Secure Evaluation of Decision Tree using Homomorphic Encryption

Pranav Vardia, Karan Naidu

1 Problem Statement

In this report we discuss about how we can employ Homomorphic encryption techniques to ensure secure evaluation of the Decision Tree model. The aim of the project is to ensure 2 things:-

- Ensure the Model owner learns nothing about the client inputs.
- The client does not come to know anything about the model architecture and parameters.

2 Setup

The setup is as follows:

- The scenario is a two party model. The client and the model owner.
- We use Public-Key Homomorphic encryption.
- The client encrypts the data using Public Key **pk** and using the same key the model encrypts its threshold values and decision values (labels).
- The client sends encrypted data to the model for evaluation and after evaluation the model sends back the encrypted label value back to the client which can be decrypted by the client's secret key **sk**.
- The above points are depicted in Figure 1.

3 Algorithms

3.1 Secure Comparison

This algorithm is at the heart of the entire setup. Since decision trees work on the principle based on filtering data using some threshold values we need an algorithm which can perform secure comparison. Consider an l-bit encrypted input x and

Figure 1: Client-Server Model

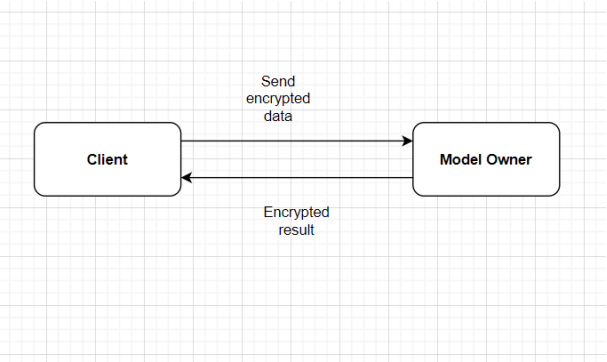
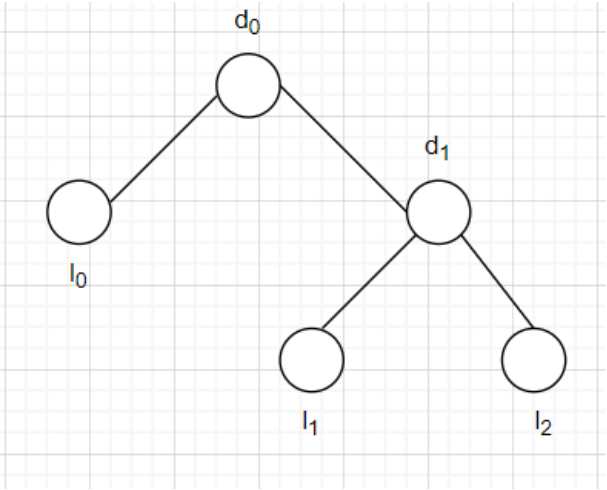


Figure 2: Decision Tree Evaluation



l-bit encrypted threshold value y.
The algorithm SecComp([[x]], [[y]]) returns [[1]] if $x < y$.

- $b = (x_{l-1} < y_{l-1}) \vee ((x_{l-1} = y_{l-1}) \wedge (x_{l-2} < y_{l-2})) ..... \vee ((x_{l-1} = y_{l-1})(x_{l-2} = y_{l-2}) .... (x_1 = y_1)(x_0 < y_0))$
- $(x_i < y_i) = (1 - x_i)(y_i)$
- $(x_i = y_i) = (1 \oplus x_i \oplus y_i)$
- $x_i \oplus y_i = x_i + y_i - 2x_i y_i$

- The value b is computed for each decision node.

3.2 Polynomial evaluation

- We evaluate each decision node d_0 and d_1 to get b_0 and b_1.
- Now, each path is evaluated in the decision tree and the final result calculated as sum of all paths.
- $[[l]] = b_0 l_0 + (1 - b_0)(b_1 l_1 + (1 - b_1) l_2)$
- Now this encrypted result is sent back to the client which will decrypt to either l_0, l_1 or l_2

4 Implementation

The algorithm stated above presents a lot of complexity while executing. The paper originally uses three kinds of encryption : AES , an SWHE, and MKHE (Multi-key HE) which is their own kind of key-setup.

The Machine-Learning division of the problem is executed in Python, so the Homomorphic Encryption Scheme has been implemented in python as well. There is only one library which has our requirements of SWHE in python, which is the PyFhel library, so the HE Schemes are implented in PyFhel. This library runs on Helib and SEAL backend.

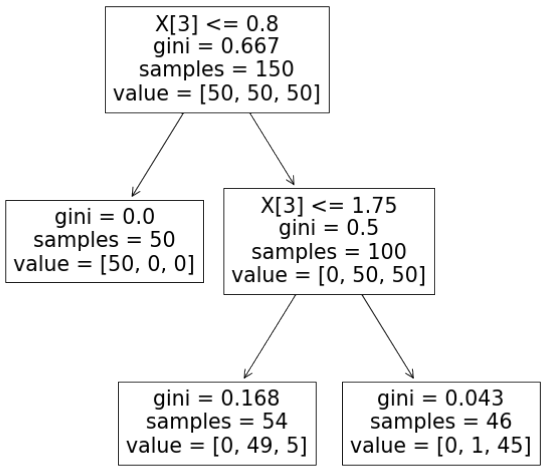
4.1 Setup

- Python Language is being used for the programming.
- We have used the PyFhel libraries for encryption and computation.
- Decision Trees are trained using the Pandas and Sklearn libraries
- Systems used for training consist of Intel Dual Core i5, 8 GB RAM, 2.2 GHz Proceesor

4.2 Decision Tree

We have created the decision tree using Iris dataset.In the iris dataset, using different lengths and widths of the flowers, each flower is classified into 3 different classes. Different modules are written for the Model Owners and the Clients to create their different functions. The tree generated is of depth 2 and for easier implementation the values have been rounded off.

Figure 3: Decision Tree Generation



Initially, the model owners train the decision tree models.The dataset is split into train and test sets and the test set is known to the client. This is done so as to obtain the thresholds and features for splitting that the tree uses. We will use these thresholds further when evaluating queries using SecComp. A decision tree we obtain by pretraining for evaluation is shown in Figure 3.

4.3 Client

Clients have the role of sending their queries so as to know the outputs. For the client, the role is it encrypts the feature vector input or the feature of the flower as the input that it has to send after encrypting the values by bit and decrypting back the output after receiving the output using the secret key. In the code, we can select a random row from all the rows and send that as a query.

4.4 PyFhel

We have used Python library PyFhel, running on HeLib backend as our implementation. PyFhel contains the necessary operations required for executing our protocol Secure Comparison such as equality, addition etc. Due to high computational complexity in implementation, we have used only PyFhel as our encryption for a single model owner and applied encryption for that key. We have tested the algorithm with single-model owner and have successfully generated the results.

4.5 Challenges

- Synchronizing the client-server model for different stages of the algorithm was difficult.
- The functionalities of their proposed MKHE is not clear on some points.
- Proposed MKHE functions better on C++, creating problems while implementing in Python.
- Joint-key generation became complex to execute

5 Conclusion and Future Work

We have implemented the functionalities present in this paper and have generated good results consistent with the paper itself.However, there is a scope of improvement present within this implementation

- Like the paper, due to unavailability of good libraries and resources, we have not implemented a multi-key scheme and this implementation is a good way
- Better Python libraries should be generated for BGV schemes and multi-key schemes
- This should be extended to Neural Networks and could be a future hotbed of research.