Rapport

Introduction

Objectif du projet

L'objectif de ce projet est de réussir à produire des statistiques basiques en utilisant le chiffrement homomorphe. Nous avons un dataset représentant des personnes ayant un risque de CHD (Coronary heart Disease) dans les 10 prochaines années. Chaque colonnes du tableau représente un attribut qui peut jouer sur le risque de CHD. L'idée est d'avoir un client voulant faire des calculs sur ces données sans avoir la puissance de calcul suffisante. Le client va donc envoyer ses données sur un serveur (le cloud par exemple) pour faire ses calculs. Les données du client sont personnelles, elles doivent donc rester confidentielles. Le client va alors devoir chiffrer ses données avant de les envoyer et le serveur va effectuer ses calculs sur ces données chiffrées.

Choix du langage et des bibliothèques

Nous avons choisi de réaliser le projet en Python pour plusieurs raisons :

- Tous les membres du projet sont déjà familiarisés avec le langage
- De nombreuses bibliothèques pour chiffrement homomorphe (tenSEAL, Pyfhel, HElib ...)
- Python permet de rapidement développer des applications

Nous avons choisi Tenseal pour la bibliothèque de chiffrement pour plusieurs raisons :

- Mieux documentée que celles précédemment essayées.
- La facilité d'utilisation, la syntaxe est simple.
- Supporte le schéma CKKS (Cheon-Kim-Kim-Song), qui permet de faire des opérations sur des réels
- OpenMined a développé cette bibliothèque. Openmined est une communauté open source travaillant sur des technos de confidentialité et de sécurité pour l'IA

Chiffrement Homomorphe

Aperçu

Le Chiffrement Homomorphe permet de chiffrer des données et ensuite d'effectuer des opérations sur ces dits chiffrés. Ce chiffrement permet donc de garantir la confidentialité sur les données manipulées. Le chiffrement homomorphe peut être de différentes natures :

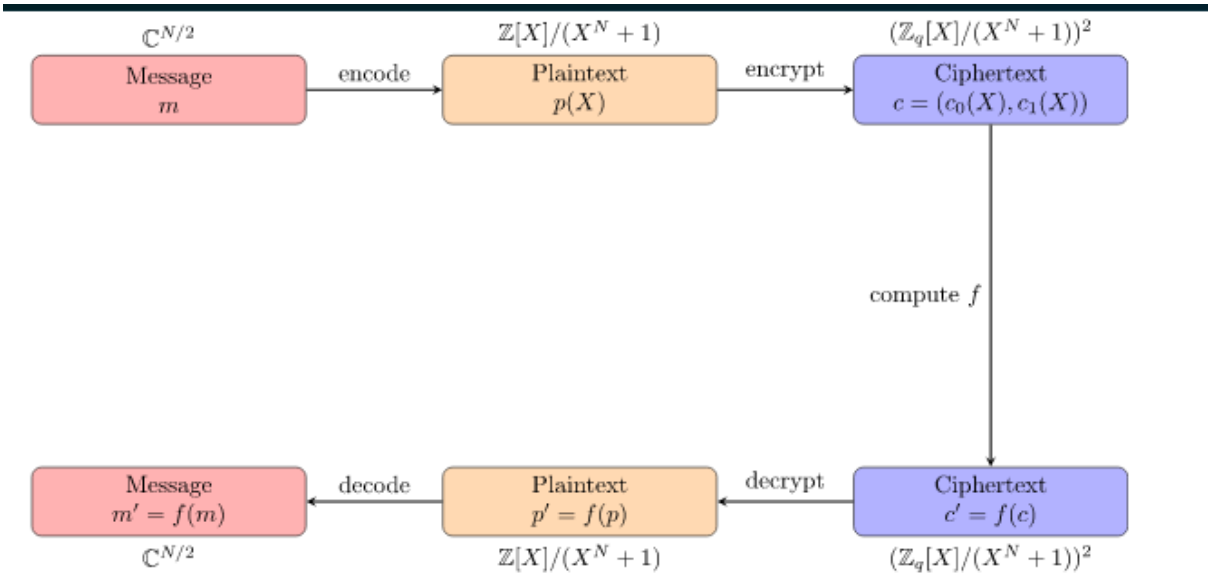
- Un système peut être partiellement homomorphe si les opérations sur les chiffrés sont restreintes et ne permet qu'un seul type d'opération (ex : seulement l'addition ou la multiplication)
- Un système peut être totalement homomorphe (FHE) si l'on peut l'addition et la multiplication
- Il y a aussi les systèmes presque homomorphes qui limitent le nombre d'operations

Lorsque l'on effectue des opérations sur des chiffrés, les résultats se dégradent. Les FHE permettent un nombre illimités d'opérations en se reposant sur le bootstraping qui permet de nettoyer le chiffré.

Il y a différents schémas de chiffrement homomorphes: les plus connus sont BFV et CKKS. BFV permet des opérations sur des entiers avec une précision de 100% alors que CKKS permet des opérations sur des flottants mais ses calculs ne sont pas complètement précis. Nous avons choisi pour notre projet CKKS car nous allons effectuer des opérations sur des réels.

CKKS

Je vais ici survoler le schéma CKKS qui permet d'effectuer des opérations sur réels. CKKS peut être décrit en plusieurs étapes (cf schéma)



La première étape est une étape d'encodage. Le but est de transformer un vecteur de valeur sur lequel on veut effectuer des opérations en un polynôme. Cette étape est nécessaire pour faire fonctionner le chiffrement et déchiffrement de ces valeurs. Le polynôme aura donc des coefficients entiers. L'intérêt d'utiliser des polynômes est de réduire le temps de calcul pour une sécurité équivalente. Le vecteur encodé sera appelé le PlainText.

Après avoir encodé un vecteur en polynôme on passe à la phase de chiffrement et de déchiffrement. Utiliser des polynômes est bien plus efficace qu'utiliser des vecteurs pour un schéma de chiffrement homomorphe.

Pour expliquer CKKS, il faut voir LWE (Learning with error). Learning with error est l'algorithme utilisé si les vecteurs n'étaient pas encodés. On part donc du principe que nous avons ici toujours des vecteurs.

On considère:

- u un vecteur de taille n, dont chaque valeur est modulo q. C'est le vecteur que l'on veut chiffrer
- A une matrice de dimension n x n, dont chaque valeur est modulo q, les valeurs sont aléatoires
- e un vecteur qui représente l'erreur qui est modulo q, les valeurs suivent une gaussiens centrées en 0. Le but est que l'erreur soit négligeable
- p = (-A*s + e, A). p représente notre clé publique.

- s est un vecteur de taille n qui est modulo q, il représente notre secret (clé privée)

Pour chiffrer u on fait l'opération suivante :

c = (u,0) + p = (u - A*s + e, A) = (c0, c1) , où c représente le chiffré

Pour déchiffrer on fait l'opération suivante :

û = c0 + c1*s = u -A*s + e + A*s = u + e =, on considère que e est négligeable et donc que û = u

RWLE repose sur le même principe que LWE sauf que nous remplaçons les vecteurs par des polynômes ce qui réduit les temps de calcul.

Le problème LWE/RWLE permet de facilement additionner 2 chiffrés entre eux (l'addition de deux erreurs reste négligeable), et aussi d'additionner/multiplier un chiffré avec un PlainText. Les polynômes sont modulo (X^(N) + 1), où N est une puissance de 2 et représente le degrés des polynômes.

Pour RWLE on a les valeurs suivantes :

- p = (b,a) = (-a*s + e,a), avec a des polynômes aléatoires , s la clé secrète, et e l'erreur.

Le problème est que l'on veut aussi pouvoir multiplier deux chiffrés pour avoir un schéma de chiffrement complet. Si nous exécutons une simple multiplication de chiffrés, la taille du résultat augmente de manière exponentielle et une multiplication de ciphertext devient un polynôme de degrés 3.

On prend 2 chiffrés c et c' que l'on veut multiplier:

- DecryptMult(CMult(c,c′),s) = Decrypt(c,s).Decrypt(c′,s) = c0$c'0$ + $(c0$*c′1+c′0$c1).s$ + $c1$*c′1s^2 = $d0$+$d1$*s+d2*s^2
- On peut donc considérer le déchiffrement de la multiplication comme un polynôme de degrés 2. Donc on a CMult = (d0,d1,d2)

Nous allons donc utiliser la relinéarisation, l'objectif est de rendre la taille des chiffrés constante, en échangeant le polynôme de degrés 2 par un polynôme de degré 1.

La relinéarisation permet d'avoir un couple de polynômes, et non un triplet, de telle sorte qu'une fois déchiffré à l'aide du circuit de déchiffrement régulier qui ne nécessite que la clé secrète et non sa puissance 2, nous obtenions la multiplication des deux textes clairs sous-jacents. On va utiliser une clé d'évaluation qui est une clé utilisée pour permettre la relinéarisation des résultats d’un multiplication entre

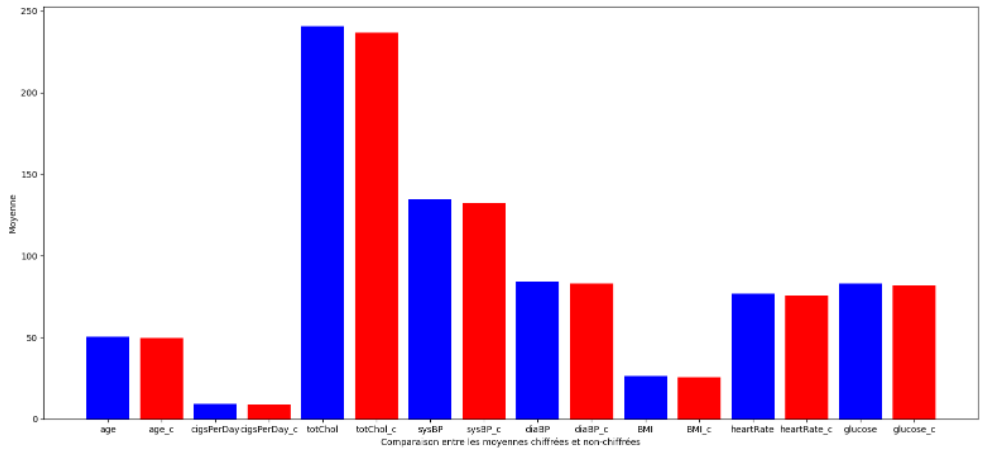
des chiffrés (ciphertexts). Donc lorsque nous effectuons une multiplication nous avons les opérations suivantes :

- Une multiplication: cmult=CMult(c,c')=(d0,d1,d2)
- La relinéarisation: crelin=Relin((d0,d1,d2),evk) (evk la clé d'évaluation)
- Decrypt(crelin,s)≈μ*μ' (Donc resultat de la multiplication en clair)

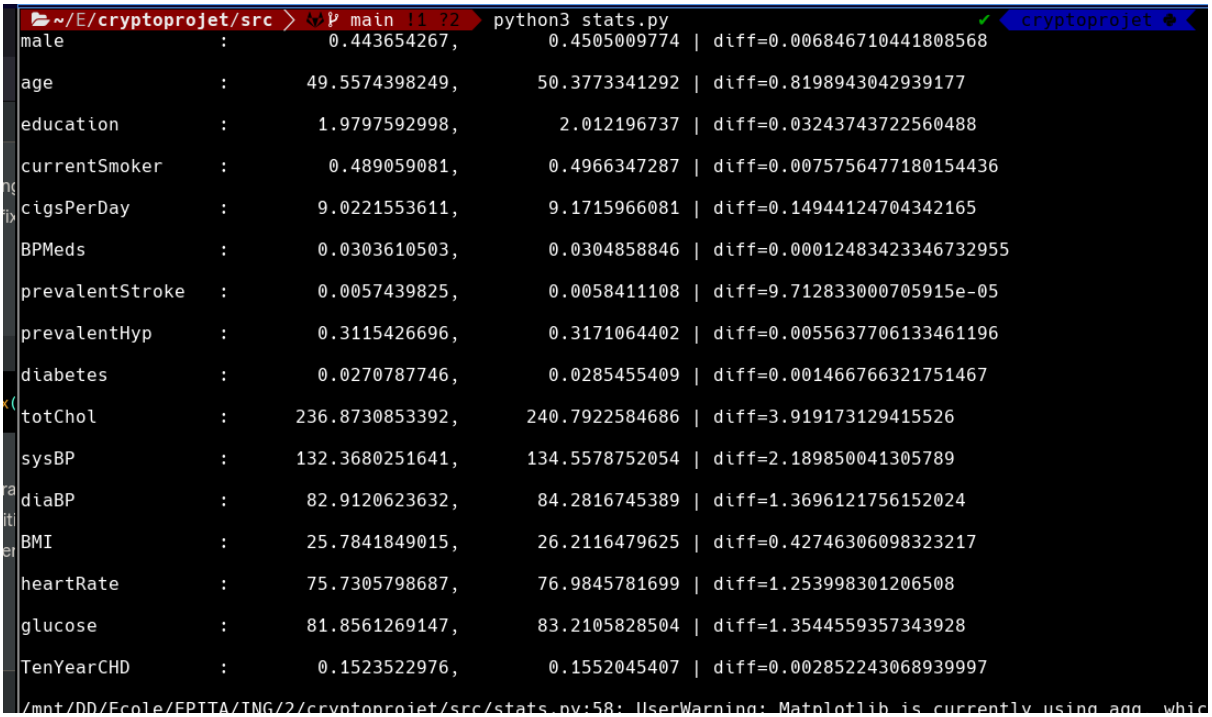
Le dernier point à aborder pour CKKS est le "rescaling". En multipliant les opérations sur les chiffrés le bruit augmente et si le bruit est trop grand il sera alors impossible de correctement déchiffrer notre valeur. Il faut donc choisir un global scale (une puissance de 2) qui permettra de faire plus d'opérations et il faudra aussi ajuster le modulo des polynômes en adéquation.

Les statistiques

Nous avons simplement fait une comparaison des moyennes des différentes colonnes. On voit une différence négligeable entre les calculs faits avec les données chiffrées et les données en claire



On voit ici les valeurs chiffrées à droite et non chiffrées à gauche pour chaque colonne. On voit que les différences entre chaque moyenne sont négligeables.



Futures améliorations et difficultés rencontrées

Difficultés

Nous avons rencontré de multiples difficultés durant ce projet.

La manière d'agencer nos données n'était pas optimale. On a choisi de chiffrer le csv ligne par ligne et de ranger chaque ligne dans une liste. Cette manière de faire nous a causé les problèmes suivants :

- Problème pour envoyer les données au serveur, car il a fallu choisir un séparateur pour chaque ligne chiffrée. Pour que le séparateur (retour à la ligne) fonctionne nous avons dû encoder en base64 chaque chiffré.
- Côté serveur, cela nous a empêché de réaliser plus simplement des corrélations, car il était impossible d'indexer une ligne en particulier.

L'avantage de cette méthode est la réduction du temps de chiffrement car on ne chiffre pas une matrice dans son entiereté.

Une autre difficulté a été l'impossibilité de réaliser des comparaisons entre chiffrés avec Tenseal. On s'est donc contenté de moyennes pour cet exercice.

Tenseal ne proposait pas de moyen d’envoyer des données entre un client et un serveur intégré à la bibliothèque. On a donc dû utiliser un serveur de fichier pour s’échanger les chiffrés.

Futures améliorations

Si le projet était à continuer, il y aurait plusieurs améliorations possibles :

- Si le projet était à refaire, nous passerions plus de temps à expérimenter plusieurs bibliothèques (ex : possibilité de faire des comparaisons qui a manqué).
- Nous aurions essayé de produire plus de graphiques et plus de statistiques dont une matrice de corrélation.
- Amélioration de l’interaction client-serveur pour plus facilement ajouter des calculs