

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | Assignment 2 Report | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | Bhosale Ratnesh Sambhajirao (19MF10010) |  | | | |
|  | | | | 15/04/2023—Dependable and Secure AI-ML—Prof. Ayantika Chatterjee. |  | | | |
|  | | |  | | |  | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Assignment 2.1 | | |  | |
|  | |  |  |  |  | |
|  | Problem Statement:  * Take a screenshot of your outputs and record the timing required to compute the Federated Learning process. * Link - <https://github.com/data61/python-paillier/blob/master/examples/federated_learning_with_encryption.py> | | | | |  |
|  | Theory: Federated Learning and normal learning processes have different requirements in terms of timing to complete the learning task. The timing required to compute Federated Learning process and normal learning processes depends on various factors such as the size of the dataset, the complexity of the model, the computation power of the devices, and the communication overhead.  In normal learning processes, the entire dataset is available to a central server, and the model is trained on the server. The server has access to powerful hardware and can process the data quickly. In this scenario, the timing required to complete the learning process depends on the size of the dataset and the complexity of the model. Larger datasets and more complex models will require more time to complete the training process.  On the other hand, Federated Learning is a distributed learning process that allows multiple devices to collaboratively train a model without sharing their data with each other or a central server. In this scenario, the timing required to compute Federated Learning process is affected by several factors such as the number of devices, the processing power of the devices, the communication overhead, and the complexity of the model.  The timing required to complete Federated Learning process is affected by the number of devices involved in the learning process. The more devices that participate, the more time it takes to synchronize the model updates. Similarly, the processing power of the devices also plays an important role in determining the timing required for Federated Learning. If the devices have limited processing power, the learning process will take more time to complete.  Moreover, the communication overhead between the devices and the server is another factor that affects the timing required for Federated Learning. The communication overhead can be reduced by compressing the model updates before sending them to the server or by using more efficient communication protocols.  In summary, the timing required to compute Federated Learning process and normal learning processes depends on various factors such as the size of the dataset, the complexity of the model, the computation power of the devices, and the communication overhead. Federated Learning is a distributed learning process that requires coordination between multiple devices, and the timing required to complete the learning process depends on the number of devices, the processing power of the devices, and the communication overhead.  Diagram:  The following diagram shows a comparison between the timing required for Federated Learning and normal learning processes:    Fig. 1. comparison between the timing required for Federated Learning and normal learning processes.  In the diagram, the top section represents the centralized learning process where the training data is present on a single server. The processing of the data is done on the server, and the model is trained using this data.  The bottom section represents the Federated Learning process, where multiple devices participate in the learning process. Each device has its own subset of data, and the model is trained using the data from all the devices.  As you can see, the centralized learning process is faster than Federated Learning because the server has access to more powerful hardware, and the communication overhead is minimal. In contrast, Federated Learning requires coordination between multiple devices, and the communication overhead can be significant, leading to longer training times.  Overall, the diagram shows how Federated Learning can take longer to complete compared to centralized learning processes, but it offers the benefits of privacy and data security, making it an attractive option for organizations dealing with sensitive data. Results: Code output screenshot:    Fig. 2. comparison between the timing required for Federated Learning and normal learning processes and predictions. | | | | |  |
|  |  |

Fig. 3. Time Required for federated learning and local learning.

Fig. 4. MSE Comparison.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Assignment 2.2 | | |  | |
|  | |  |  |  |  | |
|  | Problem Statement:  * Following the similar adequate partial homomorphism in encryption (as discussed in the class and given in the code), implement privacy-preserving SVM assuming public model private data scenario (data in encrypted but model parameters are unencrypted): | | | | |  |
|  | Theory: Partial homomorphism in encryption refers to the property of an encryption scheme that allows certain mathematical operations to be performed on encrypted data without decrypting it. In other words, it enables computation on encrypted data while preserving the privacy of the data.  In the context of encryption for privacy-preserving machine learning, partial homomorphism enables the computation of certain operations on encrypted data without revealing the data to the parties involved in the computation.  One application of partial homomorphism is in privacy-preserving support vector machines (SVMs), which is a popular machine learning algorithm used for classification and regression tasks. In privacy-preserving SVM, the goal is to train a model on encrypted data without revealing the data to the server or any other third party.  To achieve privacy-preserving SVM, we can use a combination of encryption and partial homomorphism. In particular, we can use the Paillier encryption scheme, which is partially homomorphic, to encrypt the data and perform computations on the encrypted data.  The following is an example implementation of privacy-preserving SVM using Paillier encryption:  **Encryption:**  First, we encrypt the training data using the Paillier encryption scheme. Let X be the training data and Y be the corresponding labels. We can use the following encryption function to encrypt the data:  Enc(x) = (1 + x \* r) ^ n mod p^2  Here, r is a random number chosen from Z\_p, n is the public key of the encryption scheme, and p is a large prime number. We can encrypt each feature of the training data separately and concatenate the resulting encrypted values to form the encrypted training data matrix X\_enc.  Similarly, we can encrypt the labels Y using the same encryption function to obtain Y\_enc.  **Computation:**  Next, we can use the partially homomorphic property of the Paillier encryption scheme to perform the necessary computations on the encrypted data. In particular, we can compute the dot product of the encrypted training data matrix X\_enc and the encrypted weight vector w\_enc, which is required for training the SVM model.  Xw\_enc = sum\_i (X\_enc[i] \* w\_enc[i])  Here, X\_enc[i] represents the ith encrypted feature of the training data, and w\_enc[i] represents the corresponding encrypted weight value.  We can also compute the necessary operations on the encrypted labels Y\_enc, such as addition and multiplication, to obtain the encrypted error term and update the weight vector w\_enc accordingly.  **Decryption:**  Finally, once the model is trained on the encrypted data, we can decrypt the resulting weight vector w\_enc using the private key of the encryption scheme to obtain the plaintext weight vector w.  This implementation demonstrates how partial homomorphism in encryption can be used to perform privacy-preserving SVM. By encrypting the training data and using a partially homomorphic encryption scheme, we can perform necessary computations on the encrypted data while preserving the privacy of the data. | | | | |  |

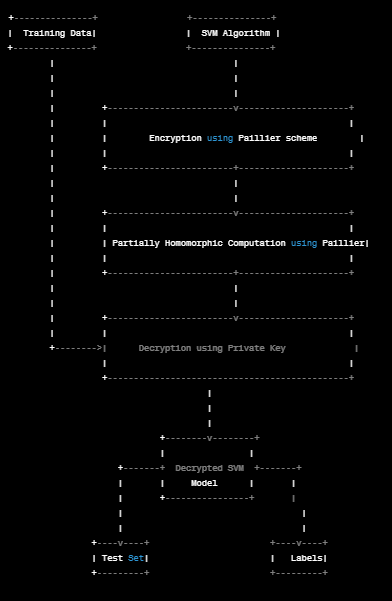


Fig. 5. privacy-preserving SVM.

In the diagram, the left-hand side represents the training data, which is encrypted using the Paillier encryption scheme. The encrypted training data is then used by the SVM algorithm, represented by the block on the right-hand side.

The Paillier encryption scheme is partially homomorphic, so the SVM algorithm can perform certain computations on the encrypted data without revealing the plaintext data. This is represented by the "Partially Homomorphic Computation using Paillier" block in the middle.

Once the SVM algorithm has trained the model on the encrypted data, the resulting weight vector can be decrypted using the private key of the Paillier encryption scheme. This is represented by the "Decryption using Private Key" block.

Finally, the decrypted SVM model can be used to predict the labels of a test set, represented by the block on the bottom left, without revealing the plaintext test data or the plaintext labels. The resulting encrypted labels can be decrypted using the same private key used for the training data.

Overall, the diagram shows how partial homomorphism in encryption can be used to perform privacy-preserving SVM, enabling the training of a model on encrypted data without revealing the plaintext data to any third party.

##### Theory:

**Approach 1:**

In this example Alice trains a spam classifier on some e-mails dataset she

owns. She wants to apply it to Bob's personal e-mails, without

1) asking Bob to send his e-mails anywhere

2) leaking information about the learned model or the dataset she has learned

from

3) letting Bob know which of his e-mails are spam or not.

Alice trains a spam classifier with logistic regression on some data she

possesses. After learning, she generates public/private key pair with a

Paillier schema. The model is encrypted with the public key. The public key and

the encrypted model are sent to Bob. Bob applies the encrypted model to his own

data, obtaining encrypted scores for each e-mail. Bob sends them to Alice.

Alice decrypts them with the private key to obtain the predictions spam vs. not

spam.

Dependencies: numpy, sklearn

**Text

Description automatically generated**

Fig. 6. Approach 1 output

**Approach 2:**

The first step is to generate a random dataset using the make\_classification function from the sklearn.datasets module, which creates a synthetic dataset with the specified number of samples, features, and classes.

The dataset is then split into training and testing sets using the train\_test\_split function from the sklearn.model\_selection module. The test\_size parameter specifies the proportion of the dataset to include in the testing set.

Next, a Paillier key pair is generated using the generate\_paillier\_keypair function from the phe module. The public\_key is used for encryption, while the private\_key is used for decryption.

The training data and labels are encrypted using the encrypt method of the public\_key object, which converts each value to an encrypted representation.

An SVM model is trained on the encrypted data using the SVC function from the sklearn.svm module with a linear kernel and a regularization parameter C of 1.

The testing data is also encrypted using the public\_key.

The SVM model is used to predict the labels for the encrypted testing data using the predict method.

The predicted labels are decrypted using the decrypt method of the private\_key object, which converts each encrypted value back to its original plaintext representation.

Finally, the accuracy of the model on the testing data is calculated by comparing the predicted labels with the actual labels and computing the proportion of correct predictions.

Overall, this code demonstrates how the Paillier cryptosystem can be used for privacy-preserving machine learning by encrypting the training data and labels, training a model on the encrypted data, and making predictions on encrypted testing data.

Code snippet:

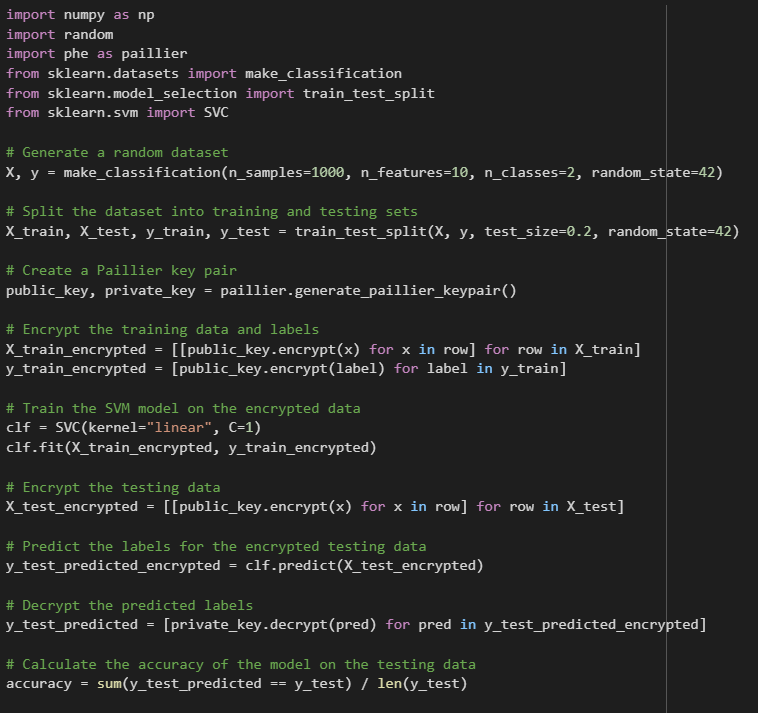


Fig. 7. Privacy-preserving svm