Experimental Secure Multi-Party Computation on Real Data using SPDZ

Abstract

Secure Multi-Party Computation (MPC) is an area of cryptography, which enables the computation on sensitive data from multiple sources while maintaining privacy guarantees. However, theoretical protocols often do not scalable efficiently to real world size data. This project investigates the efficiency of of the SPDZ framework [1], which provides implementation of MPC protocols with malicious security, in the context of popular machine learning (ML) algorithms. In particular, we choose applications such as linear regression and logistic regression, which have been implemented and evaluated using semi-honest MPC techniques [2, 8]. We demonstrate that the SPDZ framework outperforms these previous implementations while providing stronger security.

1 Introduction

Many machine learning techniques, including regression analysis, aim to build a model that fits a set of predictors to a dependent variable. Such techniques are widely used to model and analyze big data. In many settings, however, the input data for such ML analysis tools is partitioned among different parties, which have strict privacy policies. For example, the Center for Disease Control is interested in identifying disease outbreak and many individual hospitals have their own patients' data which could benefit this study. The problem is that openly sharing this data for prediction or model-building purposes would be against modern day privacy laws as it would leak private individual data. This is one of many examples, which could benefit of the functionality that allows to evaluate to output of the analysis without revealing more about the private inputs.

1.1 Multi-Party Computation

Secure MPC addresses the above problem by providing a mechanism through which different parties can run a joint computation over their private inputs with guarantees that the only thing revealed about the inputs, is the output of the computation and whatever can be inherently inferred from it. There are two main types of MPC protocols in terms of their security guarantees: semi-honest and malicious protocols [3, 5]. In semi-honest security, it is assumed that the parties will follow the protocol as specified, but they can try to infer information about the input from the protocol messages. In malicious security, dishonest parties may attempt to deviate from the specified protocol, and the protocol must guarantee that these parties cannot learn about the inputs. Since malicious protocols have to satisfy stronger guarantees in general such construction are less efficient than semi-honest protocols.

Two recent works [2, 8] propose efficient implementation for several central building blocks for machine learning such as conjugate gradient decent (CGD) and stochastic gradient descent (SGD) as well as applications using them such as linear and logistic regression. The work of Gascon et al. [2] uses the framework for semi-honest computation Obliv-C [9] and proposes several different methods for solving systems of linear equations. Their main premise is to use an iterative method such as CGD and show trade-offs that save a lot of computation and hence efficiency for the MPC in return for a small accuracy loss. They further propose a modification of CGD that has stable behavior using fix-point arithmetic since emulating floating point with the underlying MPC representation introduces substantial efficiency overhead.

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

The work of Mohassel and Zhang [8] considers stochastic gradient descent as a method for learning linear regression and logistic regression by using different activation functions. The authors consider arithmetic representation for the computation and propose new secure computation techniques for matrix computation, which generalize the approach for generating multiplicative triples in a preprocessing step. Similarly to the work of Gascon et al. [2], this paper considers techniques for approximation that save computation, for example, using a piece-wise approximation for the logistic function. The author also propose new techniques for more efficient approximate computation of fixed-point encodings.

The techniques in both of the above works are restricted to the setting of two party computation. We choose the SPDZ [1, 7, 6] framework for our implementations since it is one of the main and most comprehensive implementations for multiparty computation protocols, which provide malicious security and support more than two parties.

2 ML Functionalities

For our implementation we consider the same algorithms and functionalities as in the above two papers. Next we provide a brief overview of these classic algorithms (details can be found in [4]).

2.1 Direct vs. Iterative Decomposition for Solving a Linear System

Solving a system of linear equations, which underlies linear regression learning, can be done using techniques for direct and indirect decomposition. LDLT and Cholesky are both variants of direct decomposition methods which decompose a Hermitian, positive-definite matrix into a lower triangular matrix and its conjugate transpose. The algorithms are cubic in complexity with asymptotic run time of $O(d^3)$, where d is the dimension of the input matrix. The difference between LDLT and Cholesky is that Cholesky requires a square root. The representation of square root computation as an arithmetic circuit used in the MPC computation in SPDZ introduces considerable overhear. That is why we used the iterative Newton method as a way of approximating the square root computation. It computes x_i, where $x_i^2 = S$, with repetition of the following update function $x_{n+1} = 1/2(x_n + S/x_n)$.

In terms of an iterative approach to regression, we used the approach proposed by Gascon et al. [2], which uses a normalized version of CGD that preserves stability and convergence rate with fixed-point number representation. Similarly to other MPC implementations using floating point representation in SPDZ introduces substantial efficiency overhead.

2.2 Stochastic Gradient Descent

Stochastic gradient descent is an iterative approximation method that converges to the global minimum for convex problems, like linear and logistic regression. It is also a driving mechanism for non-convex problems like neural networks. An SGD iteration updates a weight vector $\mathbf{w}$ using a randomly selected sample from the training input as follows: $w_j := w_j - \alpha(\partial C_i(\mathbf{w})/\partial w_j)$ with learning rate α. In this update C_i is the cost function, which can be instantiated with different concrete functions to obtain computation for linear regression and logistic regression. A common technique for SGD computation is called *mini-batch* – instead of selecting one sample per iteration, a small batch of size B samples are selected and the update function is performed averaging the partial derivatives across all samples. We use the mini-batch SGD in our implementation to obtain accuracy benefits. While the work of Mohassel and Zhang [8] has optimizations for matrix computation, which can be used with mini-batch, for SPDZ this does not lead to additional savings.

2.2.1 Linear and Non-Linear Activation Functions

To obtain a solution for linear regression using SGD, we instantiate the cost function as $w_j = w_j - \alpha(X_i \cdot w * -y_i)X_{ij}$, where X is the input matrix and y is the input vector. In this update function, the weights are adjusted element-wise by the error from the predicted and expected value at a rate determined by α.

Logistic regression is a classification algorithm for modeling a binary dependent variable. Logistic regression fits the logistic function $f(u) = \frac{1}{1+e^{-u}}$ to the input. The corresponding update function for mini-batched SGD for logistic regression is $w = w - \frac{1}{|B|}\alpha X_B^T \times f(X_B \times w - Y_B)$, where f maps the predicted value into the binary output space. Mohassel and Zhang [8] proposed the following piecewise function as approximation for f:

$$f(u) = \begin{cases} 0 & \text{if } u < -0.5 \\ u + 0.5 & \text{if } -0.5 \leq u \leq 0.5 \\ 1 & \text{if } u > 0.5 \end{cases}$$

We compare the results of this MPC-friendly piecewise function to a more standard approximation approach of taking the Taylor Series expansion to varying degrees.

3 Experiments

3.1 Experimental Setup

For our evaluation, we implemented all algorithms both in the SPDZ framework as well as in python as a plaintext verification of the algorithm. The main metrics of evaluations were the latency of the MPC computation and the accuracy error, and we aimed to explore the trade-offs between accuracy and efficiency. We varied the precision after the decimal point depending on what was used in the works that we compared against (32 and 64 bits for the linear regression, less for SGD).

We evaluated our methods on both real-world datasets (MNIST, Arcene, and 9 other UCI open-source datasets) as well as synthetically generated data. These real-world datasets allow us to compare the accuracy results to existing works and to demonstrate that SPDZ can be used in practical settings. We used synthetic data in order to explore larger ranges of data characteristics such as dimension (d = 10, 20, 50, 100, 200, 500), condition number (cd = [1,10]), and number of examples (n = 1000, 100000).

Most of our experiments were ran using machines on the same local area network where there is no network latency. We performed tests where both parties were deployed on separate Amazon EC2 m4.large instances. We also ran experiment with up to four parties.

3.2 Results

In this section we present empirical results for our SPDZ implementations evaluated with real and synthetic databases. We compare the five different algorithms in terms of accuracy and run time for various parameters.

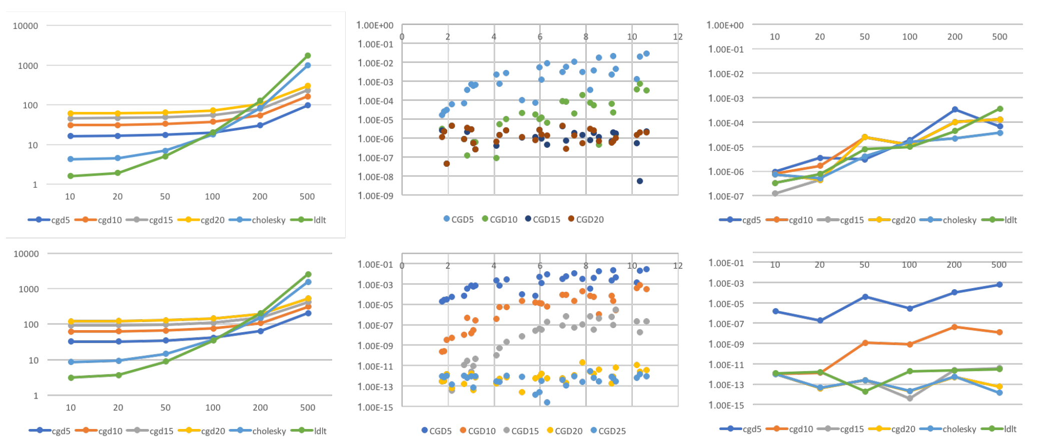

Figure 1: (Left) Run time as a function of input dimension. (Middle) Condition number as a function of accuracy. (Right) Accuracy as a function of the input dimension. (Top) Fixed-point with 60 bits of precision. (Bottom) Fixed-point with 28 bits of precision

For LDLT, Cholesky and various iterations of CGD, we evaluated on synthetically generated data of varying sizes and condition numbers. The larger the condition number is, the larger the error in approximations of the solution is. The direct decomposition methods grew exponentially in run time as input size increases, which is shown in the left column of Figure 1 – this unlikely to be suitable for large size real data. Alternatively, the iterative CGD runtime increases at a much slower rate. In the middle column of Figure 2, we find that about 20 iterations are sufficient to reach maximum accuracy given the number of allocated bits even with varying condition numbers, meaning the data might not be as well-formed. Particularly for the 64-bit case, shown on the bottom right, the accuracy is identical for CGD after 15 iterations and Cholesky/LDLT.

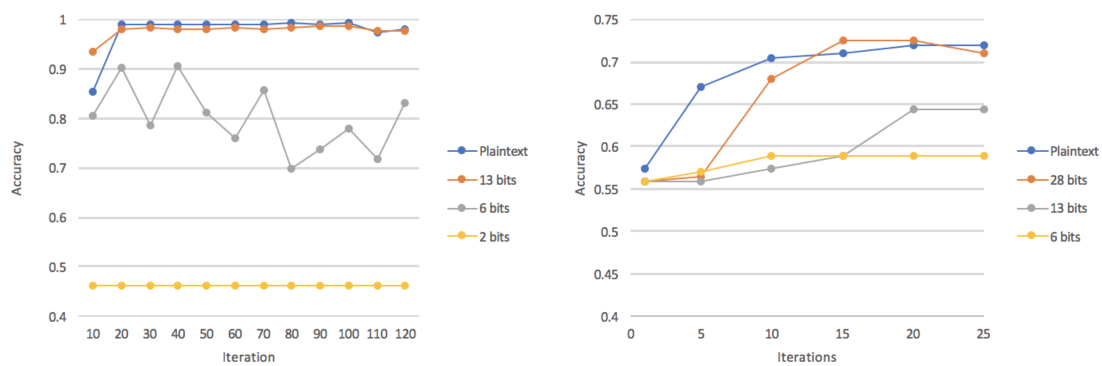

Figure 2: Comparing accuracy of privacy preserving linear regression with various fixed point precisions and plaintext training on floating point for MNIST (left) and Arcene (right).

Figure 2 compares SGD on MNIST and Arcene results. It shows that the number of bits of precision needed to get good accuracy is highly dependent on the dataset. For MNIST, 13 bits was sufficient to match plaintext accuracy but 28 bits were needed for Arcene. The MNIST data contains only 784 features while there are 10,000 in the Arcene data, 3,000 of which are considered "probes" with no predictive power, which could explain the lower overall accuracy [8]. While the numbers in the MNIST data ranged from 0 to 9, Mohassel and Zhang [8] only used 0s and 1s labels from the dataset, reducing it to a binary problem. We replicated this approach and present the results below. We did run the computation to predict all 10 digits, but found that SGD only achieved a much lower accuracy of about 19%. We also compared the root mean squared error (RMSE) of SGD on 9 UCI open-sourced datasets of ranging sizes to results in [3]. Our results in the SPDZ secure setting typically increased RMSE by about $5-20\%$ compared to plaintext computation, but still outperformed RMSE results from [3] in both CGD and SGD.

In terms of logistic regression, for SPDZ, we did not find that the new activation function was a better alternative to taking a Taylor Series approximation for the exponential function as shown in Table 1. We found that for SPDZ, which is based on arithmetic circuits, the extra time to take a few extra degrees in the approximation was negligible.

Table 1: Comparing the validation accuracy for different activation functions for logistic regression.

	Plaintext	New Activation function	Polynomial Approximation			
			degree 2	degree 5	degree 7	degree 10
MNIST	99.9%	95%	97%	85%	91%	99.5%
Arcene	72.0%	44.0%	44.0%	44.5%	65%	72%

3.3 Conclusion

We found that the SPDZ outperformed Obliv-C even in a distributed machine setting by an order of magnitude and achieves comparable results to SecureML for SGD instantiations such as Linear and Logistic Regression. The accuracy findings were comparable for LDLT, Cholesky, and CGD results for SPDZ and Obliv-C, but we found that SPDZ outperformed Obliv-C on SGD. SPDZ was able to match SecureML results on SGD and Logistic Regression.

References

[1] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[2] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *PoPETs*, 2017(4):345–364, 2017.

[3] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[4] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition*. Springer series in statistics. Springer, 2009.

[5] Carmit Hazay and Yehuda Lindell. A note on the relation between the definitions of security for semi-honest and malicious adversaries. Cryptology ePrint Archive, Report 2010/551, 2010. `https://eprint.iacr.org/2010/551`.

[6] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, pages 158–189, 2018.

[7] Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure mpc with dishonest majority. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, 2013.

[8] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38, 2017.

[9] Samee Zahur and David Evans. Obliv-c: A language for extensible data-oblivious computation. Cryptology ePrint Archive, Report 2015/1153, 2015. `https://eprint.iacr.org/2015/1153`.