Laboratory in Secure Computation in the Cloud  
Dr. Adi Akavia University of Haifa, Fall 2019

Privacy-preserving machine learning SVM inference on FHE encrypted data

Savranevski Gal

# Abstract

# Introduction

With the popularity of Cloud computing, more and more user’s data are stored on Cloud servers.

Cloud servers also enable computation power service by demand.

This raises concerns about privacy. The traditional encryption schemes do not allow running any computation on encrypted data. So, we need to choose between storing our data encrypted in the cloud and downloading it to perform any useful operations or providing the decryption key to service providers which risks our privacy.

Homomorphic encryption can solve this problem. If the user encrypts the data using homomorphic encryption, the Cloud can perform meaningful computation on the encrypted data in order to provide services to users without revealing the user’s privacy.

Machine learning algorithms are being more and more popular and efficient, but for training and making predictions, one need to share his data.

In may cases this data is private, for example a hospital wants to predict wither a patient is at hurt attack risk group. For model training the hospital need to share many patients medical data, and for prediction the patient his person data. Another example is a business company with a commercial data, that wants to train a machine learning model and then use it to predict some desired events.

Privacy-preserving machine learning enables secure outsourcing of machine learning tasks to an untrusted service provider (server) while preserving the privacy of the user’s data (client). Attaining good concrete efficiency for complicated machine learning tasks, such as support vector machine (SVM), is one of the challenges in this area

Microsoft’s Simple Encrypted Arithmetic Library (SEAL) is a [free](https://en.wikipedia.org/wiki/Free_software) and [open-source](https://en.wikipedia.org/wiki/Open-source_software) [cross platform](https://en.wikipedia.org/wiki/Cross_platform) software library developed by [Microsoft Research](https://en.wikipedia.org/wiki/Microsoft_Research) that implements various forms of [homomorphic encryption](https://en.wikipedia.org/wiki/Homomorphic_encryption).

This project task was to train a SVM model on plaintext data, design an arithmetic circuit for SVM inference, implement secure inference on encrypted data using SEAL.

Main contribution of my work is a generalized high-performance concept implementation of the SVM algorithm which suitable for all SVM models with linear and polynomial kernels.

Two SIMD technics are shown and implemented in my work; these technics can be used stand alone in various other SEAL applications.

The first is a technic for SIMD inner product implementation.

Second technic is embedding many samples into one cipher text to simultaneously calculating many predictions.

I show a model where the svm coefficients are used as a plaintext but a simple adoption can be made for an encrypted svm coefficients.

# Theoretical background

## **F**ully **H**omomorphic **E**ncryption (FHE)

The idea of fully homomorphic encryption was first proposed by Rivest et al. in the 1970s. Compared with the general encryption algorithm, homomorphic encryption can let you do computation on ciphertexts without any decryption process.

HE, clients who sent encrypted data to an untrusted server are guaranteed data privacy, and the server can perform any operations over the encrypted data

. An encryption function with homomorphism is an encryption function in which two plaintexts a and b satisfy Dec(Enc(a) LEnc(b)) = a Nb, where Enc stands for encryption, and Dec stands for decryption, L N , corresponds to operations on the plaintext and ciphertext. When L represents addition on the plaintext, the encryption is said to be an additive homomorphism: when L represents multiplication on the plaintext, the encryption is said to be a multiplicative homomorphism. A cryptosystem that supports arbitrary computation on ciphertexts is known as fully homomorphic encryption (FHE). That is, Dec(f(Enc(m1), Enc(m2), ..., Enc(mk))) = f(m1, m2, ..., mk), or written as: f(Enc(m1), Enc(m2), ..., Enc(mk)) = Enc(f(m1, m2, ..., mk)), for arbitrary function f. How to construct a fully homomorphic encryption scheme is an open challenge. Until 2009, Gentry proposed the first fully homomorphic cryptosystem based on the ideal lattice[1], which made a breakthrough in this field. Then many cryptographers have done meaningful work in the research of the homomorphic encryption scheme.

https://shaih.github.io/pubs/he-chapter.pdf

## SVM

<https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation>

# Implementation details

Tools:

### Scikit-learn

Scikit-learn is an open source machine learning Python library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities.

Scikit-learn provides dozens of built-in machine learning algorithms and models, called [estimators](https://scikit-learn.org/stable/glossary.html#term-estimators). Each estimator can be fitted to some data using its [fit](https://scikit-learn.org/stable/glossary.html#term-fit) method.

Scikit provides build in methods for:

1. A pre-processing step that transforms or imputes the data.
2. Fitting in the training phase
3. A final predictor that predicts target values
4. Functions for model evaluation like cross validation
5. Functions for automatically find the best parameter combinations (via cross-validation).

For further details one can refer to: <https://scikit-learn.org/stable/index.html>

### Anaconda and Jupyter notebook

Anaconda is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source) distribution of the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) and [R](https://en.wikipedia.org/wiki/R_(programming_language)) programming languages for [scientific computing](https://en.wikipedia.org/wiki/Scientific_computing) , that aims to simplify [package management](https://en.wikipedia.org/wiki/Package_management) and deployment. Package versions are managed by the [package management system](https://en.wikipedia.org/wiki/Package_manager) [conda](https://en.wikipedia.org/wiki/Conda_(package_manager)" \o "Conda (package manager)). The Anaconda distribution includes data-science packages suitable for Windows, Linux, and MacOS.

Anaconda Navigator is a desktop [graphical user interface (GUI)](https://en.wikipedia.org/wiki/Graphical_user_interface) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using [command-line commands](https://en.wikipedia.org/wiki/Command-line_interface). Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for [Windows](https://en.wikipedia.org/wiki/Windows), [macOS](https://en.wikipedia.org/wiki/MacOS) and [Linux](https://en.wikipedia.org/wiki/Linux).

Jupyter notebook is a [web-based interactive](https://en.wikipedia.org/wiki/Rich_Internet_application) computational environment for creating Jupyter notebook documents . Notebook documents can contain phyton code that can be executed on the spot through a notebook kernel. A notebook *kernel* is a “computational engine” that executes the code contained in a [Notebook document](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document). The *ipython kernel* executes python code. Kernels for many other languages exist.

In my work I found more convenient to work with Jupyter notebook , rather with more sophisticated editor like phycharm.

### SEAL

Microsoft SEAL is a homomorphic encryption library that allows additions and multiplications to be performed on encrypted integers or real numbers. Other operations, such as encrypted comparison, sorting, or regular expressions, are in most cases not feasible to evaluate on encrypted data using this technology. Therefore, only specific privacy-critical cloud computation parts of programs should be implemented with Microsoft SEAL.

Microsoft’s SEAL Homomorphic Encryption library implements two homomorphic cryptographic schemes, BFV scheme and CKKS scheme.

### Visual studio

Microsoft visual studio 2019 is used as the code editor and compiler.

### GitHub

I have used GitHb as my source control.

All the code the uploaded to the following repository :

<https://github.com/GALSAV/SecureCloudComputingSEAL>

Work details

### Assignment

The laboratory assignment was to train an

### DataSets and Training phase

In my work I’ve used 2 datasets

1. Iris dataset : <https://archive.ics.uci.edu/ml/datasets/Iris>
2. Mashroom dataset : <https://archive.ics.uci.edu/ml/datasets/mushroom>

Iris dataset , is a simple datasets which is used in many beginners guide’s for machine learning.

The data set contains 3 classes of 50 samples each, where each class refers to a type of iris plant.

Each sample has 4 attributes : sepal length, sepal width, petal length, petal width .

All attributes are measured in cm.

One class is linearly separable from the other 2. The Predicted attribute is a class of iris plant.

In my work I’ve have used only 2 classes Iris setosa and Iris versicolor which are linear separable.

Mashroom dataset, contains 8124 samples with 22 attributes each.

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended.  This latter class was combined with the poisonous one.

The Predicted attribute is one of two classes: edible or poisonous.

### Training Pahse

For training phase I used scikit to train a model with 20% of the data , and get the best separation parametrs .

#### Iris dataset

Iris Clasiffication can be done with linear kernel with 100% accurancy, the scikit phyton training code can be found at :

From the scikit we get the svm model parametrs which is used with c# code described bellow.

Futher details and classification of this dataset can be found at :

<https://www.kaggle.com/ash316/ml-from-scratch-with-iris>

<https://www.kaggle.com/biphili/seaborn-matplotlib-plot-to-visualize-iris-data>

#### Mashroom dataset

Mashroom Clasiffication can be done with linear polynimial with 100% accurancy, the scikit phyton training code can be found at :

It worth to mantion that this dataset needed to “data preparation” and

No dimension reduction was applied to the dataset intentionally to challenge the secure version of SVM

Futher details and classification of this dataset can be found at :

<https://www.kaggle.com/uciml/mushroom-classification/kernels>

<https://www.kaggle.com/mig555/mushroom-classification>

<https://www.kaggle.com/nirajvermafcb/comparing-various-ml-models-roc-curve-comparison>

### Implement Plain SVM

For better understanding and performance comparison I’ve implemented the plain version of SVM which is:

1. Initialized with the classification parametrs
2. Run prediction for a given example
3. The code also configurability outputs performance data such as
   1. Intermediate and final calculation result to help debugging the secure code
   2. Time performance data

The Plain SVM implementation can be found in PlainSVC project .

The code was adopted to c# from generated java and c++ code by phython sklearn library named **sklearn-porter . More information for this library can be found at** [**https://pypi.org/project/sklearn-porter/**](https://pypi.org/project/sklearn-porter/) **the library is distributed undel MIT licence.**

### Implement Secure SVM

The work of implementing secure svm was iterative from a very custom low performance implementation to iris datset only, to a gerelized secure svm implementation with powerfull batching and high performance.

I choose to work with CKKS scheme , as the SEAL documentation recommended for machine learning applications and because of the real vectors used in SVM algorithms.

The alternative was to work with BFV Scheme and to work with integer value datasets or to translate real numbers to integers ( multiply by factor) and after all calculations translate back to real result with some precision lost .

#### Simple custom implementation

At first I’ve implemented a very customize code for iris dataset only , no loops , custom made data structures (fixed size array ) , linear kernel only.

The code can be found at IrisSVNSecured project , IrisSimple file , IrisSecureSVC class .

<https://github.com/GALSAV/SecureCloudComputingSEAL/blob/master/IrisSVNSecured/IrisSimple.cs>

#### Generalized linear SVM implementation

Implementation goal was to generalize the simple implementation for running secure svm on any classification dataset. It used common programing tools such as : loops, general data structures .

This implementation handles linear kernel only.

This implementation can be found at IrisSVNSecured project , IrisGeneral file , IrisSecureSVC.class

<https://github.com/GALSAV/SecureCloudComputingSEAL/blob/master/IrisSVNSecured/IrisGeneral.cs>

#### SIMD Inner product Generalized linear SVM implementation

Implementation goal was to implement inner product functionality as a SIMD (**Single instruction, multiple data**) operation.

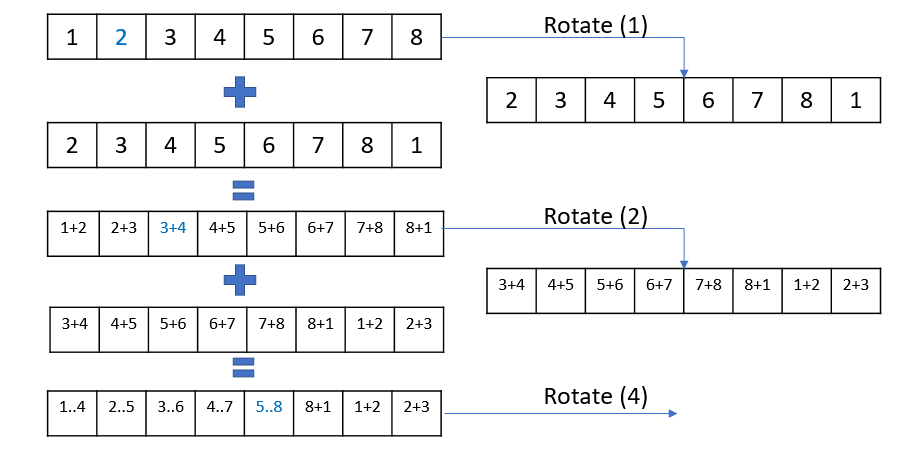
Motivation : SVM formula for linear and polynomial kernels contains calculation of inner product between sample feature vectors and support vectors feature vectors.

In Seal we can encode each of the vectors as a single ciphertext or plaintext ,then vector multiply them

The result is a ciphertext contains ( 0<i<n).

Another operation which can be applied to a ciphertext is rotation which rotate the encrypted vector left or right by any number.

To sum calculation can be done by the Illustrated algorithm:



In each iteration i ( 0 < i < ) we rotate the ciphertext left by and add it to ciphertext before the rotation. After steps , the first element of the ciphertext (after decrypting and decoding) contains the sum of all vector elements.

The code using SIMD inner product can be found :

The code can be found at IrisSVNSecured project , IrisGeneralBatch file , IrisSecureSVC class .

<https://github.com/GALSAV/SecureCloudComputingSEAL/blob/master/IrisSVNSecured/IrisGeneralBatchPoly.cs>

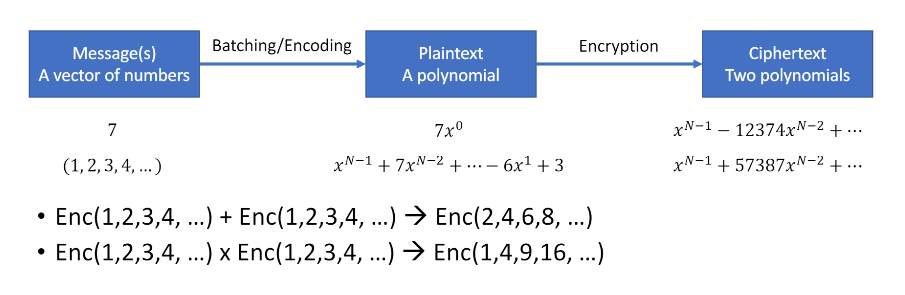
This code is also contains implementation of the polynomial kernel.

#### Batch Generalized linear SVM implementation

Implementation goal was to implement a batched version of the classification algorithm.

The main idea is to encode multiple samples for classification to one cipher text and to take advantage of SEAL native SIMD to classify many samples simultaneously.

Each SEAL computation is a SIMD computation, e.g the operation is an element-wise operation.



The encoded plaintext or ciphertext contains N/2 slots , where N = PolyModulusDegree.

Each slot encodes one real or complex number.

PolyModulusDegree representing the degree of a power-of-two cyclotomic polynomial (must be a positive power of 2).

Larger PolyModulusDegree makes ciphertext sizes larger and all operations slower, but enables more complicated encrypted computations.

The CKKSEncoder will implicitly pad the encoded plaintext with zeros to full size (N / 2).

All the operations in SEAL are element-wise vector operation, I took advantage of this feature to embed in one cipher text many samples , so in one predict function call I make classification for many samples.

The amount of samples that can be embeeded in one ciphertext is depends on the number of features of each sample and the choosen PolyModulusDegree .

Another small notice, Because the computation of Inner product using vector rotation and summing, after each sample I pad with |feature size| zeros , so the values of the presiding sample won’t mix with a former one.

I used PolyModulusDegree of 16384 and 32768 so the number of slots is 8192 and 16384 respectively.

Iris sample has 4 features, so in each cipher text can be embedded 1024 (8192/8) and 2048 (16384/8) samples (though all the dataset is 100 samples).

Mushroom sample has 22 features, so in each cipher text can be embedded ~180 (8192/44) and ~370(16384/8) samples. For running prediction on all the dataset , 46 iterations needed for PolyModulusDegree of 16384 , and 23 for PolyModulusDegree of 32768.

The code can be found at SecureSVC project , SVC file , SecureSVC class .

<https://github.com/GALSAV/SecureCloudComputingSEAL/blob/master/SecureSVC/SVC.cs>

#### Parallel computation

The motivation is to take advantage of many cores of the server side.

The server can initiate many instances of the classification class and run prediction in parallel.

In this case the client can send to the server the encryption parameters which are idenditical for all the predictions, and all the cipher text samples .

The server will run as much predictions in parallel as his available cores.

This parallel pattern is shown for concept purpose only therefor the parallel implementation is merged with the Batch Generalized linear SVM implementation. The split here is demonstrated at client side , but the same concept can be applied at the server side.

The code can be found at MashroomSecured project , MashroomSecured file, Main class .

<https://github.com/GALSAV/SecureCloudComputingSEAL/blob/master/MashroomSecured/MashroomSecured.cs>

while setting IsParallel variable to true.

# Machine Learning

# SVM

“Support Vector Machine” (SVM) is a supervised [machine learning algorithm](https://courses.analyticsvidhya.com/courses/introduction-to-data-science-2?utm_source=blog&utm_medium=understandingsupportvectormachinearticle) which can be used for both classification or regression challenges. However,  it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the classes.

A classification task usually involves separating data into training and testing sets. Each instance in the training set contains one “target value” (i.e. the class labels) and several “attributes” (i.e. the features or observed variables). The goal of SVM is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes. Given a training set of instance-label pairs (xi , yi), i = 1, . . . , l where xi ∈ Rn and y ∈ {1, −1} l , the support vector machines (SVM) (Boser et al., 1992; Cortes and Vapnik, 1995) require the solution of the following optimization problem: min w,b,ξ 1 2 w T w + C X l i=1 ξi subject to yi(w T φ(xi) + b) ≥ 1 − ξi , (1) ξi ≥ 0. 1 Table 1: Problem characteristics and performance comparisons.

Here training vectors xi are mapped into a higher (maybe infinite) dimensional space by the function φ. SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. C > 0 is the penalty parameter of the error term. Furthermore, K(xi , xj ) ≡ φ(xi) T φ(xj ) is called the kernel function. Though new kernels are being proposed by researchers, beginners may find in SVM books the following four basic kernels: • linear: K(xi , xj ) = x T i xj . • polynomial: K(xi , xj ) = (γxi T xj + r) d , γ > 0. • radial basis function (RBF): K(xi , xj ) = exp(−γkxi − xjk 2 ), γ > 0. • sigmoid: K(xi , xj ) = tanh(γxi T xj + r). Here, γ, r, and d are kernel parameters.

We propose that beginners try the following procedure first: • Transform data to the format of an SVM package • Conduct simple scaling on the data • Consider the RBF kernel K(x, y) = e −γkx−yk 2 • Use cross-validation to find the best parameter C and γ • Use the best parameter C and γ to train the whole training set5 • Test

2 Data Preprocessing 2.1 Categorical Feature SVM requires that each data instance is represented as a vector of real numbers. Hence, if there are categorical attributes, we first have to convert them into numeric data. We recommend using m numbers to represent an m-category attribute. Only one of the m numbers is one, and others are zero. For example, a three-category attribute such as {red, green, blue} can be represented as (0,0,1), (0,1,0), and (1,0,0). Our experience indicates that if the number of values in an attribute is not too large, this coding might be more stable than using a single number. 5The best parameter might be affected by the size of data set but in practice the one obtained from cross-validation is already suitable for the whole training set. 3 2.2 Scaling Scaling before applying SVM is very important. Part 2 of Sarle’s Neural Networks FAQ Sarle (1997) explains the importance of this and most of considerations also apply to SVM. The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Because kernel values usually depend on the inner products of feature vectors, e.g. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems. We recommend linearly scaling each attribute to the range [−1, +1] or [0, 1]. Of course we have to use the same method to scale both training and testing data. For example, suppose that we scaled the first attribute of training data from [−10, +10] to [−1, +1]. If the first attribute of testing data lies in the range [−11, +8], we must scale the testing data to [−1.1, +0.8]. See Appendix B for some real examples. 3 Model Selection Though there are only four common kernels mentioned in Section 1, we must decide which one to try first. Then the penalty parameter C and kernel parameters are chosen

Find a linear decision surface (“hyperplane”) that can separate patient classes and has the largest distance (i.e., largest “gap” or “margin”) between border-line patients (i.e., “support vectors”);

• If such linear decision surface does not exist, the data is mapped into a much higher dimensional space (“feature space”) where the separating decision surface is found; • The feature space is constructed via very clever mathematical projection (“kernel trick”).

Binary classification:

<http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf>

Given training data (xi, yi) for i = 1 ...N, with xi ∈ Rd and yi ∈ {−1, 1}, learn a classifier f(x) such that f(xi) ( ≥ 0 yi = +1 < 0 yi = −1 i.e. yif(xi) > 0 for a correct classification.

# SEAL