École polytechnique fédérale de Lausanne

Semester project fall 2023

Master in Mathematics

Secure and Interpretable: Federated Differentially Private Decision Trees

Author:
Hannes Gubler

Supervisor:
Dr. Guillaume Obozinski

Contents

1 Introduction

The biomedical field has experienced a rapid growth in the volume of available data over the past years. Together with the widespread success of machine learning, patient records have become increasingly crucial in the healthcare landscape. This surge in data availability allows machine learning algorithms to extract meaningful insights and enhance diagnostic accuracy, potentially improving overall treatment outcomes.

Yet, relying on patient records for machine learning raises concerns about individual privacy. Often, it is desirable to utilize patient data across different hospitals to further enhance the performance and generalizability of machine learning models. This collaborative approach requires careful analysis, as centralizing all data in one location is not always feasible due to the need to safeguard sensitive patient information, sometimes even due to privacy regulations. Centralizing all data requires a trusted entity responsible for storing, managing, and processing the combined datasets. However, assuming such a condition does not always align with many real-world scenarios, where establishing such a trusted entity can be very challenging. This is why in our work, we do not make this assumption and rely on a decentralized approach, where the raw data never leaves an owner's servers. Our approach to protect individual patient data is to combine federated learning [1], a decentralized machine learning training approach, with differential privacy [2], a technique to protect individual data during the model training process. This hybrid approach enables a decentralized, collaborative training across hospitals while ensuring that the privacy of individual patients is maintained.

While machine learning models, such as neural networks, have remarkable predictive power, they often fall short in terms of human interpretability [3]. In the medical domain, understanding the decisions made by a model is crucial for healthcare professionals to understand and trust the reasoning behind those models. This is why we focus on decision trees in our work, as they excel in providing highly interpretable results[4, 5].

Given the consideration presented above, in the following sections we focus on the design of a decision tree algorithm that yields differential privacy and can be trained in a decentralized way across multiple parties using federated learning. We introduce our method in the context of binary classification, however it is easily extendable to multiclass classification. Moreover, this report is written assuming a scenario where multiple hospitals collaborate to collectively learn model based on data from different patients.

The outline of the remainder of this work is as follows: In Section 2, we introduce the necessary theory about decision trees, federated learning, multi-party summation and differential privacy needed for the development of the algorithm. Afterwards, we introduce the framework for our federated differentially private decision tree algorithm in Section 3. We first provide a baseline algorithm relying on so-called private histograms, which we further improve in Section 3.5 through reducing the number of split candidates by lower bounding a feature's split impurity. Finally, in Section 4 we evaluate and analyze the models in different setups through a simulation study with synthetic data.

2 Background

2.1 Decision Trees

A decision tree is a non-parametric supervised learning method for classification purposes. As its name suggests, the model is structured as a tree, composed of nodes, edges and leaves. The root node, as well as the internal nodes, represent a binary decision on a specific feature of the dataset. For continuous features, this binary decision corresponds to whether or not the feature value is greater than a certain threshold, whereas for categorical features, it corresponds to the equality of the feature value to one of the categories of the feature. The leaf nodes of a decision tree represent the final outcomes or predictions. Each leaf node is labelled with a specific class or, often more useful in biomedical applications, with associated class probabilities. In essence, a decision tree partitions data space based on these tests until it reaches a leaf node, where the final predictions or classifications are made. Figure 1 presents a visual representation of a decision tree.

One of the key strengths of a decision tree is its high interpretability [4, 5]. Furthermore, decision trees have the ability to discover non-linear relationships among the features [6]. To prevent a decision tree from overfitting, stopping criteria such as the maximal tree depth allowance or a minimum samples per leaf requirement are often used.

During training, every node uses the subset of training samples that meet the sequence of decision criteria leading to this node when following the path from the root node to the node of interest to decide where to split. The CART algorithm [7], on which we base our algorithm from section 3, constructs a decision tree using the feature and threshold that yield the largest information gain in each node. More specifically, in CART each node is split to minimize the impurity G defined in the following Section.

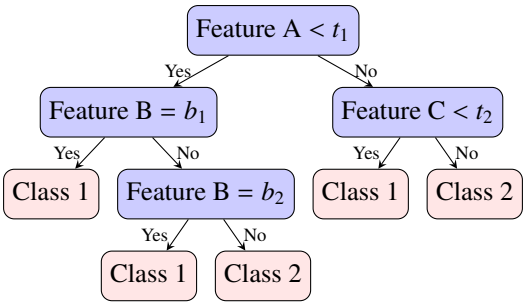

Figure 1: Example of a decision tree for binary classification with continuous features (A, C) and a categorical feature B taking values in $\{b_1, ..., b_l\}$.

2.1.1 Mathematical formulation

In the following we present the CART algorithm in the case of binary classification and using the Gini entropy. Given training samples $x_1, \dots, x_n \in \mathbb{R}^p$ with binary labels $y_1, \dots y_n \in \{0, 1\}$, let Q denote the root node containing all n samples. For a candidate split $\theta = (j, t)$ consisting of a threshold t on feature j, we define the left and right child nodes as

$$Q^{\text{left}}(\theta) = \left\{(x, y) \in Q \,\middle|\, x_j \leq t\right\}$$
$$Q^{\text{right}}(\theta) = \left\{(x, y) \in Q \,\middle|\, x_j > t\right\}.$$

Letting $n^{\text{left}} = \left|Q^{\text{left}}\right|$ and $n^{\text{right}} = \left|Q^{\text{right}}\right|$, the quality of a candidate split θ is evaluated using its impurity $G(Q, \theta)$, defined through an entropy function h:

$$G(Q, \theta) = \frac{n^{\text{left}}}{n} h(Q^{\text{left}}(\theta)) + \frac{n^{\text{right}}}{n} h(Q^{\text{right}}(\theta)).$$

In the following Sections, as well as the implementations corresponding to our work, we use the Gini entropy, defined by:

$$h(Q^{\text{left}}(\theta)) = \hat{p}^{\text{left}}(1 - \hat{p}^{\text{left}}), \quad \text{where } \hat{p}^{\text{left}} = \frac{1}{n^{\text{left}}} \sum_{i: x_i \in Q^{\text{left}}(\theta)} y_i,$$

and similarly for $h(Q^{\text{right}}(\theta))$ with $\hat{p}^{\text{right}}$.

Finally the CART algorithm selects the split θ^* that minimizes the impurity, that is

$$\theta^* = \text{argmin}_\theta \, G(Q, \theta).$$

Once we found the split θ^* that minimizes the impurity, we recursively repeat the same procedure for the nodes $Q^{\text{left}}(\theta^*)$ and $Q^{\text{right}}(\theta^*)$ until a stopping criterion, such as the maximum allowable tree depth, is met. In that case, a leaf is created and labelled with the majority class of the training samples that fall into that leaf.

Note that the above tree building procedure can be derived through empirical risk minimization, in which the Gini entropy corresponds to using the squared loss.

2.2 Federated Learning

The term *federated learning* was introduced by McMahan et al. [1]. It refers to a decentralised approach during the training process of models, which leaves the training data distributed on the provider's devices or servers. Instead of using a shared database with all the participants' data, federated learning trains a collaborative model through the sharing of training parameters across participants during the training process. A key challenge in federated learning is to minimize the leakage of information about the data during the exchange of the training parameters when training a federated model. It has been indicated that federated learning alone is not enough to guarantee privacy [8], as every federated query leaks some information about the data of the other participants. This is where differential privacy [2] comes into play, a framework which is able to quantify the privacy loss, based on a clear mathematical definition (see Section 2.3).

In this work, we consider a setting with K data holders, which we call parties throughout the report. In the scenario where the parties correspond to hospitals, K is usually not too large. For this reason, in our simulation study in Section 4, we evaluate the models in a scenario with $K = 5$ parties. However, the tree algorithms introduced in Section 3 are applicable for any number of parties K. Our only assumption is that the data is horizontally partitioned, meaning that every party maintains the same p features, where the k-th party holds n_k samples.

2.3 Differential Privacy

Differential Privacy [2] is motivated by the need to protect single individuals in a dataset while still allowing valuable data analysis. When dealing with sensitive data like clinical records, it is important to ensure that the final model does not disclose particular information about individual patients. The approach of differential privacy is to randomize the algorithm creating the final model to prevent it from revealing patterns belonging to specific samples (e.g. patient records).

2.3.1 Definition

The definition of differential privacy formalizes the concept of how much the behaviour of a randomized algorithm changes on similar datasets. To quantify the privacy on an individual level, differential privacy compares outputs on neighbouring datasets.

Definition 1. *(Neighbouring Datasets) Two datasets D and D' are called neighbouring datasets, denoted $D \sim D'$, if we can reach D' from D by adding or removing one single datapoint.*

By quantifying the output change of a randomized algorithm on neighbouring datasets, we quantify the output change when adding or removing one single datapoint, hence the contribution of an individual record to the outcome. The definition of differential privacy describes exactly this output change.

Definition 2. *(ϵ-Differential Privacy [2]) A randomized algorithm $\mathcal{A} : \mathcal{X}^* \to \mathcal{Y}$ is ϵ-differentially private if for every pair of neighbouring datasets $D, D' \in \mathcal{X}^*$ and for every output subset $Y \subset Range(\mathcal{A}) = \mathcal{Y}$ we have*

$$\mathbb{P}_{\mathcal{A}}\left(\mathcal{A}(D) \in Y\right) \leq e^{\epsilon}\mathbb{P}_{\mathcal{A}}\left(\mathcal{A}(D') \in Y\right),$$

where the probability is taken over the randomized algorithm $\mathcal{A}$.

Of course, Definition 2 is symmetrical as the roles of D and D' are interchangeable. So we have

$$e^{-\epsilon}\mathbb{P}_{\mathcal{A}}\left(\mathcal{A}(D') \in Y\right) \leq \mathbb{P}_{\mathcal{A}}\left(\mathcal{A}(D) \in Y\right) \leq e^{\epsilon}\mathbb{P}_{\mathcal{A}}\left(\mathcal{A}(D') \in Y\right).$$

Intuitively, it can be said that when ϵ is small, the behaviour of an ϵ-differentially private algorithm does not change when adding or removing one datapoint. This characteristic results into the protection of individual datapoints, as even with the full specification of all data in D and D', it is not possible to determine if the datapoint included in only one of D or D' was part of the data effectively used for training $\mathcal{A}$.

2.3.2 Composition Theorems

To design a differentially private algorithm, it is often beneficial to break down the algorithm into sub-parts. For example, a decision tree algorithm can be divided into nodes and leaves. It is then the goal to protect computations from each node and leaf with differential privacy, and afterwards combine these sub-parts to create the whole tree. This illustrates the importance of composition theorems, allowing us to compose differentially private algorithms to obtain a new differentially private algorithm.

Differential privacy possesses sequential and parallel composition properties [9], as shown in Theorems 1 and 2. Sequential composition deals with algorithms that sequentially use the same part of the data, whereas parallel composition deals with algorithms that, in parallel, use disjoint subsets of the data.

Theorem 1. *(Sequential Composition [9]) Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be algorithms that satisfy respectively $\epsilon_1, \ldots, \epsilon_k$-differential privacy. Then the sequence $\mathcal{A}_1, \ldots, \mathcal{A}_k$ satisfies $\sum_{i=1}^{k} \epsilon_i$-differential privacy.*

Theorem 2. *(Parallel Composition [9]) Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be algorithms that satisfy respectively $\epsilon_1, \ldots, \epsilon_k$-differential privacy. Assume that they are applied to disjoint subsets of the data. Then the sequence $\mathcal{A}_1, \ldots, \mathcal{A}_k$ satisfies $\max_{i=1}^{k}\{\epsilon_i\}$-differential privacy.*

Another highly useful property is that a differentially private algorithm is immune to post-processing in the sense specified by the next theorem.

Theorem 3. *(Post-Processing [10]) Let $\mathcal{A} : \mathcal{X}^* \to \mathcal{Y}$ satisfy ϵ-differential privacy and let $f : \mathcal{Y} \to \mathcal{Y}'$ be an arbitrary (randomized) mapping. Then $f \circ \mathcal{A} : \mathcal{X}^* \to \mathcal{Y}'$ satisfies ϵ-differential privacy.*

This property allows us to perform any computation on the output of an ϵ-differentially private algorithm without affecting the ϵ-differential privacy.

As the composition properties allow us to break down an algorithm into its sub-parts and then equip these sub-parts with differential privacy, the privacy parameter ϵ is often referred to as the *privacy budget*. We can think of an algorithm having a total budget of ϵ available, which is then distributed among the sub-parts to equip them with differential privacy. Sub-parts that receive a smaller budget will lead to noisier results, thus it is essential to distribute the privacy budget in an optimal way such that more important parts receive a larger budget. For example in a decision tree, it is useless to have perfect splits if the leaf labels are selected almost at random.

2.3.3 Basic Mechanisms

Protecting an algorithm with differential privacy is done by randomizing specific parts of this algorithm. While the Laplace mechanism 4 ensures differential privacy by adding noise to the output of a computation, the exponential mechanism 5 achieves differential privacy by randomly selecting a decision out of multiple possibilities.

For both mechanisms, we need the concept of the ℓ_1-sensitivity, capturing a functions magnitude by which a single datapoint can change its output.

Definition 3. *(ℓ_1-sensitivity) The ℓ_1-sensitivity of a function $f : \mathcal{X}^* \to \mathbb{R}^k$ is given by*

$$\Delta f = \max_{D \sim D'} \|f(D) - f(D')\|_1.$$

The sensitivity of a function f provides an upper bound on how much we much we must perturb its outcome to preserve privacy. In the Laplace mechanism, the sensitivity of f directly influences the variance of the noise we add to its outcome.

Definition 4. *(Laplace Mechanism) Given a function $f : \mathcal{X}^* \to \mathbb{R}^k$, the Laplace mechanism is defined as*

$$\mathcal{A}_L(D, f(.), \epsilon) = f(D) + (Y_1, \ldots, Y_k),$$

where $Y_1, \ldots, Y_k \overset{i.i.d.}{\sim} \text{Laplace}\,(0, \Delta f/\epsilon)$. A random variable has a $\text{Laplace}\,(\mu, b)$ distribution if its probability density function is

$$f\,(x \,|\, \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right).$$

Theorem 4. *The Laplace mechanism preserves ϵ-differential privacy [10].*

The exponential mechanism is defined with respect to a utility function $u : \mathcal{X}^* \times \mathcal{R} \to \mathbb{R}$, where $\mathcal{R}$ is some arbitrary range. In the context of decision trees, $\mathcal{R}$ could be all candidate splits within a node, and $u(D, r)$ computes the negative impurity of candidate split $r \in \mathcal{R}$ (where a higher negative impurity is better).

We denote the sensitivity of the utility function u as

$$\Delta u = \max_{r \in \mathcal{R}} \max_{D \sim D'} |u(D, r) - u(D', r)|.$$

The intuition behind the exponential mechanism is to output each possible $r \in \mathcal{R}$ with probability proportional to $\exp\left(\epsilon u\left(D, r\right)/\Delta u\right)$, that is $\mathbb{P}(\mathcal{A}(D) = r) \propto \exp\left(\epsilon u\left(D, r\right)/\Delta u\right)$, and therefore the privacy loss for any $r \in \mathcal{R}$ is approximately

$$\ln\left(\frac{\mathbb{P}(\mathcal{A}(D) = r)}{\mathbb{P}(\mathcal{A}(D') = r)}\right) = \ln\left(\frac{\exp\left(\epsilon u(D, r)/\Delta u\right)}{\exp\left(\epsilon u(D', r)/\Delta u\right)}\right) = \frac{\epsilon\left(u(D, r) - u(D', r)\right)}{\Delta u} \leq \epsilon,$$

where D, D' are neighbouring datasets.

However, the actual mechanism reserves half the privacy budget for changes in the normalization term and hence outputs element r with probability proportional to $\exp\left(\frac{\epsilon u(D,r)}{2\Delta u}\right)$.

Definition 5. *(Exponential Mechanism) The exponential mechanism $\mathcal{A}_E(D, u, \mathcal{R})$ with utility function $u : \mathcal{X}^* \times \mathcal{R} \to \mathbb{R}$ selects and outputs element $r \in \mathcal{R}$ with probability proportional to $\exp\left(\frac{\epsilon u(D,r)}{2\Delta u}\right)$.*

Theorem 5. *The exponential mechanism preserves ϵ-differential privacy [10].*

Other mechanisms, such as e.g. the Gaussian mechanism, can be found in the book by Abowd and Dwork [10].

2.4 Multi-Party Summation

Training a federated model is usually done by privately aggregating locally-computed updates. In our work, the only federated aggregation we carry out is summation. For completeness, we provide a method to privately compute a sum with inputs from the different parties, using homomorphic encryption [11]. An encryption system is called homomorphic if it allows a computation to be carried out on the ciphertext, in our case the encrypted summands of the parties, and then generates a ciphertext that, when decrypted, yields the result of the computation. Algorithm 1, introduced in [12, 13], is based on modulo addition to compute the private sum.

Algorithm 1 Private Sum

Input: Public upper bound for the total sum R, integers x_k held by party k, $k = 1, \ldots, K$, pairwise secret keys S_{kj} held by party k, $k, j = 1, \ldots, K$, $k \neq j$ such that $S_{kj} + S_{jk} = 0 \bmod R$.

1: Every party k calculates $Enc(x_k, S_k, R) = x_k + S_k = x_k + \sum_{j \neq k} S_{kj} \bmod R$.
2: Ever party sends $Enc(x_k, S_k, R)$ to the aggregator.
3: The aggregator calculates $\sum_{k=1}^{K} Enc(x_k, S_k, R) = \sum_{k=1}^{K} \sum_{j \neq k} x_k + k_{kj} = \sum_{k=1}^{K} x_k$.

Output: Securely calculated sum $\sum_{k=1}^{K} x_k$.

In Algorithm 1, any party can have the role of the aggregator. In cases where there are doubts about whether the aggregator might change the final result, its role can also be duplicated. To generate the secret keys S_{kj}, the Diffie-Hellman key-exchange [14] can be used, which allows two users to share a secret key over an insecure channel.

2.4.1 Distributed Differential Privacy

In order to guarantee privacy, we need to add noise to the private sum from Algorithm 1. This can be done in two ways: 1) An external trusted server takes the role of the aggregator and adds noise to the computed private sum, or 2) each party individually adds noise to their value before computing the private sum. Since the first approach relies on the strong assumption of having a trusted external server, we opt for the second approach, referred to as *distributed* differential privacy [15, 16, 17]. Given that in our framework we use the Laplace mechanism to guarantee differential privacy of the private sums, we make use of distributed perturbation Laplace algorithms (DLPA). Due to the infinite divisibility of the Laplace distribution [18], a random variable with such a distribution can be computed by summing up K other random variables. This allows every party to generate partial noise and add it to their local value, before summing up the values across all parties. Once we sum up the locally perturbed values, the sum of the partial noises follows a Laplace distribution, hence this process yields the same result as applying the Laplace mechanism to the output of Algorithm 1. The gamma DLPA generates partial noises drawn from the gamma distribution [19]: We can simulate a random variable $Z \sim Lap(0, s)$ as follows:

$$Z = \sum_{k=1}^{K} \left(G_k - G'_k \right), \tag{1}$$

where G_k, G'_k are gamma distributed random variables with shape parameter $1/K$ and rate parameter $1/s$, denoted by Gamma$(1/K, 1/s)$.

2.5 Related Work

Much of the existing work related to federated differentially private decision trees focuses on tree ensemble methods [20, 21, 22], instead of emphasizing the training of a more interpretable single differentially private decision tree in a federated way. Furthermore, the individual decision trees in the ensembles are often only trained by one party instead of in a federated way. Vos et al. [23] have a method to train a single differentially private decision tree, however their method is not designed for the federated setting. Zein et al. [24] developed an algorithm to train a federated decision tree, but the tree is not designed to satisfy differential privacy.

So in this work, we present an algorithm to train a differentially private decision tree in the federated learning setting. A lot of work on differentially private decision trees constructs these trees using random splits, while allocating the whole privacy budget to the leaf labelling. However, these random splits do not necessarily produce good leaves, as having a worse split results into having leaves containing a mix of samples where no class label clearly outweighs the other. For this reason, these trees are usually used in the context of differentially private tree ensembles, such as in [25, 26, 27]. As we are only focusing on the training of a

single decision tree, we do not want to use random splits resulting in poor leaves. Hence our approach will be to train a tree where we allocate part of the privacy budget to the node splitting, instead of solely focusing on the leaves.

3 A FDP Framework for DT Learning

In the following, we present a federated differentially private decision tree algorithm, with an adaptation in the node splitting procedure to use the privacy budget more efficiently in Section 3.5. The challenge in designing a federated decision tree algorithm that satisfies differential privacy lies in minimizing the number of federated queries without sacrificing too much performance. Using a large number of federated queries is undesirable because these are exactly the queries that leak sensitive information of the individual parties' data and therefore need to be protected through differential privacy. If these federated queries repeatedly use the same input data, the sequential composition property leads to an extensive consumption of privacy budget.

Within our framework, we focus on binary classification, defining the labels to be either 0 or 1. However, our approach can easily be extended to a more general setting with more than two classes. Furthermore, we assume the categories for categorical features, as well as the feature ranges for numerical features, are publicly known.

All the implementations of the algorithms, as well as their evaluations in Section 4, can be found on Github[1].

3.1 Multi-Party Split Impurity Calculation

It is generally possible to train a whole decision tree in a federated way, where the only exchange between the parties is through taking private sums. In Algorithm 2 we present how to calculate the impurity of an arbitrary split $\theta = (j, t)$ in a scenario where the data is split across K parties.

Algorithm 2 Federated Split

Input: Candidate split $\theta = (j, t)$, node Q containing feature matrices $(X_k)_{k=1}^K$ and response vectors $(y_k)_{k=1}^K$ across parties.

1: Every party computes locally n_k^{left} and n_k^{right} with respect to θ.
2: Calculate using Algorithm 1 $n^{\text{left}} = \sum_{k=1}^K n_k^{\text{left}}$ and $n^{\text{right}} = \sum_{k=1}^K n_k^{\text{right}}$.
3: Every party computes locally $\hat{p}_k^{\text{left}} = \frac{1}{n_k^{\text{left}}} \sum_{i:x_{ij}^k \leq t} y_i^k$ and $\hat{p}_k^{\text{right}} = \frac{1}{n_k^{\text{right}}} \sum_{i:x_{ij}^k > t} y_i^k$.
4: Calculate using Algorithm 1 $\hat{p}^{\text{left}} = \frac{1}{n^{\text{left}}} \sum_{k=1}^K n_k^{\text{left}} \hat{p}_k^{\text{left}}$ and $\hat{p}^{\text{right}} = \frac{1}{n^{\text{right}}} \sum_{k=1}^K n_k^{\text{right}} \hat{p}_k^{\text{right}}$.
5: Calculate the impurity $G(Q, \theta) = \frac{n^{\text{left}}}{n} \hat{p}^{\text{left}}(1 - \hat{p}^{\text{left}}) + \frac{n^{\text{right}}}{n} \hat{p}^{\text{right}}(1 - \hat{p}^{\text{right}})$ where $n = n^{\text{left}} + n^{\text{right}}$.

Output: Federated impurity $G(Q, \theta)$ of candidate split $\theta = (j, t)$.

We could now use existing solutions to build a differentially private decision tree and apply it to our procedure of training a federated decision tree in Algorithm 2. However, this approach would result in a large number of federated queries (when calculating $n^{\text{left}}, n^{\text{right}}, \hat{p}^{\text{left}}, \hat{p}^{\text{right}}$ for every single split candidate), blowing up the privacy budget ϵ. Protecting all the split candidates with differential privacy is, due to the sequential composition property, not an option to efficiently handle the privacy budget. We instead decide to use a similar approach as in Vos et al. [23], with private histograms to determine the best split. This drastically reduces the number of split candidates.

3.2 Federated Private Histograms

Splitting nodes based on histograms has already found success in renowned tree boosting algorithms, such as Lightgbm [28] and Xgboost [29]. While the latter use histogram splitting due to its efficiency, we employ private histograms for their effective handling of the privacy budget: Adding noise to each bin count separately is parallel composing, ensuring that the privacy budget does not add up across these queries for the bin counts. Furthermore, the calculation of the bin counts has, in the sense of Definition 3, a sensitivity of only 1 (as adding or removing one datapoint changes a bin count maximally by 1). Having a small sensitivity is desirable as it decreases the variance of the noise added to each bin count through the Laplace mechanism, as discussed in Definition 4 and Theorem 4.

[1] `https://github.com/hgubler/federated-private-trees`

The general procedure to build federated histograms with differential privacy for one feature using the gamma DLPA is described in Algorithm 3.

Algorithm 3 Federated Private Histogram

Input: Feature vectors $(X_j^k)_{k=1}^K$ across parties $k = 1, \ldots, K$, bins $([a_i, b_i))_{i=1}^r$, privacy budget ϵ.

1: Every party locally computes the bin count for each bin.
2: **for** $i = 1, \ldots, r$ **do**
    Every party k samples $G_k^i, G_k^{'i} \sim \text{Gamma}(1/K, \epsilon)$ and adds $Z_k^i = G_k^i - G_k^{'i}$ to its i-th bin count.
    Sum up the perturbed i-th bin counts across the parties using Algorithm 1.
    end

Output: Federated ϵ-differentially private Histogram of the j-th feature.

Lemma 3.1. *Algorithm 3 satisfies ϵ-differential privacy.*

Proof. As the different bins operate on disjoint subsets of the data, it is due to parallel composition enough to only consider the privacy consumption of the first bin $[a_1, b_1)$. As stated in Section 2.4.1 in Equation 1, we have that

$$Z^1 := \sum_{k=1}^{K} Z_k^1 = \sum_{k=1}^{K} G_k^1 - G_k^{'1} \sim \text{Lap}\left(0, \frac{1}{\epsilon}\right).$$

Hence summing up the perturbed local bin counts of the first bin corresponds to adding the random variable Z_1 to the overall bin count of the first bin, which ensures ϵ-differential privacy using the Laplace mechanism. □

Various options exist for selecting bin boundaries. We decide to use an equidistant grid, which has the advantage of not consuming any privacy budget (as we are assuming that the feature ranges are publicly known). Alternatively, we could use the feature's quantiles as bin boundaries. Quantiles have the advantage of evenly dividing the samples across the bins, handling features with e.g. long tails much better. However, quantiles release information about the data, hence we would need to spend part of our privacy budget for this operation. Furthermore, existing solutions for the computation of federated quantiles, as in [30], use an iterative procedure to determine the quantiles. Consequently, this results in a high volume of federated queries, making it unfavorable for efficiently handling the privacy budget.

3.3 Federated Differentially Private Tree Construction

3.3.1 Node Splitting

This section is dedicated to introduce the process of node splitting within our algorithm, making use of federated private histograms introduced before. In each node, every party separates its data by the labels, resulting in a set of samples with class 0 labels and a set of samples with class 1 labels. Then, the parties build the federated private histograms for each class separately. Since the two sets separated by labels are disjoint, computing the federated private histograms independently for each class is parallel composition, requiring no additional privacy budget compared to building histograms for all samples simultaneously. Moreover, with these histograms separated by labels, it is possible to compute the impurities of splits located at the bin boundaries. This does not consume any additional privacy budget as per the post-processing property, given that we only utilize the output information from the federated private histograms. It is therefore a key part of our algorithm to compute the histograms for each class separately. If we denote c_{ij}^k as the i-th bin count on feature j of the federated private histograms for class $k \in \{0, 1\}$, we can compute the impurity of a candidate split $\theta = (j, t)$ on feature j, where the threshold t is located on the l-th bin border as stated in Algorithm 4.

In the multiclass classification scenario, we would proceed in the same way by computing the federated private histograms separated by label, and then use a multiclass-adapted entropy function to calculate a split impurity.

Furthermore, to handle categorical features we use the one-hot encoding technique, such that we can treat them in the same way as we proceed with numerical variables. The only difference is that, for a feature corresponding to a category of a one-hot encoded variable, we only use 2 bins for our private histogram computed by Algorithm 3.

Algorithm 4 Federated Private Impurity

Input: Bin counts c_{ij}^k for bins $i = 1, \dots r$ on j-th feature for classes $k \in \{0, 1\}$, candidate split $\theta = (j, t)$ where the threshold t is on the l-th bin border.

1: Calculate $n^{\text{left}} = \sum_{i:i\leq l} c_{ij}^0 + \sum_{i:i\leq l} c_{ij}^1$, $n^{\text{right}} = \sum_{i:i>l} c_{ij}^0 + \sum_{i:i>l} c_{ij}^1$ and $n = n^{\text{left}} + n^{\text{right}}$.

2: Calculate $\hat{p}^{\text{left}} = \frac{1}{n^{\text{left}}} \sum_{i:i\leq l} c_{ij}^1$ and $\hat{p}^{\text{right}} = \frac{1}{n^{\text{right}}} \sum_{i:i>l} c_{ij}^1$.

3: Calculate the impurity of candidate split θ: $\frac{n^{\text{left}}}{n} \hat{p}^{\text{left}} \left(1 - \hat{p}^{\text{left}}\right) + \frac{n^{\text{right}}}{n} \hat{p}^{\text{right}} \left(1 - \hat{p}^{\text{right}}\right)$.

Output: Impurity of candidate split θ.

3.3.2 Leaf Labelling

In the case where a node meets a stopping criteria, a leaf is created. Utilizing private sums across the parties, we count the number of datapoints whose labels fall into that leaf for class 0 and class 1 separately. We protect these sums using the Laplace mechanism with sensitivity 1, as adding or removing one datapoint changes the sums maximally by one. As before, through parallel composition, we can handle the two sums separately without increasing the privacy budget. The above procedure is equivalent to applying private histograms from Algorithm 3 with 2 bins to the class labels, and it therefore satisfies ϵ-differential privacy by Lemma 3.1.

Now we can label the leaf with the majority class, which is post processing and therefore does not consume any additional privacy budget. Furthermore, we can also calculate the class 0 and class 1 probabilities by dividing the Laplace protected sums of each class by the total number of samples in this leaf (obtained by summing up the private sums for class 0 and class 1).

Another possibility for the leaf labelling would be to sample the label according to probabilities determined by the exponential mechanism. In this approach, however, we would not have access to probabilistic estimates of the labels, opposed to the approach with the Laplace mechanism.

3.3.3 Algorithm

In this section, we combine the ideas of the node splitting and leaf labelling procedures to present the algorithm of the federated differentially private decision tree. As stopping criteria, we use two parameters: The maximal depth allowance of the tree *max_depth*, as well as the minimum samples per leaf requirement *min_samples*, indicating that a leaf should be created when either a node reaches the maximally allowed depth or contains less samples than the minimally required samples per leaf. The overall procedure to create the tree is described in Algorithm 5.

Some remarks regarding Algorithm 5:

- While the feature matrices $(X_k)_{k=1}^K$, as well as the response vectors $(y_k)_{k=1}^K$ are denoted as input, they of course never leave the server of the respective party and are only used for calculating federated differentially private sums.
- As we add noise to the bin counts of the federated differentially private histograms, determining the number of samples inside a node leads to a different value when using different features. Therefore, checking the minimum samples per leaf condition is performed on all features. The same goes for the test whether we only have samples from one class inside a node.
- The parameter *leaf_prop* specifies the proportion of the privacy budget that we allocate to the leaf labelling. It is generally not obvious how to distribute privacy budget to the sub-parts of the algorithm, in order to enhance its performance. Further insights to this problem can be found in Section 3.6.

3.3.4 Operations across Features - Parallel or Sequential Composition?

In Algorithm 5, when calculating federated differentially private histograms for a single feature j, we allocate a privacy budget of ϵ_{node}/p because we do not treat this operation as parallel composition across features. However, other work such as PrivaTree from Vos et al. [23] consider this operation as parallel composition across the features. Although one can justify using parallel composition by viewing different features as disjoint subsets of the data, releasing histograms of two features provides more information about individual datapoints than releasing histograms of only one feature. For example, it is easier to determine whether a specific datapoint was involved in the computations if we have the histograms of two features, compared to having only access to the histogram of one feature. However, using parallel composition across features corresponds to not spending any additional budget for the second feature after the computation of the first

Algorithm 5 Federated Differentially Private Tree

Input: Feature matrices $(X_k)_{k=1}^K$ with p features and binary response vectors $(y_k)_{k=1}^K$ across parties, maximal depth allowance max_depth, minimum samples per leaf $min\_samples\_leaf$, number of bins num_bins, privacy budget ϵ, leaf privacy budget proportion $leaf\_prop$.

$\epsilon_{\text{leaf}} \leftarrow leaf\_prop \cdot \epsilon$.
$\epsilon_{\text{node}} \leftarrow (1 - leaf\_prop) \cdot \epsilon / max\_depth$.
procedure FitTreeRecursive($(X'_k)_{k=1}^K$, $(y'_k)_{k=1}^K$, $depth$)
  if $depth \geq max\_depth$ **then**
    return CreateLeaf($(y'_k)_{k=1}^K$) ▷ with budget ϵ_{leaf}
  end
  for j ***in*** *features* **do**
    Use Algorithm 3 to calculate private histograms H_j^0 and H_j^1 for each class. ▷ with budget ϵ_{node}/p
  end
  if $sum(H_j^0) + sum(H_j^1) < min\_samples\_leaf \ \forall j$ **then**
    return CreateLeaf($(y'_k)_{k=1}^K$) ▷ with budget ϵ_{leaf}
  end
  if $sum(H_j^0) \leq 0 \ \forall j$ ***or*** $sum(H_j^1) \leq 0 \ \forall j$ **then**
    return CreateLeaf($(y'_k)_{k=1}^K$) ▷ with budget ϵ_{leaf}
  end
  Find split $\theta^* = (j^*, t^*)$ that minimizes the impurity based on $(H_j^0)_{j=1}^p$ and $(H_j^1)_{j=1}^p$ with Algorithm 4.
  Every party partitions X'_k into $X_k'^{\text{left}}$ and $X_k'^{\text{right}}$ and y'_k into $y_k'^{\text{left}}$ and $y_k'^{\text{right}}$ according to θ^*.
  $\mathcal{T}^{\text{left}} \leftarrow$ FitTreeRecursive($(X_k'^{\text{left}})_{k=1}^K$, $(y_k'^{\text{left}})_{k=1}^K$, depth + 1).
  $\mathcal{T}^{\text{right}} \leftarrow$ FitTreeRecursive($(X_k'^{\text{right}})_{k=1}^K$, $(y_k'^{\text{right}})_{k=1}^K$, depth + 1).
  return Node($\theta^*, \mathcal{T}^{\text{left}}, \mathcal{T}^{\text{right}}$)
end procedure
return FitTreeRecursive($(X_k)_{k=1}^K$, $(y_k)_{k=1}^K$, depth = 0)

Output: A federated ϵ-differentially private decision tree.

feature, or equivalently it corresponds to considering the released information of two histograms to be the same as of one histogram.

Mathematically, let us denote $\mathcal{A}$ as an ϵ-differentially private algorithm for computing the histograms of a feature, and X_j as the j-th feature vector, representing the j-th column of the feature matrix X. Adding or removing a single datapoint corresponds to adding or removing a row in X. If we denote X' as the feature matrix obtained by adding or removing a datapoint, each column (feature) X'_j of X' can be obtained by adding or removing a single entry in the corresponding column (feature) X_j of X. Therefore, due to the ϵ-differential privacy of Algorithm $\mathcal{A}$, we have for any outcome H_j of $\mathcal{A}$ the inequality:

$$\mathbb{P}\left(\mathcal{A}(X_j) = H_j\right) \leq e^{\epsilon}\mathbb{P}\left(\mathcal{A}(X'_j) = H_j\right). \tag{2}$$

In total, for p independent copies $\mathcal{A}_1, \ldots, \mathcal{A}_p$ of algorithm $\mathcal{A}$ to compute the differentially private histograms across all features, we have:

$$\begin{aligned}
&\mathbb{P}\left(\mathcal{A}(X_1) = H_1, \ldots, \mathcal{A}(X_p) = H_p\right) \\
&= \prod_{j=1}^{p} \mathbb{P}\left(\mathcal{A}(X_j) = H_j\right) && \text{(Independence of } \mathcal{A}_1, \ldots, \mathcal{A}_p\text{)} \\
&\overset{2}{\leq} \prod_{j=1}^{p} e^{\epsilon}\mathbb{P}\left(\mathcal{A}(X'_j) = H_j\right) \\
&= e^{p\epsilon}\mathbb{P}\left(\mathcal{A}(X'_1) = H_1, \ldots, \mathcal{A}(X'_p) = H_p\right). && \text{(Independence of } \mathcal{A}_1, \ldots, \mathcal{A}_p\text{)}
\end{aligned}$$

As in the definition of differential privacy, the probabilities above are always taken over the randomization of the algorithms, so the data remain fixed.

Note that we observe Inequality 2 because adding or removing one datapoint from X changes an entry in every column, making X_j and X'_j neighbouring datasets for all features. If, however, we consider a disjoint

partition of X across the samples, adding or removing one datapoint from X only affects one partition, allowing us to use parallel composition.

In conclusion, the above arguments justify that the sequential composition may be more suitable than parallel composition for repeating operations across features. So in this work, we use parallel composition only when we have disjoint subsets across the samples and employ sequential composition for repeating operations across the features to guarantee the privacy of the individual datapoints. This comes with a cost where, especially in the case of a high dimensional feature space and relatively low number of samples, we need to use higher privacy budgets to reach a certain accuracy compared to methods that use parallel composition for operations across the features.

3.4 Proof of Differential Privacy

This Section is dedicated to provide a proof for the differential privacy of Algorithm 5. We use the same notations as in Algorithm 5 for the parameters.

Theorem 6. *Algorithm 5 satisfies ϵ-differential privacy.*

Proof. First, let us analyze the privacy budget consumption of one node. Algorithm 5 computes two private histograms for each feature, one for the samples belonging to class 0 and one for the samples belonging to class 1. By Lemma 3.1, Algorithm 3 with a privacy budget of ϵ_{node}/p ensures differential privacy for each of these two histograms. As separating the samples by class creates two disjoint subsets of the data, the computation of the two histograms leads, due to parallel composition, to a privacy budget consumption of ϵ_{node}/p. Repeating this step for all p features leads to a total privacy consumption of ϵ_{node} due to the sequential composition. As the calculation of the impurities for all candidate splits only uses the information from the private histograms, this step is post processing and does not consume any additional privacy budget. Hence the calculations from one single node consume a privacy budget of ϵ_{node}.

Secondly, we study the privacy budget consumption of a leaf. As this corresponds to just applying Algorithm 3 with privacy budget ϵ_{leaf} and 2 bins to the labels falling into that leaf, this operation consumes a privacy budget of ϵ_{leaf} by Lemma 3.1.

Now, we consider the privacy budget consumption of one branch (a path from the root node to a leaf) of our decision tree. As the length of one branch is bounded by the parameter max_depth, using sequential composition (because the nodes of one branch repeatedly use the same part of the data), we get that the privacy budget consumption of one branch can be bound by

$$max\_depth \cdot \epsilon_{\text{node}} + \epsilon_{\text{leaf}} = max\_depth \frac{(1 - leaf\_prop) \cdot \epsilon}{max\_depth} + leaf\_prop \cdot \epsilon = \epsilon. \tag{3}$$

To conclude, we observe that two different branches operate on disjoint subsets of the data as soon as they separate from each other. Hence, due to parallel composition, it is enough to consider the privacy budget consumption of one branch, which is bound by ϵ according to Equation 3. □

In every federated query, especially federated sums in our case, we protect the output with distributed differential privacy, ensuring the privacy guarantees of our model extend between different parties. While there is a small potential privacy loss, as each party can subtract the partial noise they added from the output, this generally does not compromise the protection between parties in our model [13]. Moreover, as the number of parties K increases, the impact of partial noises relative to the aggregated noise diminishes.

3.5 Lower Bound for Split Impurities to Save Privacy Budget

As observed in Section 3.3.3, the total privacy budget scales with the number of features when fixing the budget available for computing the private histograms of a feature. This is not an ideal property, as in many applications we have a high dimensional feature space. Especially combined with a low sample size, this property could result into poor performance of Algorithm 5. So in this section, the goal is to remove some features from the candidate splits before we create their private histograms. This allows us to spend less privacy budget in the computations of a node, as we can save the privacy budget we would need to spend for the computations on the now removed features.

Our approach is to calculate a lower bound for the best split impurity of a feature j, such that any possible threshold on that feature will have a larger impurity than this lower bound. Thus, if we find a split on any feature $i \neq j$ whose impurity underestimates the lower bound of feature j, we do not need to consider feature j anymore in the current node and therefore do not need to spend privacy budget for computations on that feature (i.e. the computation of the private histograms).

Theorem 7. *For a node Q consisting of n samples and for a feature j, the impurity of any candidate split $\theta = (j, t)$ on the j-th feature satisfies*

$$G(Q, \theta) \geq \frac{1}{n} \sum_{k=1}^{K} n_k G(Q_k, \theta_{kj}^*),$$

where Q_k denotes the samples from node Q that belong to the k-th party and θ_{kj}^* is the split that minimizes the impurity of node Q_k on the j-th feature.

Proof. First, let us denote that we have

$$H(Q) = \hat{p}(1 - \hat{p}) = \min_a \frac{1}{n} \sum_{i:x_i \in Q} (y_i - a)^2, \tag{4}$$

for any node Q with n samples. Now we can calculate for any split $\theta = (j, t)$:

$$\begin{aligned}
G(Q, \theta) &= \frac{n^{\text{left}}}{n} h(Q^{\text{left}}(\theta)) + \frac{n^{\text{right}}}{n} h(Q^{\text{right}}(\theta)) \\
&\overset{4}{=} \frac{1}{n} \left(\min_{a^{\text{left}}} \sum_{i:x_i \in Q^{\text{left}}(\theta)} (y_i - a^{\text{left}})^2 + \min_{a^{\text{right}}} \sum_{i:x_i \in Q^{\text{right}}(\theta)} (y_i - a^{\text{right}})^2 \right) \\
&= \frac{1}{n} \left(\min_{a^{\text{left}}} \sum_{k=1}^{K} \sum_{i:x_i \in Q_k^{\text{left}}(\theta)} (y_i - a^{\text{left}})^2 + \min_{a^{\text{right}}} \sum_{k=1}^{K} \sum_{i:x_i \in Q_k^{\text{right}}(\theta)} (y_i - a^{\text{right}})^2 \right) \\
&\geq \frac{1}{n} \left(\sum_{k=1}^{K} \min_{a_k^{\text{left}}} \sum_{i:x_i \in Q_k^{\text{left}}(\theta)} (y_i - a_k^{\text{left}})^2 + \sum_{k=1}^{K} \min_{a_k^{\text{right}}} \sum_{i:x_i \in Q_k^{\text{right}}(\theta)} (y_i - a_k^{\text{right}})^2 \right) \\
&= \frac{1}{n} \sum_{k=1}^{K} \left(n_k^{\text{left}} h(Q_k^{\text{left}}(\theta)) + n_k^{\text{right}} h(Q_k^{\text{right}}(\theta)) \right) \\
&= \frac{1}{n} \sum_{k=1}^{K} n_k G(Q_k, \theta).
\end{aligned}$$

Now for any party $k \in \{1, \ldots, K\}$ we have that $G(Q_k, \theta) \geq G(Q_k, \theta_{kj}^*)$, where θ_{kj}^* is the split that minimizes the impurity of node Q_k on the j-th feature. Thus we can conclude

$$G(Q, \theta) \geq \frac{1}{n} \sum_{k=1}^{K} n_k G(Q_k, \theta_{kj}^*)$$

for any split $\theta = (j, t)$ on the j-th feature.

□

Equivalently to Theorem 7, we can state the inequality as

$$nG(Q, \theta) \geq \sum_{k=1}^{K} n_k G(Q_k, \theta_{kj}^*),$$

such that the summands do not depend on the overall number of samples in node Q and compare splits by the impurity scaled by the number of samples in a node. We use this form in the implementation described in Algorithm 6.

The key point of the inequality in Theorem 7 is that the summands of the lower bound depend only on data from one single party and can be computed locally. Subsequently, we apply Algorithm 1 to compute the private sum and utilize distributed differential privacy with the Laplace mechanism, using the gamma DLPA introduced in Section 2.4.1. The sensitivity of this operation can be bounded by 5/4, as shown in Lemma 3.2 (we only need to analyze the sensitivity of one summand, as adding or removing one datapoint from the data only affects one party). We could get a tighter bound, however, it is important that this bound does not depend on any local information of a party.

Lemma 3.2. *For any party $k \in \{1, \dots, K\}$, the sensitivity of the computation of $n_k G(Q_k, \theta^*_{kj})$, where θ^*_{kj} is the split that minimizes the impurity of node Q_k on the j-th feature, can be bound by* $5/4$.

Proof. Let $\theta_k = (j, t)$ be an arbitrary split of node Q_k with n_k^{left} samples smaller than t and n_k^{right} samples larger than t. W.l.o.g. we assume that we add one datapoint on the left side of t, and denote $Q_{k,\text{new}}$ as the resulting node. This yields

$$\hat{p}^{\text{left}}_{k,\text{new}} = \frac{n_k^{\text{left}}}{n_k^{\text{left}} + 1} \hat{p}_k^{\text{left}} + \begin{cases} 0, & \text{if we add one point with label 0} \\ \dfrac{1}{n_k^{\text{left}} + 1}, & \text{if we add one point with label 1,} \end{cases}$$

so we have using $0 \leq \hat{p}_k^{\text{left}} \leq 1$

$$|\hat{p}_k^{\text{left}} - \hat{p}^{\text{left}}_{k,\text{new}}| \leq \frac{1}{n_k^{\text{left}} + 1}. \tag{5}$$

Hence we can calculate

$$\begin{aligned} &|n_k^{\text{left}} h(Q_k^{\text{left}}(\theta)) - (n_k^{\text{left}} + 1) h(Q^{\text{left}}_{k,\text{new}}(\theta))| \\ &\leq n_k^{\text{left}} |\hat{p}_k^{\text{left}}(1 - \hat{p}_k^{\text{left}}) - \hat{p}^{\text{left}}_{k,\text{new}}(1 - \hat{p}^{\text{left}}_{k,\text{new}})| + \underbrace{|\hat{p}^{\text{left}}_{k,\text{new}}(1 - \hat{p}^{\text{left}}_{k,\text{new}})|}_{\leq \frac{1}{4}} \qquad (\triangle\text{-inequality}) \\ &\overset{5}{\leq} n_k^{\text{left}} \left(\frac{1}{n_k^{\text{left}} + 1} - \left(\frac{1}{n_k^{\text{left}} + 1} \right)^2 \right) + \frac{1}{4}, \end{aligned}$$

and thus in total we get

$$|n_k G(Q_k, \theta) - (n_k + 1) G(Q_{k,\text{new}}, \theta)| \leq n_k^{\text{left}} \left(\frac{1}{n_k^{\text{left}} + 1} - \left(\frac{1}{n_k^{\text{left}} + 1} \right)^2 \right) + \frac{1}{4} \leq \frac{5}{4}.$$

The above calculations are similar in the case where we remove (instead of add) one datapoint.

So we showed that the sensitivity of the (scaled) impurity of any split θ_k can be bound by $5/4$. Thus the (scaled) impurity of the split θ^*_{kj} that minimizes the impurity of node Q_k on feature j can maximally change by $5/4$. □

Now we use these lower bounds for the impurity of every feature as follows: We split the privacy budget of a node into ϵ_{bounds} and ϵ_{hists} such that $\epsilon_{\text{node}} = \epsilon_{\text{bounds}} + \epsilon_{\text{hists}}$. Next, we sort them in ascending order and create the private histograms for the feature corresponding to the lowest lower bound using Algorithm 3 and budget $\epsilon_{\text{hists}}/p$. Then, we compute the minimal impurity for this binned feature using Algorithm 4. If this impurity underestimates any lower bound of another feature, we remove this feature from further computations in this node (creating private histograms), allowing us to save a budget of $\epsilon_{\text{hists}}/p$ per removed feature. We repeat this procedure for the feature with the next lowest lower bound, until we have no more features left to look at. Letting q denote the number of features we were able to remove before the computations of their private histograms, we save a total budget of $\epsilon_{\text{saved}} = q\epsilon_{\text{hist}}/p$. We then pass down the fully saved budget to each of the child nodes, where we subsequently now have a total node budget of $\epsilon_{\text{node}} + \epsilon_{\text{saved}}$ available. Since the child nodes only use the extra budget not consumed in their parent node, the privacy budget of one branch of the tree does not change, and therefore we obtain ϵ-differential privacy by the same arguments as in the proof of Theorem 6 showing differential privacy of Algorithm 5.

Algorithm 6 describes the splitting procedure with the budget saving procedure described above. The final decision tree algorithm then works as in Algorithm 5, besides its node splitting procedure with the feature bounds and its ability to pass down the unused privacy budget to the child nodes.

This adaptation enables the tree to automatically distribute the privacy budget between nodes, allowing a node that is easier to split to pass down the non-needed budget to its child nodes instead of spending it all. This should lead to a more efficient usage of the privacy budget, which we verify in a simulation in Section 4. In this simulation, we analyze both the federated differentially private decision tree from Algorithm 5 and its adapted version with the budget saving mechanism introduced in Algorithm 6.

3.6 Distributing the Privacy Budget

Dividing the total privacy budget is a crucial aspect of differential privacy, which enables breaking down an algorithm into differentially private components using only a sub-part of the total privacy budget. Determining

Algorithm 6 Privacy Budget Saving Node Splitting

Input: Node Q, node privacy budget ϵ_{node}, saved privacy budget ϵ_{saved} from parent node, feature bounds privacy proportion $bounds\_prop$.

$\epsilon_{\text{node}} \leftarrow \epsilon_{\text{node}} + \epsilon_{\text{saved}}$.
$\epsilon_{\text{bounds}} \leftarrow bounds\_prop \cdot \epsilon_{\text{node}}$.
$\epsilon_{\text{hists}} \leftarrow (1 - bounds\_prop) \cdot \epsilon_{\text{node}}$.
for $j = 1, \ldots, p$ **do**
  Calculate using Algorithm 1 with distributed differential privacy and sensitivity $5/4$:
  $feature\_bounds[j] \leftarrow \sum_{k=1}^{K} n_k G(Q_k, \theta^*_{kj})$. ▷ with budget ϵ_{bounds}
  end
Sort $feature\_bounds$ in ascending order.
$feature\_removed[j] \leftarrow$ False $\forall j = 1, \ldots, p$.
$\epsilon_{\text{saved}} \leftarrow 0$.
$best\_impurity \leftarrow$ inf.
for *bounds in* $feature\_bounds$ **do**
  Find feature j corresponding to *bounds*.
  if $feature\_removed[j]$ *is True* **then**
    $\epsilon_{\text{saved}} \leftarrow \epsilon_{\text{saved}} + \epsilon_{\text{hists}}/p$.
    continue
  end
  Compute private histograms of j-th feature with Algorithm 3. ▷ with budget $\epsilon_{\text{hists}}/p$
  Sum up bin counts to find n.
  Find split $\theta^*_j = (j, t)$ that minimizes the impurity on the j-th feature using Algorithm 4.
  if $G(Q, \theta^*_j) < best\_impurity$ **then**
    $best\_impurity \leftarrow G(Q, \theta^*_j)$.
    $best\_split \leftarrow \theta^*_j$.
  end
  Set $feature\_removed$ to True for all features l with $feature\_bounds[l] > n \cdot best\_impurity$.
  end
Create child nodes $Q^{\text{left}}, Q^{\text{right}}$ according to $best\_split$.

Output: Child nodes $Q^{\text{left}}, Q^{\text{right}}$, saved budget ϵ_{saved}.

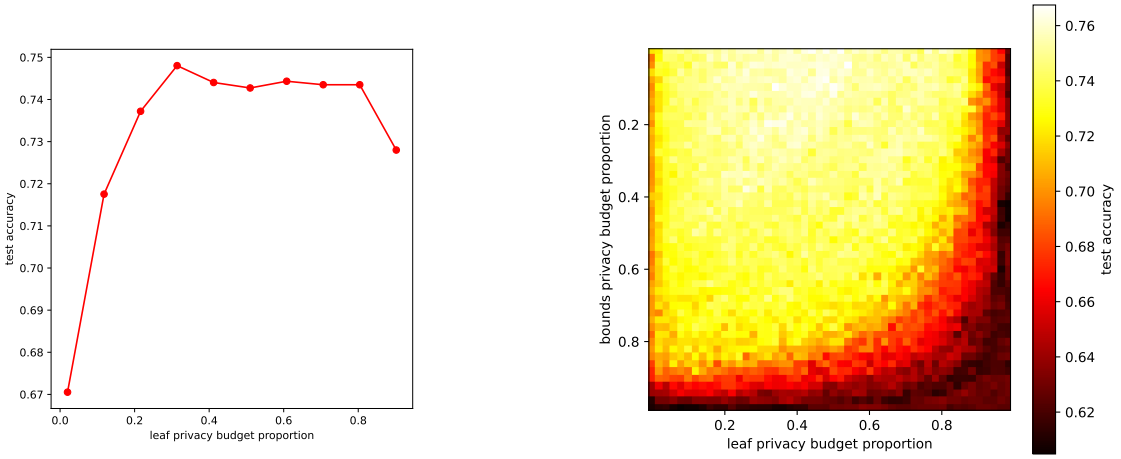

Figure 2: Left: Test accuracy vs leaf privacy budget proportion for our federated differentially private decision tree without budget saving node splitting. Right: Heatmap of test accuracy for our federated differentially private decision tree with budget saving node splitting. For both plots, the data is generated from a Gaussian mixture model with 300 samples and 10 features, and accuracy scores were averaged over 50 runs.

the optimal allocation of the privacy budget among these components to enhance the utility of the results is often not straightforward. In our case, we divide the total budget ϵ into ϵ_{leaf} and ϵ_{node}, controlled by the parameter $leaf\_prop$ specifying the proportion of ϵ used to protect the leaves. In the case where we use budget saving node splitting as in Algorithm 6, we additionally divide the node budget ϵ_{node} into ϵ_{bounds} and ϵ_{hists}, controlled by the parameter $bounds\_prop$ specifying the proportion of ϵ_{node} used to protect the impurity bounds of the features.

In Figure 2, we visualize the impact of different privacy budget allocations on the test accuracy for both with and without privacy budget saving node splitting. The test accuracies are averaged over 50 runs, and we use synthetic data from a Gaussian mixture model with 300 samples, 10 features and a correlation setting of $\rho = 0.3$. More details about the data can be found in Section 4.1.1. For the decision trees, we use a maximum depth allowance of 7, a minimal samples per leaf requirement of 10 and 10 bins for each feature.

We observe that in the case where we use Algorithm 5 without budget saving node splitting, a leaf privacy budget proportion between 0.3 and 0.8 leads to higher test accuracies than outside this range. With budget-saving node splitting from Algorithm 6, a combination of the two proportion parameters with at least one of them being large results in lower test accuracy.

Figure 2 primarily serves as an illustration, highlighting the impact of varying privacy budget allocations on test accuracy. Different datasets can yield considerably different outcomes. However, using half of the budget for leaf labeling seems generally reasonable, as correct leaf labels are crucial for good performance. In our following simulations, we set $leaf\_prop = 0.5$, and we use $bounds\_prop = 0.25$ consistently.

4 Experiments

In this section, we evaluate the performance of our federated differentially private decision tree from Algorithm 5 and its adaptation with privacy budget saving node splitting from Algorithm 6 on synthetic data split across $K = 5$ parties. Due to the latter's ability to automatically distribute unneeded budget to its child nodes, leading to more available budget, we expect this adaptation to outperform our federated differentially private tree without this budget saving mechanism.

We compare these two algorithms with a centralized decision tree fitted on the database containing the combined data from all parties, as well as a local decision tree fitted only with the data of one party. Both decision trees use the Gini entropy as in our algorithms, and share the same stopping criteria: maximum depth allowance and minimum samples per leaf requirement. We evaluate the performance of the algorithms with their test accuracies averaged over 50 rounds, where in each round we generate new training and test data

according to Section 4.1.1.

4.1 Experiment Configuration

4.1.1 Data Parameters

Letting n denote the number of samples, for each of the two classes we generate $n/2$ samples using a Gaussian mixture model with 5 components. For p features, each of these components has a covariance matrix Σ defined as $\Sigma_{ij} = \rho^{|i-j|}$, where $\rho \in [0, 1]$ specifies the correlation setting. The i-th component of class 0 has a mean vector $(i/5, \ldots, i/5) \in \mathbb{R}^p$, $i = 1, \ldots, 5$, while the mean vector of the i-th component corresponding to class 1 is the negative of the i-th mean vector of class 0. We use equal weights for the components of the mixture models. After sampling $n/2$ samples for each class, we shuffle the data and divide it into 5 subsets—one for each party, with each party holding $n/5$ randomly selected samples from the entire dataset

The test sets are always sampled from the same distribution as the training set and consist of the same number of samples n (with $n/2$ samples for each class).

4.1.2 Simulation Parameters

In the following figures, we consistently create 6 subplots with varying number of features $p \in \{10, 25, 50\}$ and maximum depth allowance $max\_depth \in \{5, 10\}$. The other tree parameters remain fixed throughout our simulations: We set the minimum number of samples per leaf requirement as well as the number of bins per feature to 10, the leaf privacy budget proportion to 0.5 and the bounds privacy budget proportion to 0.25. In the main part of this report, we use a sample size of 250, however in Appendix A we repeat the experiments in a low data volume setting with $n = 100$ and a high data volume setting with $n = 1000$. Lastly, for each data volume setting we always consider a high correlation scenario with $\rho = 0.9$ and a low correlation scenario with $\rho = 0.3$.

4.2 Results

Upon analyzing Figure 3, we first observe that our privacy budget saving version consistently yields higher accuracies than our federated differentially private decision tree without budget saving. This trend is particularly notable with an increasing number of features, as well as with an increasing maximal depth allowance. Hence it seems that, as expected, the budget saving version is able to handle the privacy budget more efficiently by distributing the non-needed budget down to the child nodes. With an increasing number of features, there are more possible features to remove from further computations inside a node, hence more possible budget to save. The same applies to an increasing number of maximal depth allowance, where the general availability of more nodes in a single branch leads to an increasing possible privacy budget to save. We conclude that, based on the underlying simulation results, the budget saving adaptation from Algorithm 6 of our federated differentially private decision tree successfully deals with the problem of high dimensional feature spaces described in Section 3.3.4, and furthermore also has beneficial properties for larger trees.

Secondly, we observe that our methods with differential privacy seem to perform better in the high correlation setting than in the low correlation setting, when comparing them with the local and centralized decision trees. In some cases, the budget saving method even yields higher test accuracies than the centralized decision tree. In that case, the noise we add to our models to satisfy differential privacy acts like a regularization term, preventing the differentially private models from overfitting the data, while the standard decision trees overfit at least part of the training data. As the high correlation scenario generally leads to lower test accuracies of the centralized and local decision trees, this phenomena is especially evident in that scenario.

While our method without privacy budget saving is often not able to beat the local decision tree in test accuracy, this is almost always the case for our budget saving differentially private decision tree after a certain privacy budget (usually between 0 and 2). Only in the case with 50 features, our budget saving method sometimes yields lower accuracies than the locally trained decision tree, or reaching the accuracy of our local tree at a rather high privacy budget. Beating the local version of the decision tree justifies the adoption of our method instead of training a local model. Additionally, our resulting federated decision tree satisfies differential privacy, whereas the local decision tree does not have that property.

The comparison between our federated differentially private decision trees with and without privacy budget saving yields the same results in the low (100 samples) and high (1000 samples) data volume settings in Appendix A: The budget saving method consistently yields higher test accuracies in all scenarios, again indicating that we handle the privacy budget more efficiently in our budget saving method. Furthermore, in both of these settings, our methods with differential privacy again perform better in the high correlation scenario when

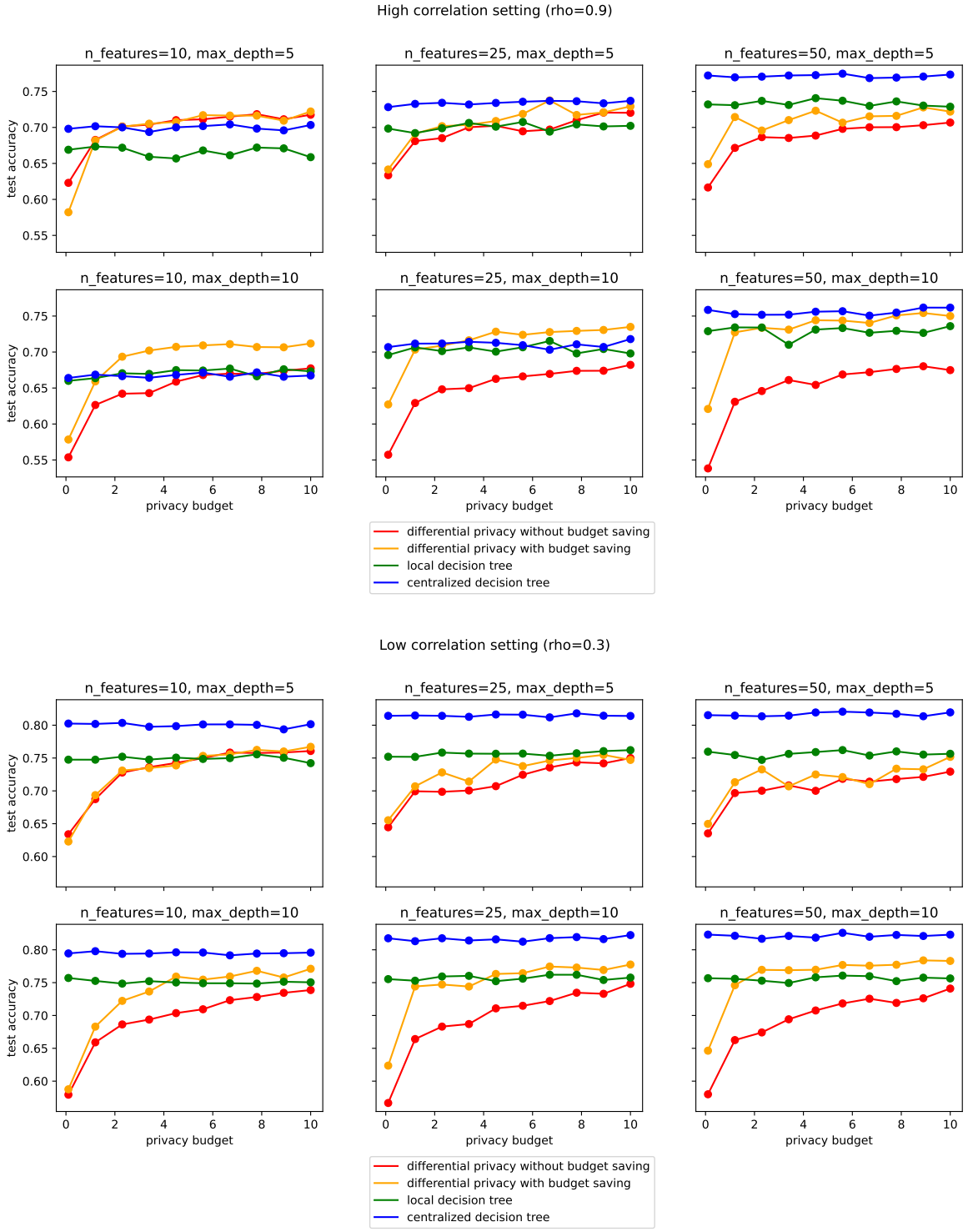

Figure 3: Test accuracy vs privacy budget with 250 samples averaged over 50 runs, comparing our federated differentially private decision trees with a local and centralized decision tree. The data comes from a Gaussian mixture model with 5 components for each class, where we consider a high correlation scenario (top) and a low correlation scenario (bottom).

comparing with the local and centralized decision trees, so we observe similar effects as in Figure 3 with 250 samples.

For the high data volume setting in Figure 4, we would expect a better performance of our methods with differential privacy: When we add noise to the bin counts to obtain private histograms, this has a relatively smaller impact on higher bin counts, which is often the case when the data availability is high in general. It seems reasonable to assume that hiding patterns of individual datapoints in a model fitted on a large training base is more feasible, indicating that differentially private models should yield better performance with larger data availability. In the high data volume with high correlation setting, the above considerations seem to hold, as especially our budget saving method yields much higher accuracies than the local decision tree for already low privacy budgets between 0 and 2, and only in the case where we have 50 features reaches the local decision tree accuracy a bit later. However, in the high data volume with low correlation setting, this is not the case anymore and our methods have difficulties in reaching the test accuracies of a local tree. In this scenario, across all settings, the local and centralized decision trees achieve the highest accuracies and the local model, with 200 samples out of 1000 across all 5 parties, may already form a robust model for this data. Consequently, the impact of noise addition to prevent overfitting might be less pronounced, explaining the performance gap between the differentially private methods and those without differential privacy in this scenario.

In the low data volume setting in Figure 5, our budget saving method usually reaches the local decision tree test accuracy for a privacy budget between 2 and 5. Although the low data availability challenges the methods with differential privacy due to the higher impact of the noise addition, the local decision trees only have 20 datapoints available in this setting and are thus not able to generalize well on the test data. Hence even with a low overall sample size, it could be an advantage to use our models with differential privacy, as the data of only one single party might not be enough to train a solid tree in this case.

5 Conclusion

This report introduces a new solution for federated differentially private decision trees, that could be particularly interesting for medical applications dealing with patient records due to its decentralized learning approach, privacy guarantees and interpretability. It was generally not obvious how to combine and apply the frameworks of federated learning and differential privacy to decision trees while trying keep the performance on a satisfying level. Our approach, introduced in Algorithm 5, bins every feature into federated private histograms and afterwards determines the split impurities of all candidate splits in a node based on these binned features. It turned out that higher dimensional feature spaces lead to, due to the sequential composition property of differential privacy, a less efficient usage of the privacy budget. Therefore we introduced an adaptation of our federated differentially private decision tree framework in Section 3.5, enabling the nodes to save unneeded privacy budget by removing features from further computations, through using a lower bound estimation of the Gini impurity for every feature. Afterwards, the nodes can then hand down this saved budget to their child nodes.

In an evaluation with synthetic data, it turned out that this adaptation with its ability to save and redistribute privacy budget strictly improves our first introduced federated differentially private decision tree based on private histograms, especially in high dimensional feature spaces and with increasing maximal depth allowance. In most cases, after a certain threshold for the privacy budget which is usually between 0 and 2, this privacy budget saving adaptation yields a higher test accuracy than a local decision tree trained with only data from one party, justifying the adoption of our method compared to training a local decision tree. In the case where the sample sizes from the individual parties are too small to train a model that is able to generalize well on the test data, our methods yield the best test accuracies when comparing them to a locally trained decision tree. In that case, they are a viable alternative to the local decision tree to increase performance, as well as to a centralized decision tree, trained on the combined data, to ensure privacy.

Appendix A More Simulation Results

A.1 High Data Volume Setting

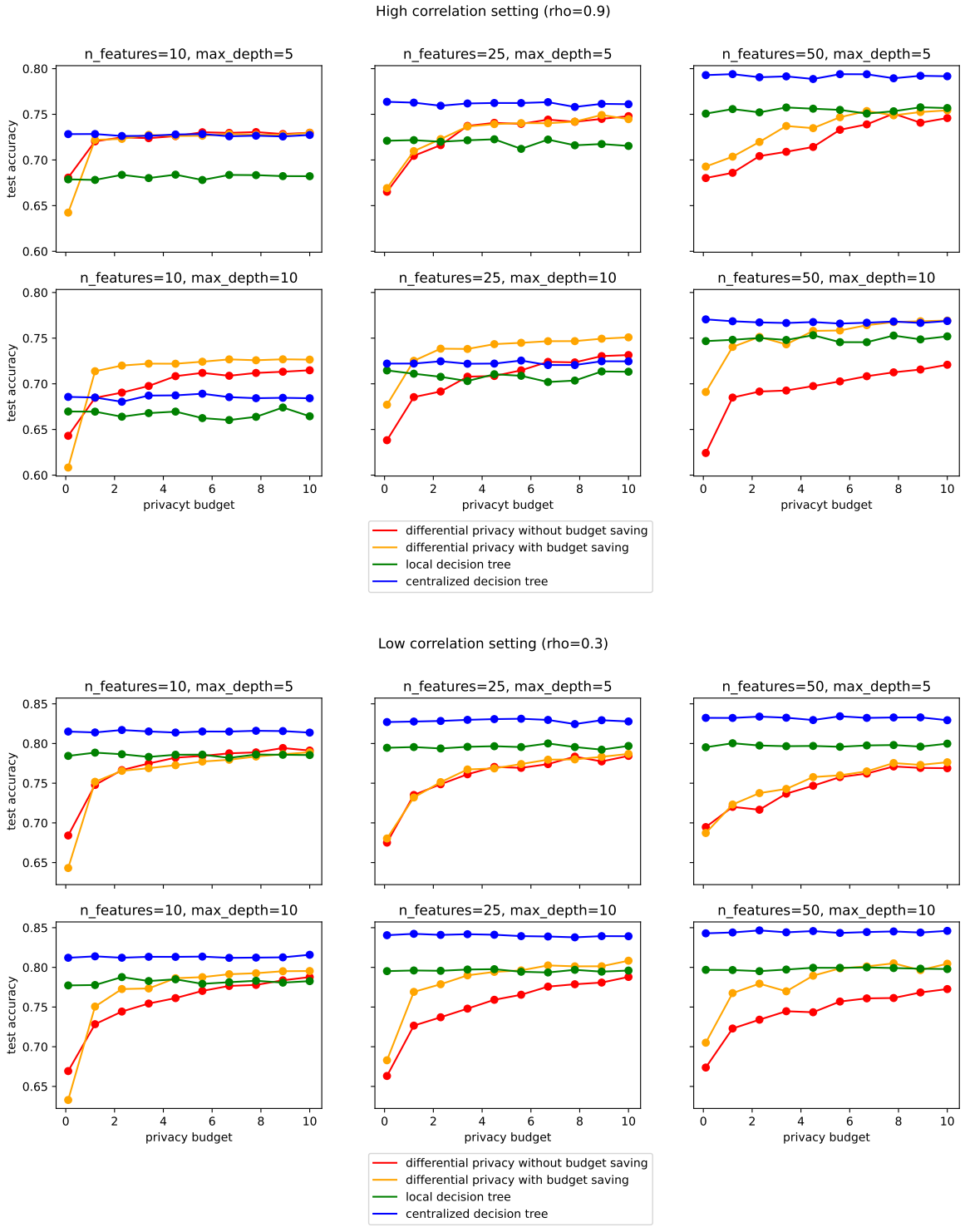

Figure 4: Test accuracy vs privacy budget with 1000 samples averaged over 50 runs, comparing our federated differentially private decision trees with a local and centralized decision tree. The data comes from a Gaussian mixture model with 5 components for each class, where we consider a high correlation scenario (top) and a low correlation scenario (bottom).

A.2 Low Data Volume Setting

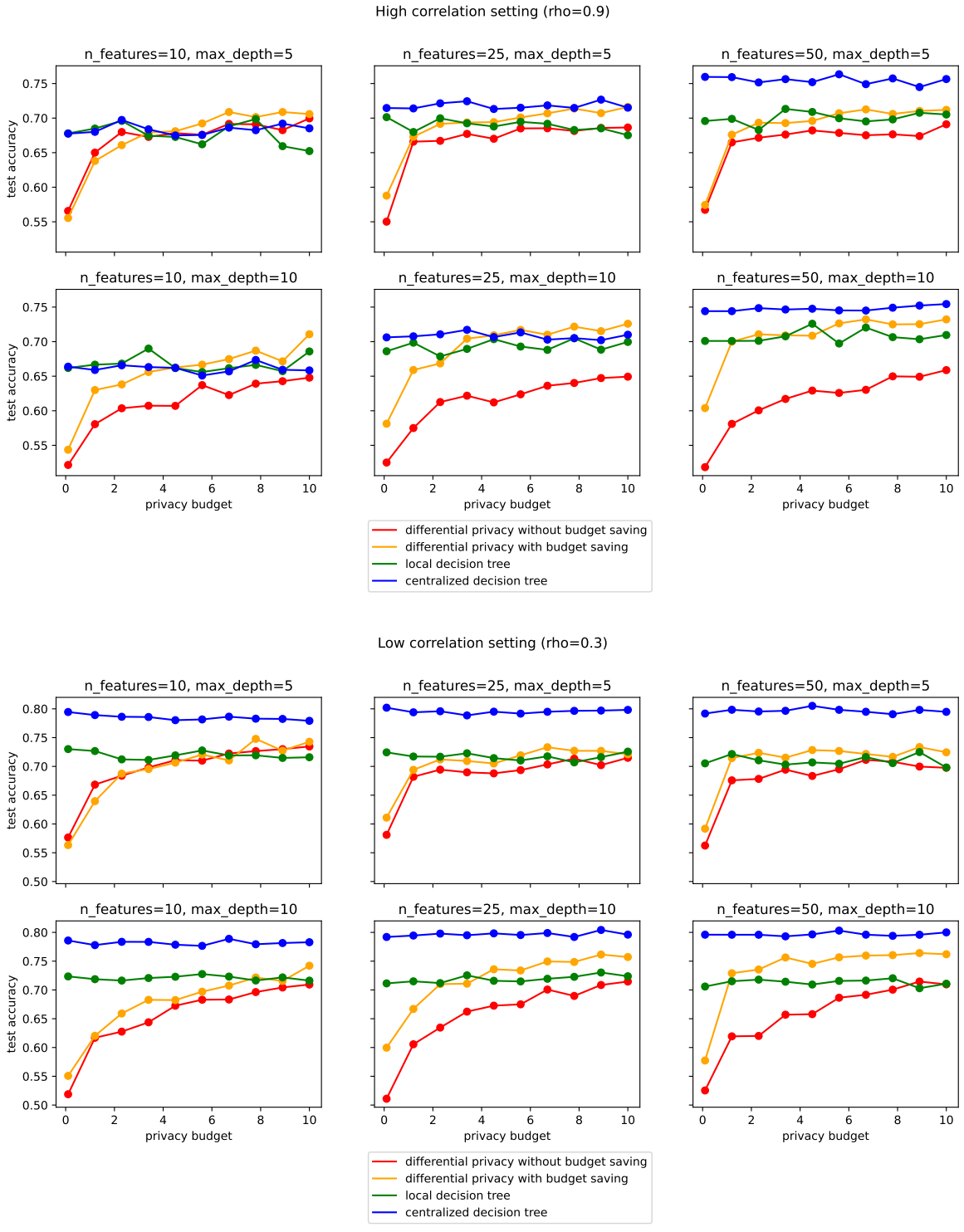

Figure 5: Test accuracy vs privacy budget with 100 samples averaged over 50 runs, comparing our federated differentially private decision trees with a local and centralized decision tree. The data comes from a Gaussian mixture model with 5 components for each class, where we consider a high correlation scenario (top) and a low correlation scenario (bottom).

References

[1] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[2] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 265–284. Springer, 2006.

[3] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[4] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

[5] Benjamin Letham, Cynthia Rudin, Tyler H McCormick, and David Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. 2015.

[6] Jiawei Han, Micheline Kamber, and Data Mining. Concepts and techniques. *Morgan kaufmann*, 340:94104–3205, 2006.

[7] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[8] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019.

[9] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30, 2009.

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

[11] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[12] Claude Castelluccia, Einar Mykletun, and Gene Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *The second annual international conference on mobile and ubiquitous systems: networking and services*, pages 109–117. IEEE, 2005.

[13] Mikko A Heikkilä, Antti Koskela, Kana Shimizu, Samuel Kaski, and Antti Honkela. Differentially private cross-silo federated learning. *arXiv preprint arXiv:2007.05553*, 2020.

[14] Whitfield Diffie and Martin E Hellman. New directions in cryptography. In *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, pages 365–390. 2022.

[15] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*, pages 486–503. Springer, 2006.

[16] Peter Kairouz, Ziyu Liu, and Thomas Steinke. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *International Conference on Machine Learning*, pages 5201–5212. PMLR, 2021.

[17] Naman Agarwal, Peter Kairouz, and Ziyu Liu. The skellam mechanism for differentially private federated learning. *Advances in Neural Information Processing Systems*, 34:5052–5064, 2021.

[18] Samuel Kotz, Tomasz Kozubowski, and Krzysztof Podgórski. *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*. Number 183. Springer Science & Business Media, 2001.

[19] Gergely Ács and Claude Castelluccia. I have a dream!(differentially private smart metering). In *International Workshop on Information Hiding*, pages 118–132. Springer, 2011.

[20] Richard Nock and Wilko Henecka. Boosted and differentially private ensembles of decision trees. *arXiv preprint arXiv:2001.09384*, 2020.

[21] Samuel Maddock, Graham Cormode, Tianhao Wang, Carsten Maple, and Somesh Jha. Federated boosted decision trees with differential privacy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2249–2263, 2022.

[22] Lingchen Zhao, Lihao Ni, Shengshan Hu, Yaniiao Chen, Pan Zhou, Fu Xiao, and Libing Wu. Inprivate digging: Enabling tree-based distributed data mining with differential privacy. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2087–2095. IEEE, 2018.

[23] Daniël Vos, Jelle Vos, Tianyu Li, Zekeriya Erkin, and Sicco Verwer. Differentially-private decision trees with probabilistic robustness to data poisoning. *arXiv preprint arXiv:2305.15394*, 2023.

[24] Yamane El Zein, Mathieu Lemay, and Kvin Huguenin. Privatree: Collaborative privacy-preserving training of decision trees on biomedical data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2023.

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

[26] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, and Yann LeCun. Differentially-and non-differentially-private random decision trees. *arXiv preprint arXiv:1410.6973*, 2014.

[27] Sam Fletcher and Md Zahidul Islam. Differentially private random decision forests using smooth sensitivity. *Expert systems with applications*, 78:16–31, 2017.

[28] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[29] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[30] Zhihua Tian, Rui Zhang, Xiaoyang Hou, Jian Liu, and Kui Ren. Federboost: Private federated learning for gbdt. *arXiv preprint arXiv:2011.02796*, 2020.