Rapport du Projet - Confidentialité Différentielle avec Privelet

Auteurs: Aline N'DEKO & Franck Lagou
Lien github: https://github.com/Musk-max/projet_sbd_Aline_Franck/tree/module2

I. Explication du projet

La confidentialité différentielle est aujourd'hui un standard incontournable pour publier des statistiques sur des données sensibles tout en préservant la vie privée des individus.

Dans ce projet, nous avons étudié et implémenté l'algorithme **Privelet**, une méthode originale basée sur la **transformée de Haar** pour garantir la confidentialité différentielle sur des données ordinales.

Nous avons appliqué cette méthode sur l'attribut **"niveau d'éducation"** du jeu de données **Adult** de l'UCI Machine Learning Repository.

L'objectif final est de répondre à une requête statistique tout en respectant un fort niveau de confidentialité.

1. Chargement et nettoyage

La première étape a consisté à **charger** les données depuis un fichier CSV public.
Après l'inspection des données, nous avons constaté la présence de valeurs manquantes, identifiées par le symbole..

1. Requête statistique initiale

Nous avons formulé la **requête H1** suivante :*"Compter le nombre d'individus pour chaque niveau d'éducation."* La réponse brute à cette requête est un **histogramme** représentant la distribution de l'attribut `education`.

Cette distribution constitue notre **donnée sensible** sur laquelle nous appliquerons l'algorithme Privelet.

II. Privelet Algorithm

A. Description de l'algorithme Privelet

Privelet est un algorithme permettant de garantir la confidentialité différentielle par l'intermédiaire de plusieurs étapes :

1. Application de la **transformée de Haar** sur les données,
2. **Ajout de bruit Laplacien** calibré sur les coefficients obtenus,
3. **Reconstruction** des données bruitées,

4. **Analyse** de la qualité du résultat.

Nous allons détailler ici chacune des étapes, en précisant les fonctions utilisées.

B. Décomposition en étapes et rôles des fonctions

1. Préparation des données

Fonction : `pad_to_power_of_two(arr)`

Rôle :
La transformée de Haar classique nécessite que le nombre de points soit une **puissance de deux**. Si ce n'est pas le cas, cette fonction **ajoute** des zéros à la fin de l'histogramme jusqu'à atteindre la taille requise. Garantit la validité de la transformation Haar.

2. Application de la transformée Haar

Fonction : `haar_wavelet_transform(data)`

Rôle :
La fonction transforme les données en une série de **coefficients de Haar**, représentant :

- **Les tendances globales** (moyennes),
- **Les détails locaux** (différences entre groupes de données).

Chaque étape de la transformation réduit la résolution des données, capturant ainsi à différentes échelles l'information. Elle permet de structurer l'information de manière hiérarchique, ce qui est crucial pour l'ajout efficace de bruit.

3. Calcul des poids de Haar

Fonction : `compute_whaar_weights(num_coeffs)`

Rôle :
Attribuer un **poids spécifique** à chaque coefficient en fonction de son niveau de granularité.

- Les coefficients représentant **des détails fins** ont généralement **moins d'importance**.
- Ceux représentant **des moyennes globales** sont **plus sensibles**.

Ces poids servent à **calibrer** la quantité de bruit ajouté à chaque coefficient pour équilibrer confidentialité et précision.

4. Ajout de bruit Laplace

Fonction : `add_laplace_noise(coeffs, epsilon)`

Rôle : Ajouter un **bruit aléatoire** de type **Laplacien** à chaque coefficient transformé. Le bruit est **proportionnel au poids** du coefficient et **inversement proportionnel** au paramètre ε (epsilon), garantissant :

- Plus ε est petit → plus de bruit → plus de confidentialité,
- Plus ε est grand → moins de bruit → plus de précision.

Cette étape est celle qui **assure la confidentialité différentielle**.

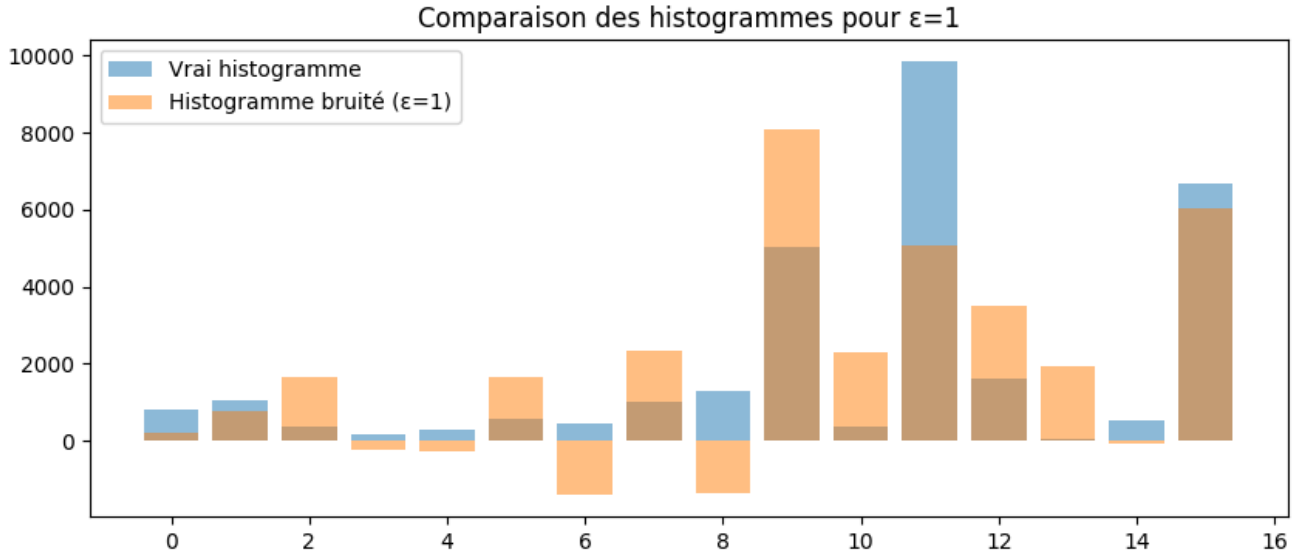
5. Reconstruction inverse

Fonction : `inverse_haar_wavelet_transform(coeffs)`

Rôle : Revenir du domaine des coefficients bruités vers un **histogramme bruité**.

- La reconstruction utilise l'inverse de la transformation Haar,
- Les données sont ramenées à leur échelle originale.

C'est ce résultat final qui sera publié comme réponse à la requête, garantissant la confidentialité.



II. Experimental Evaluation

Évaluation de la qualité des résultats

Fonction : `wasserstein_distance(u, v)` **(de** `scipy.stats`**)**

Rôle: Mesurer la **distance de Wasserstein** (aussi appelée Earth Mover's Distance) entre l'histogramme original et celui bruité.

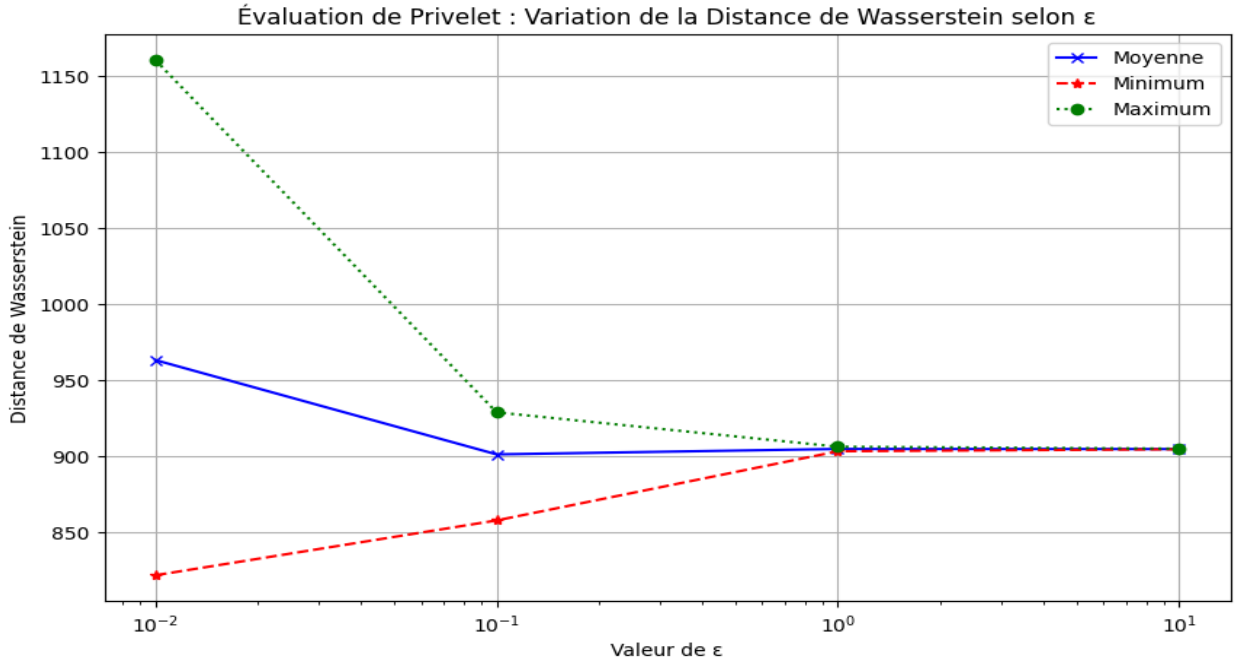
- Plus la distance est faible, plus l'histogramme bruité est proche du vrai histogramme.
- Cela nous donne une **quantification objective** de la perte d'information.

1. Protocole expérimental

Nous avons évalué la qualité en suivant cette méthode :

- Pour chaque valeur de ε parmi {0.01, 0.1, 1, 10},
- Répéter l'expérience **20 fois** pour compenser l'aléa du bruit,
- Calculer les **moyennes**, **minimum** et **maximum** des distances.

2. Analyse des résultats



Les résultats confirment que :

- **Avec ε = 0.01**, la distance est grande → forte perte d'information mais confidentialité maximale,
- **Avec ε = 10**, la distance est faible → précision quasi intacte mais confidentialité moindre.

On retrouve donc le **compromis classique** entre **confidentialité et utilité**.

III. Conclusion

À travers ce projet, nous avons pu :

- Comprendre l'intérêt de **l'approche multi-échelles** de Privelet,
- Vérifier **empiriquement** l'impact du paramètre ε sur la précision des résultats,
- Confirmer que la **transformée Haar** permet un ajout plus intelligent de bruit que l'ajout naïf.

En résumé, Privelet s'avère être une méthode performante pour les données ordinales lorsque la préservation de la confidentialité est essentielle.

Annexe : Liste des fonctions et leur rôle

Fonctions	Rôles
`pad_to_power_of_two(arr)`	Adapter la taille des données
`haar_wavelet_transform(data)`	Transformée de Haar
`compute_whaar_weights(num_coeffs)`	Calcul des poids pour le bruit
`add_laplace_noise(coeffs, epsilon)`	Ajout du bruit Laplacien
`inverse_haar_wavelet_transform(coeffs)`	Reconstruction de l'histogramme bruité
`wasserstein_distance(u, v)`	Mesure de la perte d'information