Automatic, Fine-Grained Algorithmic Choice for Differential Privacy

Jacob Imola
School of Computer Science, Carnegie Mellon University

Jean Yang
School of Computer Science, Carnegie Mellon University

March 7, 2018

1 Introduction

The rapid technological increase in data collection, speed, and storage has brought about revolutionary insights and ideas and will continue to do so. However, with huge amounts of private data comes the concern of preventing data from ending up in the wrong hands. In order to prevent data leakage, we must lay a strong privacy foundation and give data analysts the tools for them to implement privacy efficiently and correctly.

Consider a healthcare database with records like patient weight, age, some genetic information, or whether they are HIV positive. Granting access rights, or policies, to just patients and their doctors protects as much privacy as possible, and developing tools for verifying information flow policies is an interesting question in its own right. However, sometimes it's okay to release a few general statistics about a database so that an analyst can find risk factors for people who have HIV. On the other side of the spectrum, publicly releasing the entire database doesn't protect privacy at all, yet it would be an analyst's dream. In order to appease both data analysts and patients, a middle ground area must be chosen where a blurry snapshot of the database is released, comprehensive enough so that meaningful conclusions may be drawn yet blurry enough so that individuals are mostly protected. We call this the *privacy-accuracy* tradeoff. We can always sacrifice one for the other before we disclose our database snapshot. However, after we disclose, it is impossible to take any privacy back, so we have to be absolutely sure that privacy guarantee will not fail under any attack. The most promising method for doing such a disclosure is Differential Privacy [2].

Differential Privacy (DP) is considered to be the gold-standard of privacy and has been researched intensely since its conception in 2005. Its goal is to provide guarantees on what can be done with the information being released from a dataset making minimal assumptions about an attacker's powers. Previous attempts were not mathematically rigorous and did not reliably prevent attacks. Notably, before DP, researchers were able to reidentify users in a Netflix dataset given an auxiliary dataset from IMDB and form a generalized attack against the state-of-the-art privacy algorithms of the time [16]. The strong privacy guarantee of DP, on the other hand, has a rigorous mathematical foundation that makes it impervious to the post-processing attacks that exploited the Netflix dataset, and more recently, AirBnB and Instagram. Differential Privacy has certainly stood the test of time as a sturdy way to protect privacy.

However, just building a suite of DP algorithms is not satisfactory. Privacy adds a new layer of complexity to the data analyst's job because the performance of algorithms may vary wildly depending on the database or on other parameters, complexity that an analyst may not fully understand. Most data analysts are not Differential Privacy experts. This leads to the central problem of this work:

Problem: How can we help data analysts dispatch Differential Privacy algorithms in a way that helps them understand the privacy and accuracy tradeoff space?

This problem is noted by the authors of DPComp [6], who comment that currently, "the practitioner is lost and incapable of deploying the right algorithm". DPComp allows the analyst to visualize the performances of different histogram querying algorithms on certain datasets. An example dataset is shown in Figure 1. We feel this is a step in the right direction, but there are two main drawbacks: their visualization tools are limited to 2D histogram algorithms, and they don't help the programmer enough on his actual dataset, instead using publicly available knowledge. To address both of these issues, we take a programming-language-based approach. The approach is to move the manual exploration of algorithms that an analyst might do with DPComp into the runtime of our programming language, which we call `Jostle`. This will preserve the insights that an analyst may gleam from DPComp (since the algorithmic choice will be publicly available

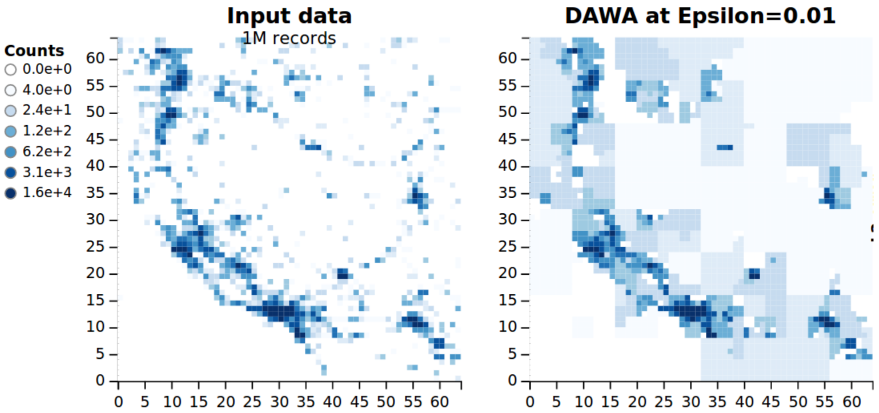

Figure 1: A screenshot of a DPComp graph depicting the original dataset and the noise that a DP algorthm, DAWA [11], adds at $\epsilon = 0.01$. This would help an analyst decide whether to apply DAWA on his own dataset.

to him) and will confer the advantages of a programming language:

- **Abstraction** Creating an abstraction for algorithmic choice in a programming language allows for the suppression of complicated privacy machinery, allowing analysts to view the choice as a black box. In fact, a programmer could view all of DP as a black box and still use `Jostle` by simply trying many DP black boxes and knowing that the best algorithm will be used [1]. Programmers write code faster with fewer bugs when they can think about complicated ideas as black boxes.

- **Generalization** As previously noted, it is infeasible to conduct experiments like DPComp on all use cases that an analyst may encounter. Writing a sophisticated programming language that automatically makes this choice would eliminate the need for this while still allowing for high-grade code performing as if the experiments had been conducted.

The most central idea in `Jostle` is the `NoisyIf` statement, illustrated in Figure 2. A programmer will use `NoisyIf` when they are unsure of which choice to make at a certain point in their code. If they have certain beliefs that may aid `Jostle` in making the choice, then they may specify an *advice* function (denoted by `g` in Figure 2). Jostle will combine this advice

[1] We could call these programmers privacy-agnostic.

```
def f(D):
    NoisyIf(g D):
        do A
    else:
        do B
```

Figure 2: Example `NoisyIf` statement.

along with previous executions of this particular `NoisyIf` to make the best decision.

We will begin with a background section on DP and related work. Then we will introduce decision trees and highlight the existence of Problem with them. Thirdly, we will solve the decision tree problem with different `NoisyIf` statements, and finally, we will illustrate how `NoisyIf` statements can be generalized to make `Jostle`.

2 Background

2.1 Differential Privacy

DP is motivated by the example of a subject being asked to participate in a study involving an algorithm $\mathcal{A}$ being run on a database D. If $\mathcal{A}(D)$ changes as a result of the subject participating, then this poses a privacy concern. An attacker may be able to infer something about the participant's input. This means that non-trivial deterministic algorithms are already unacceptable; their output may change depending on just a single addition to D. The strength ϵ of a DP guarantee is the maximum factor that the randomized output of $\mathcal{A}$ changes for two databases D and D' which differ in one row. Mathematically, we are saying:

Definition 1. *$\mathcal{A}$ satisfies ϵ-DP if for all D and D' such that $|D - D'|_1 = 1$ and for all o in the range of $\mathcal{A}$,*

$$\Pr\left(\mathcal{A}(D) = o\right) \leq e^{\epsilon} \Pr\left(\mathcal{A}(D') = o\right)$$

There is also a weaker definition:

Definition 2. *$\mathcal{A}$ satisfies (ϵ, δ)-DP if for all D and D' such that $|D - D'|_1 = 1$ and for all o in the range of $\mathcal{A}$,*

$$\Pr\left(\mathcal{A}(D) = o\right) \leq e^{\epsilon} \Pr\left(\mathcal{A}(D') = o\right) + \delta$$

For much of this paper, we will focus on ϵ-DP, but it is worth knowing the more general case so we can import the well-known privacy theorems in their most general form. Below is perhaps the most important result, and its proof comes cleanly from the definition of DP.

Theorem 1. *(Post-Processing [2]) If $\mathcal{A}$ satisfies (ϵ, δ)-DP, and F is any function that takes the output of $\mathcal{A}$ as input, then $F(\mathcal{A})$ satisfies (ϵ, δ)-DP.*

This theorem is the reason why DP is such a useful guarantee. Data analysts can be sure that once they run their algorithm $\mathcal{A}$ and release its output, then the DP guarantee gets no weaker *no matter what an adversary does with the data.* This prevents the headaches where an analyst realizes retroactively that the data he released can be combined in some way to reveal much more information than was intended. Another useful, intuitive result is:

Theorem 2. *(Composition [2]) Given algorithms M_1 and M_2 satisfying ϵ_1 and ϵ_2 DP, respectively, along with a database D, the algorithm $M = (M_1(D), M_2(D))$ has $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$ DP.*

Composition is like the union bound from probability; it's convenient to apply but often is a pessimistic bound, as we will see later. Because of composition, we often refer to ϵ as a privacy budget—if we string together many private computations, it's like we spend some of our budget on them, and our total budget is ϵ.

Finally, we can talk about the disjointness of our queries—if D is split into disjoint parts, before mechanisms are applied to it, then out of all its possible neighbors, only one of the parts will be different. Thus, only the worst mechanism will affect the guarantee:

Theorem 3. *(Disjointness [2]) Given disjoint subsets D_1, D_2 of D with two mechanisms M_1 and M_2 providing (ϵ_1, δ_1) and (ϵ_2, δ_2)-privacy, then $((M_1(D_1), M_2(D_2))$ satisfies $(\max\{\epsilon_1, \epsilon_2\}, \max\{\delta_1, \delta_2\})$-DP.*

So, what's a simple example of a DP algorithm? Suppose each row of our database D is 0 or 1, so $D \in \{0,1\}^n$, and that we are trying to release the sum of the elements of D. If this sum is S, then all neighboring databases D' have sum S or $S+1$. We can add noise to S so that it looks very similar in distribution to $S+1$. The distribution we are looking for is the Laplace distribution:

Definition 3. *The Laplace(λ) distribution has probability mass function $f(x) = \frac{1}{2\lambda} e^{-|x|/\lambda}$.*

This distribution fits perfectly with the definition of DP because of the exponentials. If X, Y are i.i.d. from Laplace $\left(\frac{1}{\epsilon}\right)$, then it is straightforward

to show that the distributions of $S+X$ and $S+1+X$ satisfy $(\epsilon, 0)$ DP. To generalize this statement, we will use the following definition:
Definition 4. *(Sensitivity) A function f is Δ-sensitive if for all x, y such that $|x-y|_1 = 1$, we have*

$$|f(x) - f(y)| \leq \Delta$$

This can equivalently be rephrased as

$$\max_{|x-y|_1=1} |f(x) - f(y)| = \Delta$$

We will denote the sensitivity of f by $\Delta(f)$.

This gives us the following mechanism:

```
Algorithm 1: Laplace Mechanism
Input  : $D$, a database; $f$, a function $\mathcal{D} \to \mathbb{R}^n$; and $\epsilon$
Output: An estimate for $f(D)$ satisfying $\epsilon$-DP.
$X$, a vector of $n$ i.i.d. variables drawn from Laplace $\left(\frac{\Delta(f)}{\epsilon}\right)$;
return X+f(D)
```

Theorem 4. *The Laplace Mechanism 1 satisfies $(\epsilon, 0)$-DP [2].*

For the counting or histogram queries such as our example above, we have $\Delta = 1$ so we add Laplace $\left(\frac{1}{\epsilon}\right)$ noise to our function.

However, what if we wanted to compute the maximum value in a set? If we had n elements, we certainly wouldn't want to apply Composition n times, obtaining $n\epsilon$-DP, just to have `Laplace` $\left(\frac{1}{\epsilon}\right)$ noise added to our answer. A better way is to use ReportNoisyMax 2. Instead of paying $n\epsilon$, ReportNoisyMax allows us to pay ϵ for the exact same noise on our answer. However, ReportNoisyMax only works on monotone queries, or where $f(x, D) < f(x, D')$ for all $x \in X'$. A version that works on queries in general

```
Algorithm 2: ReportNoisyMax
Input  : $D \in \mathcal{D}$, $\mathcal{X}$, a domain; $f$, a function $\mathcal{X} \times \mathcal{D} \to \mathbb{R}$; and $\epsilon$
Output: $x \in \mathcal{X}$ that attains maximum value on $f(S)$, satisfying $\epsilon$-DP.
$X$, a vector of $|\mathcal{X}|$ i.i.d. variables drawn from Laplace $\left(\frac{\Delta(f)}{\epsilon}\right)$;
return $\arg\max_{i=1}^{|\mathcal{X}|}\{X + f(\mathcal{X})\}$
```

```
Algorithm 3: Exponential Mechanism
Input  : $D \in \mathcal{D}$; $\mathcal{X}$, a domain; $f : \mathcal{X} \times \mathcal{D} \to \mathbb{R}$, a utility function, $\epsilon$
Output: $x \in \mathcal{X}$ where $f(x, D)$ is more likely to be high.
Pick $x \in \mathcal{X}$ where $\Pr(x = k) \propto \exp\left(\frac{\epsilon f(k,D)}{2\Delta(f)}\right)$;
return $x$
```

```
double NoisyCount(double epsilon){
    if(myagent.apply(epsilon)){
        return mysource.Count() + Laplace(1.0/epsilon);
    }else{
        throw new Exception("Access Denied")
    }
}
```

Figure 3: NoisyCount Implemented in PINQ.

is the Exponential Mechanism 3. ReportNoisyMax doesn't have a factor of 2 in its Laplacian noise, and this means a lighter tail. Thus, for monotone queries, we use ReportNoisyMax.

2.2 Related Work

Perhaps the best-known example of a language that attempts to help practitioners use DP is PINQ [14]. PINQ builds off Microsoft's C-sharp LINQ database system and provides an interface between the database and the programmer that can accomplish simple differential-privacy mechanisms. PINQ makes extensive use of the Laplace Mechamism 1 to noise histogram queries such as `Count`, `Average`, and `Median`. It also uses Composition 2 extensively when many of these queries are executed, and it attempts to abstract the composition into a `PINQAgent` class which keeps track of privacy budget. For example, the `NoisyCount` function is implemented in Figure 3 The amount of code that can be written in PINQ is surprisingly large; indeed, McSherry provides implementations of k-means and Social Networking. Also, the interface is a step in the right direction of thinking about privacy as a black box. However, the drawback of PINQ is that an analyst must still think about noise at every step of the computation. If an `Access is denied` error is thrown or the results are unacceptably noisy, the analyst will have no idea how to fix their code without diving into privacy.

Several languages have been built off PINQ to try to increase its functionality [17] [9]. wPINQ [17] uses a weighted database which is a function $f : \mathcal{D}^n \rightarrow \mathbb{R}$, which can be thought of a generalization of regular databases where each element appears a natural number of times. Proserpio et al. define `Join` on two weighted databases in a way that greatly reduces the weights of elements that appear many times in the resulting join.

For instance, in a complete bipartite graph with n vertices, there are no triangles. However, the addition of a single edge can add $O(n)$ triangles to the graph, so the triangle-counting function is very sensitive. If we were to count the number of triangles given a database D of all the edges, we could start with the standard outer Join, $\texttt{Join}(D, D)$, to produce paths of length 2. Indeed, this produces a database where $O(n)$ elements could vary when a single edge is changed. To prevent this problem, the authors define a new type of Join, where the weight of an element in $\texttt{Join}(D, D)$ is inversely proportional to the number of times each vertex in the path appears in any path of length 2. This curbs the sensitivity immensely, and the authors succeed in counting the number of triangles (or any design) using their version of `Join`.

The authors of [9] overcome the large sensitivity of `Join` by forcing the predicate of the Join to be just equality of keys. This greatly reduces the amount that a single record can change the result of `Join`.

In Fuzz [19], a type system is implemented to guarantee differential privacy and sensitivity. Each type is endowed with a metric, and judgments are given for richer programming constructs such as sums, products, recursive types, and lambda expressions. Once the type of the program is known, its metric is known and its sensitivity can be inferred from the input variables. Sensitivity allows us to add the proper amount of noise via the Laplace Mechanism. To go from an ϵ-sensitive function to an ϵ-DP function, noise is added via a monad by applying the function $add\_noise : \mathbb{R} \multimap \bigcirc\mathbb{R}$.

For instance, counting the number of people in a database older than 40 could be achieved by:

$$add\_noise\ (size\ (filter\ over_{40}\ db)) : db \multimap \bigcirc\mathbb{R}$$

However, there are two shortcomings with this method. First, the analyst is still responsible for adding noise and making decisions that affect the sensitivity of a computation. Perhaps it’s better to defer the adding of noise of the

2.3 Existing Methods for Algorithmic Choice

Limited versions of algorithmic choice have been demonstrated in previous work. These methods focus on giving the analyst the ability to choose an ϵ, evaluate whether the computations is satisfactory, and then have the option of increasing ϵ if the computation is deemed unsatisfactory. For instance, if the analyst is training a private linear regression model that requires her to release $(X^T X)^{-1} X^T y$ for a matrix X and output vector y, then it's easy for her to compute the sensitivity of her function and apply the Laplace Mechanism to the output. However, it's not easy to know ex-ante exactly how high ϵ should be to result in a fit that has, for example, less than 5 mean square error.

This problem is addressed by Koufogiannis et. al [10] who give a general method for releasing any computation based on the Laplace Mechanism *gradually.* In their version of the Laplace Mechanism, an analyst selects a set of increasing ϵ values, $(\epsilon_1, \epsilon_2, \ldots, \epsilon_T)$. Their main result is finding a way to generate Laplace variables $(v_1, v_2, \ldots, v_T)$ such that knowing a prefix $(v_1, v_2, \ldots, v_t)$ is ϵ_t-DP and, with high probability, the noise generated by v_i is no worse than the noise added by the standard Laplace Mechanism for guaranteeing ϵ_t-differential privacy. They accomplish this by correlating the v_i's in an intelligent way. Ligett et. al [12] take this one step further by describing a way to iterate through the v_i's until a suitably accurate algorithm is selected. Accuracy can be specified by the analyst as an arbitrary function that takes in a v_i. Of course, the release of the most-accurate mechanism has to be done in a differentially-private manner as well, and Ligett et. al use an adaptation of the AboveThreshold mechanism [2]. This results in an additional privacy and accuracy penalty over simply using the Laplace Mechanism with a fixed ϵ. Specifically, if v_t is chosen, then the computation is $\epsilon_t + \epsilon_A$-DP.

We notice the following problems with the methods addressed above:

- AboveThreshold does not work on releasing (ϵ, δ)-DP mechanisms with $\delta > 0$.
- A privacy and accuracy penalty is taken when selecting the best algorithm. This method is rather arbitrary; how do we select ϵ_A? Is this even the optimal way of algorithm selection?
- The papers only work on the specialized case of the Laplace Mechanism.

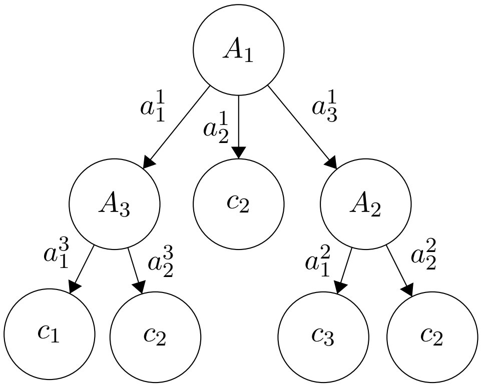

Figure 4: Example Decision Tree.

More papers to talk about: [21], [13], [7].

2.4 Decision Trees

Decision Trees are a powerful tool for data mining due to their high human interpretability, non-parametric design, low computational cost, ability to discover non-linear relationships among attributes, resilience to missing values, ability to handle both discrete and continuous data, and ability to handle non-binary labels [4]. Let our database again be D, and suppose it has k columns or attributes, the ith of which can take values in $\mathcal{A}_i$. Let $\mathcal{C}$ be the output attribute, or class, which we are trying to predict. We assume for simplicity that $\mathcal{A}_i$ and $\mathcal{C}$ are discrete sets. A decision tree classifies points by branching on attribute $\mathcal{A}_i$, forming $|\mathcal{A}_i|$ subtrees. Once certain criteria are met, no more branching occurs, and instead a leaf node predicts the class. An example Decision Tree is given in Figure 4.

The most widely-used algorithm for training decision trees is the C4.5 algorithm. C4.5 grows trees top-down, and it creates a branch up to a certain depth specified by the user. For each branch, it selects the attribute that produces the lowest conditional entropy and splits the dataset on it. Conditional Entropy is defined as:

Definition 5. *(Entropy) The Entropy of a discrete random variable X which attains k values with probabilities $p_1, p_2, \ldots, p_k$ is*

$$H(X) = -\sum_{i=1}^{k} p_i \log(p_i)$$

The conditional entropy of X given a discrete random variable Y which attains values $a_1, \ldots, a_\ell$ is

$$H(X \mid Y) = \sum_{i=1}^{\ell} \Pr[Y = i] H(X \mid Y = i)$$

Conditional entropy, being a measure for information, is minimized so as to prioritize those attributes which produce large information gain. For leaf nodes, the class that has the largest representation in the remaining dataset is selected. The C4.5 algorithm appears in Algorithm 5. We denote by $D_x^{(i)}$ to be the subset of D which attains value j on attribute i and $D_{x,y}^{(i)}$ to be the subset of $D_x^{(i)}$ which also has class y. Let's let $\tau_{x,y}^{(i)}$ be the size of $D_{x,y}^{(i)}$. In the code, we represent this as `D[i=x]` and `D[class=y]`, respectively. Defining conditional entropy in our new notation, we get, for attribute with index i:

$$H_i(D) = \sum_{j \in \mathcal{A}_i} \frac{\tau_j^{(i)}}{\tau} \sum_{c \in \mathcal{C}} \frac{\tau_{j,c}^{(i)}}{\tau_j^{(i)}} \ln \left(\frac{\tau_j^{(i)}}{\tau_{j,c}^{(i)}} \right) \to \sum_{j \in \mathcal{A}_i} \sum_{c \in \mathcal{C}} \tau_{j,c}^{(i)} \ln \left(\frac{\tau_j^{(i)}}{\tau_{j,c}^{(i)}} \right) \tag{1}$$

We omit a $\frac{1}{|D|}$ factor on the right side as we often simply compare computations on a fixed D. Two other estimates for the quality of a split which are generally less good in the non-private setting are Gini and Max:

$$G_i(D) = \sum_{j \in A_i} \tau_j^{(i)} \left(1 - \sum_{c \in C} \left(\frac{\tau_{j,c}^{(i)}}{\tau_j^{(i)}} \right)^2 \right) \tag{2}$$

$$M_i(D) = \sum_{j \in A_i} \max_c (\tau_{j,c}^{(i)}) \tag{3}$$

When converting this mechanism to the differentially-private version, the user is left with several questions as noted in [4]:

- How large of a budget has been alotted or should be alotted? There isn't a clear way to decide the budget, as the performance of the algorithm may vary wildly with the budget.
- How many times should the data be queried? How would pick d in C4.5? Should one alter line (1) to something different?
- Might the sensitivity of some of the queries prevent an accurate choice? Specifically, even though it is widely agreed that the entropy function

```
def dtree(D, atts, clss, d):
  if(d = 0 or len(atts) = 0):
    best_class = max over c in clss of len(D[clss=c, ])
    return Leaf(pred=best_class)
  else:
    best_att = max over A in atts of:
      c_entropy(D, A, clss)
    C = map(best_att, lambda a: dtree(D[best_att=a, ],
      att-best_att, clss, d-1))
    return Node(att=best_att, children=C)
```

Figure 5: C4.5 Algorithm.

performs best in the non-private setting, could a lower-quality function be substituted because its computation is more accurate?

- How does the size of D impact performance? Is there enough data to provide accurate results in the private setting?

As we will see, the answers to these questions are data-dependent and the there is never one answer that always dominates.

3 Decision Tree Examples

3.1 Private Decision Trees

We assume, as does most of the literature, that the $\mathcal{A}_i$ and $\mathcal{C}$ are public and D is private. Also, denote our total privacy budget to be β. For recursive decision tree algorithms set up like 5, the recursive calls on the same depth of the tree all operate on disjoint subsets of D. Therefore, a conservative estimate of the privacy usage, using composition and disjointness, is

$$\sum_{i=0}^{d} \max_{n \in \texttt{Nodes on lvl } i} \epsilon_n \tag{4}$$

where ϵ_n is the privacy used by node n. We will go into details on how to pick ϵ_n later.

The most naive way to create a DP version of C4.5 is to take all the histogram queries in algorithm 5 and change them to `NoisyCount`. This amounts to computing the dataset size, the most common class of a leaf, and the conditional entropy 1 of a branch. Due to the disjointness of the $D_{j,c}^{(i)}$ with each other and the $D_j^{(i)}$ with each other with a fixed i, we can do a noisy count on each $|D_j^{(i)}|$ and $|D_{j,c}^{(i)}|$ with $\frac{\epsilon'}{2}$ budget (sensitivity 1), and compute the conditional entropy spending just ϵ' budget. Unfortunately, doing this over all attributes is not disjoint, so we have to split our ϵ_n budget over up to k attributes and use composition. This adds a lot of noise to our computations, namely each NoisyCount gets $\frac{\beta}{dk}$ budget. This method is presented in [1] as a proof of concept rather than a high-performing algorithm.

To fix this accuracy problem, Friedman and Schuster [5] use entropy as a utility function on a call to the exponential mechanism. Specifically, they operate over the domain $\{1, 2, \dots, k\}$, the indices of the attributes, and utility function $u(D, i) = -H_i(D)$, where the entropy is negated because we want to find the attribute with greatest entropy reduction. They also try functions other than entropy as a quality estimator because entropy has a rather high sensitivity, shown in Theorem 5.

Theorem 5. *The entropy function on disjoint histogram counts $a_1, a_2, \dots, a_n$ produced from a database D has sensitivity bounded by $\frac{1}{|D|}\left(\frac{1}{\ln(2)} + \log(|D|)\right)$.*

Proof. Let $A = \sum_{i=1}^n a_i$. Then, the entropy is

$$\sum_{i=1}^n \frac{a_i}{A}\log\left(\frac{A}{a_i}\right) = \frac{1}{A}\sum_{i=1}^n a_i \log A - \frac{1}{A}\sum_{i=1}^n a_i\log(a_i) = \log(A) - \frac{1}{A}\sum_{i=1}^n a_i\log(a_i)$$

Suppose bucket a_j is reduced by 1, and the entropy change is

$$\begin{aligned}
&\log(A) - \log(A-1) - \frac{1}{A}a_j\log(a_j) + \frac{1}{A-1}(a_j-1)\log(a_j-1)\\
&\leq \frac{1}{\ln(2)(A-1)} - \frac{1}{A}(a_j-1)\log(a_j-1) + \frac{1}{A-1}(a_j-1)\log(a_j-1)\\
&= \frac{1}{\ln(2)(A-1)} + \frac{1}{A(A-1)}(a_j-1)\log(a_j-1) \leq \frac{1}{\ln(2)(A-1)} + \frac{1}{A}\log(A)
\end{aligned}$$

□

Another big change is the stopping criteria (Line 2 in Figure 5). As noted in [4], the stopping criteria could be different in differential privacy versus the regular setting because Laplace $\left(\frac{1}{\epsilon}\right)$ creates much higher error in smaller leaves, and the depth of a tree affects the amount noise added. On a high

```
1  def dtree_private(D, atts, clss, d, e):
2    t = max over a in atts of len(a)
3    size = noisyCount(D, e/2)
4    if(d = 0 or len(atts) = 0 or size/(t*len(clss)) < sqrt(2)/e):
5      best_class = max over c in clss of
6        noisyCount(len(D[clss=c, ]), e/2)
7      return Leaf(pred=best_class, size=size)
8    else:
9      U = map(attrs, lambda a: -c_entropy(a, clss, D))
10     best_att = exp_mech(domain=attrs, utilities=U, epsilon=e/2)
11     C = map(best_att, lambda a: dtree_private(D[best_att=a, ],
12       att-best_att, clss, d-1, e))
13     return Node(att=best_att, children=C)
14
15 def dtree_pvt_top(D, atts, clss, d, budget):
16   t = dtree_private(D, atts, clss, d, budget/(d+1))
17   return C4.5_Prune(t)
```

Figure 6: Private C4.5 proposed by Friedman and Schuster [5].

level, this suggests that shorter, sparser trees will perform better in the DP setting, but the ultimate relationship remains unclear. In [5], an additional stopping parameter depending on the NoisySize of D and some other public parameters are used in addition to the stopping parameters in C4.5. Their goal is to ensure that a certain signal is larger than the noise of NoisyCount:

$$\frac{|D|}{t|C|} < \frac{\sqrt{2}}{\epsilon}$$

where $\frac{|D|}{t|C|}$ is a signal for how large the partitions that the children nodes will make. The sizes of D for each node is used again when doing the standard C4.5Prune algorithm [18] which doesn't use a validation set but estimates a confidence interval from the results of the training set. To do this, it needs an estimate on the sizes of $D[class = c]$ for each $C \in \mathcal{C}$ at each node, and knowing the size of D helps Friedman and Schuster estimate this. Their algorithm is shown in Figure 6. We've seen that Friedman and Schuster make three big design choices that are based on experimentation and heuristics:

- **Stopping Criteria** The `if` statement on Line 4 decides whether to continue branching or to stop, and in particular, the comparison of the signal to the noise is, in the words of the authors, "arbitrary". Also, the second pass done in `C4.5_Prune` (Line 17) may clip a branch and is based on the noisy sizes collected in Line 3.

- **Non-Leaf Queries** For nodes that are not leaves, the exponential mechanism is used with different utility functions. The question of which utility function to use arises, as this is unclear given their different sensitivities.
- **Number of Trees** It's well-known that a decision forest τ trees and a simple majority vote can often outperform a single decision tree. The number of trees to train in the DP setting could be much different than the answer in the non-private setting.

Another very important question is how much privacy budget to give to each of these components. If we give 0 budget to the exponential mechanism, then we are essentially picking a random attribute. Random decision forests have been known to outperform decision trees in the non-private setting. A summary of other works and their decisions on the above three criteria appear in the table below:

	Stopping Criteria in addition to $d = 0 \Vert k = 0$	ϵ_{stop}	Non-Leaf-Queries	ϵ_{NLQ}	ϵ_{LQ}	τ
Friedman & Schuster [5]	$\frac{D}{t*\lvert C\rvert} < \frac{\sqrt{2}}{\epsilon}$; d' user-defined; second pass with C4.5Prune	$\frac{\beta}{2d'}$	Exp. Mech with entropy, Gini, max	$\frac{\beta}{2d'}$	$\frac{\beta}{2d'}$	1
Mohammed et al. [15]	d' user-defined	0	Exp. Mech with entropy	$\frac{\beta}{d'}$	$\frac{\beta}{d'}$	1
Jagannathan et al. [8]	$d' =$ min of $\frac{k}{2}$ and $\log_b(\lvert D\rvert) - 1$	0	Random	0	$\frac{\beta}{\tau}$	10
Patil & Singh [20]	$\frac{D}{t*\lvert C\rvert} < \frac{\sqrt{2}}{\epsilon}$; d' user-defined; second pass with C4.5Prune	$\frac{\beta}{2d'}$	Exp. Mech with entropy	$\frac{\beta}{2d'\tau}$	$\frac{\beta}{2d'\tau}$	10
Fletcher & Islam [3]	$\log_b(\lvert D\rvert) < \frac{\sqrt{2}\lvert C\rvert}{\epsilon}$	0	Random	0	$\frac{\beta}{\tau}$	m

Mohammed et al. [15] propose not collecting the size (Line 3) and eliminating the size criterion on Line 4. This allows us to use the entire ϵ budget for the node on the exponential mechanism and on the NoisyCount (Lines 10 and 6, respectively). Jagannathan et al. [8] propose choosing a random attribute on Line 9, spending no budget, and setting $d = \min\{\frac{k}{2}, \log_b(|D|) - 1\}$ where b is the average branching factor of the attributes, $\frac{1}{k}\sum_{i=1}^{k} |\mathcal{A}_i|$. Patil and Singh [20] do the same thing as Fletcher and Schuster but use multiple trees. Finally, Fletcher and Islam [3] do random trees but without a predefined depth; instead, they estimate the support at each node by assuming that datapoints are cut uniformly at each branch, and they stop when the support is less than the noise, similar to Friedman and Schuster.

3.2 Experiments

In this section, we plot the performances of the five algorithms on different datasets to highlight the existence of data dependence. Graphs of the data for different datasets appear in Figure 7. The datasets are described below:

Name	No. Points	No. Attrs	Avg. Branch Size	Class size
HIV-1 protease cleavage	1625	8	20	2
Nursery	12960	8	3.375	5
Contraceptive	1473	9	4.88	3

The HIV dataset is a sparse dataset; there are far more possible attribute values than the number of data points actually in the dataset. As we go farther down the decision tree, after only a short while, the vast majority of nodes will have no training data at all—after just 3 branches, there are 20^3 possible nodes which means almost 80% of the nodes have to be empty, and most of the nodes have large support [insert a graph of this]. As is well known in decision trees, low-support nodes give noisier estimates of the classes, and it rarely pays to branch at these nodes. This explains the performances of the algorithms: A1, having the most sophisticated stopping criteria, performs the best because it knows which nodes have low support. A2, having almost no stopping criteria at all, performs the worst (and takes the longest to train, going to all 20^d leaves), and A5, with a stopping criterion almost as weak, took too long to train.

The Nursery dataset, on the other hand, is dense. The stopping criteria does not matter as much as all nodes have a healthy support (in fact, the dataset has points for every possible value of the attributes). Thus, A2, which doesn't waste any budget on the stopping criterion, performs best.

The Contraceptive dataset is a mix between the two other datasets. Here, one advantage of A3 is very pronounced: its excellent performances at small values of ϵ. A3 gives the most budget to its leaf nodes, resulting in very accurate predictions at even small ϵ assuming the random trees are chosen in a fairly simple way. Because Contraceptive has no “bogus” attributes [insert graph of this], a randomly-chosen subset of the attributes is a fair set to work with, until the other, biased algorithms have enough budget to catch up.

In light of these experiments, we notice that stopping criteria, non-leaf-queries, and number of trees in the forest have a huge impact on the performance of the algorithm. What an analyst would really like to do is to shed light on these complicated relationships in an automated way. It would be easy for her to summarize all of the decision tree algorithms in a chart as in Figure 8. Each of the two places where she is uncertain is marked with a label, L1 or L2. The goal of `Jostle` is to learn automatically properties of the database that may lend to a certain decision at L1 or L2.

Our goal is to decide which of the heuristics provided by all the decision trees may perform best, along with any other heuristic the analyst may provide us.

References

[1] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: The sulq framework. In *24th ACM SIGMOD International Conference on Management of Data / Principles of Database Systems, Baltimore (PODS 2005)*, Baltimore, Maryland, USA, June 2005.

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

[3] Sam Fletcher and Md Islam. A differentially private random decision forest using reliable signal-to-noise ratios, 11 2015.

[4] Sam Fletcher and Md Zahidul Islam. Decision tree classification with differential privacy: A survey. *CoRR*, abs/1611.01919, 2016.

[5] Arik Friedman and Assaf Schuster. Data mining with differential privacy. In *Proceedings of the 16th ACM SIGKDD International Con-*

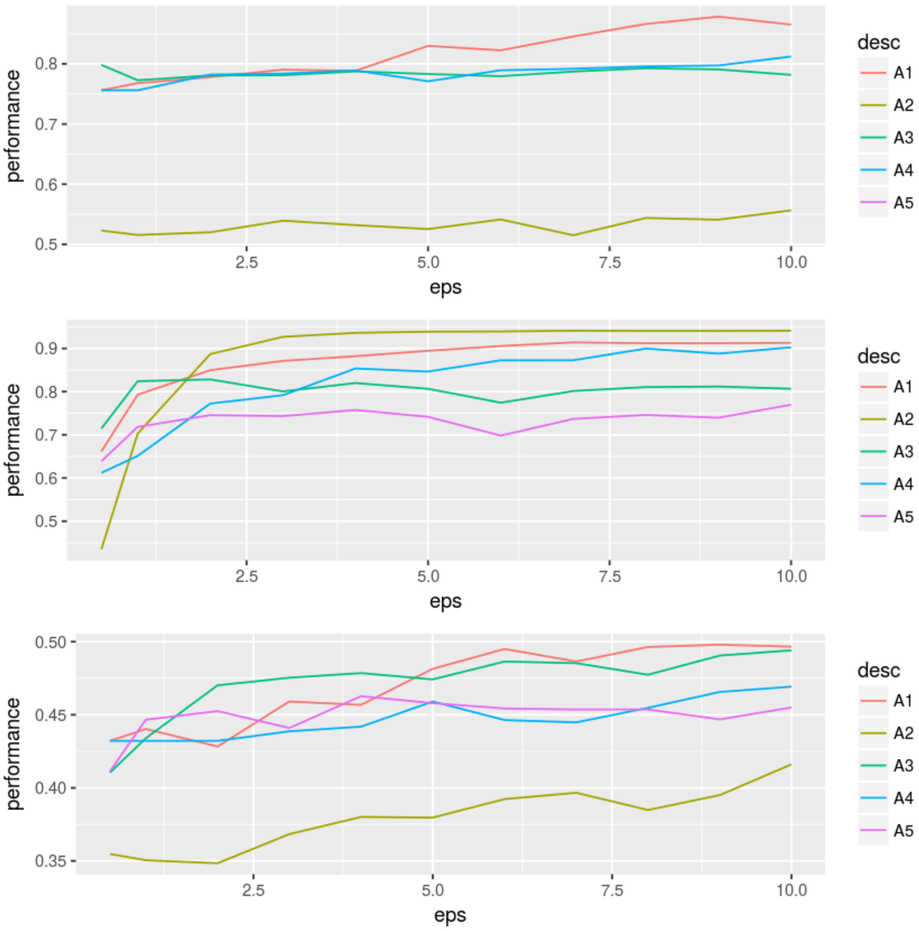

Figure 7: Performances of the five decision tree algorithms. The performance is measured from the prediction success rate on a validation set using a 30/70 validation to training split. Graph 1 does not have A5 because it takes such a long time to train, having a weak stopping criterion.

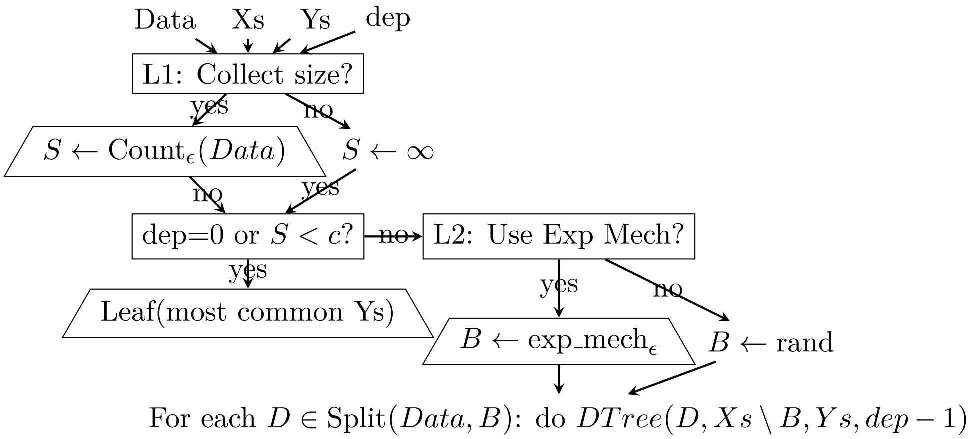
Data
Xs
Ys
dep
L1: Collect size?
yes
no
$S \leftarrow \text{Count}_\epsilon(Data)$
$S \leftarrow \infty$
no
yes
dep=0 or $S < c$?
no
L2: Use Exp Mech?
yes
Leaf(most common Ys)
yes
no
$B \leftarrow \text{exp\_mech}_\epsilon$
$B \leftarrow$ rand
For each $D \in \text{Split}(Data, B)$: do $DTree(D, Xs \setminus B, Ys, dep - 1)$

Figure 8: Our decision tree algorithm with NoisyConditionals which learn the execution path from past histories. Depending on what the NoisyConditionals say, this algorithm is capable of expressing all the decision tree algorithms in [4].

ference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010, pages 493–502, 2010.

[6] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, Dan Zhang, and George Bissias. Exploring privacy-accuracy tradeoffs using dpcomp. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2101–2104, New York, NY, USA, 2016. ACM.

[7] Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C. Pierce, and Aaron Roth. Differential privacy: An economic method for choosing epsilon. *CoRR*, abs/1402.3329, 2014.

[8] Geetha Jagannathan, Krishnan Pillaipakkamnatt, and Rebecca N. Wright. A practical differentially private random decision tree classifier. In *Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, ICDMW '09, pages 114–121, Washington, DC, USA, 2009. IEEE Computer Society.

[9] Noah M. Johnson, Joseph P. Near, and Dawn Xiaodong Song. Practical differential privacy for SQL queries using elastic sensitivity. *CoRR*, abs/1706.09479, 2017.

[10] Fragkiskos Koufogiannis, Shuo Han, and George J. Pappas. Gradual release of sensitive data under differential privacy. *CoRR*, abs/1504.00429, 2015.

[11] Chao Li, Michael Hay, Gerome Miklau, and Yue Wang. A data- and workload-aware algorithm for range queries under differential privacy. *Proc. VLDB Endow.*, 7(5):341–352, January 2014.

[12] Katrina Ligett, Seth Neel, Aaron Roth, Bo Waggoner, and Zhiwei Steven Wu. Accuracy first: Selecting a differential privacy level for accuracy-constrained ERM. *CoRR*, abs/1705.10829, 2017.

[13] Xiaoqian Liu, Qianmu Li, Tao Li, and Dong Chen. Differentially private classification with decision tree ensemble. *Applied Soft Computing*, 62:807 – 816, 2018.

[14] Frank McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53(9):89–97, September 2010.

[15] N. Mohammed, S. Barouti, D. Alhadidi, and R. Chen. Secure and private management of healthcare databases for data mining. In *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*, pages 191–196, June 2015.

[16] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR*, abs/cs/0610105, 2006.

[17] Davide Proserpio, Sharon Goldberg, and Frank McSherry. Calibrating data to sensitivity in private data analysis: A platform for differentially-private analysis of weighted datasets. *Proc. VLDB Endow.*, 7(8):637–648, April 2014.

[18] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann Publishers, 1993.

[19] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. *SIGPLAN Not.*, 45(9):157–168, September 2010.

[20] Sanjay Singh and Abhijit Patil. Differential private random forest, 09 2014.

[21] Daniel Winograd-Cort, Andreas Haeberlen, Aaron Roth, and Benjamin C. Pierce. A framework for adaptive differential privacy. *Proc. ACM Program. Lang.*, 1(ICFP):10:1–10:29, August 2017.