Privacy-Preserving Distributed Linear Regression on High-Dimensional Data

Borja Balle
Amazon Research Cambridge
(work done at Lancaster University)

Based on joint work with Adria Gascon, Phillipp Schoppmann, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans
Motivation

<table>
<thead>
<tr>
<th>Treatment Outcome</th>
<th>Medical Data</th>
<th>Census Data</th>
<th>Financial Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Attr. 1</td>
<td>Attr. 2</td>
<td>Attr. 3</td>
</tr>
<tr>
<td>-1.0</td>
<td>0</td>
<td>54.3</td>
<td>...</td>
</tr>
<tr>
<td>1.5</td>
<td>1</td>
<td>0.6</td>
<td>...</td>
</tr>
<tr>
<td>-0.3</td>
<td>1</td>
<td>16.0</td>
<td>...</td>
</tr>
<tr>
<td>0.7</td>
<td>0</td>
<td>35.0</td>
<td>...</td>
</tr>
<tr>
<td>3.1</td>
<td>1</td>
<td>20.2</td>
<td>...</td>
</tr>
</tbody>
</table>

*Note:* This is vertically-partitioned data; similar problems with horizontally-partitioned
PMPML: Private Multi-Party Machine Learning

Problem

- Two or more parties want to *jointly learn* a model of their data
- But they *can’t share* their private data with other parties

Assumptions

- Parameters of the model will be received by all parties
- Parties can engage in on-line secure communications
- External parties might be used to outsource computation or initialize cryptographic primitives
The Trusted Party “Solution”

*Receives plain-text data, runs algorithm, returns result to parties*

The **Trusted Party** assumption:

- Introduces a single point of failure (with disastrous consequences)
- Relies on weak incentives (especially when private data is valuable)
- Requires agreement between all data providers

=> Useful but unrealistic. Maybe can be simulated?
Secure Multi-Party Computation (MPC)

**Public:** \( f(x_1, x_2, \ldots, x_p) = y \)

**Private:** \( x_i \) (party \( i \))

**Goal:** Compute \( f \) in a way that each party learns \( y \) (and nothing else!)

**Tools:** Oblivious Transfers (OT), Garbled Circuits (GC), Homomorphic Encryption (HE), etc

**Guarantees:** Honest but curious adversaries, malicious adversaries, computationally bounded adversaries, coalitions
In This Talk

A PMPML system for vertically partitioned linear regression

Features:
• Scalable to millions of records and hundreds of dimensions
• Formal privacy guarantees
• Open source implementation

Tools:
• Combine standard MPC constructions (GC, OT, TI, ...)
• Efficient private inner product protocols
• Conjugate gradient descent robust to fixed-point encodings
FAQ: Why is PMPML…

*Exciting?*
Can provide access to previously ”locked” data

*Hard?*
Privacy is tricky to formalize, hard to implement, and inherently interdisciplinary

*Worth?*
Better models while avoiding legal risks and bad PR
## Related Work

<table>
<thead>
<tr>
<th>Ref</th>
<th>Crypto</th>
<th>Linear Solver</th>
<th>Examples</th>
<th>Features</th>
<th>Running Time</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>HE</td>
<td>Newton</td>
<td>50K</td>
<td>22</td>
<td>2d</td>
<td>YES</td>
</tr>
<tr>
<td>[2]</td>
<td>HE+GC</td>
<td>Cholesky</td>
<td>2K</td>
<td>20</td>
<td>6m</td>
<td>YES</td>
</tr>
<tr>
<td>[3]</td>
<td>TI/HE</td>
<td>Newton</td>
<td>50K</td>
<td>223</td>
<td>“7h”</td>
<td>NO</td>
</tr>
<tr>
<td>[4]</td>
<td>SS</td>
<td>Gauss/Chol/CGD</td>
<td>10K</td>
<td>10</td>
<td>11s</td>
<td>NO</td>
</tr>
</tbody>
</table>


Functionality: Multi-Party Ridge Regression

Training Data

\[ X = [X_1 \hspace{0.5em} X_2] \in \mathbb{R}^{n \times d} \]
\[ Y \in \mathbb{R}^n \]

Private Inputs

Party 1: \( X_1, Y \)

Party 2: \( X_2 \)

Ridge Regression

\[
\min_{\theta \in \mathbb{R}^d} \| Y - X \theta \|^2 + \lambda \| \theta \|^2
\]

(optimization)

\[
(X^\top X + \lambda I)\theta = X^\top Y
\]

(closed-form solution)
Aggregation and Solving Phases

**Aggregation**

\[ A = X^\top X + \lambda I \]

\[ b = X^\top Y \]

\[ \mathcal{O}(nd^2) \]

**Solving**

\[ \theta = A^{-1}b \]

\[ \mathcal{O}(d^3) \] (eg. Cholesky)

\[ X^\top X = \begin{bmatrix} X_1^\top X_1 & X_1^\top X_2 \\ X_2^\top X_1 & X_2^\top X_2 \end{bmatrix} \]

(cross-party products)

Approximate iterative solver \[ \mathcal{O}(kd^2) \] (eg. k-CGD)
Challenges and Trade-offs

- **MPC protocols**: out of the box vs. tailored
- **Encoding real numbers**: speed vs. accuracy
- **Scalability**: $n, d, \#$ parties
- **Privacy guarantees**: semi-honest vs. malicious
- **External parties**: speed vs. privacy
- **Interaction**: off-line vs. on-line
1. **CrP** distributes correlated randomness
2. **DPS** run multiple inner product protocols to get additive share of $(A,b)$
3. **CoP** get GC for solving linear system from **CrP**
4. **DPS** send garbled shares of $(A,b)$ to **CoP**
5. **CoP** executes GC and returns solution to **DPSs**

**Alternative:** **CrP** and **CoP** simulated by non-colluding parties
Aggregation Phase – Two Protocols

\[ X_1^\top X_2 \rightarrow f(x_1, x_2) = \langle x_1, x_2 \rangle \]

(matrix product) (inner product b/w columns)

• **External pre-processing**: inner product protocol leveraging correlated randomness supplied by Trusted Initializer (TI)

• **Stand-alone**: 2-party inner product protocol based on Oblivious Transfers (OT)

**Fixed-point Encoding**

\[ \mathcal{O}(\log(n/\varepsilon)) \] bits \(\Rightarrow\) error \(\leq \varepsilon\)
Aggregation Phase - Experiments

Trade-offs

- **OT**: stand-alone, out-of-the-box MPC
- **TI**: pre-processing, external party, faster

<table>
<thead>
<tr>
<th>$n$</th>
<th>$d$</th>
<th>Number of parties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OT</td>
</tr>
<tr>
<td>$5 \cdot 10^4$</td>
<td>20</td>
<td>1m50s</td>
</tr>
<tr>
<td>$5 \cdot 10^4$</td>
<td>100</td>
<td>42m12s</td>
</tr>
<tr>
<td>$5 \cdot 10^5$</td>
<td>20</td>
<td>18m18s</td>
</tr>
<tr>
<td>$5 \cdot 10^5$</td>
<td>100</td>
<td>7h3m56s</td>
</tr>
<tr>
<td>$1 \cdot 10^6$</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>$1 \cdot 10^6$</td>
<td>200</td>
<td>-</td>
</tr>
</tbody>
</table>

* AWS C4 instances, 1Gbps
Solving Phase – Garbled Circuits

\[ A\theta = b \]

(PSD linear system)

\[ (A_i, b_i) \]

(party i’s input)

\[ A = \sum_i A_i \quad b = \sum_i b_i \]

Solver implemented in a Garbled Circuit

Floating-point computation with GC is not feasible (yet)

<table>
<thead>
<tr>
<th>Year</th>
<th>Device / Paper</th>
<th>32 bit floating point multiplication (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1961</td>
<td>IBM 1620E</td>
<td>17.7</td>
</tr>
<tr>
<td>1980</td>
<td>Intel 8086 CPU (software)</td>
<td>1.6</td>
</tr>
<tr>
<td>1980</td>
<td>Intel 8087 FPU</td>
<td>0.019</td>
</tr>
<tr>
<td>2015</td>
<td>Pullonen et al. @ FC&amp;DS</td>
<td>38.2</td>
</tr>
<tr>
<td>2015</td>
<td>Demmler et al. @ CCS</td>
<td>9.2</td>
</tr>
</tbody>
</table>
Solving Phase – Two Methods

- **Cholesky**: exact, cubic, used in [Nikolaenko et al.’13]
- **Conjugate Gradient Decent (CGD)**: approximated, “quadratic”
Fixed-point + Conjugate Gradient Descent

**Textbook CGD**

\[
\begin{align*}
g_0 & := Ax_0 - b \\
p_0 & := g_0 \\
\text{repeat for } k = 1 \ldots K \\
\alpha_k & := \frac{g_k^T p_k}{p_k^T A p_k} \\
x_{k+1} & := x_k - \alpha_k p_k \\
g_{k+1} & := g_k - \alpha_k A p_k \\
\beta_k & := \frac{p_k^T A g_{k+1}}{p_k^T A p_k} \\
p_{k+1} & := g_{k+1} - \beta_k p_k
\end{align*}
\]

**Fixed-point CGD**

\[
\begin{align*}
g_0 & := Ax_0 - b \\
p_0 & := g_0 / \|g_0\|_\infty \\
\text{repeat for } k = 1 \ldots K \\
\alpha_k & := \frac{g_k^T p_k}{p_k^T A p_k} \\
x_{k+1} & := x_k - \alpha_k p_k \\
g_{k+1} & := g_k - \alpha_k A p_k \\
\beta_k & := \frac{p_k^T A (g_{k+1} / \|g_{k+1}\|_\infty)}{p_k^T A p_k} \\
p_{k+1} & := g_{k+1} / \|g_{k+1}\|_\infty - \beta_k p_k
\end{align*}
\]
Fixed-point + Conjugate Gradient Descent

Textbook CGD ($d=50$, bits=64)

Fixed Point CGD ($d=50$, bits=64)

Bits = $N_i + N_f + 1$

$N_i$ = number of integer bits

$N_f$ = number of fractional bits
able from those of a randomly sampled approximation, and
tion of a functionally private approximation, which guarantees
ing does not provide security in general. They define the no-
evaluation of the exact functionality, and that simply round-
discussed in detail in the work of Feigenbaum et al. [25]. The
functionalities that implement approximations, and is dis-
ected in the verification phase.

◊

still a difficulty that must be overcome to formally prove se-
approximation of the linear regression functionality, there is
the computation of the first phase is evaluating an exact solu-
for the solving phase for

fast: in our experiments, garbling and execution took less than
framework presented in [65]. As expected, it runs extremely
the garbled circuit with malicious security for this verification

and the root mean squared error (RMSE) of the solution obtained by our system and an insecure implementation of ridge regression

Table 3.

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>Reference</th>
<th>$d$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Student Performance</td>
<td>[11, 14]</td>
<td>30</td>
<td>395</td>
</tr>
<tr>
<td>2</td>
<td>Auto MPG</td>
<td>[72]</td>
<td>7</td>
<td>398</td>
</tr>
<tr>
<td>3</td>
<td>Communities and Crime</td>
<td>[61, 62]</td>
<td>122</td>
<td>1994</td>
</tr>
<tr>
<td>4</td>
<td>Wine Quality</td>
<td>[12, 13]</td>
<td>11</td>
<td>4898</td>
</tr>
<tr>
<td>5</td>
<td>Bike Sharing Dataset</td>
<td>[23, 24]</td>
<td>12</td>
<td>17379</td>
</tr>
<tr>
<td>6</td>
<td>Blog Feedback</td>
<td>[8, 9]</td>
<td>280</td>
<td>52397</td>
</tr>
<tr>
<td>7</td>
<td>CT slices</td>
<td>[33]</td>
<td>384</td>
<td>53500</td>
</tr>
<tr>
<td>8</td>
<td>Year Prediction MSD</td>
<td>[5]</td>
<td>90</td>
<td>515345</td>
</tr>
<tr>
<td>9</td>
<td>Gas sensor array</td>
<td>[26, 27]</td>
<td>16</td>
<td>4208261</td>
</tr>
</tbody>
</table>

• 70-30 train-test random split
• Regularization tuned in the clear
• Implemented in Obliv-C
• 2+2 parties, 20 CGD iterations
• Data standardization inside protocol

• CGD faster for $d > 100$
• 32 bits provide good accuracy
Conclusion

Summary

- Full system is accurate and fast, available as open source
- Scalability requires hybrid MPC protocols and non-trivial engineering
- Robust fixed-point CGD inside GC has many other applications

Extensions

- Security against malicious adversaries
- Classification with quadratic loss
- Kernel ridge regression
- Differential privacy at the output

Future Work

- Models without a closed-form solution (eg. logistic regression, DNN)
- Library of re-usable ML components, complete data science pipeline
Privacy-Preserving Distributed Linear Regression on High-Dimensional Data

Adrià Gascón¹, Phillipp Schoppmann², Borja Balle³, Mariana Raykova⁴, Jack Doerner⁵, Samee Zahur⁶, and David Evans⁷

¹ University of Edinburgh
² Humboldt University of Berlin
³ Lancaster University
⁴ Yale University
⁵ Northeastern University
⁶ Google
⁷ University of Virginia

A Secure Multiparty Computation (MPC) protocol for computing linear regression on vertically distributed datasets

GitHub repository: https://github.com/schoppmp/linreg-mpc