall peak performance was achieved at $k_s = 2$.

Figures 8-4 and 8-5 show the results of experiments with varying subset sizes, and fig. 8-3 shows how model performance responds to different partition sizes with a fixed subset size. All experiments in figures 8-4, 8-5 and 8-3 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.

Chapter 9

Discussion

Our contributions in this thesis are twofold. First, we have presented a framework for processing personal data into a simple, universal representation for machine learning. Second, we presented and analyzed a practical, end-to-end system that enables learning on such data with practical and theoretical privacy guarantees. In our tests, we showed that AnonML can perform well on a number of real-world prediction problems while satisfying local differential privacy.

9.1 AnonML performance

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 8-1, 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, but we are optimistic overall about the possibilities of learning with extreme cardinality reduction. There

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.

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.

9.2 Differential privacy and anonymity

Our system comprises two privacy-preserving techniques. Anonymous networking, described in chapter 5, ensures that the aggregator cannot link any two feature packets to the same peer. Differentially private data sharing, described in chapter 7, 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.

9.3 Future work

Here we describe proposed applications for our contributions and suggest future work.

9.3.1 Personal machine learning applications

The framework we presented in chapter 3 will only be useful if it is adopted in the real world. We believe this can be achieved by focusing on building usable systems. Here, we speculate about what an implementation of our framework would look like at three different levels of abstraction, each of which builds on the previous level and targets users with a different level of technical ability.

We believe that the central challenge of machine learning for the next several years will be enabling intuitive problem definition. Despite great advances in automation, human decisions are still required to define entities and labels (i.e. instancers), and it appears this will remain so for the foreseeable future. In all three of these prospective applications, our goal is to make it easier for users to articulate the problems they want to solve, and allow machines to do as much of the rest of the work as possible.

1. **ML pipeline framework and library:** The most basic use of our framework would involve a small, personal database application that users could install on a phone or personal cloud computer, as well as a public repository of importer plugins for various common data sources. Users could install importers for apps and services they use, and their data would automatically be cleaned, collated and saved as an event stream in their personal database. Basic instancer and featurizer primitives would be available in an open-source library. An amateur data scientist with an idea for an ML model could write a custom instancer and featurizer, then plug the output feature matrix directly into a machine learning library like `scikit-learn`. Users could share their custom model frameworks with each other or package them as standalone applications and distribute them on application marketplaces.

2. **Graphical ML interface:** To make problem definition more accessible, our framework could be wrapped in a standalone application with a graphical interface. The application would include a library of importer plugins and allow users to enable and disable data sources with a single click or tap. The application would display a user's personal dataset as an event stream and allow

them to explore their data with basic visualization tools. The main purpose of the application would be to enable users without coding skills to create their own instancers. A graphical interface would allow a user to define an entity, perhaps by selecting subsets of their event stream with some pattern-matching functions. Then the user would define a label as a function of the data in a single instance of the entity. The app would allow the user to combine simple primitive functions, like sum, count, and min/max, with a graphical flow chart.

Once a user had defined an instancer, he could manually define features using the same function-composition interface used for labeling, or the app could automatically perform feature engineering with a library like `featuretools` [34]. Users could share their instancer and featurizer implementations with each other through an in-app marketplace.

3. **ML voice assistant:** The ultimate personal machine learning application would allow users to ask questions in plain English and respond by constructing a new model and querying it in real time. For example, a user might ask "How likely am I to pass the test tomorrow?" In response, the app would interpret the question as an instancer: "test" would be the entity, and "pass/fail" would be the boolean label. Then, the app would search the user's dataset for previous instances of tests, perform automatic feature engineering on the training examples it found, and build a model with the most predictive features it could generate. Finally, it would use the features available at the present time – perhaps things like "hours of sleep last night," "hours spent studying in past week," and "grades on past month's homework" – to make a prediction. All of this would happen behind the scenes, and the user would just hear the response: "Based on your recent habits, you have a 70% chance of passing." This application may seem far off, and there is certainly a lot of work to be done before it can become a reality. However, we believe that our framework a first step towards such a truly democratized machine learning solution.

9.3.2 AnonML applications

As a standalone application, AnonML is suitable for a variety of learning problems and scenarios where people want to perform machine learning, but do not want to trust a central authority with personal data. Here, we list a few potential use cases:

- 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.

- 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.

- 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.

9.3.3 Privacy

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 [2]. 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 [33] 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 suggest investigating higher-cardinality private binning and binning 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.

Finally, we believe that a general discussion about how to interpret the guarantees of ϵ-differential privacy is necessary. We introduced anonymous networking as a way to enhance the practical privacy of locally-private releases, but it's not clear how much this actually reduces the risk of disclosure. Our system – and others like it – would benefit from a more intuitive way of discussing privacy, or at least an alternative one. This is not an easy problem. Differential privacy has been so successful in part because other privacy frameworks have either failed to stand up to scrutiny, including k-anonymity, l-diversity, and t-closeness [50, 40, 37], or failed to achieve wide adoption, like [36]. But without a more flexible, accessible way to assess the privacy of data systems, we fear that the proliferation of useful privacy-preserving systems may suffer.

Appendix A

Proofs

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.

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

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

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

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,

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

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:*

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

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

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

□

Bibliography

[1] Nist randomness beacon (prototype implementation), 2014.

[2] Raef Bassily and Adam Smith. Local, private, efficient protocols for succinct histograms. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 127–135. ACM, 2015.

[3] Gordon Bell. A personal digital store. *Communications of the ACM*, 44(1):86–91, 2001.

[4] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, pages 115–123, 2013.

[5] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy preserving machine learning. Cryptology ePrint Archive, Report 2017/281, 2017. `http://eprint.iacr.org/2017/281`.

[6] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. 2015.

[7] Ramón Cáceres, Landon Cox, Harold Lim, Amre Shakimov, and Alexander Varshavsky. Virtual individual servers as privacy-preserving proxies for mobile devices. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, pages 37–42. ACM, 2009.

[8] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems*, pages 289–296, 2009.

[9] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[10] George Danezis. Statistical disclosure attacks. In *Security and Privacy in the Age of Uncertainty*, pages 421–426. Springer, 2003.

[11] Yves-Alexandre de Montjoye, Erez Shmueli, Samuel S Wang, and Alex Sandy Pentland. openpds: Protecting the privacy of metadata through safeanswers. *PloS one*, 9(7):e98790, 2014.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[13] Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052, pages 1–12, Venice, Italy, July 2006. Springer Verlag.

[14] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[15] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006.

[16] Cynthia Dwork and Kobbi Nissim. Privacy-preserving datamining on vertically partitioned databases. In *Annual International Cryptology Conference*, pages 528–544. Springer, 2004.

[17] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[18] Nathan Eagle and Alex Sandy Pentland. Reality mining: sensing complex social systems. *Personal and ubiquitous computing*, 10(4):255–268, 2006.

[19] Ericsson. Ericsson Mobility Report 2017. White paper, June 2017.

[20] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067. ACM, 2014.

[21] Barton Gellman and Laura Poitras. U.s., british intelligence mining data from nine u.s. internet companies in broad secret program. *The Washington Post*, June 2013.

[22] JM Gouweleeuw, Peter Kooiman, and PP de Wolf. Post randomisation for statistical disclosure control: Theory and implementation. *Journal of official Statistics*, 14(4):463, 1998.

[23] Bernard G Greenberg, Abdel-Latif A Abul-Ela, Walt R Simmons, and Daniel G Horvitz. The unrelated question randomized response model: Theoretical framework. *Journal of the American Statistical Association*, 64(326):520–539, 1969.

[24] Hamed Haddadi, Heidi Howard, Amir Chaudhry, Jon Crowcroft, Anil Madhavapeddy, and Richard Mortier. Personal data: thinking inside the box. *arXiv preprint arXiv:1501.04737*, 2015.

[25] Jihun Hamm, Paul Cao, and Mikhail Belkin. Learning privately from multiparty data. *arXiv preprint arXiv:1602.03552*, 2016.

[26] Jihun Hamm, Adam C Champion, Guoxing Chen, Mikhail Belkin, and Dong 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*, pages 11–20. IEEE, 2015.

[27] Andrew Dean Ho, Isaac Chuang, Justin Reich, Cody Austun Coleman, Jacob Whitehill, Curtis G Northcutt, Joseph Jay Williams, John D Hansen, Glenn Lopez, and Rebecca Petersen. Harvardx and mitx: Two years of open online courses fall 2012-summer 2014. *Available at SSRN 2586847*, 2015.

[28] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):832–844, August 1998.

[29] Susan Hohenberger, Steven Myers, Rafael Pass, et al. Anonize: A large-scale anonymous survey system. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 375–389. IEEE, 2014.

[30] Nicholas Hopper, Eugene Y Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2):13, 2010.

[31] Geetha Jagannathan, Krishnan Pillaipakkamnatt, and Rebecca N Wright. A practical differentially private random decision tree classifier. In *2009 IEEE International Conference on Data Mining Workshops*, pages 114–121. IEEE, 2009.

[32] Zhanglong Ji, Zachary C Lipton, and Charles Elkan. Differential privacy and machine learning: a survey and review. *arXiv preprint arXiv:1412.7584*, 2014.

[33] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Extremal mechanisms for local differential privacy. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2879–2887. Curran Associates, Inc., 2014.

[34] J. M. Kanter and B. Schreck. Featuretools. `https://www.featuretools.com/`, 2016-2017.

[35] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE, 2015.

[36] Daniel Kifer and Ashwin Machanavajjhala. A rigorous and customizable framework for privacy. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 77–88. ACM, 2012.

[37] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115. IEEE, 2007.

[38] M. Lichman. UCI machine learning repository, 2013.

[39] Bing-Rong Lin, Ye Wang, and Shantanu Rane. On the benefits of sampling in privacy preserving statistical analysis on distributed databases. *arXiv preprint arXiv:1304.4613*, 2013.

[40] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[41] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016.

[42] Min Mun, Shuai Hao, Nilesh Mishra, Katie Shilton, Jeff Burke, Deborah Estrin, Mark Hansen, and Ramesh Govindan. Personal data vaults: a locus of control for personal data streams. In *Proceedings of the 6th International COnference*, page 17. ACM, 2010.

[43] Steven J Murdoch and George Danezis. Low-cost traffic analysis of tor. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 183–195. IEEE, 2005.

[44] Manas Pathak, Shantanu Rane, and Bhiksha Raj. Multiparty differential privacy via aggregation of locally trained classifiers. In *Advances in Neural Information Processing Systems*, pages 1876–1884, 2010.

[45] Manas A Pathak and Bhiksha Raj. Large margin gaussian mixture models with differential privacy. *IEEE Transactions on Dependable and Secure Computing*, 9(4):463–469, 2012.

[46] Alex Pentland. Reality mining of mobile communications: Toward a new deal on data. *The Global Information Technology Report 2008–2009*, page 1981, 2009.

[47] Michael G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.

[48] Benjamin Schreck and Kalyan Veeramachaneni. What would a data scientist ask? automatically formulating and solving predictive problems. In *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*, pages 440–451. IEEE, 2016.

[49] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[50] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[51] Kalyan Veeramachaneni, Sherif Halawa, Franck Dernoncourt, Una-May O'Reilly, Colin Taylor, and Chuong Do. Moocdb: Developing standards and systems to support mooc data science. *arXiv preprint arXiv:1406.2015*, 2014.

[52] Stanley L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[53] Yong Ming Jeffrey Woo and Aleksandra B Slavković. Logistic regression with variables subject to post randomization method. In *International Conference on Privacy in Statistical Databases*, pages 116–130. Springer, 2012.