Pythia: Data Dependent Differentially Private Algorithm Selection

Ios Kotsogiannis
Duke University
iosk@cs.duke.edu

Ashwin Machanavajjhala
Duke University
ashwin@cs.duke.edu

Michael Hay
Colgate University
mhay@colgate.edu

Gerome Miklau
University of Massachusetts
miklau@cs.umass.edu

ABSTRACT

Differential privacy has emerged as a preferred standard for ensuring privacy in analysis tasks on sensitive datasets. Recent algorithms have allowed for significantly lower error by adapting to properties of the input data. These so-called data-dependent algorithms have different error rates for different inputs. There is now a complex and growing landscape of algorithms without a clear winner that can offer low error over all datasets. As a result, the best possible error rates are not attainable in practice, because the data curator cannot know which algorithm to select prior to actually running the algorithm.

We address this challenge by proposing a novel meta-algorithm designed to relieve the data curator of the burden of algorithm selection. It works by learning (from non-sensitive data) the association between dataset properties and the best-performing algorithm. The meta-algorithm is deployed by first testing the input for low-sensitivity properties and then using the results to select a good algorithm. The result is an end-to-end differentially private system: Pythia, which we show offers improvements over using any single algorithm alone. We empirically demonstrate the benefit of Pythia for the tasks of releasing histograms, answering 1- and 2-dimensional range queries, as well as for constructing private Naive Bayes classifiers.

1. INTRODUCTION

Differential privacy is one of the primary technical approaches of managing the significant risks of personal disclosure that arise in today's age of massive data collection and rampant data sharing. In a database where a record corresponds to an individual, the promise of differential privacy is that the participation of any single individual does not significantly alter the output of a differentially private algorithm. This is generally accepted as a persuasive privacy guarantee (for suitable settings of the privacy parameter ϵ), allowing researchers to focus on the accuracy that can be achieved under this definition.

For many data analysis tasks, the best accuracy achievable under ϵ-differential privacy on a given input dataset is not known. There are general-purpose algorithms (e.g. the Laplace Mechanism [6] and the Exponential Mechanism [18]), which can be adapted to a wide range of settings to achieve differential privacy. However, the naive application of these mechanisms nearly always results in sub-optimal error rates. For this reason, the design of novel differentially-private mechanisms has been an active and vibrant area of research [8][13][14][21]-[23][27]. Recent innovations have had dramatic results: in many application areas, new mechanisms have been developed that reduce the error by an order of magnitude or more when compared with general-purpose mechanisms and *with no sacrifice in privacy.*

While these improvements in error are absolutely essential to the success of differential privacy in the real world, they have also added significant complexity to the state-of-the-art. First, there has been a proliferation of different algorithms for popular tasks. For example, in a recent survey [9], Hay *et al.* compared 16 different algorithms for the task of answering a set of 1- or 2-dimensional range queries. Even more important is the fact that many recent algorithms are *data-dependent*, meaning that the added noise (and therefore the resulting error rates) vary between different input datasets. Of the 16 algorithms in the aforementioned study, 11 were data-dependent.

Data-dependent algorithms exploit properties of the input data to deliver lower error rates. As a side-effect, these algorithms do not have clear, analytically computable error rates (unlike simpler data-independent algorithms). When running data-dependent algorithms on a range of datasets, one may find that error is much lower *for some datasets*, but it could also be much higher than other methods on other datasets, possibly even worse than data-independent methods. The difference in error across different datasets may be large, and the "right" algorithm to use depends on a large number of factors: the number of records in the dataset, the setting of epsilon, the domain size, and various structural properties of the data itself.

As a result, the benefits of recent research advances are unavailable in realistic scenarios. Both privacy experts and non-experts alike do not know how to choose the "correct" algorithm for privately completing a task on a given input.

SIGMOD'17, May 14-19, 2017, Chicago, Illinois, USA
© 2017 ACM. ISBN 978-1-4503-4197-4/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3035918.3035945

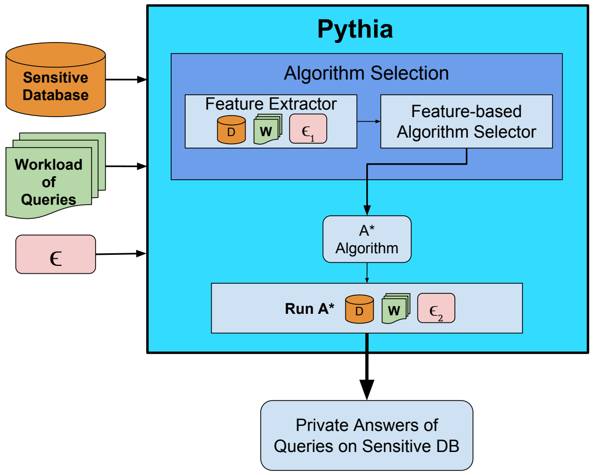


Figure 1: The Pythia meta-algorithm computes private query answers given the input data, workload, and epsilon. Internally, it models the performance of a set of algorithms, automatically selects one of them, and executes it.

Algorithm Selection

Motivated by this, we introduce the problem of differentially private *Algorithm Selection*, which informally is the problem of selecting a differentially private algorithm for a given specific input, such that the error incurred will be small.

One baseline approach to Algorithm Selection is to arbitrarily choose one differentially private algorithm (perhaps the one that appears to perform best on the inputs seen so far). We refer to this strategy as *Blind Choice*. As we will show later adopting blind choice does not guarantee an acceptable error for answering queries under differential privacy. A second baseline approach is to run all possible algorithms on the sensitive database and choose the best algorithm based on their error, we refer to this strategy as *Informed Decision*. This approach, while seemingly natural, leads to a privacy leak since checking the error of a differentially private algorithm requires access to the sensitive data.

Our approach

We propose *Pythia*, an end-to-end differentially private mechanism for achieving near-optimal error rates using a suite of available privacy algorithms. Pythia is a *meta-algorithm*, which safely performs automated Algorithm Selection and executes the selected algorithm to return a differentially private result. Using Pythia, data curators do not have to understand available algorithms, or analyze subtle properties of their input data, but can nevertheless enjoy reduced error rates that may be possible for their inputs.

Pythia works in three steps, as illustrated in Fig. 1. First it privately extracts a set of feature values from the given input. Then, using a Feature-based Algorithm Selector Pythia chooses a differentially private algorithm A^* from a collection of available algorithms. Lastly, it runs A^* on the given input. An important aspect of this approach is that Pythia does not require intimate knowledge of the algorithms from which it chooses, treating each like a black-box. This makes Pythia extensible, easily accommodating new advances from the research community as they appear.

Contributions

Our main technical contributions are as follows:

- We formalize Algorithm Selection as the problem of choosing an algorithm from a suite of differentially private algorithms $\mathcal{A}$ with the least error for performing a task on a given input dataset. We require solutions to be (a) differentially private, (b) treat each algorithm like a black box, and (c) offer competitive error on a wide range of inputs. An algorithm's competitiveness on a given input is measured using *regret*, or the ratio of its error to the minimum achievable error using any algorithm from $\mathcal{A}$.
- We implement the Feature-based Algorithm Selector using decision trees over features extracted from the private data instance, the workload and epsilon. We propose a regret based learning method to learn a decision tree that models the association between the input parameters and the optimal algorithm for that input.
- We build Pythia, a system for answering 1- and 2-dimensional range queries an important class of queries that supports among others: histogram estimation, marginals estimation, as well as building more complex systems (e.g. a Naive Bayes Classifier). It is also a class that has been intensively studied by privacy researchers. For this class of queries we identify a set of dataset features, which can be estimated privately and used to direct the selection of an algorithm.
- We comprehensively evaluate Pythia's performance on a total of 6,294 different inputs across multiple tasks and use cases (answering a workload of queries and building a Naive Bayes Classifier from sensitive data). On average, Pythia has low regret ranging between 1.27 and 2.27 (an optimal algorithm has regret 1). Our Pythia based implementation of the Naive Bayes Classifier outperforms the state of the art implementation by up to 60% in terms of misclassification rate on two real world datasets.

Our results have two important consequences. First, because our Feature-based Algorithm Selector is interpretable, the output of training phase can provide insight into the space of algorithms and when they work best. (See for example Fig. 8). Second, we believe our approach can have a significant impact on future research efforts. An extensible meta-algorithm, which can efficiently select among algorithms, shifts the focus of research from generic mechanisms (which must work well across a broad range of inputs) to mechanisms that are specialized to more narrow cases (e.g., datasets with specific properties). One might argue that algorithms have begun to specialize already; if so, then effective meta-algorithms justify this specialization and encourage further improvements.

Organization. In Section 2 we introduce preliminary definitions and notation. In Section 3 we define and motivate the problem of Algorithm Selection, and discuss the limitations of baseline approaches. We introduce Pythia in Section 4 and discuss its learning procedure in Section 5 and Section 6. In Section 7 we discuss implementation details of Pythia and in Section 8 we perform a thorough empirical evaluation of our system. Lastly, in Section 9 and Section 10 we discuss related work and recap our main results.

2. PRELIMINARIES

In this section we describe the data model, workloads, differentially private algorithms, and our error metric.

Data Model. The database D is a multiset of records, each having k attributes with discrete and ordered domains. Let $\mathcal{D}$ denote the set of all possible input databases. We describe D as a vector $\mathbf{x} \in \mathbb{N}^d$ where $d = d_1 \times \ldots \times d_k$, and d_j is the domain size of the j^{th} attribute. We denote the i^{th} value of $\mathbf{x}$ with x_i.

Given a dataset $\mathbf{x}$, we define three of its key properties: its *scale* is the total number of records: $s_{\mathbf{x}} = \|\mathbf{x}\|_1$; its *shape* is the empirical distribution of the data: $\mathbf{p}_{\mathbf{x}} = \mathbf{x}/s_{\mathbf{x}}$; and its *domain size* is the number of entries $d_{\mathbf{x}} = |\mathbf{x}|$.

Queries. A *query workload* is a set of queries defined on $\mathbf{x}$ and we use matrix notation to define it. A query workload $\mathbf{W}$ is an $m \times d$ matrix where each row represents a different linear query on $\mathbf{x}$. The answer to this workload is defined as $\mathbf{y} = \mathbf{W}\mathbf{x}$. An example of a workload is $\mathbf{P}$, an upper triangular matrix with its non-zero elements equal to 1. This workload is called the *prefix workload* and contains all prefix queries on a dataset vector – i.e., $\forall i: \ q_i = x_1 + \ldots + x_i$.

Usually a data curator is not interested in answering one specific workload, but rather a collection of similar workloads. For that reason we define a *task* T as a collection of relevant workloads. Examples of tasks include 1D range queries, 2D range queries, marginal queries, etc.

Differential Privacy. Differential privacy protects the individuals' records by enforcing that the output distribution of the algorithm changes only by a multiplicative factor in the absence or presence of a single tuple. Let $\text{nbrs}(D)$ denote the set of databases differing from D in at most one record; i.e., if $D' \in \text{nbrs}(D)$, then $|(D - D') \cup (D' - D)| = 1$.

Definition 2.1 (Differential Privacy [6]). *A randomized algorithm* A *is* ϵ*-differentially private if for any instance* D*, any* $D' \in \text{nbrs}(D)$*, and any subset of outputs* $S \subseteq$ *Range*(A)*,*

$$Pr[\mathrm{A}(D) \in S] \leq \exp(\epsilon) \times Pr[\mathrm{A}(D') \in S]$$

Theorem 1 (Sequential Composition [19]). *Let* $\mathrm{A}_1, \ldots \mathrm{A}_k$ *be algorithms, where each* A_i *satisfies* ϵ_i*-differential privacy. Then their sequential execution on the same dataset satisfies* $\sum_i \epsilon_i$*-differential privacy.*

Sequential composition allows for building complex differentially private algorithms.

Error Measurement. For a differentially private algorithm A, dataset $\mathbf{x}$, workload $\mathbf{W}$, and privacy parameter ϵ we denote the output of A as $\tilde{\mathbf{y}} = \mathrm{A}(\mathbf{W}, \mathbf{x}, \epsilon)$. Then the *error* is the L_2 distance between the vectors of the true answers and the noisy estimates: $\text{error}(\mathrm{A}, \mathbf{W}, \mathbf{x}, \epsilon) = \|\tilde{\mathbf{y}} - \mathbf{y}\|_2$

Algorithms. Differentially private algorithms can be broadly classified as *data-independent* and *data-dependent* algorithms. The error introduced by data independent algorithms is independent of the input database instance. Classic mechanisms like the Laplace mechanism [6] are data independent. For the task of answering range queries, alternative data-independent techniques can offer lower error. One example is Hb [21], which is based on hierarchical aggregation – i.e., it computes counts for both individual bins of a histogram as well as aggregate counts of hierarchical subsets of the bins.

Algorithm Name	Tasks	Prior Work
Data Independent		
Laplace	General Purpose	[6]
Hb	Range Queries	[21]
Privelet	Range Queries	[22]
Data Dependent		
Uniform	General Purpose	n/a
DAWA	Range Queries	[13]
MWEM	General Purpose	[8]
AHP	General Purpose	[27]
AGrid	2d Range Queries Queries	[14]
DPCube	2d Range Queries	[23]

Table 1: Algorithm Overview

Data-dependent algorithms usually spend a portion of the budget to learn a property of the dataset based on which they calibrate the noise added to the counts of $\mathbf{x}$. A category of data-dependent algorithms are *partition-based*; these algorithms work by learning a partitioning of $\mathbf{x}$ and add noise only to the aggregate counts of the partitions. The value of any individual cell of $\mathbf{x}$ is given by assuming uniformity on its partition. While this technique reduces the total noise added to $\mathbf{x}$, it also introduces a bias factor because of the uniformity assumption on the partitions. Hence, the overall error greatly depends on the *shape* of $\mathbf{x}$. Examples of data-dependent partitioning algorithms include DAWA, AGrid, AHP, and DPCube. Other data-dependent algorithms (like MWEM) use other data adaptive strategies.

Table 1 lists the algorithms that Pythia chooses from for answering the task of 1- and 2-dimensional range queries.

3. ALGORITHM SELECTION

In this section we formally define the problem of Algorithm Selection, describe the desiderata of potential solutions, and discuss the limitations of three baseline approaches.

Algorithm selection is performed given a collection of algorithms, a dataset, a workload, and a desired epsilon:

Definition 3.1. ***Algorithm Selection.*** *Let* $\mathbf{W}$ *be a workload of queries to be answered on database* $\mathbf{x}$ *under* ϵ*-differential privacy. Let* $\mathcal{A}$ *denote a set of differentially private algorithms that can be used to answer* $\mathbf{W}$ *on* $\mathbf{x}$*. The problem is to select an algorithm* $\mathrm{A}^* \in \mathcal{A}$ *to answer* $\mathbf{W}$ *on* $\mathbf{x}$*.*

We identify the following desiderata for Algorithm Selection solutions: (a) *differentially private*, (b) *algorithm-agnostic*, and (c) *competitive*.

Differentially Private: Algorithm Selection methods must be differentially private. If the input data is relevant to an Algorithm Selection method, any use of the input data must be included in an end-to-end guarantee of privacy.

Agnostic: Algorithm Selection methods should treat each algorithm $\mathrm{A} \in \mathcal{A}$ as a black box, i.e., solutions should only require that algorithms satisfy differential privacy and should be agnostic to the rest of the details of each algorithm. Agnostic methods are easier to deploy and are also readily extensible as research provides new algorithmic techniques.

Competitive: Algorithm Selection methods should provide an algorithm A^* that offers low error rates on a wide variety of inputs (multiple workloads, different datasets).

We measure the competitiveness of an Algorithm Selection method using a *regret* measure defined to be the ratio of the error of the selected algorithm to the least error achievable from any algorithm of $\mathcal{A}$. More precisely, given a set of differentially private algorithms $\mathcal{A}$, a workload $\mathbf{W}$, a dataset $\mathbf{x}$, and a privacy budget ϵ, we define the (relative) regret with respect to $\mathcal{A}$, of an algorithm $A \in \mathcal{A}$ as follows:

$$\text{regret}(\mathrm{A}, \mathbf{W}, \mathbf{x}, \epsilon) = \frac{\text{error}(\mathrm{A}, \mathbf{W}, \mathbf{x}, \epsilon)}{\min_{\mathrm{A} \in \mathcal{A}} \text{error}(\mathrm{A}, \mathbf{W}, \mathbf{x}, \epsilon)}$$

3.1 Baseline Approaches

As we mentioned in Section 1, two baseline approaches to Algorithm Selection are *Blind Choice* and *Informed Decision*. We also consider a third baseline, *Private Informed Decision* and explain how each of these approaches violate our desiderata.

Blind Choice. This baseline consists of simply selecting an arbitrary differentially private algorithm and using it for all inputs. It is a simple solution to Algorithm Selection and clearly differentially private. But such an approach will only be competitive if there is one algorithm that offers minimal, or near-minimal error, on all inputs. Hay *et al.* demonstrated [9] that the performance of algorithms varies significantly across different parameters of the input datasets, like domain size, shape, and scale. One of the main findings is that there is no single algorithm that dominates in all cases. Our results in Section 8.2 confirm this, showing that the regret of Blind Choice (for any one algorithm in $\mathcal{A}$) is high.

Informed Decision. In *Informed Decision* the data curator first runs all available algorithms on the given input and records the error of each algorithm. He then chooses the algorithm that performed the best. While Informed Decision solves Algorithm Selection with the lowest possible regret, it violates differential privacy since it needs to access the true answers in order to compute the error.

Theorem 2. *There exists a set of differentially private algorithms $\mathcal{A}$, an input $(\mathbf{W}, \mathbf{x}, \epsilon)$ such that if Informed Decision is used to choose $\mathrm{A}^* \in \mathcal{A}$ for the input $(\mathbf{W}, \mathbf{x}, \epsilon)$ then releasing $\mathrm{A}^*(\mathbf{W}, \mathbf{x}, \epsilon)$ violates Differential Privacy.*

See Appendix A.1 for proof.

Private Informed Decision. This strategy follows the same steps as Informed Decision except that estimation of the error of each algorithm is done in a differentially private manner. Naturally, this means the total privacy budget must be split to be used in two phases: (a) private algorithm error estimation, and (b) running the chosen algorithm. This kind of approach has already been proposed in [4], where the authors use this method to choose between differentially private machine learning models.

The main challenge with this approach is that it requires that algorithm error has low sensitivity; i.e., adding or removing a record does not significantly impact algorithm error. However, we are not aware of tight bounds on the sensitivity of error for many of the algorithms we consider in Section 8. This means that Private Informed Decision cannot be easily extended with new algorithms. So, while Private Informed Decision satisfies differential privacy and may be more competitive than Blind Choice, it violates the algorithm agnostic desideratum.

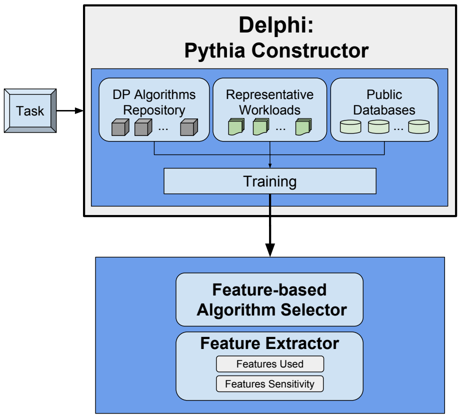


Figure 2: Delphi: Building of Pythia

4. PYTHIA OVERVIEW

Our approach to solve Algorithm Selection is called *Pythia* (see Fig. 1) and works as follows. Given an input $(\mathbf{W}, \mathbf{x}, \epsilon)$, Pythia first extracts a set of features F from the input, and perturbs each $f \in F$ by adding noise drawn from $\textsc{Laplace}(d \cdot \Delta f / \epsilon_1)$, where Δf denotes the sensitivity of f, and d is the number of sensitive features. The set of features and their sensitivities are predetermined. Next it uses a Feature-based Algorithm Selector (FAS) to select an algorithm $\mathrm{A}^\star$ from an input library of algorithms $\mathcal{A}$ based on the noisy features of the input. Finally, Pythia executes algorithm $\mathrm{A}^\star$ on $(\mathbf{W}, \mathbf{x}, \epsilon_2)$ and outputs the result. It is easy to see that this process is differentially private.

Theorem 3. *Pythia satisfies ϵ-Differential Privacy, where $\epsilon = \epsilon_1 + \epsilon_2$.*

Proof. Feature extraction satisfies ϵ_1-Differential Privacy and executing the chosen algorithm satisfies ϵ_2-Differential Privacy. The proof follows from Theorem 1. □

The key novelty of our solution is that the Feature-based Algorithm Selector is constructed using a learning based approach, called Delphi (see Fig. 2). Delphi can be thought of as a constructor to Pythia: given a user specified task T (e.g., answering 1-dimensional range queries) it utilizes a set of differentially private algorithms $\mathcal{A}_T$ that can be used to complete the task T, and a set of public datasets to output the set of features F, their sensitivities ΔF as well as the Feature-based Algorithm Selector (FAS). To learn the FAS, Delphi constructs a training set by (a) generating training inputs $(\mathbf{W}, \mathbf{x}, \epsilon)$ that span diverse datasets and workloads, and (b) measuring the empirical error of algorithms in $\mathcal{A}_T$ on training inputs. Delphi never accesses the private input database instance, but rather uses public datasets to train the FAS. This allows Delphi to (a) trivially satisfy differential privacy with $\epsilon = 0$, and (b) be run once and re-used for Algorithm Selection on different input instances.

Next we describe the design of Delphi and Pythia in detail. Section 5 describes the training procedure employed by Delphi to learn a Feature-based Algorithm Selector. Section 6 describes specific implementation choices for the task of answering range queries. Section 7 describes the Pythia algorithm as well as optimizations that help reduce error.

5. DELPHI

Delphi's main goal is to build a Feature-based Algorithm Selector that can be used by Pythia for algorithm selection. The design of Delphi is based on the following key ideas:

Data Independent: As mentioned in the previous section, we designed Delphi to work without knowledge of the actual workload $\mathbf{W}$, database instance $\mathbf{x}$, or privacy parameter ϵ that will be input to Pythia. Delphi only takes the task (e.g., answering range queries in 1D) as input. First, this saves privacy budget that can be used for extracting features and running the chosen algorithm later on. Secondly, this allows the FAS output by Delphi to be reused for many applications of the same task.

Rule Based Selector: The FAS output by Delphi uses rules to determine how features are mapped to selected algorithms. In particular we use Decision Trees [16] for algorithm selection. Decision trees can be interpreted as a set of rules that partition the space of inputs (in our case $(\mathbf{W}, \mathbf{x}, \epsilon)$ triples), and the trees Delphi outputs shed light into the classes of $(\mathbf{W}, \mathbf{x}, \epsilon)$ for which an algorithm has the least error. Moreover, prediction is done efficiently by traversing the tree from root to leaf. We discuss our decision tree implementation of FAS in Section 5.1.

Supervised Approach: Delphi constructs a training set, where each training instance is associated with features extracted from triples $(\mathbf{W}, \mathbf{x}, \epsilon)$ and the empirical error incurred by each $\mathrm{A} \in \mathcal{A}$ for answering $\mathbf{W}$ on $\mathbf{x}$ under ϵ-differential privacy. We ensure the training instances captures a diverse set of ϵ values as well as databases $\mathbf{x}$ with varying shapes, scales and domain sizes. Unlike standard supervised learning where training sets are collected, Delphi can (synthetically) generate as many or as few training examples as necessary. Training set construction is explained in Section 5.2.

Regret-based Learning: Standard decision tree learning assumes each training instance has a set of features and a label with the goal of accurately predicting the label using the features. This can be achieved by associating each training instance with the algorithm achieving the least error on the instance. However, standard decision tree algorithms view all mispredictions as equally bad. In our context this is not always the case. Recent work [9] has shown that for datasets $\mathbf{x}$ with large scales (e.g. $\geq 10^8$ records), algorithms like MWEM have a high regret (in the hundreds), while algorithms like HB and DAWA have low regrets (close to 2) for the task of 1D range queries. A misprediction that offers a competitive regret should not have the same penalty as a misprediction whose regret is in the hundreds. Towards this goal, Delphi builds a decision tree that partitions the space of $(\mathbf{W}, \mathbf{x}, \epsilon)$ triples into regions where the average regret attained by some algorithm is low. Delphi does not distinguish between algorithms with similar regrets (since these would all be good choices), and thus is able to learn a FAS that selects algorithms with lower regret than models output by standard decision tree learning. Our learning approach is described in detail in Section 5.3.

5.1 Feature-based Algorithm Selector

We use decision trees to implement the Feature-based Algorithm Selector. The FAS is a binary tree where the internal nodes of the tree are labeled with a feature and a condition of the form $f_i \leq v$. Leaves of the tree determine the outcome, which in our case is the chosen algorithm. The decision tree divides the space of inputs into non-overlapping regions – one per leaf. All inputs in the region corresponding to the leaf satisfy a conjunction of constraints on features $\ell_1 \wedge \ell_2 \wedge \ldots \wedge \ell_h$, where $\ell_i = (f_i \leq v)$ if the leaf is in the left sub-tree of an internal node with that condition, and $\ell_i = (f_i > v)$ if the leaf is in the right sub-tree.

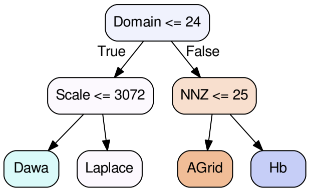


Figure 3: Example of an FAS for 2D range queries.

Given an unseen input set of features, prediction starts at the root of the tree. The condition on the internal node is checked. Traversal continues to the left child if the condition is true and to the right if the condition is false. Traversal stops at the leaf which determines the outcome. Figure 3 shows an example FAS for the task of 2-dimensional range queries. For instance, the FAS selects the LAPLACE mechanism for inputs with small domain size (≤ 24) but a large number of records (> 3072). Similarly, the FAS picks AGRID for large domain sizes (> 24) with a small number of non-zero (NNZ$\leq$ 25) counts.

5.2 Training Data

For a task T, Delphi chooses a set of differentially private algorithms $\mathcal{A}_T$ for T. Then using a library of representative workloads for the task T and a benchmark of public datasets, Delphi constructs a set of inputs $\mathcal{Z}_T$ of the form $z = (\mathbf{W}, \mathbf{x}, \epsilon)$. Details on how $\mathcal{Z}_T$ is constructed can be task dependent, and the implementation for range queries is described in Section 6.

Next, from an input $z = (\mathbf{W}, \mathbf{x}, \epsilon)$, we extract a feature vector to be used in FAS. Features can be derived from the workload $\mathbf{W}$, the input dataset $\mathbf{x}$, or the privacy budget ϵ. Let F be a set of real valued functions over input triples. For $f \in F$, we denote by f_z the value of feature f on input triple z, and by $\mathbf{f}_z$ the feature vector $[f_{1_z}, \ldots, f_{m_z}]^\intercal$. Examples of features include the number of records in the dataset (or scale), or the domain size. Section 6 describes the precise set of features used for the task of range queries. Delphi also records the performance of each algorithm $\mathrm{A} \in \mathcal{A}_T$ on each input $z \in \mathcal{Z}_T$ and creates a regret vector for each z: $\mathbf{r}_z$ that contains the regret for all algorithms in $\mathcal{A}_T$ for input z.

$$\mathbf{r}_z = \left[\mathrm{regret}_{\mathrm{rel}}(\mathrm{A}, z)\right]^\intercal_{\forall \mathrm{A} \in \mathcal{A}_T}$$

Finally, Delphi records the algorithm with the least error on z, say A^*_z, which will have a regret of 1. Thus, the final training data is a set $\mathcal{I}$ consisting of triples of the form $i = (\mathbf{f}_z, \mathrm{A}^*_z, \mathbf{r}_z)$. We use the notation $i.\mathbf{f}_z, i.\mathrm{A}^*_z, i.\mathbf{r}_z$ to refer to the different members of the training instance i.

5.3 Regret-based Learning

Decision trees are typically constructed in a top-down recursive manner by partitioning the training instances $\mathcal{I}$ into a tree structure. The root node is associated with the set of

Algorithm 1 Cart $(\mathcal{I})$ [3, 16]

```
Start at the root node, containing all training data $\mathcal{I}$.
For each feature $f$ find the value $s^*$ such that splitting on $(f, s^*)$ results in children whose weighted average of node impurity (NI) is minimized. Repeat the process for all features and choose $(f^*, s^*)$ that minimizes the weighted average of NI of the children.
Recurse on each child until the stopping criterion is met.
```

all training examples. An internal node v that is associated with a subset of training examples $V \subset \mathcal{I}$, is split into two child nodes $v_{f \leq s}$ and $v_{f>s}$ based on a condition $f \leq s$. The children are associated with $V_{f \leq s} = \{i \in V | i.f_z \leq s\}$ and $V_{f>s} = \{i \in V | i.f_z > s\}$, respectively. The split condition $f \leq s$ is chosen by computing the values f^*, s^* according to a *splitting criterion.* Recursive tree construction ends when a *stopping condition* is met. The two conditions we consider are: (a) when no split of the node v results in an improvement and (b) when the tree has reached a maximum depth $h_{\max}$. Algorithm 1 describes a standard decision tree construction algorithm called Cart. Note that the computation of f^* implies that features are automatically selected in order from the system.

The splitting criterion we use in this work chooses $(f^\star, s^\star)$ to maximize the difference between the *node impurity* (NI for short) of the parent node, and the weighted average of the node impurities of the children resulting from a split.

$$\operatorname*{argmax}_{f,s} \Big(|V| \mathrm{NI}(v) - \big(|V_{f \leq s}| \mathrm{NI}(v_{f \leq s}) + |V_{f>s}| \mathrm{NI}(v_{f>s}) \big) \Big)$$

Node impurity NI is a function that maps a set of training instances to a real number in the range $[0, 1]$ and measures the homogeneity of the training examples within a node with respect to predicted values. In our context, $\mathrm{NI}(v)$ should be low if a single algorithm achieves significantly lower error than all other algorithms on instances in V, and high if many algorithms achieve significantly lower error on subsets training examples. Decision tree construction methods differ in the implementation of NI. We next describe four alternate implementations of NI that result in four splitting criteria – best algorithm, group regret, minimum average regret and regret variance criterion. As the names suggest, the first criterion is based just on the best algorithm for each training instance (and is an adaptation of a standard splitting criterion). The other three splitting criteria are novel and are based on the regrets achieved by all algorithms in $\mathcal{A}_T$ on a training instance z. In Section 8.4 we make a quantitative comparison between all splitting criteria we consider.

Best Algorithm Criterion. This approach treats the problem of Algorithm Selection like a standard classification problem, where each training instance is associated with a label corresponding to the algorithm with the least error on it. If multiple algorithms achieve a regret of 1, one of them is chosen arbitrarily as the label. The $\mathrm{NI}(V)$ implementation we consider is the Gini impurity [16], which measures the likelihood that a randomly chosen training instance in V will be misclassified if a label was predicted based on the empirical distribution of labels in V. More specifically, for node v of the tree let $\mathbf{t}_v$ denote the empirical distribution over labels.

$$\mathbf{t}_v = \left[\frac{1}{|V|} \big| \{ i \in V | s.t.\ i.\mathrm{A}^*_z = \mathrm{A} \} \big| \right]^{\intercal}_{\forall \mathrm{A} \in \mathcal{A}_T}$$

That is, $\mathbf{t}_v[A]$ is the fraction of training instances for which A is the best algorithm. The Gini impurity on node v is defined as follows:

$$\mathrm{NI}(v) = \textsc{Gini}(v) = 1 - \mathbf{t}_v^{\intercal} \cdot \mathbf{t}_v$$

As discussed before, the best algorithm criterion views all algorithms that are not the best as equally bad. Delphi employs a *regret-based* splitting criterion discussed next, which allow to rank different splits based on their average regret. Recall that $i.\mathbf{r}_z$ denotes the vector of regrets for all algorithms $A \in \mathcal{A}_T$ on training instance z. We define the average regret vector of training instance in V as:

$$\mathbf{r}_v = \frac{1}{|V|} \sum_{i \in V} i.\mathbf{r}_z$$

Group Regret Criterion. We now present our best splitting criterion for algorithm selection, which we call the Group Regret Criterion. The key idea behind this splitting criterion is to (a) cluster algorithms with similar average regrets for a set of training instances, (b) associate training instances of a node v to the group of v with the least average regret, and (c) compute the Gini impurity criterion on the empirical distribution of the groups rather than on the empirical distribution over the labels (i.e., the best algorithm). The intuition is that choosing any algorithm from the same cluster would result in similar average regret, and thus algorithms in a cluster are indistinguishable.

Let $\mathcal{C}$ a partitioning of $\mathcal{A}_T$, then for a node v let $\mathbf{g}_{v_\mathcal{C}}$ denote the empirical distribution over the clusters of $\mathcal{C}$:

$$\mathbf{g}_{v_\mathcal{C}} = \left[\frac{1}{|V|} \big| \{ i \in V | s.t.\ i.\mathrm{A}^*_z \in C \} \big| \right]^{\intercal}_{\forall C \in \mathcal{C}}$$

That is, $\mathbf{g}_{v_\mathcal{C}}[C]$ is the fraction of training instances for which some $\mathrm{A} \in C$ is the algorithm that attains the least error.

Definition 5.1 (θ-Group Impurity)**.** *Given a node v associated with a set of training examples V and a threshold $\theta \in \mathbb{R}^+$, we define a θ-clustering of algorithms $\mathcal{A}_T$ to be a partitioning $\mathcal{C} = \{C_1, \ldots, C_k\}$ such that $\forall C \in \mathcal{C}$ and $\forall \mathrm{A}, \mathrm{A}' \in C$, $\big| \mathbf{r}_v[\mathrm{A}] - \mathbf{r}_v[\mathrm{A}'] \big| \leq \theta$. The θ-Group Impurity of v is defined as:*

$$\mathrm{NI}(v) = \mathrm{GI}_\theta(v) = \min_{\theta\text{-clusterings } \mathcal{C}} 1 - \mathbf{g}_{v_\mathcal{C}}^{\intercal} \cdot \mathbf{g}_{v_\mathcal{C}} \tag{1}$$

For a node v, the clustering $\mathcal{C}^*$ that achieves the minimum $\mathrm{GI}_\theta(v)$ is called the θ-Group Clustering (θGC).

The intuition behind θ-Group Impurity is the following: suppose A is the best algorithm for an instance z (regret is 1). Other algorithms A' that are in the same cluster in a θGC have regret at most $\theta + 1$, and hence the model should not be penalized for selecting A' instead of A. However, the FAS must be penalized for selecting algorithms that are not in the same cluster as A in the θGC.

θ-group clusterings can be efficiently computed due to the following property:

Claim 5.1. *Let $\mathcal{C}$ be a θGC for a set of algorithms in node v of the* FAS*. For any three algorithms* k, l, m *such that*

$\mathbf{r}_v[\text{K}] \leq \mathbf{r}_v[\text{L}] \leq \mathbf{r}_v[\text{M}]$*, if* K *and* L *are in the same cluster* $C \in \mathcal{C}$*, then* L *is also in the same cluster* C*.*

See Appendix A.2 for proof.

As a consequence of Claim 5.1, if the algorithms in $\mathcal{A}_T$ are sorted in increasing order of their regrets, then the θGC always corresponds to a range partitioning of the sorted list of algorithms. More precisely, if $\{\text{A}_1, \text{A}_2, \dots\}$ are such that $\mathbf{r}_v[\text{A}_i] \leq \mathbf{r}_v[\text{A}_j]$ for all $i \leq j$, then every cluster $C \in \mathcal{C}^*$ is a range $[k, m]$ such that $\forall \ell \in [k, m] : \text{A}_\ell \in C$. When the cardinality of $\mathcal{A}_T$ is low (like in our experiments) one can enumerate over all the range partitions of the sorted list of algorithms to find the θGC. In cases where $\mathcal{A}_T$ is large we can use dynamic programming (like in [12]) since the optimization criterion (Equation 1) satisfies the optimal substructure property.

Minimum Average Regret Criterion. With minimum average regret (MAR) criterion our goal is to promote splits in the tree where the resulting *average* regret of the children is less than the average regret of the parent node. This is achieved by choosing a Node-Impurity that measures the average regret of the node:

$$\text{NI}(v) = \text{MAR}(v) = \frac{\|\mathbf{r}_v\|_1}{|\mathcal{A}_T|}$$

Regret Variance Criterion. The next criterion we consider is to promote splits where the variance of the regret vectors of the children is smaller than the variance of the regret of the parent node. In this case Node-Impurity(v) is simply the variance of v:

$$\text{NI}(v) = \text{Var}(v) = \frac{1}{|\mathcal{A}_T|} \sum_{A \in \mathcal{A}_T} \left(\mathbf{r}_v[A] - \frac{\|\mathbf{r}_v\|_1}{|\mathcal{A}_T|}\right)^2$$

6. DELPHI FOR RANGE QUERIES

In this section we present how Delphi generates the set of input instances $\mathcal{Z}_T = \{(\mathbf{W}, \mathbf{x}, \epsilon)\}$ for tasks of range queries. Section 6.1 details how we generate $\mathbf{x}$'s, and Sections 6.2 and 6.3 explain how we handle workloads and epsilon values in the training phase.

6.1 Generating Datasets

Recent work [9] on the empirical evaluation of differentially private algorithms for answering range queries identified that algorithm error critically depends on three parameters of a dataset $\mathbf{x}$: *scale*, *shape*, and *domain size*. The characteristics of the input to Pythia are not know a priori, thus we must ensure that Delphi creates training data that spans a diverse range of scales, shapes, and domain sizes.

Delphi starts with a benchmark of public datasets $\mathcal{D}_{public}$. One or two dimensional datasets are constructed by choosing one or two attributes from the dataset, respectively. For each choice of attribute(s), if the domain is categorical it is made continuous using kernel density estimation. This process results in an empirical density, which we call the shape $\mathbf{p}$. We denote by P the set of all shapes constructed.

Next, the continuous domain is discretized using equi-width bins (in 1- or 2-dimensions) to get various domain sizes. We denote by K the set of domain sizes for each shape. Finally, to get a dataset of scale s, given a domain size k and shape $\mathbf{p}$, we scale up the shape $\mathbf{p}$ by s to get a total histogram count of s. The set of scales generated is denoted by S. Thus the space of all datasets corresponds to $P \times K \times S$. We denote by $\mathcal{X}$ the resulting set of datasets.

6.2 Workload Optimization

Replicating training examples for every possible workload for a given task would make training inefficient. Hence, we use the following optimization. Delphi maps each task T to a set of representative workloads $\mathcal{W}_T$, which contains workloads relevant to the task. For example if T is *"Answer range queries on* $1D$ *datasets"*, then $\mathcal{W}_T$ contains $\mathbf{I}$ and $\mathbf{P}$, the identity and prefix workloads respectively. The identity workload is effective as answering short range queries, while the prefix workload is a better choice for answering longer random range queries. Given a new task T, Delphi selects a set of differentially private algorithms $\mathcal{A}_T$, a set of representative workloads $\mathcal{W}_T$, and a privacy budget ϵ. Delphi also generates a set of input datasets $\mathcal{X}$ (as described above).

For every workload $\mathbf{W} \in \mathcal{W}_t$ Delphi generates a set of training instances $\mathcal{I}_\mathbf{W}$ by running all algorithms of $\mathcal{A}_T$, for all datasets $\mathbf{x} \in \mathcal{X}$, workload $\mathbf{W}$, and privacy budget ϵ. Then Delphi uses the Cart algorithm with training data $\mathcal{I}_\mathbf{W}$ and creates a set of FAS's: $\{\text{FAS}_\mathbf{W} \mid \forall \mathbf{W} \in \mathcal{W}_T\}$. Lastly, Delphi creates a root r connecting each $\text{FAS}_\mathbf{W}$ where edges incident to r have rules based on workload features. The resulting tree with root r is the FAS returned by Delphi.

6.3 Privacy Budget Optimization

As with workloads, Delphi could train different trees for different ϵ values. However, this would either require knowing ϵ (or a range of ϵ values) up front, or would require building an infinite number of trees. Delphi overcomes this challenge by learning a FAS for a *single* value of $\epsilon = 1.0$; i.e., all training instances have the same value of ϵ. At run-time in Pythia, if $z = (\mathbf{W}, \mathbf{x}, \epsilon')$, where $\epsilon' \neq \epsilon$, Pythia transforms the input database $\mathbf{x}$ to a different database $\mathbf{x}' = \frac{\epsilon'}{\epsilon}\mathbf{x}$, and runs algorithm selection on $z' = (\mathbf{W}, \mathbf{x}', \epsilon)$. This strategy is justified due to the scale-epsilon exchangeability property defined below.

Definition 6.1. *Scale-epsilon exchangeability [9] Let* $\mathbf{p}$ *be a shape,* $\mathbf{W}$ *a workload. For datasets* $\mathbf{x_1} = s_1\mathbf{p}$ *and* $\mathbf{x_2} = s_2\mathbf{p}$*, a differentially private algorithm* A *is scale-epsilon exchangeable if* $\text{error}(\text{A}, \mathbf{W}, \mathbf{x_1}, \epsilon_1) = \text{error}(\text{A}, \mathbf{W}, \mathbf{x_2}, \epsilon_2)$ *whenever* $\epsilon_1 s_1 = \epsilon_2 s_2$*.*

Recent work [9] showed that all state-of-the-art algorithms for answering range queries under differential privacy satisfy scale-epsilon exchangeability. We can show that under asymptotic conditions, the algorithm selected by a FAS on $(\mathbf{W}, \mathbf{x}, \epsilon')$ that is trained on input instances with privacy parameter ϵ' would be identical to algorithm selected by a FAS' on $(\mathbf{W}, \frac{\epsilon'}{\epsilon}\mathbf{x}, \epsilon)$ trained on input instances with privacy parameter ϵ.

Let $\mathcal{X}$ be $P \times K \times \mathbb{R}^+$ a set of datasets. We construct inputs $\mathcal{Z}_1 = \{(\mathbf{W}, \mathbf{x}, \epsilon_1) | \forall \mathbf{x} \in \mathcal{X}\}$ and $\mathcal{Z}_2 = \{(\mathbf{W}, \mathbf{x}, \epsilon_2) | \forall \mathbf{x} \in \mathcal{X}\}$. We construct $\mathcal{I}_1$ and $\mathcal{I}_2$ by executing epsilon-scale exchangeable algorithms $\mathcal{A}$, on $\mathcal{Z}_1$ and $\mathcal{Z}_2$ respectively. Let the Feature-based Algorithm Selectors constructed from these training datasets: $\text{FAS}_1 = \text{Cart}(\mathcal{I}_1)$, and $\text{FAS}_2 = \text{Cart}(\mathcal{I}_2)$.

Theorem 4. *Consider instances* $z_1 = (\mathbf{W}, \mathbf{x}_1, \epsilon_1)$ *and* $z_2 = (\mathbf{W}, \mathbf{x}_2, \epsilon_2)$ *such that* $\epsilon_1\mathbf{x}_1 = \epsilon_2\mathbf{x}_2$*. During prediction, let the traversal of* z_1 *on* FAS_1 *result in leaf node* v_1*, and let the traversal of* z_2 *on* FAS_2 *result in leaf node* v_2*. Then, we* *have* $\mathbf{t}_{v_1} = \mathbf{t}_{v_2}$*. Thus, the algorithm selected by* FAS_1 *on* z_1 *is the same as the algorithm selected by* FAS_2 *on* z_2*.*

Algorithm 2 $\textsc{Pythia}(\mathbf{W}, \mathbf{x}, \epsilon, \rho)$

```
1: $\epsilon_1 = \rho \cdot \epsilon$
2: $\epsilon_2 = (1 - \rho) \cdot \epsilon$
3: $d = \textsc{Nnz}(\mathbf{\Delta F})$
4: $\mathbf{f}_z = \mathrm{F}(\mathbf{W}, \mathbf{x}, \epsilon)$
5: $\tilde{\mathbf{f}}_z = \mathbf{f}_z + \mathbf{\Delta F}^T \,\mathrm{Lap}(d/\epsilon_1)$
6: $\mathrm{A}^* = \mathrm{FAS}(\tilde{\mathbf{f}}_z)$
7: $\tilde{\mathbf{y}} = \mathrm{A}^*(\mathbf{W}, \mathbf{x}, \epsilon_2)$
8: return $\tilde{\mathbf{y}}$
```

See Appendix A.3 for the proof.

7. PYTHIA

Pythia is a meta-algorithm with the same interface as a differentially private algorithm: its input is a triple $z = (\mathbf{W}, \mathbf{x}, \epsilon)$, and its output is $\mathbf{y}$, the answers of $\mathbf{W}$ on $\mathbf{x}$ under ϵ-differential privacy. Pythia works in three steps: feature extraction, algorithm selection, and algorithm execution. First, using ϵ_1 privacy budget it extracts a differentially private estimate of the features $\tilde{\mathbf{f}}_z$ from the input z. Then based on $\tilde{\mathbf{f}}_z$ it uses its FAS to choose an algorithm A^*, which runs with input $(\mathbf{W}, \mathbf{x}, \epsilon_2)$ and returns the result.

In Algorithm 2 we see an overview of Pythia. In lines 2-3 of Algorithm 2 we split the privacy budget to ϵ_1 and ϵ_2 to be used for feature extraction and algorithm execution, respectively. In line 4 we compute the number of total features that need to be privately computed ($\textsc{Nnz}$ is a function that returns the number of non-zero elements of a given vector). In line 5 we extract the true features $\mathbf{f}_z$ and in line 6 we use the $\textsc{Laplace}$ Mechanism to produce a private estimate $\tilde{\mathbf{f}}_z$. In line 7 we apply the FAS on the noisy features $\tilde{\mathbf{f}}_z$ and we get the chosen algorithm A^*. In line 8 we run A^* with input $z = (\mathbf{W}, \mathbf{x}, \epsilon_2)$ and return the answer.

Feature Extraction. Delphi provides Pythia with the set of features F of the input $z = (\mathbf{W}, \mathbf{x}, \epsilon)$. As a reminder, features extracted from the sensitive dataset $\mathbf{x}$ might potentially leak information about $\mathbf{x}$; for that reason we need to privately evaluate the values of these features on $\mathbf{x}$. To do so, we use the vector of sensitivities $\mathbf{\Delta F}$ of each individual feature. We add noise to the features in the following manner: we assign a privacy budget ϵ_1 for feature extraction, and then use the Laplace Mechanism to privately evaluate each feature's value by using a fraction ϵ_1/d for each feature, where d is the total number of *sensitive* features. This process guarantees that feature extraction satisfies ϵ_1-differential privacy.

7.1 Deployment Optimizations

The first optimization we consider is *dynamic budget allocation*, and the second is *post-processing via noisy features*. In Algorithm 3 we show Pythia utilizing both optimizations.

Dynamic Budget Allocation. The first optimization we consider is to dynamically reallocate the privacy budget between feature extraction and the execution of the selected algorithm. Recall that the feature extraction step of Pythia consumes privacy budget ϵ_1 to recover d sensitive features from $\mathbf{x}$. Then $\tilde{\mathbf{f}}_z$ is used to traverse the decision tree FAS to choose an algorithm A^*. In reality, not all features are necessarily used at the tree traversal step. For example, in Fig. 3, while there are 2 sensitive features (scale, number of non-zero counts) in the FAS, any input traversing that FAS will only utilize one sensitive feature (either scale, or $\textsc{Nnz}$). In this example we have spent $\epsilon_1/2$ to extract an extra sensitive feature that we do not use.

Algorithm 3 $\textsc{Pythia}(\mathbf{W}, \mathbf{x}, \epsilon, \rho)$ – w/ Optimizations

```
1: $\epsilon_1 = \rho \cdot \epsilon$
2: $\epsilon_2 = (1 - \rho) \cdot \epsilon$
3: $d = \textsc{Nnz}(\mathbf{\Delta F})$
4: $\mathbf{f}_z = \mathrm{F}(\mathbf{W}, \mathbf{x}, \epsilon)$
5: $\tilde{\mathbf{f}}_z = \mathbf{f}_z + \mathbf{\Delta F}^T \,\mathrm{Lap}(d/\epsilon_1)$
6: $\mathrm{A}^*, \tilde{\mathbf{f}}_z' = \mathrm{FAS}(\tilde{\mathbf{f}}_z)$
7: $\epsilon_2' = \epsilon_2 + (d - |\tilde{\mathbf{f}}_z'|)/d\epsilon_1$
8: $\tilde{\mathbf{y}} = \mathrm{A}^*(\mathbf{W}, \mathbf{x}, \epsilon_2')$
9: $\bar{\mathbf{y}} = \textsc{Optimize}(\tilde{\mathbf{y}}, \mathbf{W}, \tilde{\mathbf{f}}_z')$
10: return $\bar{\mathbf{y}}$
```

Dynamic Budget Allocation recovers the privacy budget spent on extracting features that are not utilized in the tree traversal step and instead spends it on running the chosen algorithm A^*. More specifically, given $d' < d$ sensitive features were used to traverse the tree, we update the privacy budget of the algorithm execution step to $\epsilon_2' = \epsilon_2 + (d - d')/d \cdot \epsilon_1$. Lines 7 and 8 of Algorithm 3 reflect this optimization. In the example of Fig. 3 this means that we will run the chosen algorithm with privacy budget $\epsilon_2 + \epsilon_1/2$ and thus achieve higher accuracy on the release step.

Post-Processing via Noisy Features. The second deployment optimization we propose is a post-processing technique on the noisy output $\tilde{\mathbf{y}}$ of Pythia by reusing the noisy features. The intuition behind our method is the following, the true features extracted from the dataset $\mathbf{f}_z$ impose a set of constraints on the true answers of the workload $\mathbf{y}$. We describe these constraints as a set $\mathbb{C}$, i.e., $\mathbf{y} \in \mathbb{C}$. Since $\tilde{\mathbf{y}}$ is a noisy estimate of $\mathbf{y}$, it might be the case that $\tilde{\mathbf{y}} \notin \mathbb{C}$. In the case that $\mathbb{C}$ is a convex set, we can project the noisy answer to $\mathbb{C}$ and get another estimate: $\bar{\mathbf{y}} = \mathit{Proj}_{\mathbb{C}}(\tilde{\mathbf{y}})$, where $\mathit{Proj}_{\mathcal{A}}(\mathbf{x}) \triangleq \arg\min_{\mathbf{y} \in A} \|\mathbf{x} - \mathbf{y}\|$. Doing this guarantees that the error of $\bar{\mathbf{y}}$ will be smaller than $\tilde{\mathbf{y}}$.

Theorem 5. *Let a convex set* $\mathbb{C}$*, and points* $\mathbf{y}$, $\mathbf{y}'$ *where* $\mathbf{y} \in \mathbb{C}$*. Then* $\|\mathbf{y} - \mathbf{y}^*\|_2 \leq \|\mathbf{y} - \mathbf{y}'\|_2$ *where* $\mathbf{y}^* = \textsc{Projj}_{\mathbb{C}}(\mathbf{y}')$*.*

At deployment time we do not know the true features $\mathbf{f}_z$, instead we have a noisy estimate $\tilde{\mathbf{f}}_z$. We overcome this challenge by creating a relaxed convex space $\tilde{\mathbb{C}}$ based on the noisy features and project to that. As an example, consider dataset $\mathbf{x}$ and workload $\mathbf{W} = \mathbf{I}$ the identity workload, at run-time suppose that the scale $\tilde{s}_z$ is used. Then we create the constraint $\|y\|_1 \leq \tilde{s}_z + \xi$, where $\xi \sim 1/\epsilon_1$ is a slack parameter, to account for the noise added. Lastly we project the noisy answer $\tilde{\mathbf{y}}$ to space defined by our constraint. We show experimentally significant improvements in the quality of the final answer $\tilde{\mathbf{y}}$ using this technique.

8. EXPERIMENTS

In our experimental evaluation we consider two different tasks: 1D and 2D range queries. For each task we train a *single* version of Pythia that is evaluated on all use cases for that task. We consider the standard use case of *workload answering* and we also demonstrate that Pythia can be very effective for the use case of building a *multi-stage differentially private system*, specifically a Naive Bayes classifier.

In Pythia we always set $\rho = 0.1$ to split the privacy budget for the feature extraction step. Tuning the budget allocation between the two phases is left for future work. For algorithms used by Pythia, we parameterized using default values whenever possible.

Sumary of Results. We evaluate performance on a total of 6,294 different inputs across multiple tasks and use cases. Our primary goal is to measure Pythia's ability to perform algorithm selection, which we measure using regret. Our main findings are the following:

- On average, Pythia has low regret ranging between 1.27 and 2.27. If we compare Pythia to the strategy of picking a single algorithm and using it for all inputs, we find that Pythia always has lower average regret. This is indirect evidence that Pythia is not only selecting a good algorithm, on average, it is selecting *different* algorithms on different inputs.

- For the multi-stage use case, we learn a differentially private Naive Bayes classifier similar to Cormode [5] but swap out a subroutine with Pythia. We find that this significantly reduces error (up to $\approx 60\%$). In addition, results indicate that for this use case Pythia has very little regret: it performs nearly as well as the (non-private) baseline of Informed Decision.

We also examine some aspects of the training procedure for building Pythia.

- We show that our regret-based learning technique using the group impurity measure results in lower average regret compared to the standard classification approach that uses the Gini impurity measure. The reduction is more than 30% in some cases.

- The learned trees are fairly interpretable: for example, the tree learned for the task of 2D range queries reveals that Pythia: selects DAWA when features suggest the data distribution is uniform or locally uniform, selects Laplace for small domains, and AHP for large scales.

In terms of run time, Pythia adds negligible overhead to algorithm execution: some algorithms take up to minutes for certain inputs, but Pythia runs in milliseconds. Training is somewhat costly due to the generation of training data (which takes about 5 hours). However, once the training data is generated, the training itself takes only seconds.

In Section 8.1, we describe the inputs supplied to the training procedure Delphi. For each use case, we describe the setup and results in Sections 8.2 and 8.3. Section 8.4 illustrates the interpretability of the Feature-based Algorithm Selector and the accuracy improvements due to our regret based learning procedure.

8.1 Delphi setup

Recall that Pythia is constructed by the Delphi training procedure described in Sections 5 and 6. To instantiate Delphi for a given task, we must specify the set of algorithms $\mathcal{A}_T$, the inputs $\mathcal{Z}_T$, and the features used.

Dataset Name	**Domain Size**	**Original Scale**	**Prior Work**
Task: 1D Range Queries			
ADULTFRANK	4,096	32,561	[8],[13]
HEPTH	4,096	347,414	[13]
INCOME	4,096	20,787,122	[13]
MEDCOST	4,096	9,415	[13]
NETTRACE	4,096	25,714	[1],[10],[24],[28]
SEARCHLOGS	4,096	335,889	[1],[10],[24], [28]
PATENT	4,096	27,948,226	[13]
Task: 2D Range Queries			
ADULT-2D	256 x 256	32,561	[8],[13]
BJ-TAXI-S	256 x 256	4,268,780	[11]
BJ-TAXI-E	256 x 256	4,268,780	[11]
SF-TAXI-S	256 x 256	464,040	[20]
SF-TAXI-E	256 x 256	464,041	[20]
CHECKING-2D	256 x 256	6,442,863	[9]
MD-SALARY-2D	256 x 256	70,526	[9]
LOAN-2D	256 x 256	550,559	[9]
STROKE-2D	256 x 256	19,435	[9]

Table 2: Overview of the datasets used for each task T.

Algorithms. The set of algorithms $\mathcal{A}_T$ is equal to the set of algorithms shown in Table 1, except for AGRID and DPCUBE, which were specifically designed for data with 2 or more dimensions and are therefore not considered for the task of answering range counting queries in 1D.

Inputs. We construct $\mathcal{Z}_T$, the set of triples (W, x, e), as follows. The value of ϵ is fixed to 1.0, leveraging the optimization discussed in Section 6.3. The datasets $\mathbf{x}$ are constructed using the methods described in Section 6.1, with the parameters set as follows: $\mathcal{D}_{public}$ consists of datasets for a given task as described in Table 2; the set of scales is set to $S = \{2^5, 2^6, \ldots, 2^{24}\}$; and the set of domain sizes is $K = \{128, 256, \ldots, 8192\}$ for 1D and $K = \{4 \times 4, 8 \times 8, \ldots, 128 \times 128\}$ for 2D. This yields 980 datasets for the 1D task and 1080 datasets for 2D.

The workload $\mathbf{W}$ comes from the set of representative workloads, $\mathcal{W}_T$, which varies by task. For 1D, we use 2 representative workloads: *Identity* is the set of all unit-length range queries; and *Prefix* is the set of all range queries whose left boundary is fixed at 1. For 2D, we use 4 workloads, each of consisting of 1000 random range queries, but differing in permitted lengths. The *Short* workload has queries such that their length m satisfies $m < d/16$ for domain size d, *Medium* has $d/16 \leq m < d/4$, *Long* has $m \geq d/4$ and *Mixed* consists of a random mix of the previous types.

By taking every combination of workload, dataset, and ϵ described above, we have $2 \times 980 \times 1 = 1,960$ inputs for 1D and $4 \times 1080 \times 1 = 4,320$ inputs for 2D. For each input, we run every algorithm in $\mathcal{A}_T$ on it 20 times (with different random seeds) and estimate the algorithm's error by taking the average across random trials. We use this to empirically determine the regret for each algorithm on each input.

Features. Recall that in Delphi, each input $(\mathbf{W}, \mathbf{x}, \epsilon)$ is converted into a set of features. The *dataset features* and their corresponding sensitivities are as follows:

- The *domain size*, denoted d. This feature has sensitivity zero because the domain size of neighboring datasets is always the same, i.e., the domain size of a dataset is public information.

- The *scale* is defined as $\mathrm{S}(\mathbf{x}) = \|\mathbf{x}\|_1$, and corresponds to the total number of tuples in the dataset. Since the absence or presence of any tuple in the dataset the scale can change at most by 1, we have $\Delta\,\mathrm{S} = 1$.
- The *number of non-zeros* is $\textsc{Nnz}(\mathbf{x}) = |\{x_i \in \mathbf{x}|\ x_i \neq 0\}|$. Changing any tuple in $\mathbf{x}$ alters the number of non-zeros by at most 1 so $\Delta\textsc{Nnz} = 1$.
- The *total variation* between the uniform distribution and $\mathbf{x}$ is:

  $$\mathrm{tvd_u}(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{d}\Big|x_i - u\Big|$$

  where $u = \|\mathbf{x}\|_1/|\mathbf{x}|$. We have $\Delta\,\mathrm{tvd_u} = 1 - \frac{1}{d} \leq 1$ and the proof is in Appendix B.
- The *partitionality* of $\mathbf{x}$ is denoted Part and is a function that returns minimum cost partition of $\mathbf{x}$ according to the partition score defined in Li et al. [13]. Given the analysis of Li et al. [13], it is straightforward to show that ΔPart $= 2$. Part has low values for datasets whose histograms can be summarized using a small number of counts with low error.

The *workload features* vary by task. For the task of 1D range queries, we use the binary feature "is the average query length less than $d/2$?" For 2D range queries, we use a feature that maps a workload to one of 4 types: short, medium, long, or mixed. If all queries are short then it is mapped to short, similarly for medium and long; otherwise, it is mapped to mixed. As discussed in Section 6.2, the workload feature is used at the root of the tree to map a test instance to the appropriate subtree. For 2D, workloads are mapped directly by the above function; for 1D, workloads with average query length of less than $d/2$ are mapped to the Identity subtree and the rest are mapped to the Prefix subtree. Workload features have sensitivity zero because they do not depend on the private input $\mathbf{x}$.

8.2 Use Case: Workload Answering

The first use case we consider is *workload answering*: answering a single workload of queries $\mathbf{W}$ on a dataset $\mathbf{x}$ given a privacy budget of ϵ. Our goal is to evaluate Pythia's ability to select the appropriate algorithm for a given input. We measure this ability by calculating regret: given a test input $\mathbf{z} = (\mathbf{W}, \mathbf{x}, \epsilon)$ we run each algorithm in the set $\{Pythia\} \cup \mathcal{A}_T$ on this input 20 times using different random seeds and calculate average error for each algorithm. Average error is then used to derive regret with respect to $\mathcal{A}_T$. Note that when Pythia is invoked without optimizations (see Algorithm 2), even if one assumes it chooses the best algorithm A^* for an input z, its regret will be > 1. This is because Pythia has to execute A^* for privacy budget $\epsilon_2 > \epsilon$.

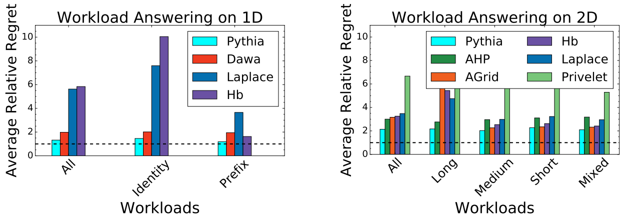
Datasets. The test inputs that we use are drawn from the set $\mathcal{Z}_T$, which was described in the previous section on training. Of course this poses an additional challenge: we should not evaluate Pythia on an input $\mathbf{z}$ that was used in training. To ensure fair evaluation, we employ a kind of *stratified* ℓ-fold cross-validation: $\mathcal{Z}_T$ is partitioned into ℓ folds such that each fold contains all of the inputs associated with a common source dataset from $\mathcal{D}_{public}$. This ensures that the training procedure does not have access to any information about the private datasets that are used in testing. The number of source datasets varies by task: as indicated in Table 2, for the 1D task, $|\mathcal{D}_{public}| = 7$ and thus $\ell = 7$; for 2D, $|\mathcal{D}_{public}| = \ell = 9$. Reported results are an aggregation across all folds.

Algorithms Compared. We compare Pythia against the baselines of Section 3.1. More specifically, we compare against Informed Decision, which always achieves a regret of 1 but is non-private and Blind Choice, which uses a single algorithm for all inputs.

In addition, the optimizations described in Section 7.1 are used: budget reallocation is used for both 1D and 2D and post-processing is used for 1D only.

Results. Fig. 4 shows the results for both tasks. Each bar in the "All" group corresponds to the average regret over all test inputs. The other bar groups report average regret over subsets of the test inputs based on workload type. The dotted line corresponds to Informed Decision with regret = 1. Algorithms whose average regret exceeds 10 were omitted, namely AHP, MWEM, Privelet, and Uniform for 1D and DAWA, MWEM, Uniform, and DPCube for 2D. Additionally, in Appendix C we provide more detailed results where we analyze the regret of different algorithms for fixed values of shape, domain size, and scale.

The results show that Pythia has lower average regret than all other techniques. In addition, Pythia's regret is generally low, ranging between 1.27 (Prefix 1D) and 2.27. (Short $2D$). It is also interesting to see that among the single algorithm strategies, the algorithm with lowest regret changes depending on the subset of inputs: for example, Hb has lower regret than DAWA for 1D Identity workload whereas the opposite is true for the 1D Prefix workload. The results provide indirect evidence that Pythia is selecting *different* algorithms depending on the input and achieving lower error than any fixed algorithm strategy.



(a) 1D Range Queries

(b) 2D Range Queriest

Figure 4: Use Case: Workload Answering

8.3 Use Case: Multi-Stage Task

In this section, we evaluate Pythia by building a multi-stage differentially private system, namely a Naive Bayes Classifier (NBC) [17]. Fitting an NBC for binary classification requires computing multiple 1D histograms of possibly heterogeneous domain sizes and shapes. We use Pythia to automatically select the most appropriate algorithm to use for each histogram. We evaluate performance using two datasets from the UCI repository [15] that, for the purposes of evaluating Pythia, represent two extreme cases: one has a small number of homogeneous histograms, the other has a larger number of more diverse histograms. This way we can see whether the benefit of algorithm selection increases with the heterogeneity of the input.

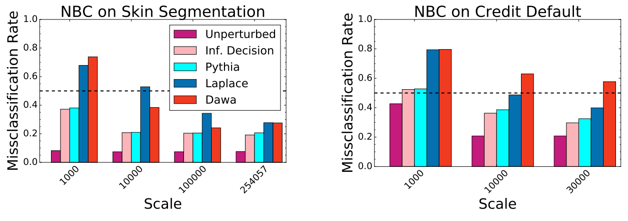
Given a k-dimensional dataset, with attributes $\{X_1, \ldots, X_k\}$ and a binary label Y, an NBC requires computing a histogram on Y and, for each attribute X_i, a histogram on X_i conditioned on the value of Y for each possible value of Y. In total, this requires estimating $2k + 1$ histograms. In addition, once the histograms are computed, they are used to fit a statistical model. We consider two different models: the Gaussian [26] and Multinomial [17] models. To compute an NBC under ϵ-differential privacy, each histogram can be computed using any differentially private algorithm provided it receives only an $\epsilon' = \epsilon/(2k+1)$ share of the privacy budget.

Datasets. The first dataset is the *Skin Segmentation* [2] dataset. Tuples in the dataset correspond to random pixel samples from face images of individuals of various race and age groups. In total there are $245K$ tuples in the dataset. Each tuple is associated with 3 features R, G, B and the labels are $\{Skin, NoSkin\}$. The second dataset we use is the *Credit Default* dataset [25] with $30K$ tuples. Tuples correspond to individuals and each tuple consists of 23 features consisting of demographic information of the individual, as well as her past credit payments and credit status. The binary label indicates whether or not the borrower defaults. Note that as a pre-processing step, we removed 7 features that were not predictive for the classification task. To get test datasets of diverse scales, we generate smaller datasets by subsampling. For Skin Segmentation, we sample three datasets of sizes $1K$, $10K$, and $100K$, and for Credit Default, two datasets of sizes $1K$ and $10K$.

Note that these datasets are used for *testing only.* Pythia is trained on different inputs, as described in Section 8.1.

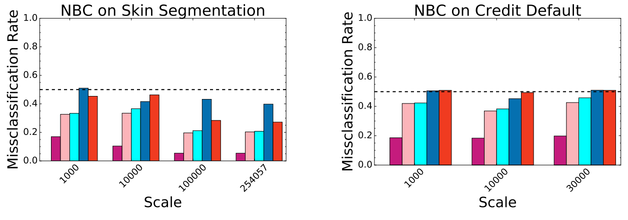
Algorithms Compared. We are interested in evaluating how the choice of algorithm for computing each histogram affects the accuracy of the resulting classifier. We consider 5 ways of computing histograms: (1) non-private unperturbed histograms, (2) non-private Informed Decision, which for each histogram selects the algorithm that achieves lowest error, (3) Pythia, (4) the Laplace mechanism, and (5) DAWA. We evaluated these approaches for both Gaussian and the Multinomial NBCs. Note that NBC with the Laplace mechanism and Multinomial model corresponds to the algorithm proposed by Cormode [5]. Accuracy is measured on a 50/50 random training/testing split. We repeat the process 10 times for different random trials and report the average misclassification rate across trials.

Results. Figs. 5 and 6 report classifier error for the Gaussian and Multinomial NBCs respectively. The results indicate that Pythia achieves lower error than any other differentially private strategy. In many cases, it achieves error that is almost as low as that of Informed Decision, which is not private. Fig. 6 also indicates that an NBC built with Pythia outperforms the existing state of the art approach (Multinomial with Laplace) of Cormode [5]. Somewhat surprisingly, Pythia is very effective even on the Skin Segmentation dataset whose histograms are fewer and homogeneous in terms of domain size. This is because Pythia almost always chooses Laplace for releasing the histogram on the label attribute (which has a domain size of 2) and DAWA for the the conditional distributions. This is close to the optimal choice of algorithms. Using Laplace or DAWA alone for all the histograms results in much higher error.



(a) Skin Segmentation Dataset (b) Credit Card Default Dataset

Figure 5: Use Case: Naive Bayes Classifier (Gaussian)



(a) Skin Segmentation Dataset (b) Credit Card Default Dataset

Figure 6: Use Case: Naive Bayes Classifier (Multinomial)

8.4 Evaluation of Training

We also examine some aspects of the training procedure for building Pythia.

Learned Tree. Fig. 8 illustrates the tree learned by Delphi for the task of 2D range queries on the Short workload. Internal nodes indicate a measured feature and leaves are labeled with the name of the algorithm that is selected for inputs that reach that leaf. The fraction shown in a leaf indicates for what fraction of those training inputs that were mapped to that leaf the selected algorithm was optimal. The tree can be fairly easily interpreted and offers insight into how Pythia chooses among algorithms. For instance, Pythia tends to select DAWA when measures indicate the data distribution is uniform (low TVD) or locally uniform (low Partitionality). It tends to select Laplace for small domains, and AHP for large scales.

Effect of Regret-based Learning. We also compare our approach of regret-based learning (Section 5.3), which uses Group Regret as its split criteria, against some alternatives including the standard Gini criterion measure, the Minimum Average Regret (MAR) and Regret Variance (VAR) criteria, all described in Section 5.3.

Fig. 7 compares these measures for the task of workload answering. The figure shows average error across the test inputs, exactly as was described in Section 8.2. It shows that the group impurity measure results in a roughly 30% reduction in average regret for 1D to the standard classification approach that uses the Gini impurity measure. For 2D, the effect is less pronounced (14%) but still the group regret criterion achieves the lowest average regret.

9. RELATED WORK

The approach proposed by Chaudhuri et al [4] can be used to implement a version of Private Informed Decision. In that work, the authors address the problem of differentially private parameter tuning in machine learning applications. In

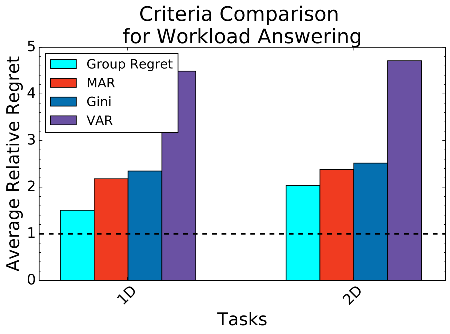


Figure 7: Criteria Comparison for Workload Answering

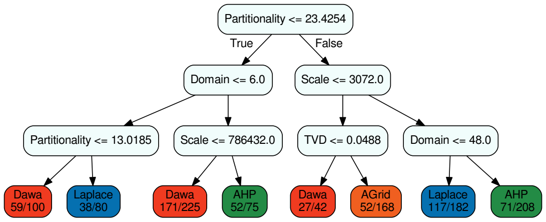


Figure 8: Tree learned by Delphi for the Short workload on 2D.

particular, a machine learning model (e.g., a classifier) is trained on a dataset $\mathcal{T}$ and its performance is validated on a hold-out dataset $\mathcal{V}$. To choose the correct parameters for the model, the process is repeated until the best parameter values, w.r.t. the performance on $\mathcal{V}$, are found. The authors assume that both $\mathcal{V}$ and $\mathcal{T}$ contain sensitive information, and try to provide an efficient differentially private parameter tuning method. First, they assume a set of models $\mathcal{H} = \{h_i, \cdots\}$, each corresponding to a different parameter value, and define stability conditions on the quality measure $q(\cdot)$ of a classifier h. Based on the stability conditions, they evaluate the quality of each $h \in \mathcal{H}$ with a privacy budget ϵ_1, and choose the best classifier and output its model with privacy budget ϵ_2. Applying the same algorithm in the context of Algorithm Selection results in a Private Informed Decision strategy. More specifically, the set of classifiers corresponds to the set $\mathcal{A}_T$ of differentially private algorithms, sets $\mathcal{T}$ and $\mathcal{V}$ are identical and correspond to the private input $\mathbf{x}$, the quality measure $q(\cdot)$ corresponds to an error metric, and lastly the stability condition is the sensitivity of the error measure. As discussed in Section 3.1, this approach is not practical because computing the sensitivity of the error for each algorithm in $\mathcal{A}_T$ is a non-trivial task. It is challenging for known algorithms and limits the extensibility of the approach as new algorithms are proposed.

In Cost Sensitive Learning [7] (CSL) the learning procedure assigns a different cost factor for correct and incorrect predictions. The motivation behind this model is that certain incorrect predictions always have greater cost than some other incorrect predictions. For example, misclassifying a movie as "PG-13" might have a greater cost than misclassifying it as "R" (since inappropriate material might be labeled as suitable for minors). Applying CSL in algorithm selection means that misclassifying an input z with some $\mathrm{A}_i \in \mathcal{A}_T$ has lower cost than misclassifying it with $\mathrm{A}_k \in \mathcal{A}_T$. As we saw earlier, this is not always the case. Algorithms have different costs depending on the subset of inputs z they are applied on. In other words, our regret-based learning method treats the cost (regret) of each label (algorithm) in a dynamic fashion, by calculating it for each subset of training instances.

10. CONCLUSIONS

In this paper we explore the problem of Algorithm Selection in the context of differential privacy, a problem motivated by recent work [9] that shows that across inputs there is no clear winner among differentially private algorithms. Simple solutions have limitations that result in either (a) high error, (b) a violation of differential privacy, or (c) impractical implementations. To address this problem, we designed Pythia, a meta-algorithm that measures features of the input to automatically select the algorithm to execute from among a suite of available ones. Pythia is an end-to-end differentially private algorithm that has demonstrably good utility across a large and diverse set of inputs. Further, Pythia is agnostic to the details of the differentially private algorithms from which it selects. This has the added benefit that it is easily extensible and can readily incorporate new algorithms developed by the research community.

Pythia's approach to algorithm selection centers around a Feature-based Algorithm Selector (FAS). The FAS is learned by Delphi, a data independent (and hence inherently private) supervised learning technique. While the training process can be time-consuming (≈ 5 hours in our experiments), a key property of our approach is that this needs to be done only once for a given class of task (e.g. 1D range queries). The learned model can then be deployed inside Pythia and used in an arbitrary number of instances of that task.

We evaluate the performance of Pythia for the two tasks of 1D and 2D range queries and multiple use cases, including computing the histograms needed to fit a Naive Bayes classifer. We compare Pythia with state of the art algorithms, as well as with non-private baselines. Overall we see that Pythia not only achieves the best average performance across inputs, but also offers small variability in its error.

The current work focuses on a relatively narrow class of tasks which we hope to expand in future work. Doing so introduces new challenges such as identifying new dataset features that are predictive for the new class of tasks. Ideally, we would like to develop methods that automatically identify and extract the features from the training instances. This may provide more insight into the conditions under which existing algorithms work well and may motivate the development of algorithms that are increasingly specialized. Those algorithms could be readily added to Pythia and selected whenever the input appears to match the conditions in which they are designed to work well.

11. ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under grants 1253327, 1408982, 1409125, 1443014, 1421325, and 1409143; and by DARPA and SPAWAR under contract N66001-15-C-4067. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon.

The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

12. REFERENCES

[1] G. Acs, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 1–10, 2012.

[2] R. Bhatt and A. Dhall. Skin segmentation dataset.

[3] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[4] K. Chaudhuri and S. A. Vinterbo. A stability-based validation procedure for differentially private machine learning. In *Advances in Neural Information Processing Systems 26*, pages 2652–2660. 2013.

[5] G. Cormode. Personal privacy vs population privacy: Learning to attack anonymization. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1253–1261, New York, NY, USA, 2011. ACM.

[6] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, pages 265–284, Berlin, Heidelberg, 2006. Springer-Verlag.

[7] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'01, pages 973–978, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[8] M. Hardt, K. Ligett, and F. Mcsherry. A simple and practical algorithm for differentially private data release. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2339–2347. Curran Associates, Inc., 2012.

[9] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, 2016.

[10] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proceedings of the VLDB Endowment*, 3(1-2):1021–1032, sep 2010.

[11] X. He, G. Cormode, Ashwin Machanavajjhala, C. M. Procopiuc, and Divesh Srivastava. DPT : Differentially Private Trajectory Synthesis Using Hierarchical Reference Systems. *Vldb*, 8(11):1154–1165, 2015.

[12] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 275–286, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[13] C. Li, M. Hay, G. Miklau, and Y. Wang. A Data- and Workload-Aware Algorithm for Range Queries Under Differential Privacy. 7(5):341–352, 2014.

[14] N. Li, W. Yang, and W. Qardaji. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, pages 757–768, Washington, DC, USA.

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

[16] W.-Y. Loh. Classification and regression trees. *Wiley*

Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 1(1):14–23, 2011.

[17] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. *AAAI-98 workshop on learning for text categorization*, 752:41–48, 1998.

[18] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, 2007.

[19] F. D. McSherry. Privacy integrated queries. *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09*, page 19, 2009.

[20] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. CRAWDAD dataset epfl/mobility (v. 2009-02-24). Downloaded from http://crawdad.org/epfl/mobility/20090224, Feb. 2009.

[21] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *Proc. VLDB Endow.*, 6(14):1954–1965, Sept. 2013.

[22] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Trans. on Knowl. and Data Eng.*, 23(8):1200–1214, Aug. 2011.

[23] Y. Xiao, J. Gardner, and L. Xiong. Dpcube: Releasing differentially private data cubes for health information. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, 2012.

[24] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. *The VLDB Journal*, 22(6):797–822, apr 2013.

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

[26] H. Zhang. The optimality of naive bayes. 2004.

[27] X. Zhang, R. Chen, J. Xu, X. Meng, and Y. Xie. *Towards Accurate Histogram Publication under Differential Privacy*, chapter 66, pages 587–595.

[28] X. Zhang, R. Chen, J. Xu, X. Meng, and Y. Xie. Towards Accurate Histogram Publication under Differential Privacy. *Proc. SIAM SDM Workshop on Data Mining for Medicine and Healthcare*, pages 587–595, 2014.

APPENDIX

A Detailed Proofs

Here we present proofs for the theorems stated in the main body of the paper.

A.1 Proof of Theorem 2

Proof. Let $\mathbf{W}$ be a query workload and let $\mathbf{x}$ and $\mathbf{y}$ be two neighboring datasets (i.e., $\|\mathbf{x}-\mathbf{y}\|_1 = 1$) that have distinct outputs on $\mathbf{W}$. That is, $\mathbf{Wx} \neq \mathbf{Wy}$. Let A_x and A_y be two algorithms such that A_x always outputs $\mathbf{Wx}$ independent of the input, and A_y always outputs $\mathbf{Wy}$ independent of the input. Since A_x, and A_y are constant functions, they trivially satisfy differential privacy for any ϵ value.

Consider the Algorithm Selection problem where $\mathcal{A} = \{\mathrm{A}_x, \mathrm{A}_y\}$. For input $x = (\mathbf{W}, \mathbf{x}, \epsilon)$ informed decision picks the algorithm that results in the least error which is A_x. For informed decision ID to satisfy ϵ-differential privacy, we want $\forall S \in \mathit{Range}(\mathrm{ID})$:

$$P(\mathrm{ID}(\mathbf{x}) \in S) \leq \exp(\epsilon) \times P(\mathrm{ID}(\mathbf{y}) \in S)$$

But we know that $P(\mathrm{ID}(\mathbf{x}) = \mathbf{Wx}) = 1$, while $P(\mathrm{ID}(\mathbf{y}) = \mathbf{Wx}) = 0$, resulting in contradiction. □

A.2 Proof of Claim 5.1

Before we prove Claim 5.1, we extend our notation to help us with the proof. Let a θ-clustering $\mathcal{C}$, then the partial sum of a cluster $C_i \in \mathcal{C}$ is: $S_i = \mathbf{g}_{v_\mathcal{C}}[C_i]^\intercal \mathbf{g}_{v_\mathcal{C}}[C_i]$, it follows that $\mathbf{g}_{v_\mathcal{C}}^\intercal \cdot \mathbf{g}_{v_\mathcal{C}} = \sum_{C_i \in \mathcal{C}} S_i$. Also let $g(\mathcal{C}) = \mathbf{g}_{v_\mathcal{C}}^\intercal \cdot \mathbf{g}_{v_\mathcal{C}}$.

Claim 5.1. We prove by contradiction. Let $\mathcal{C}^*$ a $\theta-$Group Clustering for node v and algorithms $\mathcal{A}_T$. This implies that $\mathcal{C}^* = \mathrm{argmax}_\mathcal{C}\, g(\mathcal{C})$. Assume that $\mathcal{C}^*$ does not satisfy the claim, i.e., there exist algorithms $\textsc{k}, \textsc{l}, \textsc{m} \in \mathcal{A}_T$ such that $\mathbf{r}_v[\textsc{k}] \leq \mathbf{r}_v[\textsc{l}] \leq \mathbf{r}_v[\textsc{m}]$ with $\textsc{k}, \textsc{m} \in C_i$ and $\textsc{l} \in C_j$, where $C_i, C_j \in \mathcal{C}^*$. It is obvious that $\textsc{l}$ is admissible to C_i (since it is bounded by $\textsc{k}$ and $\textsc{m}$ already in C_i.

Also note that since $\max_{[h] \in C_j} |\mathbf{r}_v[\textsc{h}] - \mathbf{r}_v[\textsc{l}]| \leq \theta$, at least one of $\textsc{k}, \textsc{m}$ is admissible to C_j.

We consider two cases, regarding the partial sums of C_i and C_j. If $S_i^* \geq S_j^*$: we construct another solution $\mathcal{C}'$ by removing $\textsc{l}$ from C_j and adding it to C_i, i.e. $\mathcal{C}' = \{C \mid \forall C \in \mathcal{C} \backslash \{C_i, C_j\}\} \cup \{C_j \backslash \{\textsc{l}\}\} \cup \{C_i \cup \{\textsc{l}\}\}$. The value of this solution is computed as follows:

$$\begin{aligned} g(\mathcal{C}') &= g(\mathcal{C}^*) - S_i^{*2} - S_j^{*2} + S_i'^2 + S_j'^2 \\ &= g(\mathcal{C}^*) - S_i^{*2} - S_j^{*2} + (S_i^* + \mathbf{t}_v[\textsc{l}])^2 + (S_j^* - \mathbf{t}_v[\textsc{l}])^2 \\ &= g(\mathcal{C}^*) + 2\mathbf{t}_v[\textsc{l}]^2 - 2\mathbf{t}_v[\textsc{l}]S_j^* + 2\mathbf{t}_v[\textsc{l}]S_i^* \\ &= g(\mathcal{C}^*) + 2\mathbf{t}_v[\textsc{l}]^2 + 2\mathbf{t}_v[\textsc{l}](S_i^* - S_j^*) \geq g(\mathcal{C}^*) \end{aligned}$$

If $S_i^* \leq S_j^*$: w.l.o.g. assume only $\textsc{k}$ is admissible to C_j, then we construct $\mathcal{C}'$ by removing $\textsc{k}$ from C_i and adding it to C_j, i.e. $\mathcal{C}' = \{C \mid \forall C \in \mathcal{C} \backslash \{C_i, C_j\}\} \cup \{C_j \cup \{\textsc{k}\}\} \cup \{C_i \backslash \{\textsc{k}\}\}$. The value of this solution is computed as follows:

$$\begin{aligned} g(\mathcal{C}') &= g(\mathcal{C}^*) - S_i^{*2} - S_j^{*2} + S_i'^2 + S_j'^2 \\ &= g(\mathcal{C}^*) - S_i^{*2} - S_j^{*2} + (S_i^* - \mathbf{t}_v[\textsc{k}])^2 + (S_j^* + \mathbf{t}_v[\textsc{k}])^2 \\ &= g(\mathcal{C}^*) + 2\mathbf{t}_v[\textsc{k}]^2 + 2\mathbf{t}_v[\textsc{k}]S_j^* - 2\mathbf{t}_v[\textsc{k}]S_i^* \\ &= g(\mathcal{C}^*) + 2\mathbf{t}_v[\textsc{k}]^2 + 2\mathbf{t}_v[\textsc{k}](S_j^* - S_i^*) \geq g(\mathcal{C}^*) \end{aligned}$$

□

A.3 Proof of Theorem 4

We prove Theorem 4 after showing the following lemma. Recall that in Section 6.3 we defined FAS_1, and FAS_2 trained on infinite training sets, with different epsilon values. We also define a *ϵ-stable bijection.* A bijection $f_{\epsilon,\epsilon'} : \mathcal{D} \rightarrow \mathcal{D}$ is a ϵ-stable bijection if for $f_{\epsilon,\epsilon'}(s \cdot \mathbf{p}) = s' \cdot \mathbf{p}$, any workload $\mathbf{W}$, and a scale/ϵ-exchangeable algorithm A:

$$\mathrm{error}(\mathrm{A}, \mathbf{W}, s\mathbf{p}, \epsilon) = \mathrm{error}(\mathrm{A}, \mathbf{W}, s'\mathbf{p}, \epsilon')$$

Lemma 6. *Let $f_{\epsilon,\epsilon'}$ an ϵ-stable bijection. We denote the nodes of* FAS_1 *at level i as $v_1^i, \ldots, v_{2^i}^i$, and similarly for* FAS_2*: $w_1^i, \ldots, w_{2^i}^i$. Then $\forall i, j$: $V_j^i = f[W_j^i]$ and $\mathbf{t}_{v_j^i} = \mathbf{t}_{w_j^i}$*

Proof. The infinite size of the training data as well as the scale/ϵ exchangeability of the algorithms in the labels guarantee that both roots of FAS_1 and FAS_2 share the same label distribution. Consider the first split of FAS_1: (v_1, v_2), we know that this split achieves the highest impurity improvement: θ_1. We argue that the first split of FAS_2 : (W_1, W_2) is such that $V_1 = f[W_1], V_2 = f[W_2]$, if it was any other case then the impurity improvement would be less in either FAS_1, or FAS_2. Because of f is an ϵ-stable bijection this also implies that $\mathbf{t}_{v_1} = \mathbf{t}_{w_1}$ and $\mathbf{t}_{v_2} = \mathbf{t}_{w_2}$. As tree construction is made top-down, we recursively apply the same argument and the proof follows.

□

Proof. [**Theorem 4**] From Lemma 6 we have that all non-leaf nodes v_j^i and w_j^i make a split on the same feature, more specifically $\forall f \in \mathcal{F} \backslash \{scale\}$: the split condition is the same, and that for $f = scale$ the split conditions are of the form (f, s) and $(f, s\epsilon_1/\epsilon_2)$ for FAS_1 and FAS_2 respectively.

This means that at traversal time, z_1 and z_2 will end up in the leaves v_j^i and w_j^i of FAS_1, and FAS_2. The proof follows from Lemma 6. □

B Feature Sensitivity Estimation

Let $\mathbf{x}, \mathbf{x}'$ neighboring datasets with $\|\mathbf{x}\|_1 = s$, $|\mathbf{x}| = |\mathbf{x}'| = d$, and w.l.o.g $x_1' = x_1 + 1$, and $x_j = x_j' \quad \forall j \in \{2, \ldots, d\}$. Then:

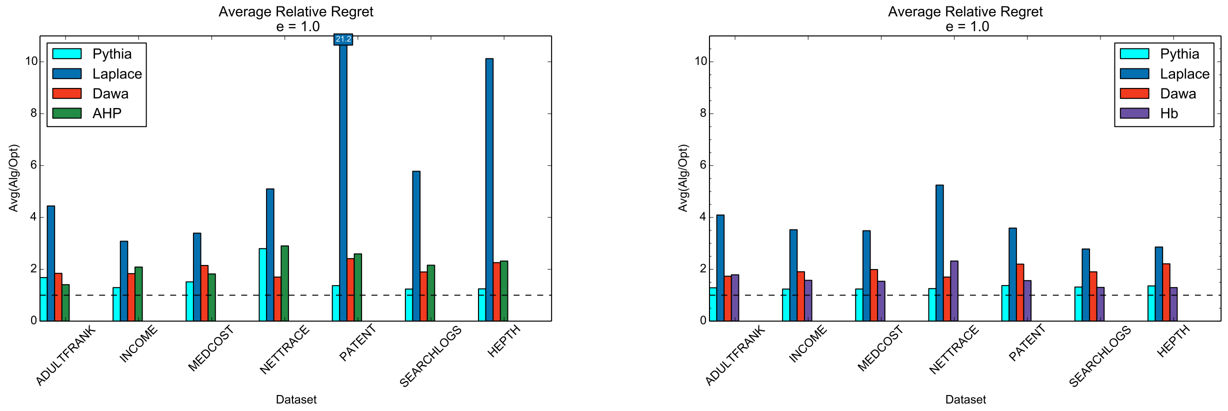
$$
\begin{aligned}
\text{tvd}_\text{u}(\mathbf{x}') &= \frac{1}{2}\sum_{i=1}^{d} \left|x_i' - (s+1)/d\right| \\
&= \frac{1}{2}\Bigg(\left|x_1' - (s+1)/d\right| + \sum_{i=2}^{d} \left|x_i - (s+1)/d\right|\Bigg) \\
&\leq \frac{1}{2}\Bigg(\left|x_1 - s/d\right| + 1 - 1/d + \sum_{i=2}^{d} \big(\left|x_i - s/d\right| + 1/d\big)\Bigg) \\
&= \frac{1}{2}\Bigg(\left|x_1 - s/d\right| + \sum_{i=2}^{d} \big(\left|x_i - s/d\right|\big) + 1 - 1/d + (d-1)/d\Bigg) \\
&= \frac{1}{2}\Bigg(\sum_{i=1}^{d} \left|x_i - s/d\right| + 2 - 2/d\Bigg) \\
&= \text{tvd}_\text{u}(\mathbf{x}) + 1 - 1/d
\end{aligned}
$$

C Further Experiments

Here we present additional experimental results that complement our analysis in Section 8.2. We further analyze the error incurred by algorithms for the task of workload answering for fixed values of shape, domain size, and scale.

In Fig. 9 we plot the average regret of each algorithm across different datasets, for the 1D tasks. Fig. 9a and 9b correspond to the identity and the prefix workload respectively. For the identity workload, Pythia has the lowest average regret amongst 5 data-sets and both AHP and DAWA have the lowest in 1 dataset. For the prefix workload, Pythia has the lowest average regret in 5 datasets and Hb has the lowest regret in 2 datasets. The key point in this case is that when Pythia is not the best it is the second-best, which means that across datasets it has consistently good error.

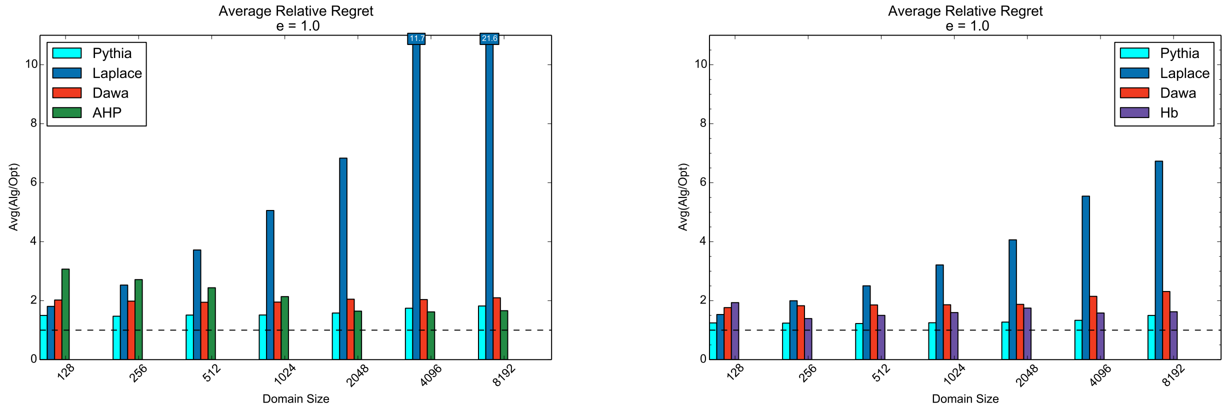
In Figures 10 and 11 we see the corresponding plots when we fix the domain size and scale respectively, and then average out the regret measure. Again we see similar trends, with Pythia being a consistently good choice.



(a) Identity workload

(b) Prefix workload

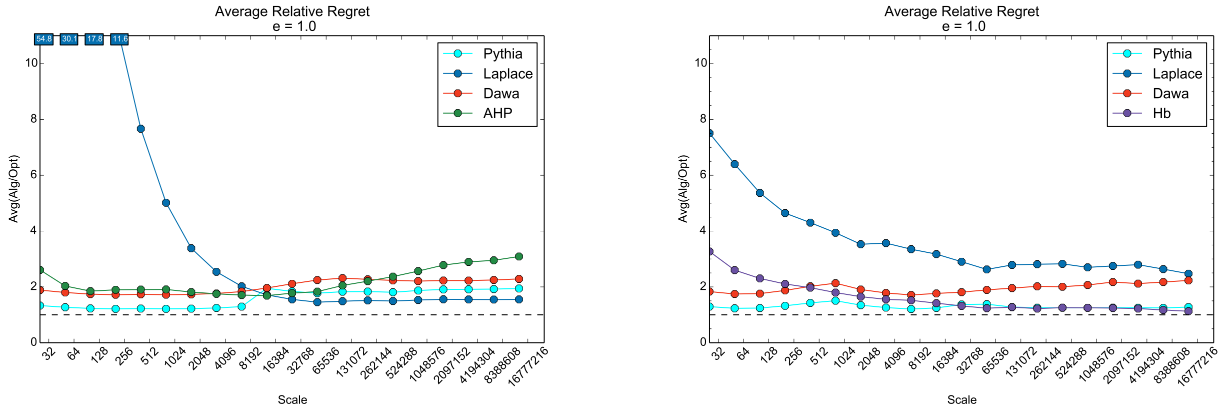
Figure 9: Average Regret vs Shape



(a) Identity workload

(b) Prefix workload

Figure 10: Average Regret vs Domain Size



(a) Identity workload

(b) Prefix workload

Figure 11: Average Regret vs Scale