Machine Learning Classification over Encrypted Data

Raphaël Bost
Université Rennes 1
MIT

Raluca Ada Popa,
ETH Zürich
MIT

Stephen Tu
MIT

Shafi Goldwasser
MIT
Classification (Machine Learning)

- Supervised learning (training)
- Classification
Problem

- The provider’s model is sensitive financial model, genetic sequences, …

- Client’s private data medical records, credit history, …
Problem

- The provider’s model is sensitive financial model, genetic sequences, …
- Client’s private data medical records, credit history, …

MPC / 2PC
Using General 2PC?

+ Works for any circuit
+ Constant number of interactions
- Have to build circuits
- Hard to ‘compose’
- Not easily reusable
Using General 2PC?

+ Works for any circuit
+ Constant number of interactions
- Have to build circuits
- Hard to ‘compose’
- Not easily reusable

➡ Ad Hoc protocols
Goal

- Enable classification without sacrificing privacy
- Secure classification, no learning, the model is already known
- Practical performance
Approach

- Classifiers as specialized 2PC
- Identify and construct reusable building blocks
- Threat model: passive adversary (honest-but-curious)
<table>
<thead>
<tr>
<th>ML Algorithm</th>
<th>Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceptron</td>
<td>Linear</td>
</tr>
<tr>
<td>Least squares</td>
<td>Linear</td>
</tr>
<tr>
<td>Fischer linear discriminant</td>
<td>Linear</td>
</tr>
<tr>
<td>Support vector machine</td>
<td>Linear</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>Naïve Bayes</td>
</tr>
<tr>
<td>ID3/C4.5</td>
<td>Decision trees</td>
</tr>
</tbody>
</table>
Insight

- Identify core operations
- Construct reusable/composable building blocks
- Choose the best fitted primitives
  - Homomorphic Encryption, FHE, Garbled Circuits, …
Related Work

• Privacy-preserving training
  • Using FHE, linear means classifier [GLN12]
  • Specific techniques for Naïve Bayes [VKC08], decision trees [BDMN05,LP00], linear discriminant [DHC04], kernel methods [LLM06]

• Privacy-preserving classification
  • Using FHE, outsource computation [BLN13]
  • Secure branching programs [BFL+09, BFL+11]
  • Specific classifiers (face recognition/detection) [SSW09, AB07]
Building Blocks

- Dot product
- Encrypted Comparison
- Encrypted (arg)max
- Private decision trees
- Encryption scheme switching
Classifiers from blocks

Linear Classifier

Naïve Bayes Classifier

Decision Tree Classifier

Dot Product

Enc. Compare

Enc. Argmax

ES Switching

Private Decision Trees
Classifiers

In Practice

• Linear Classifier
• Naïve Bayes Classifier
• Decision Trees
Linear Classifier

• Separate two sets of points
• Very common classifier
• Dot product + Encrypted compare
## Linear Classifier

<table>
<thead>
<tr>
<th>Model Size (dimension)</th>
<th>Time / protocol</th>
<th>Total</th>
<th>Comm.</th>
<th>Inter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>Dot Product</td>
<td>&lt;0.01s</td>
<td>0.194 s</td>
<td>0.204 s</td>
</tr>
<tr>
<td>47</td>
<td>Dot Product</td>
<td>0.024 s</td>
<td>0.194 s</td>
<td>0.217 s</td>
</tr>
</tbody>
</table>

Evaluation on UC Irvine ML databases
40 ms network latency
2.66 GHz Intel Core i7
Naïve Bayes Classifier

$$\arg\max_{i \in [k]} p(C = c_i) \prod_{j=1}^{d} p(X_j = x_j | C = c_i)$$
Naïve Bayes Classifier

\[
\arg\max_{i \in [k]} p(C = c_i) \prod_{j=1}^{d} p(X_j = x_j | C = c_i)
\]
Naïve Bayes Classifier

\[
\arg\max_{i \in [k]} p(C = c_i) \prod_{j=1}^{d} p(X_j = x_j | C = c_i)
\]

\[
\arg\max_{i \in [k]} \log p(C = c_i) \sum_{j=1}^{d} \log p(X_j = x_j | C = c_i)
\]

- Additive homomorphism + Encrypted argmax
### Naïve Bayes Classifier

### Evaluation on UC Irvine ML databases

<table>
<thead>
<tr>
<th># Cat.</th>
<th># Features</th>
<th>Argmax</th>
<th>Total Time</th>
<th>Comm.</th>
<th>Inter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9</td>
<td>0.40 s</td>
<td>0.48 s</td>
<td>72.47 kB</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>1.33 s</td>
<td>1.42 s</td>
<td>150.7 kB</td>
<td>42</td>
</tr>
<tr>
<td>24</td>
<td>70</td>
<td>3.38 s</td>
<td>3.81 s</td>
<td>1911 kB</td>
<td>166</td>
</tr>
</tbody>
</table>

40 ms network latency
2.66 GHz Intel Core i7
Decision Trees
Decision Tree

- Combination of other classifiers
- In this example, linear classifiers
- Linear classifier + ES Switching + Decision Trees
# Decision Tree

<table>
<thead>
<tr>
<th>Tree Specs.</th>
<th>Time / Protocol</th>
<th>Decision Tree (FHE)</th>
<th>Total</th>
<th>Comm.</th>
<th>Inter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>Depth</td>
<td>Lin. Class.</td>
<td>ES Switch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.45 s</td>
<td>1.64 s</td>
<td>0.27 s</td>
<td>2.3 s</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>1.41 s</td>
<td>7.41 s</td>
<td>0.93 s</td>
<td>9.8 s</td>
</tr>
</tbody>
</table>

Evaluation on UC Irvine ML databases
40 ms network latency
2,66 GHz Intel Core i7
## Decision Tree

<table>
<thead>
<tr>
<th>Tree Specs.</th>
<th>Time / Protocol</th>
<th>Total</th>
<th>Comm.</th>
<th>Inter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>Depth</td>
<td>Lin. Class.</td>
<td>ES Switch</td>
<td>Decision Tree (FHE)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.45 s</td>
<td>1.64 s</td>
<td>0.27 s</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>1.41 s</td>
<td>7.41 s</td>
<td>0.93 s</td>
</tr>
</tbody>
</table>

Run sequentially, can be parallelized
Building blocks library

- Designed to be modular
  Easy composition

- Easy to construct new secure classifiers
  Face detection algorithm (Viola & Jones)
Building blocks library

E.g.: Linear Classifier

Client

\( v \)  \( \rightarrow \)  \( \text{Dot Product} \)  \( \rightarrow \)  \( [\langle v, w \rangle] \)  \( \rightarrow \)  \( \text{Enc. Compare} \)  \( \rightarrow \)  \( \langle v, w \rangle > 0 \)

Server

\( w \)  \( \rightarrow \)  \( \text{Dot Product} \)  \( \rightarrow \)  \( \text{Enc. Compare} \)  \( \rightarrow \)  \( \text{SK} \)
Building blocks library

E.g.: Linear Classifier

Client

```cpp
bool Linear_Classifier_Client::run()
{
    exchange_keys();

    // values_ is a vector of integers
    // compute the dot product
    mpz_class v = compute_dot_product(values_);
    mpz_class w = 1; // encryption of 0

    // compare the dot product with 0
    return enc_comparison(v, w, bit_size_, false);
}
```

Server

```cpp
void Linear_Classifier_Server_session::run_session()
{
    exchange_keys();

    // enc_model_ is the encrypted model vector
    // compute the dot product
    help_compute_dot_product(enc_model_, true);

    // help the client to get
    // the sign of the dot product
    help_enc_comparison(bit_size_, false);
}
```
In conclusion

• Composable building blocks for secure classifiers
• Library with practical performances

Future work:

• Less roundtrips (work on the protocols)
• More parallelism (work on the implementation)
Questions?