Machine Learning on Encrypted Data using Homomorphic Encryption

Pablo Duboue, PhD

Grupo Análisis y Procesamiento de Grandes Redes Sociales y Semánticas
UNC Cordoba
Septiembre 2021

Outline

https://github.com/Textualization/riiaa21_ws11_ml_over_encrypted_data

Medical Example (1)

- Imagine a company spends 50 million USD in collecting data
  - It buys medical histories from willing participants
- Using this data, it builds a model that can predict future diseases based on current and past medical issues
- Would people use this system?
  - Disclosing medical history to third-party
  - Results might lead to increased insurance premiums

Medical Example (2)

- Machine Learning techniques on top of Homomorphic Encryption (HE) allow to solve this problem:
  - User encrypts their medical history in their own computer / device
  - User sends encrypted data to provider
    - Plus a public key to encrypt more data if needed
  - Provider executes their model arriving to an encrypted result
    - Provider cannot decrypt the result, it is opaque to them
  - User receives encrypted result from the provider and decrypts it
    - Future potential health complications can be taken into account

Spoiler alert

- The type of Homomorphic Encryption (HE) we will see (CKKS) implies linear operations
  - Moreover, it also implies **vectorized operations**
- This restricts the type of ML models and algorithms that can be used
  - Even more so for **training** models
- A reasonable compromise is to expect the model to have been trained using unencrypted data
  - If the application of the model uses only linear operations it can be brought into HE using CKKS
- Other schemes might be able to handle non-linearities

Familiar Grounds: RSA

- Chances are you have heard of Public Key Encryption (PKE) and the RSA scheme

Rivest, Shamir, Adleman. *On Digital Signatures and Public-Key Cryptosystems*. MIT Computer Science, 1977.

- Still in use, although elliptic curves is becoming more popular
- The algorithm exhibits some homomorphic encryption (HE) capabilities
  - Homomorphic encryption: a property of an encryption scheme that allows certain operations over unencrypted data to be carried over their encrypted counterparts
- We will briefly review RSA before exploring its HE capabilities in a hands on tutorial

RSA Math (1)

- PKE is based on two keys
  - Public key that is known to all parties and can be used to send an encrypted message to the owner of the private key
  - A private key used to decrypt messages
- In the case of RSA:
  - The private key are two large (300 digits in base 10) prime numbers p and q
  - The public key is their multiplication $n = pq$
  - Assumption: factoring very large numbers is very difficult
- To encrypt a message μ, we compute the value of raising μ to an agreed upon power ($e = 65537$ in most implementations):

$$c = \mu^e \mod n$$

- Recovering μ is akin to computing the $1/e$-root of c modulo n, is easy if the prime decomposition of n is known, very hard otherwise

RSA Math (2)

- RSA as defined, exhibits some HE capabilities: it allows multiplications of ciphertexts:
  - If $c = \mu^e \mod n$ and $c' = \mu'^e \mod n$, then

$$\begin{aligned} c * c' &= (\mu^e \mod n) * (\mu'^e \mod n) \\ &= (\mu * \mu')^e \mod n \end{aligned}$$

Hands-on Intro

`https://github.com/Textualization/riiaa21_ws11_ml_over_encrypted_data`

- We will compute RSA by hand,
  - Use ssh-keygen to generate two large primes
  - Read them out using OpenSSL
  - Encrypt and decrypt programmatically
  - Do some HE multiplications
- This notebook is based on the work by Laurent Boué (GitHub: Ranlot) who has shared a longer and more detailed notebook:

`https://github.com/Ranlot/public-key-encryption`

Outline

LWE: Learning With Errors

- Which encryption scheme enables more HE operations?
  - In recent years, there has been a few proposed, we will focus on

Regev. *On lattices, learning with errors, random linear codes, and cryptography*. JACM, 56(6), 1-40. 2009.

- Given a random secret key $s \in \mathbb{Z}_q^n$, we can build a public key by noticing that, given some random numbers $a_i \in \mathbb{Z}_q^n$, recovering the secret from $(1 \leq i \leq n)$:

$$(a_i, b_i) = (a_i, < a_i, s > + e_i)$$

(where e_i are small random noise injected to make the problem difficult enough) is very hard.

- So hard that not even quantum computers can solve it.

LWE (2)

- Using this scheme, expressed in matrix form as $(A, A.s + e)$, we can publish a rearranged public key $p = (-A.s + e, A)$.
- To encrypt a message $\mu \in \mathbb{Z}_q^n$:

$$c = (\mu, 0) + p = (\mu - A.s + e, A) = (c_0, c_1)$$

- To decrypt c using s:

$$\tilde{\mu} = c_o + c_1.s = \mu - A.s + e + A.s = \mu + e \approx \mu$$

- This encryption scheme allows for additions and multiplications.
  - But operations take $O(n^2)$ due to the matrix multiplication

RLWE (1)

- To obtain a usable scheme, we move away from using numbers and matrix to polynomials in integer rings.

Lyubashevsky, Peikert, Regev. *On ideal lattices and learning with errors over rings*. Eurocrypt. 2010.

- The scheme is as before but a, s and e are now randomly sampled **polynomials**
- The advantages of RLWE is that the key is no longer quadratic in size
- There is a very efficient algorithm for multiplication
  - Discrete Fourier Transform for polynomial multiplication
    - $O(n \log n)$

RLWE (2)

- HE addition
  - Given μ and μ', encrypted as $c = (c_0, c_1)$, and $c' = (c'_0, c'_1)$, then

$$c_{\text{add}} = c + c' = (c_0 + c'_0, c_1 + c'_1) = (c_{\text{add},0}, c_{\text{add},1})$$

  - If we decrypt as before:

$$\begin{aligned} \tilde{\mu}_{\text{add}} &= c_{\text{add},0} + c_{\text{add},1}.s \\ &= c_0 + c'_0 + (c_1 + c'_1).s \\ &= c_0 + c_1.s + c'_o + c'_1.s \\ &= \mu + \mu' + 2e \approx \mu + \mu' \end{aligned}$$

Relinearization

- Multiplication also works **but** the polynomial degree increases
  - That is equivalent to say the secret increased in size
  - The size of the polynomials grows exponentially with the number of multiplications
  - Unusuable in practice
- However, it is possible to use the secret key to compute a **multiplication key** that can squash the higher order terms in the resulting polynomial
- We will skip the details but the importance bit here is:
  - Multiplication is possible using this scheme
  - It needs a key (to be shared similar to the public key) that allows for relinearization

From data to polynomials

- To arrive to a usable HE scheme, we need to be able to convert vectors of data into polynomials
- This can be achieved by representing the data as the valuation of the polynomial in a set of "given points"
- This is called the **canonical embedding** into polynomials in the complex plane
- Decoding is thus simple, but encoding presents issues unless the "given points" allow for easy computation
  - We use the "roots of the cyclotomic polynomial"

From complex to real numbers

- We want to work over real not complex numbers though
- This can be done using an approximation

Cheon, **K**im, **K**im, **S**ong. *Homomorphic encryption for arithmetic of approximate numbers*. ASIACRYPT 2017.

- This is the 'scale_factor' parameter in the HE library we will be using

Hands on: CKKS

- We will now see the full process of creating the polynomials, using `numpy.polynomial`
  - For actual CKKS we will use an optimized library
  - The point is to appreciate a vector of numbers is transformed into a polynomial very different from the original numbers
  - These are the encoded plaintexts, to encrypt them using RLWE, a suitable polynomial key s and polynomial data a has to be defined and the multiplications carried on
- This notebook is based on the work by Daniel Huynh (GitHub: dhuynh95) who has shared a longer and more detailed notebook:

`github.com/dhuynh95/homomorphic_encryption_intro`

Outline

Linear everything

- HE using CKKS implies linear operations
  - Moreover, it also implies **vectorized operations**
- This restricts the type of models and algorithms that can be used
  - Even more so for **training** models
- A reasonable compromise is to expect the model to have been trained using unencrypted data
  - If the application of the model uses only linear operations it can be brought into HE using CKKS
- Other schemes might be able to handle non-linearities

Libraries

- Paillier
- Palisade
- Pyfhel
- Seal
- Syft
- Tfhe

Palisade

`https://palisade-crypto.org/`

- From the New Jersey Institute of Technology (NJIT)
- 26 different *organizations* that have contributed to its development
- C++ developed
- Focus on easy experimentation
- Plug-and-play framework
- US defense research funding and others

Linear SVM (1)

- A popular linear model in ML are linear SVMs
- Given a feature vector f_i with n entries, a linear SVM model is defined by n coefficients $\beta_i$and a bias term b
- A prediction is $\sum \beta_i f_i - b \geq 1 \implies y_i = 1$

Linear SVM (2)

- To train a SVM over data that is not linearly separable, a hinge loss can be used
  - Allow some training instances to be misclassified
- The separation hyperplane is learned by an optimization process

Other Algorithms

- In the next section, we will see a recommendation example
- There are random forests and neural networks proposed in the literature
  - Not familiar with their details
  - If interested, look at Daniel Huynh's master thesis

Runtime considerations

- It is quite slow and memory intensive
- Needs to use the vectorization to be usable
  - In the recommendation setting, trying to encode a row in the clients $\times$ products matrix takes 10Gb of RAM
- However, accessing numbers within each vector is highly non-trivial
  - Because the numbers are unknown and they are the value of evaluating the polynomial on certain numbers!

Parameters

- The main parameters to set up are:
  - Multiplicative depth
    - Not the total number of multiplications but the maximum depth of the tree formula
    - Can be reduced with clever rearrangement of formulas
    - E.g.: $A * B * C * D$ has a depth of 3 but $(A * B) * (C * D)$ has a depth of 2
  - Scale factor
    - For CKKS transformation from complex numbers to integers
  - Batch size
    - Size of the vector in vectorial calculations
  - Security level, according to the standard in `http://homomorphicencryption.org`
    - E.g., STD128 (HE standard set with more than 128 bits of security w.r.t. classical computer attacks)
    - STD128Q (same as above but security w.r.t. quantum computer attacks)

Technical issues

- Palisade is a C++ library, built using CMake
- The palisade-python-demo uses CMake and LibBoost-Python to expose a very limited functionality of the library
- Using an Ubuntu 18.04 virtual machine, we have compiled the library so it uses the same C library as Google Colab
  - The binaries are included in a zip file in the workshop's repo
- The CMake in Ubuntu 18.04 couldn't compile the python library
  - Instead we reverse engineered the compiler and linker command lines and compiled without a Makefile
- We also extended the adapter library to expose additional functionality we need for the recommender
  - The modified source code is available in the workshop's repo and the notebook starts by compiling the library
  - Any changes to the adapter library C++ code can easily be integrated

Hands on: Palisade Python demo

`https://github.com/Textualization/riiaa21_ws11_ml_over_encrypted_data`

- We will now see the python demo of the Palisade project
  - It loads trained linear SVMs for a few datasets
    - Then runs the linear SVMs on plain data, on encrypted data and using an encrypted model
    - Verifies all the runs produce the same results (up to an approximation error)
- From there, we will play encoding some numbers and performing operations on them
- This notebook is an adaptation of the Palisade Python Demo by David Bruce Cousins, Andrey Kim and Yuriy Polyakov from the NJIT:

`https://gitlab.com/palisade/palisade-python-demo`

Outline

Collaborative Filtering (Recommendation)

- Filter information using communities of users
- Key concepts: users, items and preferences
- User-based recommendation:
- For each item i that the user u has no preference:
  - for each user v that has a preference for i:
    - compute the similarity s between u and v
    - accumulate the preference of v for i, weighted by s
  - return the items with higher weighted sums

User Similarity Metrics

- Euclidean
- Tanimoto
- Log-likelihood
- Pearson

Details: `http://aprendizajengrande.net/clases/clase8`

Recommendation in Palisade (1)

- In our context, users = clients, items = products and preferences = number of times a client purchased a product
- We store the matrix of users×items in vectorized (batched) form:
  - user 1: polynomial for batch 1, polynomial for batch 2, ...
  - user 2: polynomial for batch 1, polynomial for batch 2, ...
- We will use an approximate Euclidean distance by computing the square of the Euclidean distance:
  - Euclidean = $\sqrt{\sum(x_i - y_i)^2}$
- Because the data is encrypted, we don't know which entries are zero so we cannot use the optimized version of the recommendation system
  - We compute preferences for products the clients have already expressed preference
    - We leave the filtering of those at the client-side

Recommendation in Palisade (2)

- Given a target customer for whom we are computing recommendations, we thus compute
  - The negation of its preference row, N
  - The weights for all the other users, defined as $w_i = \sum(P_i + N)^2$
- Given these weights, we need to multiply that scalar by the full row (i.e., $w_i P_i$)
  - But Palisade does not allow to do that directly
    - P_i is encoded as a few (two in our case) batched polynomials
    - w_i is also a batched polynomial, with only the first entry in the vector being non-zero
    - $w_i P_i$ zeros all the other entries in P_i
  - We wrote a "broadcast" function that transforms w_i into a vector with w_i in each coordinate
    - Uses the polynomial rotation functionality contained in Palisade

Synthetic Data

- To have recommendations that might make any sense, we run the algorithm over *synthetic data* obtained from real preference data as released by the GroupLens Research group:

  `https://grouplens.org/datasets/movielens/`

- The small sample we are using contains 600+ users and 8400+ movies from the '90s
  - It should allow easy evaluation of the preferences recommended
- Ratings above 3 were transformed as products being purchased (say, from a 1990s videoclub) by different customers
  - The number of times a product has been purchased results in the "rating" given by a customer
- To highlight a real system, this information was put into electronic invoice format

Hands on: Putting it all together

`https://github.com/Textualization/riiaa21_ws11_ml_over_encrypted_data`

- The notebook first reads the invoices and builds a table of clients×products
- It computes preferences for one client using NumPy
- It then encrypts the data using Palisade and computes encrypted preferences

Encrypted data and trust

- While the technology might work, deploying it as a commercial offering is difficult
- The user is still sharing the data in an encrypted fashion
  - If there were bugs, it might be decrypted later on
  - Alternatively, the parameters employed might not be as secured as initially mustered
- Example: Debian SSH
  - In 2006 a Debian maintainer mistakenly "improved" the OpenSSL source seriously reducing the number of keys
  - This was later discovered by Luciano Bello (CVE-2008-0166)
  - All the data was encrypted, it just a simple list of all keys
    - A single user couldn't realize that, because it'll have to see the keys used by other people
- A HE solution requires trust from the users on the company and from the company on its tools
  - And in the absence of trust, an expedite legal system to seek monetary remedies when the trust has been breached

Learning more

- en.wikipedia.org/wiki/Homomorphic_encryption
- Daniel Huynh blog series
  - https://blog.openmined.org/ckks-explained-part-1-simple-encoding-and-decoding/
- http://homomorphicencryption.org
- Palisade
  - Manual: https://palisade-crypto.org/documentation/
  - Webinars: https://palisade-crypto.org/webinars/

Conclusions

- HE ideas and implementations are coming of age
  - Libraries are available to develop some ML solutions
  - Currently it implies vectorized linear operations
    - This restricts the type of ML models and algorithms that can be used
    - Expect the model to have been trained using unencrypted data
- The field is growing at a very fast pace
  - New developments will enable greater ML algorithms to be migrated over to HE
  - What is secure and what is not is still being fully worked on
- Private data at both private (companies) and governmental levels need HE both to:
  - Live up to its full potential
  - Maintain the privacy level required by corporate governance / law