Projet SBD 2023

Projet n°6 : PinedRQ

M1 MIAGE - ISTIC - 2022/2023

Groupe D :
Roberto Chacon Suarez
Manh-Huan Nguyen
Hugo Thomas

Dans ce rapport, nous répondons aux questions du sujet 6 de SDB.
Le code que nous avons modifié tout au long du projet se trouve dans le dossier **pinedRQ**, dans le fichier **pinedrq.py**.
Nous testons le code modifié dans le jupyter-notebook du même dossier, nommé **RunMe.ipynb**.

Warm-up

Q09 : Inspect the code and identify the parameters values and the goal of the example experiment that is per-formed.

- Bins -> Nombre de nœuds qu'il y a dans l'histogramme. Plus il y a de nœuds, plus l'histogramme du nœud sera perturbé puisqu'il y aura plus de bruit.

- Degree -> Définie la quantité de bruit qui est ajoutée. Plus il est élevé, plus il y aura de bruit ajouté, mais la précision des résultats sera réduite.

- Epsilon -> Paramètre de confidentialité qui contrôle le bruit de LaPlace ajouté aux histogrammes. Plus il est grand, plus il y a de bruit et plus il est confidentiel.

- Proba dp -> Probabilité de répondre au critère de la differential privacy. Plus elle est petite, moins on a de chance d'y répondre. Par contre une probabilité élevée veut dire une complexité plus haute donc il faut trouver un équilibre.

- Frac -> Fraction de domaine de définition que la requête interroge.

- Recall -> Le nombre de vrais positifs divisé par le nombre de vrais positifs existant dans la "ground truth". Si égal à 1, tous les éléments pertinents ont été retournés.

- Précision -> nombre de vrais positifs divisé par le nombre total des données reçues.

Q10 : Why the recall is perfect and the precision extremely low.

Le recall est parfait parce qu'il n'y a pas encore de bruit, donc le nombre de vrais positifs est le nombre de vrais positif trouvés dans le ground truth.
La précision est basse au début puisque la fraction de domaine interrogé n'est qu'une partie de la totalité. C'est pour cela que, du fait qu'il n'y ait aucun bruit, la précision est à 1 quand toutes les données sont interrogées et augmente plus le `frac` (domaine interrogé) augmente.

Encryption

Q11-12 : Inspect pinedrq.py and spot the function in which the secret key and initialization vector must be generated, and the function in which the records must be encrypted. Then generate them.

La fonction dans laquelle il faut générer le vecteur d'initialisation et la clé doit être implémentée dans la classe **Keychain**. Ils sont générés dans la méthode **__init__**.

Q13 : Encrypt the records based on the AES cipher in AES.MODE_CBC mode (use the API provided by the pycryptodome module). Do not forget to pad(\cdot) the sequence so that its size is a multiple of AES.block_size .

Nous savons que *AES.MODE_CBC* fait référence au chiffrement de blocs *Advanced Encryption Standard (AES)* et *CBC (Cipher Block Chaining).*

- Le chiffrement *AES* utilise une clé de longueur fixe pour chiffrer et déchiffrer les données en blocs de 16 bytes.
- Le mode d'opération *CBC* ajoute un niveau de sécurité au chiffrement AES en enchaînant les blocs de texte en clair avant le chiffrement. Il nécessite un vecteur d'initialisation (aléatoire et unique pour chaque séquence chiffrée) qui est XORé avec le premier bloc de texte en clair avant d'être chiffré (chaque octet du IV est combiné avec l'octet correspondant du premier bloc de texte en clair en utilisant l'opération XOR).

De même, en remplissant les derniers blocs avec des données supplémentaires, le padding garantit que la taille de la séquence à chiffrer est un multiple entier de la taille de bloc d'AES, ce qui est nécessaire pour l'algorithme de chiffrement.
Tout cela est implémenté dans la fonction **encrypt** de la classe **Keychain**.

Q14 : Spot the function in which the encrypted records must be decrypted. Be careful with following the exact reverse path followed by the encryption function.

La fonction **decrypt**, tout comme la fonction **encrypt** se trouve dans la classe **Keychain,** et elle suit la même logique, mais en sens inverse. Ces fonctions peuvent être utilisées de la manière suivante :

```
#Initialisation
message = "Message secret"
keychain = Keychain()
#Encodage du message
encrypted_message = keychain.encrypt(message.encode())
#Décodage du message
decrypted_message = keychain.decrypt(encrypted_message)
```

Perturbation

Q15 - 17 : Now spot the function in which the counts of the nodes must be perturbed and compute the perturbation.

La fonction qui doit perturber le count retourné par un nœud doit être **_perturb_histogram** de la classe **Index**, permettant ainsi de répondre aux critères de la differential privacy.

Pour cette implémentation, nous avons calculé le scale factor de la distribution de LaPlace à partir de la formule du cours :

$$\frac{1}{(\frac{1}{2^i} * \epsilon_{total})}$$

Nous avons décidé d'utiliser cette formule pour mieux répartir le privacy budget entre les différents niveaux de l'arbre plutôt qu'un unique scale factor uniforme.

Donc les étapes que nous avons suivies pour implémenter les perturbations sont les suivantes :

- Nous avons calculé le scale factor de chaque niveau de l'arbre.
- Pour chaque niveau :
    - On récupère les nœuds de ce niveau.
    - Pour chaque noeud :
        - On génère la perturbation avec le scale factor de ce niveau.
        - On applique le bruit au count du nœud.

Q18 Perturb the count of each node by adding to it a random variable sampled from the Laplace distribution well parameterized (you can use the API provided by the scipy module).

Pour cela, nous avons changé la génération du bruit qui utilisait le module *numpy* :

```
noise = np.random.laplace(0,scale)
```

pour la fonction **laplace.rvs** du module *SciPy* :

```
noise = laplace.rvs(scale=scale)
```

En utilisant donc cette nouvelle fonction, nous avons accès à plus de paramètres et de méthodes statistiques pour mieux exploiter cette distribution.

Q19 : You can truncate negative values to 0. Which differential privacy property guarantees the security of this operation ?

Pour cela, il fallait vérifier si la valeur perturbée était négative, et dans le cas où elle l'était, il fallait mettre le count du noeud à 0.

Afin de répondre aux critères de la propriété de la perturbation aléatoire, il est important de mettre les valeurs perturbées négatives à 0. En effet, cela permet dans un premier temps d'avoir un histogramme valide car les enfants ne peuvent avoir un histogramme négatif. Cela permet aussi d'avoir une somme de counts des enfants en cohérence avec le parent.

Cela vérifie le critère de confidentialité *epsilon privacy*, assurant ainsi que l'ajout de bruit ne permet pas à un attaquant de distinguer la réponse obtenue à la réponse qu'il aurait obtenue si les données d'un utilisateur étaient retirées de la base de données.

Autrement dit, le bruit aléatoire ne modifie pas de manière significative la sortie de l'histogramme lorsqu'il est suffisamment grand.

Q20 : Spot the function in which the cloud processes a query.

La méthode dans laquelle le cloud traite une requête est la méthode **query** définie dans la classe **Cloud**.
La méthode prend en entrée une plage de valeurs à rechercher, et appelle la fonction du même nom de la classe **Index**, qui est responsable de trouver les feuilles de l'index qui contiennent des enregistrements se trouvant dans la plage spécifiée.
Ensuite elle retourne simplement les feuilles trouvées, qui contiennent des enregistrements chiffrés, tout cela dans une séquence des buckets.

Q21 : Design and implement a query processing strategy that improves precision (at the cost of the recall). Warning: keep the naive query processing strategy somewhere. It will be useful as a baseline.

Nous avons rencontré des difficultés dans cette question en raison d'une mauvaise analyse de notre part.

En premier temps, nous voulions comparer chaque donnée des buckets afin de s’assurer que les âges se trouvaient dans les plages données. Pour cela, nous avons déchiffré les buckets et parcouru toutes les données en comparant l'âge avec les plages.

Cependant, nous nous sommes rendus compte que cette approche n’était pas idéale car, dans le traitement de la requête, les données seront déchiffrées plus tard, spécifiquement dans la méthode **query** de la classe **Client**, et cette étape ne devait que trier les buckets.

Afin d'améliorer la précision, nous avons créé une requête qui ne renvoie que les buckets dont les nœuds sont contenus dans l'intervalle spécifié :

- Tout d'abord nous obtenons toutes les feuilles depuis le nœud root.
- Ensuite, pour chaque feuille, nous vérifions si ses limites gauches et droites se trouvent dans l'intervalle de la requête.
- Si c'est le cas, nous stockons le bucket lié à cette feuille.
- Finalement nous envoyons la liste des buckets pour qu'elle soit processée plus tard.

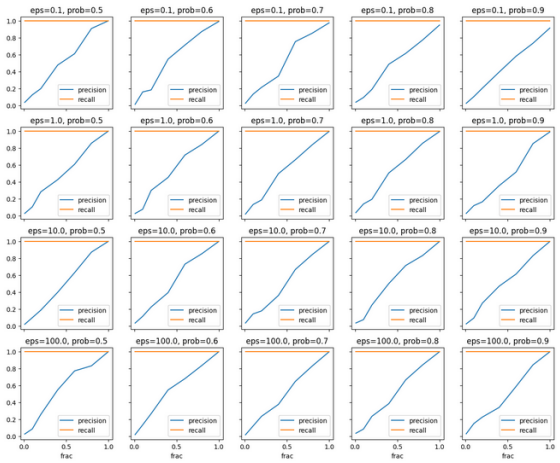
Stepping Back

Q22 à 24 : Comparer les résultats entre la requête naïve et la requête améliorée.

Méthode naïve :

```
CPU times: user 1min 30s, sys: 377 ms, total: 1min 31s
Wall time: 1min 31s
```

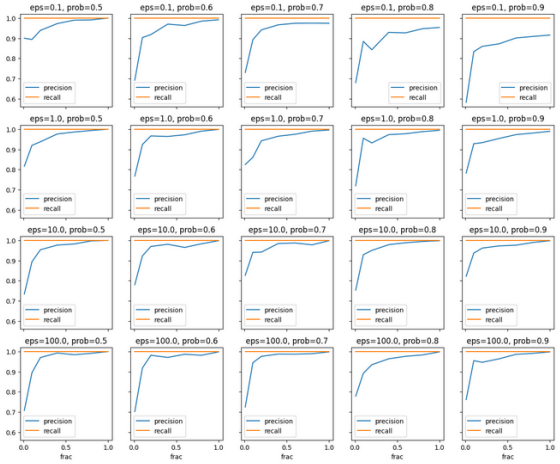
	bins	degree	epsilon	proba_dp	frac	precision	recall
count	1400.0	1400.0	1400.000000	1400.000000	1400.000000	1400.000000	1400.0
mean	50.0	5.0	27.775000	0.700000	0.444286	0.471437	1.0
std	0.0	0.0	41.893382	0.141472	0.344127	0.367332	0.0
min	50.0	5.0	0.100000	0.500000	0.010000	0.000058	1.0
25%	50.0	5.0	0.775000	0.600000	0.100000	0.090903	1.0
50%	50.0	5.0	5.500000	0.700000	0.400000	0.391634	1.0
75%	50.0	5.0	32.500000	0.800000	0.800000	0.829124	1.0
max	50.0	5.0	100.000000	0.900000	1.000000	1.000000	1.0



Méthode améliorée :

```
CPU times: user 53.8 s, sys: 520 ms, total: 54.3 s
Wall time: 54.3 s
```

	bins	degree	epsilon	proba_dp	frac	precision	recall
count	1400.0	1400.0	1400.000000	1400.000000	1400.000000	1400.000000	1400.0
mean	50.0	5.0	27.775000	0.700000	0.444286	0.930875	1.0
std	0.0	0.0	41.893382	0.141472	0.344127	0.113037	0.0
min	50.0	5.0	0.100000	0.500000	0.010000	0.142857	1.0
25%	50.0	5.0	0.775000	0.600000	0.100000	0.919565	1.0
50%	50.0	5.0	5.500000	0.700000	0.400000	0.977233	1.0
75%	50.0	5.0	32.500000	0.800000	0.800000	0.998465	1.0
max	50.0	5.0	100.000000	0.900000	1.000000	1.000000	1.0



Tout d'abord nous remarquons une amélioration significative dans la performance. Le temps d'exécution de la première approche est d'environ 1m30s tandis que le temps d'exécution de l'approche améliorée est d'environ 50s.

De plus, nous pouvons remarquer une différence significative dans la précision entre l'approche naïve et l'approche améliorée. La précision moyenne est passée de 0,46 dans l'approche naïve à 0,93 dans l'approche améliorée. La valeur minimale de la précision est également beaucoup plus élevée avec l'approche améliorée, ce qui montre que les résultats sont plus stables.

En outre, le "recall" reste à 1 pour les deux approches, ce qui indique que les deux requêtes sont capables d'identifier correctement toutes les données se situant dans les intervalles donnés. Cela est représenté dans les courbes.

La méthode query développée pour améliorer les résultats de la requête naïve permet donc une meilleure précision dans le retour des requêtes en retournant des résultats pertinents à plus de 90% et avec des performances à l'exécution bien supérieures.

Conclusion

Dans ce TP, nous avons vu comment l'encryption côté client est implémentée avec l'approche basée sur les perturbations, notamment sur l'index à l'aide de la Differential Privacy, tout en utilisant l'algorithme PINED-RQ.

Nous avons également examiné les mesures de performance couramment utilisées pour évaluer la qualité des modèles entraînés avec la differential privacy, notamment la précision et le recall.

L'objectif de PINED-RQ est de préserver la confidentialité tout en obtenant un "recall" élevé et en maintenant la précision à un niveau aussi élevé que possible, et nous avons réussi à démontrer, avec un traitement des requêtes plus précis, que cet objectif a été atteint.