Federated Computation Beyond Learning

Graham Cormode

Federated Computation

- Privacy-preserving computations distributed over collections of users at web scale
  - Can be thought of as “A privacy-preserving MapReduce”
- Data stays on client devices, only sufficient statistics are shared: data minimization, purpose limitation
- Additional privacy comes from (local or central) differential privacy and secure multiparty computation
- Federated Computation has been widely adopted in practice (Google, Apple, Meta, etc.)
- Often combined with Differential Privacy (DP) to obtain formal privacy guarantees

There's more to life than learning…

Most work has been on Federated Learning (FL) which concentrates on the core training procedure

- Typically, via collection of gradients from batches of clients over multiple epochs

A complete end-to-end federated ML solution has many additional steps:

- Feature selection
- Feature normalization
- Model calibration and evaluation
- Model maintenance / checking

These steps share the same privacy concerns as the core FL training

We badge these as "Federated Computation Beyond Learning"

Outline

Overview of three recent works looking at different aspects of federated computation:

- Baseline federated pre-training for feature selection (CCS 2022)
- Post-processing: federated AUC measurement (VLDB 2023)
- Synthetic Data Generation in the Federated Setting (KDD 2024)

Open problems and future work

Federated Boosted Decision Trees with Differential Privacy

Samuel Maddock, Graham Cormode, Tianhao Wang, Carsten Maple, Somesh Jha

Federated Boosted Decision Trees

- Goal: To develop accurate, lightweight GBDT methods for the federated setting, under DP
- Motivation:
  - “Simple” baseline models that could be used in the federated (i.e., distributed and private) setting
  - Features chosen by decision trees are most important: addresses feature selection
- Some prior work has looked at related questions:
  - Existing private tree-based methods focus on central DP and only on decision trees (DTs) or RFs
  - Existing federated GBDT methods lack DP
    - Opportunity to study private boosting and adopt tight privacy accounting methods like Rényi DP
- Horizontal Federated model : each client holds some data over all features

Recap: Gradient Boosted Decision Trees (GBDTs)

Decision Trees (DTs): Hierarchical model based on binary splits

- Each internal node splits the dataset depending on a condition e.g., age <= 30
- Each leaf node of the tree contains a weight for prediction

Random Forests (RFs):

- Build an ensemble of independent DTs in parallel
- RF predicts the average or majority vote of each tree

Gradient-boosted Decision Trees (GBDT):

- 'Boost' a set of weak learners by building models iteratively
- Trees are built (sequentially) at each round on residuals from previous rounds

Many popular centralized GBDT implementations: XGBoost, LightGBM, CatBoost

Private GBDT Framework

Break the GBDT algorithm into 3 main components:

- (C1) Split Method: “Decide which split to pick”
- (C2) Weight Update Method: “Computing the leaf weights”
- (C3) Split Candidate Method: “Computing candidate splits”

All three require querying the data (with DP noise for privacy)

Algorithm 1 General GBDT

Input: Number of trees T, maximum depth d, number of split candidates Q, privacy parameters ϵ, δ

1: For each feature $j = 1, \ldots, m$ generate Q split candidates $S_j := \{s_1^j, \ldots, s_Q^j\}$ **(C3)**
2: Initialise the forest $\mathcal{T} \leftarrow \emptyset$
3: **for** $t = 1, \ldots, T$ **do**
4: For each $(x_i, y_i) \in D$ compute the required gradient information (g_i, h_i) based on $\hat{y}_i^{(t-1)}$ **(C2)**
5: Choose a subset of features $F^{(t)} \subseteq \{1, \ldots, m\}$ with $|F^{(t)}| = k$ for the current tree f_t **(A1)**
6: **while** depth of the current node $\leq d$ **do**
7: Choose a feature split candidate pair (j, s_q^j) from $F^{(t)}$ **(C1)**
8: Split the current node with observations I into two child nodes with index sets $I_{\leq} = \{i : x_{ij} \leq s_q^j\}$ and $I_{>} = I \setminus I_{\leq}$
9: Repeat (6)-(9) recursing separately on the child nodes
10: For each leaf l calculate a weight $w_l^{(t)}$ from the examples in the leaf according to the chosen update method **(C2)**
11: Update predictions $\hat{y}_i^{(t)}$ or batch updates **(A2)**
12: Add the tree to the ensemble $\mathcal{T} = \mathcal{T} \cup \{f_t\}$
13: **return** the trained forest $\mathcal{T}$

C1: Split Methods

- Histogram-based (Hist):
  - Compute a (private) histogram over split-candidates and use this to find split-scores at each level
  - Scores can be computed easily under secure aggregation + DP
- Partially Random (PR):
  - Pick a random split-candidate for each feature and compute split scores
- Totally Random (TR):
  - Pick a feature, split-candidate pair completely at random
  - Requires only perturbing leaf weights

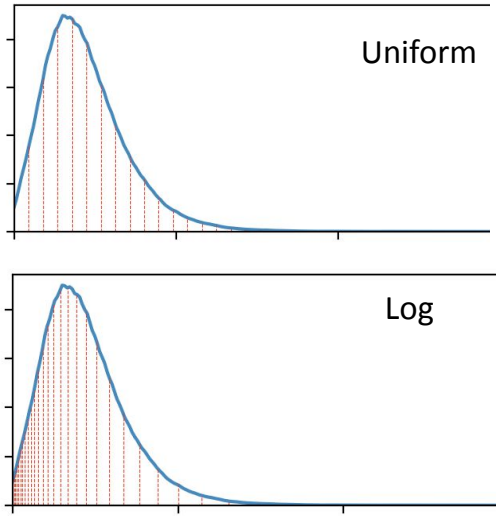
C2: Weight Updates

- Averaging: (zeroth order) compute class probabilities of leaf nodes and average across each tree
  - Leads to Random Forest updates
- Gradient: (first order) just use gradient information g_i to update weights
  - Leads to standard GBDT updates
- Newton: (second-order) use both gradient and Hessians g_i, h_i to compute weights
  - Leads to XGBoost updates

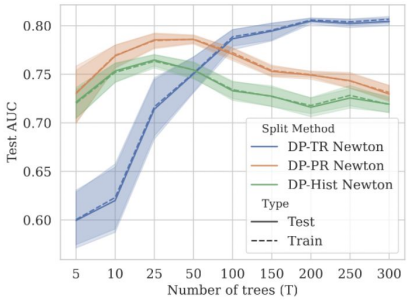
All three can be done via a single aggregation operator that computes a (secure, private) vector sum

C3: Split Candidates

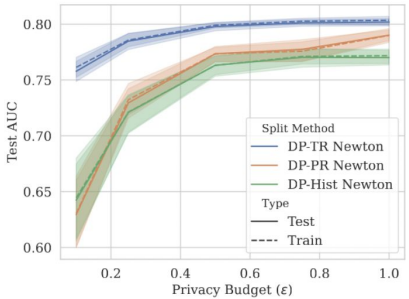
- Quantiles (non-private): Standard method used for GBDTs
- Uniform: Uniformly divide up features over their range
- Log: Divide uniformly over the log of a feature
- Iterative Hessian (IH):
  - Mimics XGBoost to form quantile sketches where Hessians are used as weights
  - Forms a private Hessian histogram over current split candidates by iteratively splitting/merging bins over s steps



Results C1 & C2: Split Methods and Weight Updates



(a) Varying T with $d = 4, \epsilon = 1$



(c) Varying ϵ

Varying split-methods on benchmark Credit 1 data: Histogram, Totally random (TR), Partially Random (PR)

Conclusions:

- TR competitive but typically requires many trees to get better results than histogram
- PR helps but still performs worse than TR

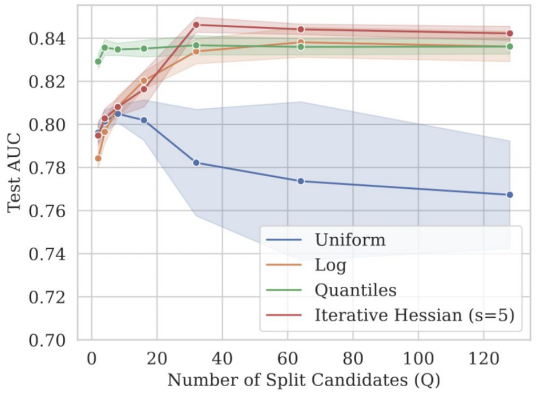
Results C3: Split Candidates

Methods: Uniform, Log, Quantiles (non-private), IH

- For datasets with skewed attributes, IH performs best
  - For more candidates, uniform splits lose performance

Conclusion:

- Refining split-candidates over rounds can help
- Only a small amount of budget needed for good results



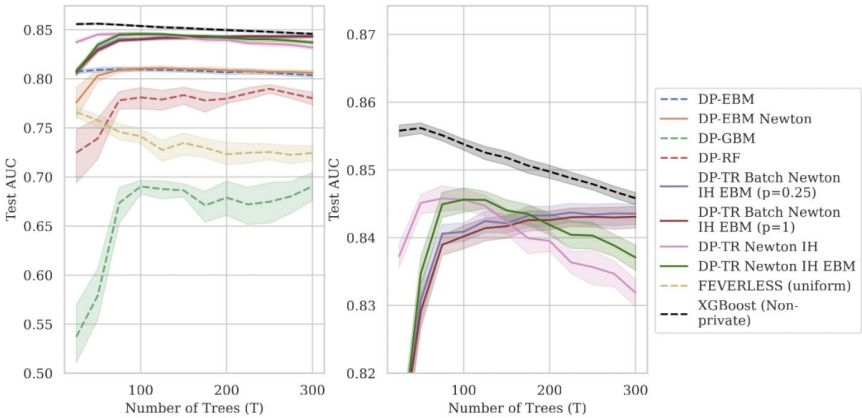
(c) Varying Q with $T = 100, d = 4, \epsilon = 1$

End-to-End Comparison

- Methods that replicate the centralized algorithm under DP perform worst
  (DP-GBM, FEVERLESS, DP-RF)
- Best results combine Newton updates, random splits, IH split candidates, large batches

Conclusions:

- The best individual components also work the best when combined together
- Batching updates reduces communication
- Best results close to non-private baselines



Credit 1 dataset, $\epsilon = 1, d = 4$

Key Takeaways

- Can achieve good performance with few rounds of communication by batching random trees
  - Batching updates is advantageous when privacy budget is very small
- Proposing split candidates over multiple rounds can often lead to better utility
  - **BUT**... do not waste 50% of your privacy budget doing so!
- Boosting doesn't always have a clear advantage over RF in high privacy settings
  - Some previous works have overstated performance of DP-GBDTs vs private RFs i.e., "GBDTs > RFs"
- **Core message:** Fully replicating the GBDT algorithm in the Federated setting is not ideal
  - Use "data-independent" or less data-intensive methods when possible

Federated Calibration and Evaluation of Binary Classifiers

Graham Cormode, Igor L. Markov

Federated Post-training statistics

Given a (binary) classifier that has been trained, we want to evaluate:

- **ROC AUC (Area Under Curve)**: a measure of quality of the classifier
- Calibration curves: a function to accurately measure the confidence of a prediction
- Other metrics: the precision, recall, accuracy etc. …

In the federated setting, each client holds examples with a ground truth label (positive or negative)

We show how to capture these via (federated) histogram and quantile primitives

Area Under Curve

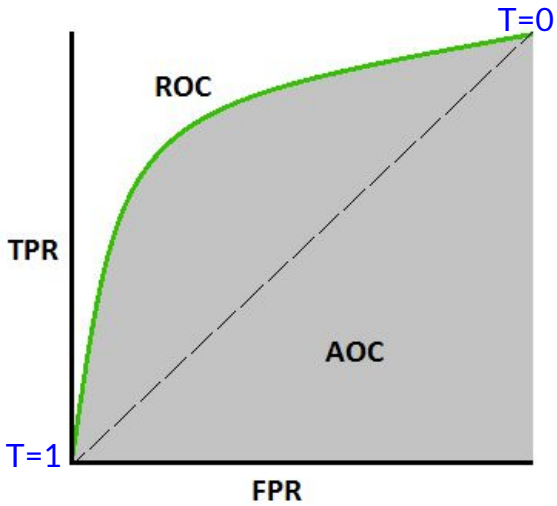
Given the score function, we predict x is positive if s(x) > T, else negative

Different choices of T give false positive (FP) / false negative (FN) tradeoffs

Receiver Operator Characteristic curve: plot FPR against TPR as T varies; Area Under Curve (AUC) measures the tradeoff, between 0.5 and 1.0

Basic calculation: sort examples by score, numerically integrate (quadrature)

But there are equivalent combinatorial calculations:



- Compute sum of ranks of positive examples in sorted scores as S
- AUC = $(S - \frac{1}{2}n^+(n^+ - 1)) / (n^+n^-)$, where n^+ (n^-) are the number of positive (negative) examples
- This effectively computes the number of (positive, negative) pairs that are disordered by s

Federated Area Under Curve

We make use of histograms to capture information about the classifier behaviour via secure aggregation:

Divide scores into B equal size bins, build **histograms** of number of negatives and positives in each bin

Compute AUC from histogram approximation by one of two (numerically equivalent) options:

a) Treating the bins as piecewise constant score function, and performing quadrature; or
b) Apply the combinatorial calculation based on sum of ranks of positive examples

Error decays as $O(1/B^2)$ under smoothness assumption on score function, or $O((1/B + 1/\varepsilon)1/B)$ with ε-DP

Federated Primitives

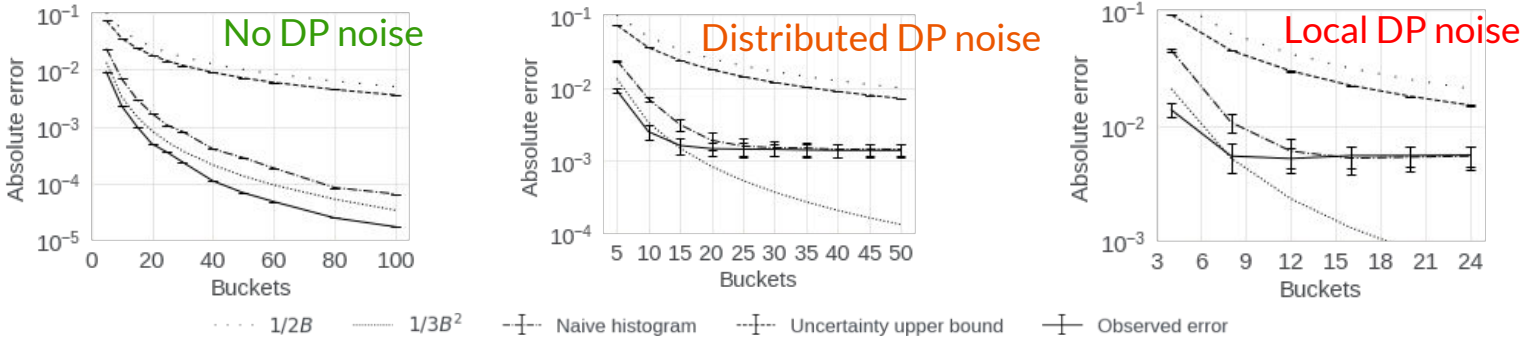
Computation of the tasks is enabled by a few basic primitives:

- **Quantile** computation (of the score distribution)
- **Histogram** materialization (of the positive and negative counts)
- **Mean** estimation (of other quantities)

All three of these are implementable in the Federated setting in different privacy models

- Secure Aggregation: server only sees the result of the aggregation computation
- Local DP: each client message achieves differential privacy locally
- Distributed DP: the clients each add enough noise so that the sum is DP

Federated AUC Results



- Error quickly becomes negligible (10^{-3} with 20 buckets, 10^{-4} with 60 buckets) for no noise (left)
- For distributed DP noise (centre), error plateaus at around 0.002
- 10-20 buckets achieves < 0.005 error for Local DP noise (right)

Federated Synthetic Data with FLAIM

Graham Cormode, Sam Maddock, Carsten Maple

Federated Synthetic Data

Synthetic Data is an important tool in working with private data

- Train a generative model on a private data set
- Ensure that the parameters are learned with a (differential) privacy guarantee
- Generate arbitrary amounts of realistic synthetic data without privacy constraints
- Enables many downstream applications for data exploration and modeling

Federated Synthetic Data follows the description, but the private training data is held by distributed clients

Challenges to Federating Synthetic Data Generation

In the federated setting, we need to ensure that:

- The privacy guarantees are met for each client
- The coordinating server is not exposed to private data
- The communication cost of the training is bounded
- Messages are sent securely and do not leak information
- The resulting model is sufficiently accurate

AIM: Adaptive and Iterative Mechanism

We start with a state-of-the-art approach to synthetic data generation in the centralized setting
"*AIM: An Adaptive and Iterative Mechanism for Differentially Private Synthetic Data*", McKenna et al. 2022

The data is represented based on marginal distributions over combinations of features

We make use of a "workload" set of queries, e.g., all pairwise feature distributions

- Select: (privately) select the query that is worst approximated by the current model
- Measure: (privately) perform the selected query on the private data and update the model
- Generate: use the updated model to generate a new set of synthetic data

FLAIM: Federated Learning AIM

The core of our approach is to translate the centralized algorithm into the federated setting:

- Perform some number of steps of the AIM algorithm locally at each client
  - Apply regularization to prevent overfitting the local distribution (heterogeneity)
- Periodically, update the central statistics by combining all client reports, tracking privacy budget
  - Client updates are weighted based on the noise added and number of examples
- The server finds the new model, choosing the new parameters to minimize a loss function

For more details, see the main presentation/poster at KDD24 on Tuesday 27th August

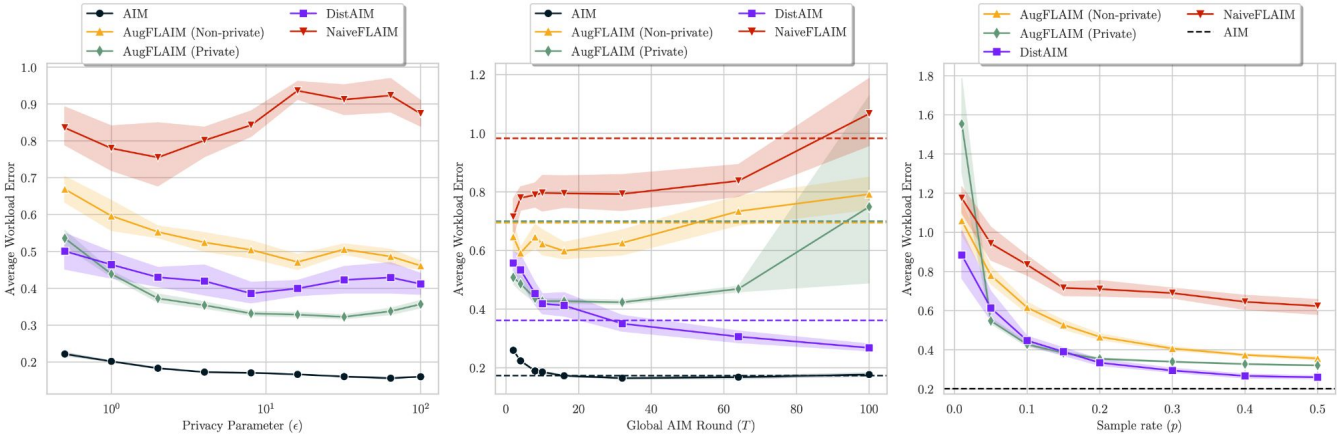
Experimental Set up

We implement and evaluate three variants as well as the original (centralized) AIM:

- DistAIM: a synchronous (expensive) translation of AIM to the federated setting on a sample of clients
- NaiveFLAIM, a first attempt to translate AIM to the federated setting
- AugFLAIM, FLAIM augmented with information on client heterogeneity to avoid overfitting

We measure workload error as we vary privacy budget, number of rounds, and fraction of participating clients

Experimental results



(a) Varying privacy budget (ε) (b) Varying global rounds (T) (c) Varying client sample-rate (p)

- Augmenting FLAIM with knowledge of client heterogeneity improves accuracy
- FLAIM can approach accuracy of centralized and (non-private) distributed AIM

Concluding Remarks

There's more to life than learning: many other ML/DM tasks are needed in the federated setting

Future work: post-training model calibration, data distribution drift measurement, richer analytics

More generally, one can think of many other tasks in federated setting

- Data analysis – querying and mining
- General purpose federated data querying systems?
- Explore other models of distributed data privacy for computation
- Debugging other federated workflows