CS 203.4860

Project Implementation: Report

Lecturer: Dr. Adi Akavia *Students: Yahel , Ammar , Rana , Aya*

Abstract

In this report we will describe our implementation for Privacy-Preserving Ridge Regression with only Linearly-Homomorphic Encryption according to this **Paper** ,and also an implementation without privacy preserving , we we'll compare between the two implementations, optimizations, tests, complexity and running time.

1 Introduction

In this report we will describe our implementation to train a ridge linear regression model using only Linearly homomorphic encryption as described in the paper. We implemented the project in Python using librariries such as numpy, Paillier and hashlib.

2 Privacy preserving implementation

2.1 Classes and functions:

We've implemented the system as described in the paper.

LabHE class: that is a cryptographic tool as described in the paper with same methods (init which is the same as localGen,labEnc and labDec).

LabeledCiphertext class: which is implementation of ciphertext for labHE.

CSP class: a crypto service provider which takes care of initializing the encryption scheme used in the system and interacts with MLE (phase 1, 2 and protocols as decribed).

MLE class: the Machine-Learning Engine which wants to run a linear regression algorithm on the dataset D obtained by merging the local datasets. (It executes phase 1, 2 as described in the paper).

DO class: the Data-Owners there are m data-owners , each data-owner has a private dataset and is willing to share it only if encrypted to preserve privacy (Phase 1), it gets a public from CSP and sends the encrypted data.

getNetWorkSetup function: it gets number of the data owners as parameter and create all the pipes and queues needed for communication.

run function:it creates the threading and run the system.

setDataByPartition function: utility function, splitting the data by the partition and give it to the data owners.

2.2 Tests and communication complexity:

For testing our implementation we defined a random output (n*d matrix), we run the system on it and we calculated the execution time and communication complexity for different n and d (n-number of data points , d-number of features), here is some of our running results:

n	**d**	**Time(seconds)**	**KB**
3	6	2.191508	37
10	10	20.6696918	197
15	10	39.149615	371
20	10	67.52147293	600
100	20	3329.709427	12390

3 Implementation without privacy preserving

3.1 Classes and functions:

Since this implementation doesn't preserve privacy we don't need threads and CSP and LabHE and the encryption, So we have to classes:

DO Class: the data owner who is sharing the datasets with MLE.

MLE class: it receives the dataset from the data owner and returns w* as defined in the paper.

3.2 Tests and execution time:

For testing our implementation we defined a random output (n*d matrix), we run the system on it and we calculated the execution time for different n and d (n-number of data points , d-number of features), we tested the same values n and d in this implementation and all of them took less than 1 second which much faster.

4 Conclusion:

It's much faster to run implementation that doesn't preserve privacy and with out encryption and it has less functions and communications, but it doesn't protect the privacy of the data which may be sensitive data that we don't want to share.

5 Optimization:

- Using Damgård-Jurik scheme

- Modifying the masking to improve efficiency

- Active security