

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université des Sciences et de la Technologie Houari Boumédiène

Faculté d'Informatique
Département d'Intelligence Artificielle et Sciences des Données

Master Systèmes Informatiques intelligents

Module : Data Mining

Rapport de la Partie 2 du projet

Réalisé par :
DIB Fella, 202031028311
IGHILAZA Lina, 202031063054

Année universitaire : 2024 / 2025

Table des matières

| | | |
|----------|--|-----------|
| 1 | Description des Algorithmes | 1 |
| 1.1 | Algorithmes de Régression | 1 |
| 1.1.1 | Arbres de Décision | 1 |
| 1.1.2 | Forêts Aléatoires | 2 |
| 1.2 | Algorithmes de Clustering | 3 |
| 1.2.1 | CLARANS (Clustering Large Applications based upon RANdomized Search) | 3 |
| 1.2.2 | DBSCAN (Density-Based Spatial Clustering of Applications with Noise) . . | 5 |
| 2 | Implémentation | 7 |
| 2.1 | Environnement de développement | 7 |
| 2.2 | Traitement des données | 7 |
| 2.3 | Métriques d'évaluation | 9 |
| 2.3.1 | Métriques pour les algorithmes de régression | 9 |
| 2.3.2 | Métriques pour les algorithmes de clustering | 10 |
| 3 | Expérimentation | 12 |
| 3.1 | Evaluation des modèles de régression | 12 |
| 3.1.1 | Modèle de Decision Tree | 12 |
| 3.1.2 | Modèle de Random Forest | 14 |
| 3.2 | Evaluation des modèles de clustering | 16 |
| 3.2.1 | Algorithmes DBSCAN | 16 |
| 3.2.2 | Algorithmes CLARANS | 17 |
| 3.2.3 | Comparaison entre l'algorithme DBSCAN et CLARANS | 18 |

Introduction

Dans un monde où les données jouent un rôle stratégique de plus en plus important, le data mining s'impose comme un outil essentiel pour extraire des connaissances pertinentes à partir de jeux de données volumineux et complexes. Ce projet s'inscrit dans cette dynamique, en explorant les différentes étapes du processus de data mining, depuis l'analyse initiale des données jusqu'à l'évaluation de modèles avancés.

La première phase du projet a été consacrée à l'analyse exploratoire et au prétraitement des données. Cette étape cruciale a permis de nettoyer et d'intégrer un jeu de données climatiques et pédologiques, tout en appliquant des techniques de réduction et de normalisation des données. Les choix relatifs aux étapes de prétraitement ont été guidés par une analyse approfondie et justifiée, garantissant ainsi un jeu de données prêt pour les modèles de data mining ultérieurs.

La seconde partie, qui constitue l'objet de ce rapport, se concentre sur l'application d'algorithmes de régression (arbres de décision et forêts aléatoires) et de clustering (DBSCAN et CLARANS). L'objectif est de comparer ces algorithmes en termes de performance, tout en explorant leurs avantages et limitations dans des contextes spécifiques.

Chapitre 1

Description des Algorithmes

Dans ce projet, nous avons implémenté et testé deux types d'algorithmes : des algorithmes de régression et des algorithmes de clustering.

1.1 Algorithmes de Régression

La régression est une technique de data mining utilisée pour prédire une variable continue à partir de variables indépendantes. Elle permet de construire des relations non linéaires et d'identifier des patterns complexes dans les données. Les deux algorithmes de régression utilisés dans cette partie sont les arbres de décision et les forêts aléatoires. [1]

1.1.1 Arbres de Décision

Un arbre de décision est un modèle d'apprentissage supervisé utilisé pour la classification et la régression. Il utilise une structure arborescente binaire où chaque nœud représente une question sur les caractéristiques des données, et chaque branche représente les réponses possibles, menant à des feuilles qui contiennent la prédiction finale. L'algorithme divise de manière récursive l'ensemble des données en sous-ensembles de plus en plus homogènes, en choisissant les caractéristiques qui minimisent l'erreur. Des critères comme l'entropie (pour la classification) ou la variance (pour la régression) sont utilisés pour déterminer les meilleurs points de division à chaque nœud.[2]

Avantages :

- Simple à comprendre et à visualiser.
- Capable de gérer des données qualitatives et quantitatives.

Inconvénients :

- Sujet à l'overfitting si l'arbre est trop profond.
- Sensible aux petites variations dans les données.

Algorithme 1 : Decision Tree (DT)

```
Entrées : dataset : ensemble des données, attributCible : attribut cible, attributs : liste des attributs.  
Output : Arbre de décision.  
si dataset est vide alors  
|   retourner Erreur // ou bien voir la classe dominante;  
fin si  
si attributs est vide alors  
|   retourner un nœud ayant la classe C la plus représentée pour attributCible;  
fin si  
si toutes les instances du dataset ont la même classe C pour attributCible alors  
|   retourner un nœud ayant la classe C;  
fin si  
sinon  
|   attributSlectionn  $\leftarrow$  attribut maximisant le gain d'information parmi attributs;  
|   attributsRestants  $\leftarrow$  suppressionListe(attributs, attributSlectionn);  
|   newNode  $\leftarrow$  nœud étiqueté avec attributSlectionn;  
|   pour chaque valeur V de attributSlectionn faire  
|   |   dataFiltrsV  $\leftarrow$  getAttributValeur(dataset, attributSlectionn, V);  
|   |   newNode.fils(V)  $\leftarrow$  DT(dataFiltrsV, attributCible, attributsRestants);  
|   fin pour chaque  
|   retourner newNode;  
fin si
```

1.1.2 Forêts Aléatoires

L'algorithme Random Forest est un modèle d'apprentissage supervisé basé sur un ensemble d'arbres de décision. Il utilise le concept de bagging (bootstrap aggregating) pour construire plusieurs arbres sur des sous-échantillons aléatoires des données et agrège leurs prédictions (moyenne pour la régression, vote majoritaire pour la classification). Chaque arbre est construit avec une sélection aléatoire de caractéristiques à chaque division, ce qui améliore la diversité entre les arbres et réduit le risque de surapprentissage (overfitting).[3]

Avantages :

- Réduit le risque d'overfitting.
- Performant sur des jeux de données avec des variables nombreuses ou bruitées.

Inconvénients :

- Plus complexe à implémenter et à interpréter.
- Peut nécessiter des ressources computationnelles importantes.

Algorithme 2 : Algorithme Random Forest

Entrées : D : ensemble de données d'entraînement,
 T : nombre d'arbres à construire,
 m : nombre d'attributs sélectionnés aléatoirement pour chaque division (où $m < |attributes|$).

Output : Un modèle Random Forest constitué de T arbres.

Début ;

pour $i \leftarrow 1$ **à** T **faire**

Générer un ensemble de données D_i en sélectionnant un échantillon bootstrap de D ;
 Construire un arbre de décision $Tree_i$ en utilisant D_i selon les étapes suivantes : ;

tant que *l'arbre n'est pas terminé* **faire**

À chaque nœud, sélectionner m attributs aléatoires parmi les attributs disponibles;
 Choisir le meilleur attribut parmi les m pour diviser le nœud en maximisant un critère ;

fin tq

Ajouter $Tree_i$ à la forêt;

fin pour

retourner la forêt contenant les T arbres;

Fin;

1.2 Algorithmes de Clustering

Le clustering est une technique d'apprentissage non supervisé utilisée pour regrouper des données similaires en clusters (groupes) en fonction de leurs caractéristiques. Contrairement aux modèles supervisés, il ne nécessite pas de variable cible. Les algorithmes de clustering cherchent à maximiser la similarité des données à l'intérieur d'un cluster tout en minimisant celle entre différents clusters.

1.2.1 CLARANS (Clustering Large Applications based upon RANdomized Search)

CLARANS est un algorithme de clustering partitionnel conçu pour travailler efficacement sur de grandes bases de données. Il est une extension de l'algorithme K-Medoids, mais utilise une approche aléatoire pour explorer les solutions possibles. CLARANS sélectionne des nœuds aléatoires dans l'espace des solutions et les évalue pour trouver la meilleure partition des données.[4]

Avantages :

- Efficace pour les grands ensembles de données.
- Flexible grâce à l'exploration aléatoire.

Inconvénients :

- Sensible aux paramètres initiaux.
- Peut être coûteux en termes de temps d'exécution.

Algorithme 3 : Algorithme CLARANS (Clustering Large Applications based on Randomized Search)

Entrées : D : ensemble de données avec n objets

k : nombre de clusters désirés

$maxNeighbor$: nombre maximal de voisins testés pour chaque nœud

$numLocal$: nombre d'itérations pour rechercher des minima locaux

Output : Un ensemble de k médoïdes représentant les clusters

Début ;

$bestClustering \leftarrow \emptyset$;

$minCost \leftarrow +\infty$;

pour $i \leftarrow 1$ **à** $numLocal$ **faire**

 Sélectionner aléatoirement un ensemble initial de k médoïdes;

Coût courant \leftarrow Calculer le coût du clustering;

pour $j \leftarrow 1$ **à** $maxNeighbor$ **faire**

 Choisir aléatoirement un médoïde et le remplacer par un non-médoïde;

Nouveau coût \leftarrow Recalculer le coût avec la nouvelle configuration;

si *Nouveau coût* $<$ *Coût courant* **alors**

 Mettre à jour les médoïdes et **Coût courant**;

 Redémarrer la boucle des voisins;

fin si

fin pour

si *Coût courant* $<$ *minCost* **alors**

$bestClustering \leftarrow$ Clustering courant;

$minCost \leftarrow$ **Coût courant**;

fin si

fin pour

Fin;

1.2.2 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN est un algorithme de clustering basé sur la densité, conçu pour identifier des groupes de points densément connectés dans l'espace. Contrairement aux méthodes partitionnelles comme K-Means, il ne nécessite pas de spécifier le nombre de clusters à l'avance. DBSCAN fonctionne en identifiant des points *cœurs* ayant au moins un nombre minimum de voisins dans un rayon donné (paramètres MinPts et ϵ). [5]

Avantages :

- Identifie les clusters de formes arbitraires.
- Gère efficacement les points aberrants.

Inconvénients :

- Sensible aux paramètres (ϵ et MinPts).
- Peut être inefficace pour des ensembles de données très volumineux.

Algorithme 4 : DBSCAN

Entrées : D : Dataset, ϵ : distance minimum de voisinage, $MinPts$: nombre minimum d'instances dans un cluster.

Output : D : Dataset étiqueté.

Début;

$C \leftarrow 0$;

pour chaque point P non visité du dataset D faire

$P.status \leftarrow \text{visité}$ // Marquer P comme déjà visité;

$PtsVoisins \leftarrow \text{epsilonVoisinage}(D, P, \epsilon)$;

si $\text{tailleDe}(PtsVoisins) < MinPts$ **alors**

$P.cluster \leftarrow \text{BRUIT}$ // Ajouter P à la liste des données bruitées;

sinon

$C \leftarrow C + 1$ // Créer un nouveau cluster;

étendreCluster($D, P, PtsVoisins, C, \epsilon, MinPts$);

fin si

fin si

fin pour chaque

Fin;

Algorithme 5 : étendreCluster

Entrées : D : Dataset, eps : distance minimum de voisinage, $MinPts$: nombre minimum d'instances dans un cluster, P : une instance de D , $PtsVoisins$: liste des voisins de P , C : numéro du cluster en cours de création.

Output : D : Dataset étiqueté.

$P.cluster \leftarrow C$ // Ajouter P au cluster C ;

pour chaque point P' de $PtsVoisins$ **faire**

si $P'.status \neq \text{visité}$ **alors**

$P'.status \leftarrow \text{visité}$ // Marquer P' comme visité;

$PtsVoisins' \leftarrow \text{epsilonVoisinage}(D, P', eps)$;

si $\text{tailleDe}(PtsVoisins') \geq MinPts$ **alors**

$PtsVoisins \leftarrow PtsVoisins \cup PtsVoisins'$ // Inclure les voisins des voisins à la liste;

fin si

fin si

si $P'.cluster = 0$ **alors**

$P'.cluster \leftarrow C$ // Ajouter P' au cluster C ;

fin si

fin pour chaque

Note : On appelle "epsilon Voisinage" d'une instance P toutes les instances de D qui sont à une distance inférieure ou égale à eps de P .

Chapitre 2

Implémentation

2.1 Environnement de développement

Le langage de programmation utilisé est Python sous la version 3.9.18.

Les bibliothèques Python suivantes ont été utilisées pour implémenter les différents modèles :

- **Pandas** [6] : utilisée pour la manipulation et l’analyse des données. Elle offre divers outils de nettoyage, filtrage, transformation et stockage d’ensembles de données.
- **Numpy** [7] : une bibliothèque pour le calcul numérique, fournissant des fonctions pour les opérations mathématiques sur différentes structures de données, notamment les tableaux et les matrices.
- **Scikit-learn** [8] : offre des outils pour la normalisation, la préparation des données et la construction de modèles d’apprentissage automatique.
- **Matplotlib** [9] : très utile pour la visualisation des données dans les projets d’apprentissage automatique, car elle permet de représenter graphiquement les résultats des modèles.

2.2 Traitement des données

- Le jeu de données obtenu à l’issue de la partie 1 est composé de 854 lignes et 48 colonnes (23 colonnes pour les données climatiques et 22 colonnes pour les données du sol).
- Les colonnes géospatiales (`lat`, `lon` et `geometry`) ont été supprimées.
- Les données ont été normalisées avec la méthode de normalisation min-max.
- Les valeurs aberrantes (“outliers”) n’ont pas été modifiées, car elles représentent des variations réelles et significatives.
- La corrélation entre attributs a ensuite été calculée. La figure suivante montre le résultat :

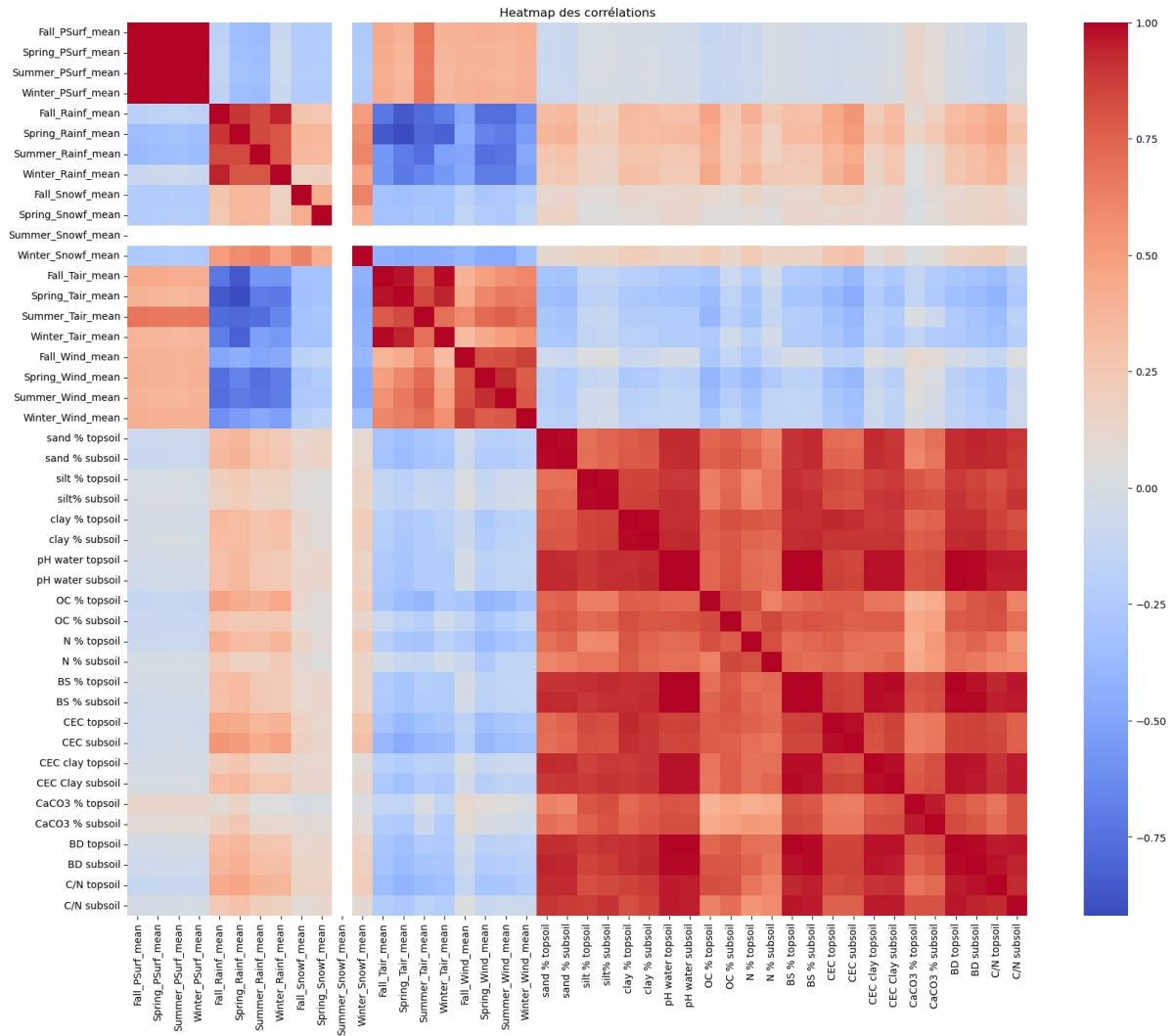


FIGURE 2.1 – Matrice de corrélation entre les attributs

Pour chaque paire ayant une corrélation de 95% ou plus, un des deux attributs a été supprimé. On obtient un jeu de données avec 25 colonnes.

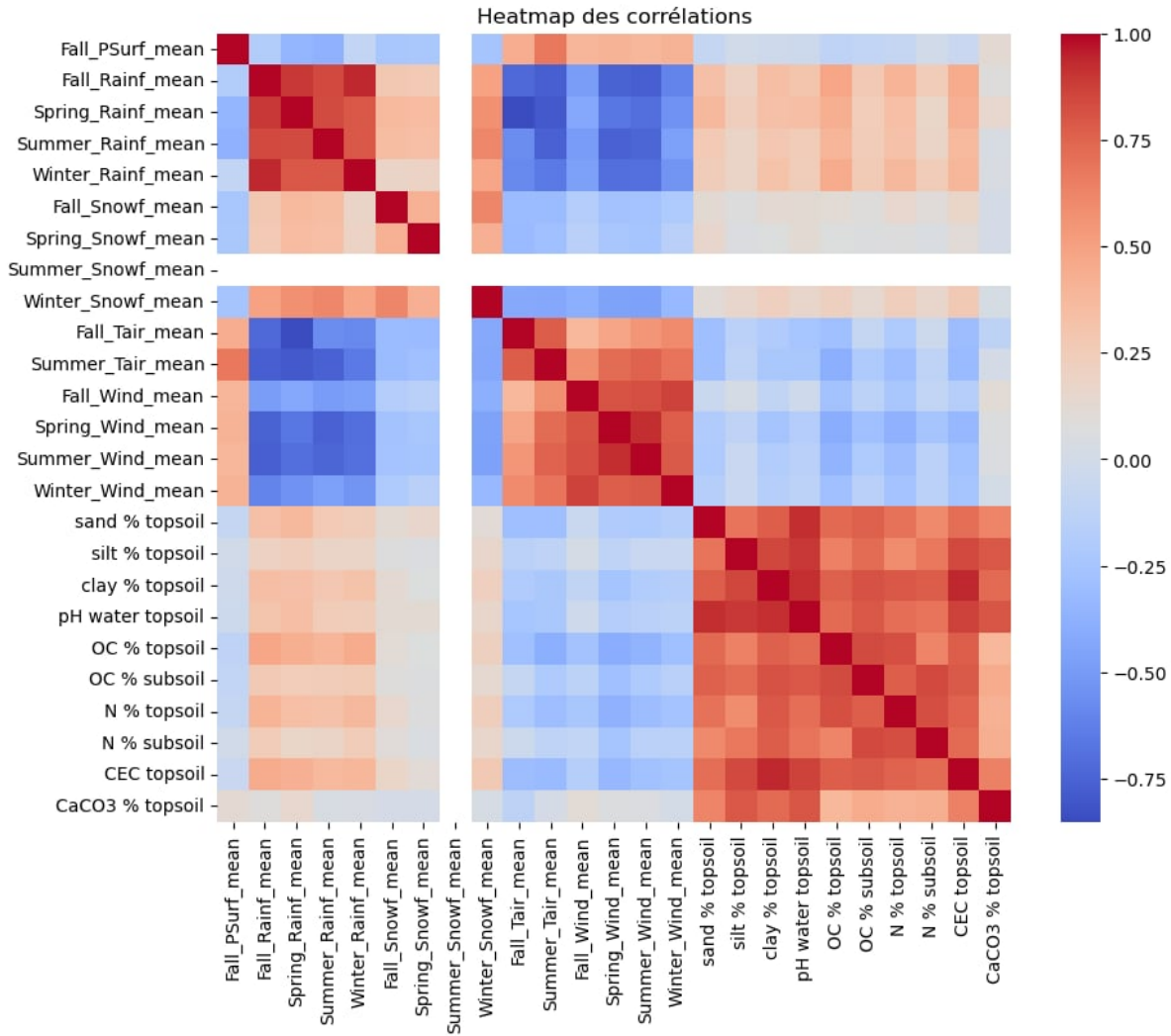


FIGURE 2.2 – Matrice de correlation entre les attributs

2.3 Métriques d'évaluation

L'évaluation des algorithmes de régression et de clustering repose sur des métriques spécifiques qui permettent de mesurer leur performance et leur efficacité. Ces métriques sont choisies en fonction des objectifs de chaque type d'algorithme.

2.3.1 Métriques pour les algorithmes de régression

Erreur absolue moyenne (Mean Absolute Error, MAE)

La MAE calcule l'écart moyen absolu entre les valeurs prédites et réelles :

Avantages : Plus robuste aux valeurs aberrantes que la MSE.

Inconvénients : Ne pénalise pas suffisamment les grandes erreurs.

Erreur quadratique moyenne (Root Mean Squared Error, RMSE)

La RMSE mesure la racine carrée de l'erreur quadratique moyenne :

Avantages : Pénalise davantage les grandes erreurs que la MAE.

Inconvénients : Sensible aux valeurs aberrantes.

Coefficient de détermination (R^2)

Le R^2 indique la proportion de la variance des données expliquée par le modèle :

Avantages : Facile à interpréter, avec des valeurs proches de 1 indiquant un bon ajustement.

Inconvénients : Peut être trompeur si utilisé seul, notamment pour des modèles non linéaires.

Temps d'exécution

Cette métrique évalue le temps nécessaire pour entraîner et tester les modèles. Elle est particulièrement importante pour des jeux de données volumineux ou des algorithmes complexes.

2.3.2 Métriques pour les algorithmes de clustering

Indice de silhouette

L'indice de silhouette mesure la qualité de la séparation des clusters. Pour chaque point i , il est défini comme suit :

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

où $a(i)$ est la distance moyenne entre i et les points de son propre cluster, et $b(i)$ est la distance moyenne entre i et les points du cluster le plus proche. La valeur moyenne de $s(i)$ pour tous les points donne l'indice global.

- Une valeur de l'indice de silhouette proche de 1 indique que les points sont bien séparés des autres clusters.
- Une valeur proche de 0 suggère que les points se trouvent sur la frontière entre deux clusters.
- Une valeur négative signifie que les points sont probablement mal assignés.

Avantages :

- Fournit une mesure intuitive de la compacité et de la séparation des clusters.

Inconvénients :

- Peut être coûteux à calculer pour de grands ensembles de données.

Indice de Davies-Bouldin

Cet indice évalue la compacité et la séparation des clusters. Il est défini comme la moyenne des ratios maximaux de la somme des dispersions intra-cluster à la distance entre les centres des clusters :

$$\text{DBI} = \frac{1}{k} \sum_{i=1}^k \max_{j \neq i} \frac{\sigma_i + \sigma_j}{d(c_i, c_j)}$$

où σ_i représente la dispersion intra-cluster pour le cluster i , et $d(c_i, c_j)$ est la distance entre les centres des clusters i et j .

- La dispersion intra-cluster (σ_i) mesure à quel point les points d'un cluster sont proches de son centre.
- La distance inter-cluster ($d(c_i, c_j)$) mesure la séparation entre deux clusters.

Un indice de Davies-Bouldin faible indique une bonne séparation des clusters et une faible dispersion intra-cluster.

Avantages :

- Facile à calculer.
- Permet une comparaison directe entre différents algorithmes de clustering.

Inconvénients :

- Sensible aux formes non sphériques des clusters.

Indice de Calinski-Harabasz

Cet indice mesure le rapport entre la variance inter-cluster et la variance intra-cluster :

$$\text{CHI} = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{n - k}{k - 1}$$

où $\text{Tr}(B_k)$ est la trace de la matrice de dispersion inter-cluster, $\text{Tr}(W_k)$ est la trace de la matrice de dispersion intra-cluster, n est le nombre total de points, et k est le nombre de clusters.

- La variance inter-cluster ($\text{Tr}(B_k)$) mesure la séparation entre les clusters.
- La variance intra-cluster ($\text{Tr}(W_k)$) mesure la compacité des clusters.

Un indice de Calinski-Harabasz élevé indique des clusters bien séparés et compacts.

Avantages :

- Fournit une évaluation fiable de la qualité des clusters.
- Convient pour des clusters bien séparés.

Inconvénients :

- Peut être biaisé par des clusters de tailles inégales.

Temps d'exécution Comme pour la régression, le temps d'exécution est une métrique importante pour évaluer l'efficacité des algorithmes de clustering, notamment pour ceux qui effectuent des recherches aléatoires comme CLARANS.

Chapitre 3

Expérimentation

Ce chapitre présente les expérimentations menées dans le cadre de ce projet. Les algorithmes de régression et de clustering ont été appliqués sur le dataset prétraité. Les performances des modèles ont été évaluées à l'aide des métriques présentées dans le chapitre précédent.

Nous avons pour chaque modèle fait varier différents paramètres, afin de trouver la combinaison qui donne les meilleurs résultats.

3.1 Evaluation des modèles de régression

Dans les tests suivants, nous avons utilisé la prédiction des 4 attributs en même temps.

3.1.1 Modèle de Decision Tree

Variation de la profondeur de l'arbre : Nous avons fixé le nombre d'instances minimum à 2 pour effectuer les tests sur la profondeur de l'arbre. Les résultats sont présentés dans le tableau suivant :

| Profondeur maximale | MAE | RMSE | R2 | Temps (s) |
|---------------------|--------|--------|--------|-----------|
| 5 | 0.0457 | 0.0637 | 0.8671 | 7.1655 |
| 10 | 0.0328 | 0.0509 | 0.9155 | 12.8763 |
| 15 | 0.0334 | 0.0521 | 0.9116 | 15.8263 |
| 20 | 0.0331 | 0.0519 | 0.9122 | 16.5502 |
| 25 | 0.0331 | 0.0519 | 0.9122 | 16.6113 |

TABLE 3.1 – Variation de la profondeur de l'arbre de décision.

D'après les résultats, une profondeur égale à 10 donne les meilleurs résultats.

Variation du nombre minimum d'instances pour la division : La profondeur de l'arbre est fixée à 10. Le tableau suivant montre les résultats obtenus :

| Nombre minimum d'instances | MAE | RMSE | R2 | Temps (s) |
|----------------------------|--------|--------|--------|-----------|
| 2 | 0.0328 | 0.0509 | 0.9155 | 12.8763 |
| 4 | 0.0329 | 0.0512 | 0.9143 | 12.3898 |
| 6 | 0.0324 | 0.0502 | 0.9176 | 12.0237 |
| 8 | 0.0326 | 0.0498 | 0.9188 | 11.7471 |
| 10 | 0.0327 | 0.0497 | 0.9189 | 11.2801 |

TABLE 3.2 – Variation du nombre minimum d'instances pour la division.

On conclut donc que le modèle est plus performant avec une profondeur de 10 et un nombre minimum d'instances égal à 10. Ces valeurs seront utilisées pour les tests suivants.

Comparaison du modèle implémenté avec le modèle prédéfini de la bibliothèque Scikit-learn :

| Modèle | MAE | RMSE | R2 | Temps (s) |
|-----------|--------|--------|--------|-----------|
| CustomDT | 0.0327 | 0.0497 | 0.9189 | 11.2801 |
| SklearnDT | 0.0316 | 0.0446 | 0.9353 | 0.0051 |

TABLE 3.3 – Comparaison entre le modèle implémenté et celui de Scikit-learn pour Decision Tree.

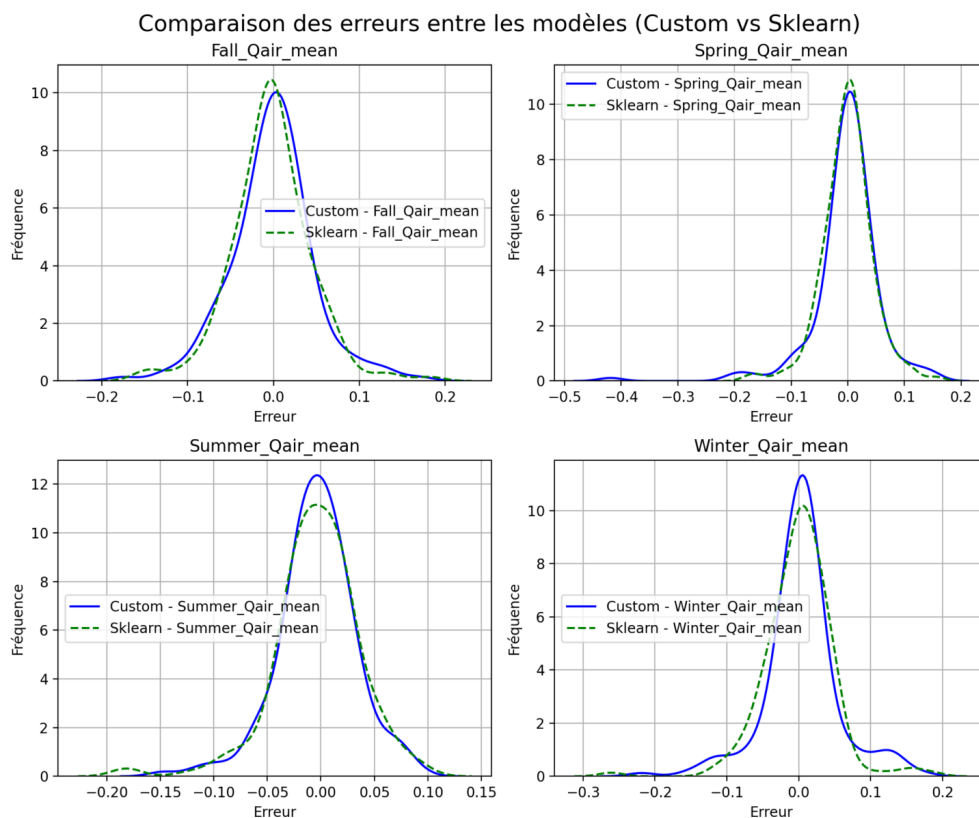


FIGURE 3.1 – Comparaison entre le modèle implémenté et celui de Scikit-learn pour Decision Tree

D'après les résultats, le modèle Scikit-learn est plus performant que l'implémentation personnalisée. Cela dit, les résultats de cette dernière restent compétitifs et pourraient être améliorés avec

d'avantage d'optimisations.

3.1.2 Modèle de Random Forest

Variation du nombre d'arbres : La profondeur maximale et le nombre minimum d'instances pour la division sont fixés à 10. Les résultats sont présentés dans le tableau suivant :

| Nombre d'arbres | MAE | RMSE | R2 | Temps (s) |
|-----------------|--------|--------|--------|-----------|
| 10 | 0.0283 | 0.0419 | 0.9429 | 48.4035 |
| 20 | 0.0294 | 0.0437 | 0.9377 | 105.7295 |
| 30 | 0.0289 | 0.0411 | 0.9449 | 180.2696 |
| 40 | 0.0275 | 0.0399 | 0.9483 | 244.0257 |
| 50 | 0.0297 | 0.0432 | 0.9392 | 303.3905 |

TABLE 3.4 – Variation du nombre d'arbres dans le modèle Random Forest.

Les résultats montrent que le modèle atteint ses meilleures performances avec 40 arbres, où la MAE est la plus faible et le R^2 le plus élevé. Toutefois, le temps d'exécution devient considérable avec un nombre d'arbres supérieur, ce qui représente un compromis entre précision et coût computationnel.

Variation de la profondeur maximale :

| Profondeur maximale | MAE | RMSE | R2 | Temps (s) |
|---------------------|--------|--------|--------|-----------|
| 5 | 0.0382 | 0.0517 | 0.9129 | 160.5937 |
| 10 | 0.0275 | 0.0399 | 0.9483 | 244.0257 |
| 15 | 0.0282 | 0.0429 | 0.9399 | 258.4137 |
| 20 | 0.0276 | 0.0415 | 0.9440 | 257.9666 |

TABLE 3.5 – Variation de la profondeur maximale dans Random Forest.

La profondeur maximale de 10 semble être la meilleure option pour Random Forest. Elle assure un bon équilibre entre la performance (avec un R^2 élevé et une MAE faible) et un temps de calcul raisonnable.

Variation du nombre minimum d'instances pour la division :

| Profondeur maximale | MAE | RMSE | R2 | Temps (s) |
|---------------------|--------|--------|--------|-----------|
| 2 | 0.0278 | 0.0409 | 0.9455 | 291.4732 |
| 6 | 0.0289 | 0.0431 | 0.9396 | 253.9611 |
| 10 | 0.0275 | 0.0399 | 0.9483 | 244.0257 |

TABLE 3.6 – Variation du nombre minimum d'instances pour la division dans Random Forest.

Le nombre d’instances optimal est de 10, avec des performances de R^2 et de MAE qui restent stables, tout en offrant des résultats de qualité en termes de temps d’exécution.

Comparaison entre le modèle implémenté et celui de Scikit-learn :

| Modèle | MAE | RMSE | R2 | Temps (s) |
|-----------|--------|--------|--------|-----------|
| CustomRF | 0.0280 | 0.0396 | 0.9490 | 246.6183 |
| SklearnRF | 0.0297 | 0.0424 | 0.9411 | 0.2383 |

TABLE 3.7 – Comparaison entre le modèle implémenté et celui de Scikit-learn pour Random Forest.

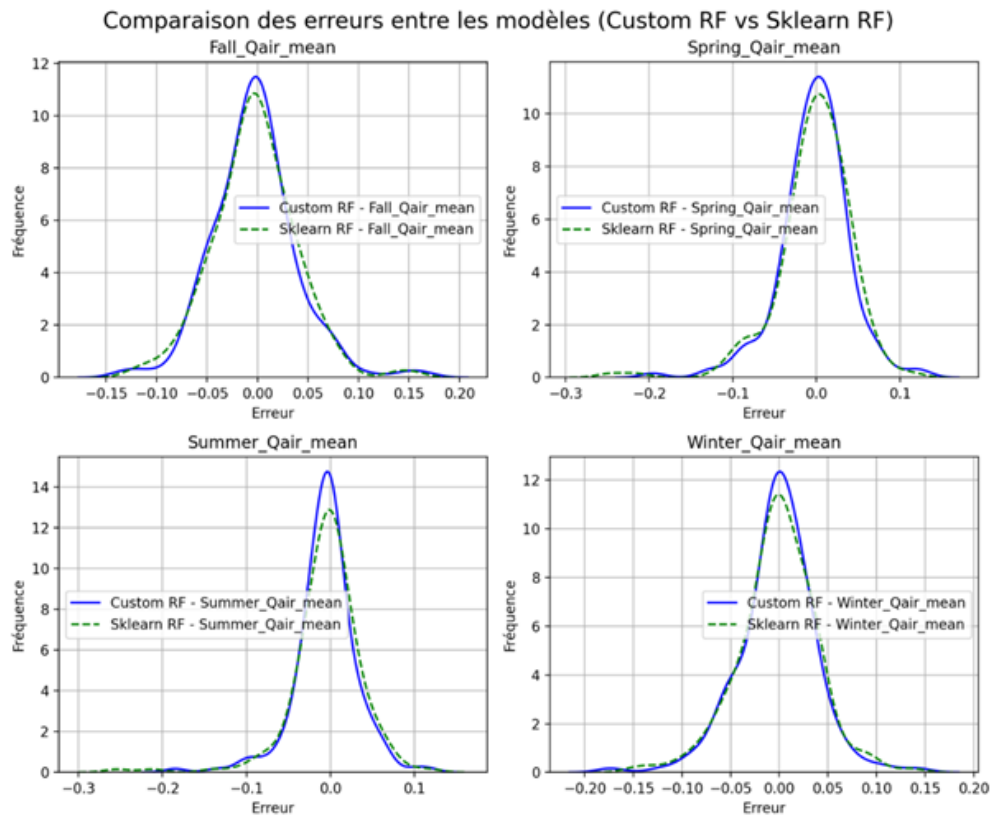


FIGURE 3.2 – Comparaison entre le modèle implémenté et celui de Scikit-learn pour Random Forest

En comparant les deux implémentations, le modèle Scikit-learn est plus rapide, mais l’implémentation personnalisée offre un meilleur R^2 et une meilleure MAE.

Analyse des résultats des modèles de regression :

Les résultats des tests de régression mettent en évidence l’importance cruciale de la configuration des hyperparamètres pour atteindre des performances optimales. L’algorithme Random Forest surpasse le Decision Tree en termes de précision, principalement grâce à sa capacité à capturer

des interactions complexes entre les attributs et à réduire le risque de surapprentissage. Cependant, cette amélioration de précision s'accompagne d'un coût : un temps de calcul plus élevé, particulièrement problématique pour les jeux de données volumineux.

Par ailleurs, bien que les algorithmes proposés par Scikit-learn offrent des performances robustes et standardisées, les implémentations personnalisées présentent parfois l'avantage de mieux s'adapter aux spécificités des données et aux besoins spécifiques des utilisateurs. Ces ajustements peuvent être particulièrement utiles dans des contextes où les données sont atypiques ou présentent des caractéristiques uniques.

Enfin, il est important de souligner que les algorithmes standards de Scikit-learn sont implémentés en Cython, un langage hybride entre Python et C. Cette implémentation permet d'exécuter une partie du code en langage C, un langage de bas niveau reconnu pour sa rapidité d'exécution. Cette optimisation explique la différence notable dans les temps d'exécution entre les algorithmes de Scikit-learn et des implémentations purement en Python. Cela met en lumière l'importance des choix technologiques dans le développement d'algorithmes, notamment lorsque la performance computationnelle est un facteur clé.

3.2 Evaluation des modèles de clustering

3.2.1 Algorithmes DBSCAN

Variation de epsilon (eps) :

| Epsilon (eps) | Silhouette | DBI | Calinski-Harabasz |
|---------------|------------|--------|-------------------|
| 0.2 | 0.1850 | 1.1649 | 52.76 |
| 0.4 | 0.2759 | 1.3194 | 138.5341 |
| 0.6 | 0.3896 | 1.6658 | 267.4116 |
| 0.8 | 0.3919 | 1.2019 | 31.3874 |

TABLE 3.8 – Variation de epsilon dans DBSCAN.

Epsilon est donc fixé à 0.6.

Variation du nombre minimum d'instances dans un cluster :

| min_samples | Silhouette | DBI | Calinski-Harabasz |
|-------------|------------|--------|-------------------|
| 5 | 0.3896 | 1.6658 | 267.4116 |
| 10 | 0.3810 | 1.8051 | 269.7637 |
| 15 | 0.3602 | 1.6276 | 258.1267 |
| 20 | 0.3862 | 1.1388 | 354.9305 |

TABLE 3.9 – Variation du nombre minimum d'instances dans un cluster dans DBSCAN

Les résultats montrent que le nombre minimum d'instances optimal se situe à 20.

3.2.2 Algorithmes CLARANS

Variation du nombre de clusters (k) :

| Nombre de clusters (k) | Silhouette | DBI | Calinski-Harabasz |
|------------------------|------------|--------|-------------------|
| 2 | 0.3708 | 0.9353 | 410.2746 |
| 4 | 0.2699 | 1.3730 | 260.1812 |
| 6 | 0.3176 | 1.3534 | 331.2260 |
| 8 | 0.3425 | 1.2789 | 296.8561 |

TABLE 3.10 – Variation du nombre de clusters dans CLARANS.

K est donc fixé à 2.

Variation du Nombre de recherches locales :

| Nombre de recherches locales | Silhouette | DBI | Calinski-Harabasz |
|------------------------------|------------|--------|-------------------|
| 10 | 0.3708 | 0.9353 | 410.2746 |
| 12 | 0.3684 | 0.9373 | 408.7271 |
| 14 | 0.3739 | 0.9062 | 410.3874 |
| 16 | 0.3663 | 0.9643 | 408.8795 |
| 18 | 0.3684 | 0.9379 | 408.6290 |
| 20 | 0.3708 | 0.9304 | 410.7891 |

TABLE 3.11 – Variation du Nombre de recherches locales dans CLARANS

Les meilleurs résultats sont obtenus avec 14 recherches locales.

Variation du Nombre maximal de voisins :

| Nombre maximal de voisins | Silhouette | DBI | Calinski-Harabasz |
|---------------------------|------------|--------|-------------------|
| 5 | 0.3702 | 0.9383 | 410.4237 |
| 6 | 0.3664 | 0.9555 | 408.1530 |
| 7 | 0.3538 | 1.0028 | 380.8440 |
| 8 | 0.3570 | 1.0129 | 398.8436 |
| 9 | 0.3720 | 0.9233 | 410.9612 |
| 10 | 0.3678 | 0.9495 | 405.1308 |

TABLE 3.12 – Variation du Nombre maximal de voisins dans CLARANS

Le nombre optimal de voisins est de 9.

Variation du seuil d'arrêt :

| Seuil d'arrêt | Silhouette | DBI | Calinski-Harabasz |
|---------------|------------|--------|-------------------|
| 0.2 | 0.3746 | 0.9015 | 411.1639 |
| 0.5 | 0.3720 | 0.9233 | 410.9612 |
| 0.8 | 0.3698 | 0.9337 | 408.1203 |

TABLE 3.13 – Variation du seuil d'arrêt dans CLARANS

Le seuil d'arrêt de 0.2 génère les meilleures valeurs.

3.2.3 Comparaison entre l’algorithme DBSCAN et CLARANS

| Modèle | Silhouette | DBI | Calinski-Harabasz |
|---------|------------|--------|-------------------|
| CLARANS | 0.3746 | 0.9015 | 411.1639 |
| DBSCAN | 0.3862 | 1.1388 | 354.9305 |

TABLE 3.14 – Comparaison entre les modèles CLARANS et DBSCAN.

Les algorithmes DBSCAN et CLARANS montrent des performances compétitives, chacun ayant des avantages spécifiques selon les métriques évaluées. DBSCAN se distingue par un score légèrement supérieur en termes de silhouette, indiquant une meilleure cohésion et séparation des clusters dans certains cas. En revanche, CLARANS excelle sur des métriques telles que Calinski-Harabasz, favorisant des clusters bien définis et compacts.

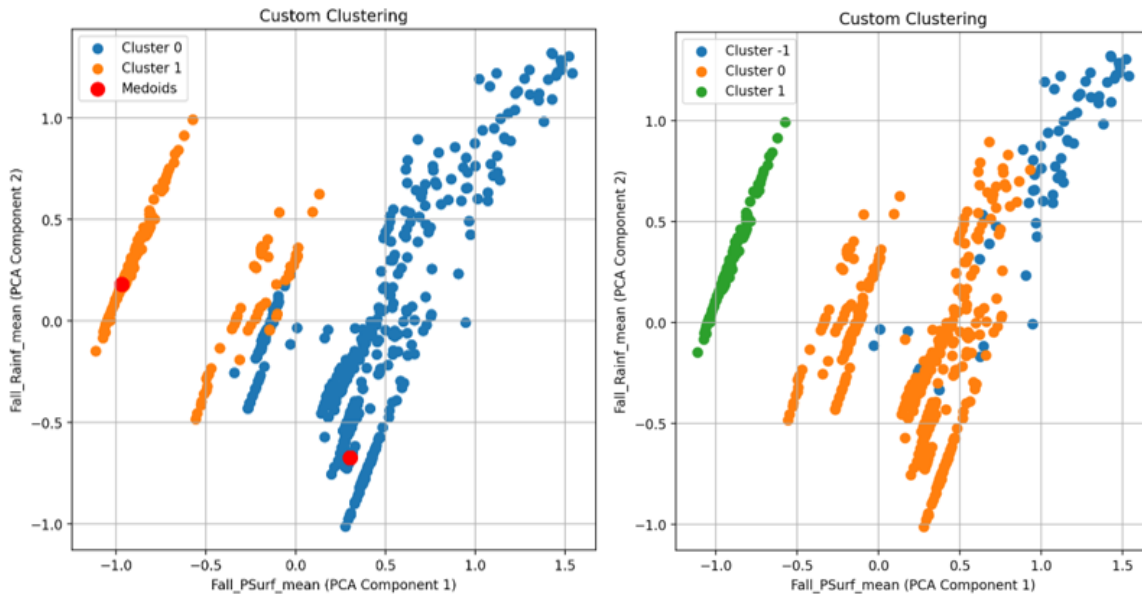


FIGURE 3.3 – Clusters générés par les modèles CLARANS et DBSCAN respectivement

D’après la figure 3.3, La relation entre PSurf (pression de surface) et Rainf (précipitations) semble suivre une tendance inverse : des pressions plus faibles (clusters avec faible PSurf) sont associées à des précipitations plus importantes, tandis que des pressions élevées correspondent à des conditions plus sèches. Les clusters générés par DBSCAN montrent une meilleure capacité à identifier des anomalies (Cluster -1), tandis que ceux générés par CLARANS, centrés autour des médoïdes, offrent une segmentation plus structurée des régimes climatiques.

Analyse générale des résultats des modèles de clustering :

Les données utilisées, permettent d’exploiter pleinement les capacités des deux algorithmes. DBSCAN, grâce à sa capacité à identifier des clusters de formes arbitraires et à gérer efficace-

ment les points de bruit, s'avère particulièrement performant dans des contextes où des variations significatives existent dans les données climatiques et du sol. Ces variations, conservées lors du traitement des données, contribuent à l'efficacité de cet algorithme.

D'un autre côté, CLARANS, avec son approche basée sur des recherches locales et son optimisation itérative, est mieux adapté aux clusters bien distincts présents dans les données. Son efficacité sur la métrique Calinski-Harabasz reflète sa capacité à créer des partitions cohérentes et compactes, exploitant la richesse des attributs sélectionnés après le traitement des corrélations.

En conclusion, le choix entre DBSCAN et CLARANS dépend des besoins spécifiques de l'analyse. DBSCAN est idéal pour des données présentant des variations complexes et bruitées, tandis que CLARANS se montre plus performant pour des clusters homogènes et bien séparés.

Conclusion

Ce projet a permis d'explorer et de comparer différents algorithmes de régression et de clustering dans le cadre du data mining. Les résultats obtenus montrent que chaque algorithme possède des forces spécifiques qui le rendent adapté à des contextes et des objectifs variés.

Pour la régression, l'algorithme Random Forest s'est révélé plus performant que les arbres de décision, grâce à sa capacité à capturer des interactions complexes tout en réduisant le surapprentissage. Cependant, cette précision accrue s'accompagne d'un coût computationnel plus élevé, mettant en lumière la nécessité d'un compromis entre précision et efficacité.

En ce qui concerne le clustering, les algorithmes DBSCAN et CLARANS ont montré des performances compétitives selon les métriques évaluées. DBSCAN s'est distingué par sa capacité à gérer efficacement les données bruitées et les clusters de formes arbitraires, tandis que CLARANS a démontré sa pertinence pour des clusters homogènes et bien séparés.

Ces analyses soulignent l'importance de choisir l'algorithme en fonction des caractéristiques des données et des objectifs de l'étude. Enfin, ce projet met en évidence le rôle crucial du prétraitement des données et de l'optimisation des hyperparamètres pour maximiser les performances des modèles. Cette démarche constitue une base solide pour des applications futures, tant en recherche qu'en pratique industrielle.

Références

- [1] : Résumé datamining 2021-2022 N.a.Houacine
- [2] : Résumé datamining 2021-2022 N.a.Houacine
- [3] : Résumé datamining 2021-2022 N.a.Houacine
- [4] : Résumé datamining 2021-2022 N.a.Houacine
- [5] : Résumé datamining 2021-2022 N.a.Houacine
- [6] : <https://pandas.pydata.org/>
- [7] : <https://numpy.org/>
- [8] : <https://scikit-learn.org/>
- [9] : <https://matplotlib.org/>