Privacy Preserving K-Means Clustering

6.857 Final Rport

Amartya Shankha Biswas, Akshat Bubna, Dustin Doss, Sarah Scheffler

May 11, 2016

1 Introduction

Given the exponential rise in the size of datasets over the past decade, novel tools are needed to process and make use of this information. Machine learning, a relatively old subfield of computer science, has gained prominence based on its ability to extract models for meaningful decision-making based on large amounts of data. Fields as wide-ranging as medical diagnosis, disaster response, image and speech recognition, and network breach detection have all benefited from advances in machine learning.

However, data-owning parties often find it undesirable or impossible to share the information they collect. Companies may have obligations towards user privacy, aspire to monetize their data, or may need to keep their data private to protect their own platform. Medical organizations have privacy obligations in the United States under the Patient Privacy Act. However, it is very often useful to share data in the development of a machine learning model, giving it more predictive power over a wider range of inputs. To solve this dilemma, we turn to cryptography to enable *privacy-preserving* machine learning, in which parties can gain an output model based on shared data, without having to reveal individual data points to each other.

We recognize that many organizations interested in using privacy-preserving learning would rather use a provided implementation than build a solution themselves. Thus, our goal in this work was to provide a prototype of a *usable* privacy-preserving machine learning implementation with particular focus on performance, to demonstrate the usefulness of such a system. We also wanted our system to be *conceptually-scalable*: we provide building blocks that can be used to build additional privacy-preserving machine learning algorithms other than the single one we implemented.

Based on the work of Saeed Samet, Ali Miri, and Luis Orozco-Barbosa [9] (often referred to in this paper as "SMO07,") we have devised and implemented a system to allow disparate parties to utilize their disjoint datasets in order to develop a k-means model usable by all parties. This system leverages multi-party computation and an open-source implementation of the Paillier cryptosystem [3] to implement a k-means clustering algorithm over data that is distributed among different parties. Our implementation can be found at `https://github.com/sarahscheffler/pp-k-means`.

There are many other homomorphic encryption-based machine learning protocols, but very few of them are accompanied by implementations, and the ones that do have an implementation are slow, unoptimized, and in some cases, incomplete. [2] [8] We chose SMO07 because it was a suitably simple algorithm that we could implement it in the time provided. We also looked for opportunities to improve the algorithm's performance and security.

The rest of this paper is organized as follows: Section 2 describes related work and other research paths we think would be useful to this paper. Section 3 explicitly defines our problem and threat model, and provides example use cases. Section 4 summarizes our protocol, including the original algorithm provided by [9] and more details about the Paillier cryptosystem. Section 5 is a description of the system we have built to run the protocol, with focus paid to networking concerns and usability. Section 6 is an initial performance analysis, and section 7 describes paths for future research on this project.

2 Related Work

Present literature presents classifiers which are trained on unencrypted datasets and subsequently classify encrypted data points. [8] adopts such an approach for hyperplane, naïve Bayes and decision tree classifiers. Work has also been done on securing data for a classifiers in distributed settings, in which individual worker nodes are treated as untrusted. [4] uses the MapReduce framework for this approach.

Applying neural networks on homomorphic encryption schemes has been studied in some detail. Since fully homomorphic encryption is slow, [2] proves that homomorphic encryption that applies only to degree bounded polynomials is a viable alternative. However, the speed and practicality of this method is still unclear.

The epsilon-diferential privacy model promised in [1] also provided interesting directions of research. This paper provides strong guarantees on securing a logistic regression model. We also looked examined of securing a machine learning system against adversarial attacks ([5], [6], [11]) to further strengthen the system.

3 Problem Description and Threat Model

This protocol is designed parties who have a shared data format, but different datasets. These parties are interested in the benefits of a k-means model trained on their data, but they all want to keep their data confidental from all other parties.

3.1 Example Use Cases

These examples have been provided to motivate potential uses for our privacy-preserving k-means algorithm.

Example A Several companies have been collecting log data about their network in an effort to detect and measure network intrusion by malicious parties. These compa-

nies want to be able to use machine learning to determine the extent to which their networks have been compromised as proposed by [10] and others. However, company policy prohibits the sharing of this information, since sharing it could help malicious parties penetrate their network. The companies have all been collecting the same kinds of data, and the data can be put into a common format with a small amount of scripting. The companies can use this protocol to use the k-means clustering algorithm to combine their information so as to better secure their networks.

Example B Medical researchers for health companies that possess sensitive patient information wish to use the k-means clustering algorithm to gain some knowledge about how their patients' symptoms might be related. For privacy reasons, they cannot share the individual data points with each other or with any other parties. They all collect the same kind of information about each patient, and can then use this protocol to run k-means on their combined data without fear that they are violating privacy agreements.

3.2 Threat Model

The adversaries in this model fall into two categories:

- External attackers (e.g., on the network at large)
- Parties involved in the computation

We are assuming that all parties involved in the computation are semi-honest; that is, they will follow the protocol correctly, but try to get as much extra information as they can. As such, our results focus on confidentiality rather than integrity.

Against external attackers, we assume that any and all inter-party communication can be eavesdropped on. As such, it is necessary to ensure both confidentiality and integrity all traffic between the parties. The goal of these actions are to prevent this attacker from learning the data being shared, and to prevent them from altering the communication without the knowledge of the parties.

We assume that parties holding data in this computation are looking out only for themselves, but also that their main goal is to learn the legitimate outcome of this protocol. We treat these parties as honest but curious: they will follow the protocol correctly but they will also try to read the other parties' data at every opportunity.

4 Protocol Description

4.1 K-means Clustering Algorithm

In the k-means clustering problem, the goal is to partition a data-set into k clusters and build a classifier that can classify other data points into one of these clusters. We represent each cluster by a "mean". Then, the cluster that a data point belongs to is the one wtih the closest mean. Mathematically, the objective is to minimize the sum of squares of distances of each data point to its nearest mean.

The algorithm works by choosing k random initial guesses. Each data point is then sorted into a cluster by finding the guess nearest to it. Then, for each cluster, the new guess for the mean is recomputed as the centroid of all points sorted into that cluster. This process is rerun with these new guesses for cluster means. This process iterates until some stopping criterion is met.

With Privacy-Preserving k-means, our goal is to calculate these k means when different parties own different datapoints, and do not want others to learn them. We build on a scheme from Samet, Miri, and Orozco-Barbosa [9] to homomorphically encrypt each party's data and learn the k means for the combined dataset keeping our threat model discussed in Section 3 in mind.

4.2 Original SMO07 algorithm

The original algorithm proposed by Samet and Miri in [9] uses a multi-party addition algorithm to perform privacy-preserving k-means clustering on horizontally-partitioned data. We first describe the multi-party addition algorithm.

4.2.1 Paillier Homomorphic Encryption

In order to implement this protocol, parties need to be able to compute functions on the data without being given access to all of it. This will be achieved by using homomorphic encryption to allow operations on encrypted data.

We use the Paillier cryptosystem[7]. This is an additive homomorphic encryption scheme which provides non-deterministic encryption. This means that given any two ciphertexts, we can compute a new cipher text which encrypts the sum of the two original plaintexts without the knowledge of the secret key. i.e. without decrypting the given ciphertexts.

The basic operation is the multiplication of two ciphertexts, which is equivalent to the addition of the underlying plaintexts. More formally, $E(m_1, r_1) \cdot E(m_2, r_2) \pmod{n^2}$ decrypts to $m_1 + m_2$. Also, we can multiply an encrypted plaintext by an unencrypted plaintext through exponentiation. Formally, $E(m_1, r_1)^{m_2} \pmod{n^2}$ decrypts to $m_1 \cdot m_2$.

Now we see how to use these properties to construct a multi-party addition protocol.

4.2.2 SMO07 Multi-Party Addition

This is a summary of the Multi-Party Addition algorithm described in [9].

Let there be k parties, $P_1, P_2, \ldots, P_k$, where each party P_i possesses a private input x_i. The goal of this protocol is for each party P_i to end with a private share ℓ_i such that

$$\sum_{i=1}^{k} x_i = \prod_{i=1}^{k} \ell_i$$

To accomplish this, we will use an additive homomorphic encryption scheme Enc Dec, and KeyGen, where KeyGen creates public encryption key e and private decryption key d.

The steps of the protocol are as follows:

1. P_1 runs KeyGen and gets the keys d and e for this protocol. They broadcast e to all parties.
2. P_1 calculates $\mathsf{Enc}_e(x_1)$ and sends it to P_2.
3. For $i = 2$ through $i = k - 1$ (in order), P_i calculates

$$\mathsf{Enc}_e(x_i) \cdot \prod_{j=1}^{i-1} \mathsf{Enc}_e(x_j)$$

   and sends the result to P_{i+1}.
4. P_k takes the value of $\prod_{i=1}^{k-1} \mathsf{Enc}_e(x_i)$ given to them by P_{k-1} and multiplies it by the encryption of their own input $\mathsf{Enc}_e(x_k)$ to compute $\prod_{i=1}^{k} \mathsf{Enc}_e(x_i)$.
5. P_k randomly selects ℓ_k (that is not the additive identity under Enc), computes ℓ_k^{-1}, and then sends $y_k = \left(\prod_{i=1}^{k} \mathsf{Enc}_e(x_i)\right)^{\ell_k^{-1}}$ to P_{k-1}.
6. For $i = k-1$ to 2, P_i randomly selects a non-identity ℓ_i and corresponding ℓ_i^{-1}, puts the value y_{i+1} they received from P_{i+1} to the power of ℓ_i^{-1}, and sends $y_i = y_{i+1}^{\ell_i^{-1}}$ to P_{i-1}.
7. P_1 decrypts y_2 and sets the calculated value as their share ℓ_1. That is,

$$\ell_1 = \mathsf{Dec}_d(y_2) \tag{1}$$

$$= \mathsf{Dec}_d\left(\left(\prod_{i=1}^{k} \mathsf{Enc}_e(x_i)\right)^{\ell_k^{-1} \cdots \ell_2^{-1}}\right) \tag{2}$$

The end result of this algorithm is that we now have shares ℓ_i of the original secret values x_i such that

$$\sum_{i=1}^{k} x_i = \prod_{i=1}^{k} \ell_i.$$

The original paper also proved correctness and gave a security analysis, which are ommitted here to save space.

4.2.3 SMO07 K-Means

As presented originally in SMO07, the SMO07 Multi-Party Addition (MPA) algorithm is leveraged to compute the means on a set of encrypted data points where different parties own different data points. The protocol is as follows assuming n parties P_1 through P_n:

1. The (public) value k is agreed upon by all parties, and k random initial guesses for the means are broadcasted to all parties.

2. Each party P_i finds the closest mean to each of its data points.

3. Consider the jth of the k means, and call this mean μ_j. Let ℓ_{ji} be the sum of data points owned by party i in cluster j, and let r_{ji} be the count of these data points. By definition,
$$\mu_j = \frac{\sum_{i=1}^{n} \ell_{ji}}{\sum_{i=1}^{n} r_{ji}}.$$
To find this, the MPA algorithm is run separately for the ℓ_{ji} values and the r_{ji} values. So we get $\sum_{i=1}^{n} \ell_{ji} = \prod_{i=1}^{n} p_{ji}$ and $\sum_{i=1}^{n} r_{ji} = \prod_{i=1}^{n} q_{ji}$.

4. P_1 receives all the p_{ji} and q_{ji} values, computes the final quotient μ_j and sends it back to each party.

5. This process is repeated for all j from 1 to k.

5 System Description

Our code can be found at `https://github.com/sarahscheffler/pp-k-means`.

Our system simulates multiple parties by running multiple independent processes. These processes communicate over sockets using the ZeroMQ library. For n parties, we create n processes, each assigned a process number from 0 to $n-1$. This allows each process to know which sockets to communicate on, and follow a protocol that determines the order they send and receive messages in.

We implement a Multi-Party Addition function (*getAddShares*), which when run simultaneously on all of these processes, securely adds the input to each of these functions and returns the result to process 0. Before every iteration of this function, a public-private key pair is generated using the Paillier cryptosystem. Process 0 broadcasts the public key to each party.

Every process has an instance of the $KMeans$ class, which is used in every iteration to compute the local centroids. After the *getAddShares* functions compute global centroids, this $KMeans$ object is updated with the new means.

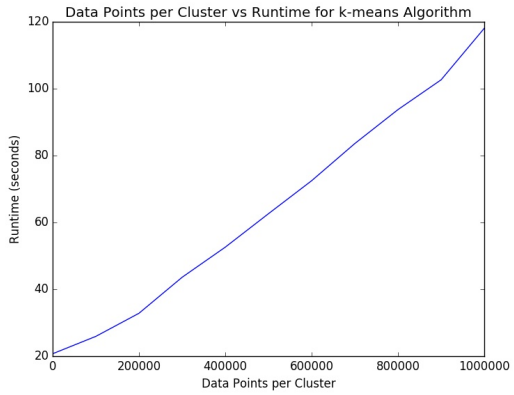

Figure 1: Number of Datapoints vs Runtime

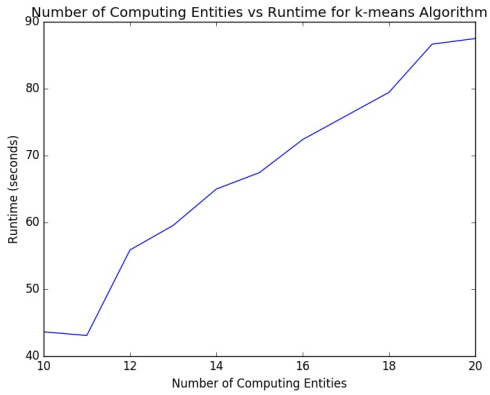

Figure 2: Computing Entities vs Runtime

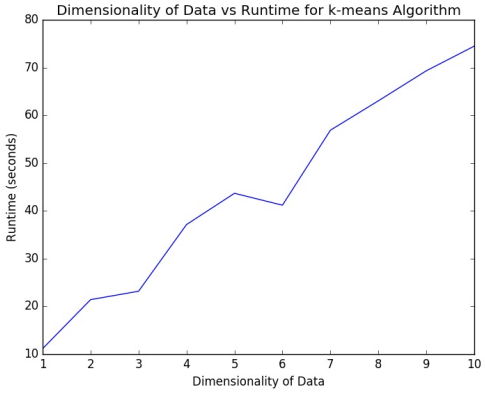

Figure 3: Dimensions of Data vs Runtime

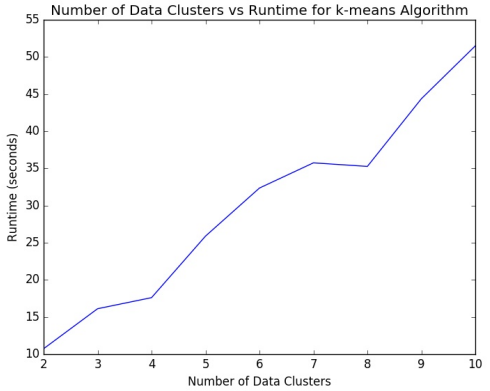

Figure 4: Data Cluster Count vs Runtime

6 Performance Analysis

The original system design upon which our implementation was based did not provide performance metrics. Given our goal of presenting a realistic, performance-optimized private learning system, accurate and clear measures of system performance under a variety of conditions was important. We present such metrics in this section.

We examine four variables independently: the total number of datapoints used, the number of independent computing entities communicating, the dimensionality, and the count of clusters (means) found in the data. Each of these is evaluated independently, with all other variables held constant, and compared to runtime. It is clear that these variables also have synergistic effect: increasing, for example, the number of datapoints as well as the dimensionality of each datapoint will lead to multiplicitive increases in runtime. However, observing changes based on the individual effects of each variable is sufficient to draw intreresting conclusions about the performance of this system.

For the purposes of these analyses, we generated random datasets with specified bounds and distributions. Each parameter-set was run and evaluated over 10 iterations– in realistic situations, the number of iterations would vary with convergence time depending on the data and random initialization of the centroids. These analyses were run on a 64-bit Linux machine with 8GB of RAM and an Intel i5-4690K (4 cores at 3.5GHz) processor.

At a high-level, we see that this system is highly performant under a variety of conditions on the data and participating entities. As shown in figure 1, a trial using 500000 datapoints (over 4 clusters, 5 clients, and 2 dimensions) was completed in under 1 minute. This suggests that even quite high counts of data yield reasonable runtimes, a property not found in many comparable secure machine-learning systems. Additionally, scaling to include 20 clients (with 100000 datapoints) was very reasonable, only increasing runtimes to around 90 seconds. Data gathered from trials varying the dimensionality and number of data clusters showed similar linearity in the results.

We do, however, acknowledge some potential weaknesses in our evaluation metrics. First of all, the lack of direct comparative data hinders us from drawing conclusions about the ability for us to improve the performance of our system. Additionally, the trials were run with networking emulated and all computation occuring on a single machine: clearly, there is substantial performance overhead in communicating between machines, but this had countering effects as the independent processes competed for computing resources. Finally, a more thorough study of this system's feasiblity would include substantially more repitition of trials and utilize multiple data-sources, to observe conditions in reasonable use-cases of this system.

7 Future Work

One of the primary goals of this paper was to explore potential extentions to the framework and algorithm implemented. This section will discuss various improvements and extensions which could be applied based on this work:

7.1 System Improvements

There are various means by which the existing implementation of our algorithm may be improved. From a performance standpoint, python is suboptimal: using a lower-level language could provide opportunities for improvement. Additionally, it may be useful to explore means by which our implementation could be parallelized–this may require reimplementing various libraries, but could provide substantial improvements in this regard.

In consideration of security, we have shown that our implementation stands up to the threat models described in section 3.2. However, it may be possible to extend this system to stand up to other types of attacks. Authentication schemes could be used to guard against faulty or malicious implementations of the protocol. A central managing controller could reduce the possibility for one party to antagonize the rest by, for example, decrypting and distributing the results of the addition protocol. Finally, it

would be valuable to provide insurances against malicious/falsified data by, for example, limiting the number of datapoints any single entity can supply.

Finally, there is more work to be done before this system fills our original goal of "usability". The development of supporting tools to aid organizations in identifying targets for collaborative learning, standardizing data formatting, and providing an easy-to-use frontend would all be valuable extensions in this space.

7.2 Related Protocols

In the process of implementing the k-means clustering algorithm, we established several multi-party computation primitives which could be used in the development of additional machine learning algorithms. The inspiring paper[9] described an implementation of a multi-party ID3 identification-trees algorithm using these same primitives. In addition, we could explore extensions of either the k-means or ID3 trees algorithms to allow for more complex or inconsistant partitionings of data. These would likely be valuable in responding to the nuances and disparities found in typical data sources.

8 Conclusions

We sought to develop a usable, conceptually-scalable implementation of a multi-party machine learning algorithm over encrypted data. We chose to implement and explore the horizontially-partitioned k-means clustering algorithm described in SMO07[9], with the potential for performance and security enhancements. The implementation was coded in python using the Paillier homomorphic encryption libraries [3]. Our evaluation of our implementation suggests that we were successful in developing a performance-friendly algorithm, but there is still substantial room for further development in usability. Future work may involve implementing some of these usability improvements or utilizing the developed computational primitives to create additional machine learning algorithms.

References

[1] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. *Advances in Neural Information Processing Systems*, 2009.

[2] Pengtao Xie et. al. Crypto-nets: Neural networks over encrypted data. *ICLR*, 2015.

[3] Mike Ivanov. Pure python paillier homomorphic cryptosystem, 2011.

[4] L. Guo Y. Guo Y. Fang K. Xu, H. Yue. Privacy-preserving machine learning algorithms for big data systems. *IEEE*, 2015.

[5] Blaine Nelson Benjamin Rubinstein J.D. Tygar Ling Huang, Anthony Joseph. Adversarial machine learning. *AISec*, 2011.

[6] Thomas Ristenpart Matt Fredrikson, Somesh Jha. Model inversion attacks that exploit confidence information and basic countermeasures. *ACM CCS*, 2015.

[7] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology—EUROCRYPT'99*, pages 223–238. Springer, 1999.

[8] Stephen Tu Shafi Goldwasser Raphael Bost, Raluca Ada Popa. Machine learning classification over encrypted data. *IACR*, 2014.

[9] Saeed Samet, Ali Miri, and Luis Orozco-Barbosa. Privacy preserving k-means clustering in multi-party environment. In Javier Hernando, Eduardo Fernández-Medina, and Manu Malek, editors, *Proceedings of the International Conference on Security and Cryptography (SECRYPT 2007)*, pages 381–385. INSTICC Press, 2007.

[10] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. *IEEE*, 2010.

[11] Tom Tistenpart. Exploiting leakage in searchable encryption and machine learning, jan 2016.