République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جامعة هواري بومدين للعلوم و التكنولوجيا
Université des Sciences et de la Technologie
Houari Boumediene

Faculté d'Informatique

Département d'Intelligence Artificielle

Master 2 Systèmes Informatiques intelligents

Module : Data Mining

Intitulé :

Apprentissage supervisé et non supervisé

Réalisé par :

— Djenane Nihad
— M'Barek Lydia

Travail demandé par :

Mme. Drias Habiba - Cours
Mme. Belkadi Widad Hassina - TP

Promotion : 2023/2024

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Le data mining représente le processus de recherche et d'analyse d'un ensemble volumineux de données brutes dans le but d'identifier des motifs et d'extraire des informations utiles. Ses applications sont diverses, allant de la gestion du risque de crédit à la détection de fraudes, en passant par la filtration des spams. De plus, il sert d'outil de recherche marketing, permettant de révéler les sentiments ou opinions d'un groupe donné de personnes.

Le data mining fait appel à des algorithmes et diverses techniques pour convertir d'importantes collections de données en résultats utiles. Parmi les types de techniques de data mining les plus populaires, on retrouve la classification et le clustering que nous aborderons dans cette deuxième partie du rapport.

Pour ce faire, nous avons utilisé le dataset 1 analysé dans la première partie. Nous allons appliquer des algorithmes de classification tels que KNN, arbres de décision et forêt aléatoire, ainsi que des algorithmes de regroupement comme K-means et DBScan. Ensuite, nous évaluerons expérimentalement ces méthodes sur nos données et les comparerons.

Chapitre 1

Données et Préparation

1.1 Introduction

Tout au long de ce rapport, nous ferons usage du dataset1 que nous avons analysé dans la première partie, comprenant des informations détaillées sur la qualité du sol. Ce chapitre débutera par un rappel de la description des données afin de préparer notre ensemble pour les étapes suivantes, où nous effectuerons des prédictions et du clustering à l'aide de modèles supervisés et non supervisés.

1.2 Récapitulation de la Description et Analyse des Données

Le dataset1 comprend des informations détaillées liées à la qualité du sol, comprenant 14 attributs décrivant des mesures telles que N, P, K, pH, etc. Toutes ces données sont numériques, et on trouve l'attribut 'fertility' qui décrit la classe de manière catégorique (0, 1 ou 2).
Ce dataset contient 885 instances, sous trouverez ci-dessous un rappel de quelques instances.

1	N	P	K	pH	EC	OC	S	Zn	Fe	Cu	Mn	B	OM	Fertility
2	138	8.6	560	7.46	0.62	0.7	5.9	0.24	0.31	0.77	8.71	0.11	1.204	0
3	213	7.5	338	7.62	0.75	1.06	25.4	0.3	0.86	1.54	2.89	2.29	1.8232	0
4	163	9.6	718	7.59	0.51	1.11	14.3	0.3	0.86	1.57	2.7	2.03	1.9092	0
5	157	6.8	475	7.64	0.58	0.94	26	0.34	0.54	1.53	2.65	1.82	1.6168	0
6	270	9.9	444	7.63	0.4	0.86	11.8	0.25	0.76	1.69	2.43	2.26	1.4792	1
7	220	8.6	444	7.43	0.65	0.72	11.7	0.37	0.66	0.9	2.19	1.82	1.2384	0
8	220	7.2	222	7.62	0.43	0.81	7.4	0.34	0.69	1.05	2	1.88	1.3932	0
9	207	7	401	7.63	0.59	0.69	7.6	0.32	0.68	0.62	2.43	1.68	1.1868	0
10	333	14.9	422	8.26	0.48	NA	8.45	0.51	3.32	1.08	9.21	0.32	2.0124	2

FIGURE 1.1 – Rappel Visuel de Quelques Instances du dataset1

Nous avons résumé la description de tous les attributs du dataset dans le tableau suivant :

	N	P	K	pH	EC	OC	S	Zn	Fe	Cu	Mn	B	OM	Fertility
Unique	61	93	63	107	71	69	153	70	387	167	429	127	68	3
Missing	0	2	0	0	0	1	0	0	0	1	0	0	0	0
Mean	247	14.55	501.34	7.51	0.54	0.62	7.55	0.47	4.13	0.95	8.65	0.59	1.06	0.59
Median	257	8.1	475	7.5	0.55	0.59	6.64	0.36	3.56	0.93	8.34	0.41	1.01	1
Mod	207	8.3	444	7.5	0.62	0.88	4.22	0.28	6.32	1.25	7.54	0.34	1.51	1

TABLE 1.1 – Statistiques descriptives des attributs du dataset1 liés à la qualité du sol

Au cours de notre analyse, nous avons identifié des corrélations significatives, notamment une corrélation entre les attributs OC (Carbone Organique) et OM (Matière Organique). Ces résultats seront pris en compte lors de l'étape de prétraitement des données.

1.3 Prétraitement des Données

Après le rappel de la description et l'analyse des données, nous avons entamé le prétraitement des données afin de les préparer pour les chapitres suivants.

1.3.1 Séparation du dataset

Dans le cadre de notre analyse, nous prévoyons de procéder à une séparation des données pour les modèles supervisés. Nous avons divisé notre ensemble de données en deux parties : 80 % pour l'ensemble d'entraînement et 20 % pour le test, dans le but d'évaluer la performance du modèle. Cette répartition équitable garantit une représentation adéquate des différentes classes dans les deux ensembles, ce qui favorise une généralisation efficace du modèle.
Après la séparation, le dataset se compose désormais de quatre ensembles distincts :

- X_train : l'ensemble des instances d'apprentissage, contient 708 lignes et 13 colonnes.
- Y_train : l'ensemble des labels d'apprentissage, contient 708 lignes et une colonne.
- X_test : l'ensemble des instances de test, contient 177 lignes et 13 colonnes.
- Y_test : l'ensemble des labels de test, contient 177 lignes et une colonne.

En revanche, pour les modèles non supervisés, nous avons utilisé la totalité de l'ensemble de données. Cette approche s'explique par le fait que les modèles non supervisés, qui regroupent les données, ne nécessitent pas d'ensembles d'entraînement et de test distincts pour l'évaluation du modèle.

NB : La segmentation du dataset intervient préalablement au prétraitement des données. Cela permet de maintenir l'intégrité des données d'apprentissage et de test, favorisant ainsi une évaluation plus fidèle des performances des modèles.

1.3.2 Traitement des valeurs manquantes et aberrantes

- D'après l'analyse des données, nous avons constaté la présence de 4 valeurs manquantes (2 dans l'attribut P, 1 dans OC et 1 dans Cu). Étant donné que toutes ces données sont numériques et non symétriques, nous avons choisi d'opter pour la méthode de remplacement par la mediane.
- Concernant les valeurs aberrantes, nous les avons laissées telles quelles, car elles pourraient être des données pertinentes.

1.3.3 Réduction de la dimensionnalité

— Nous avons identifié trois lignes dupliquées que nous avons supprimées.

— Comme il y a une corrélation entre les deux attributs OC et OM, au point où on peut le calculer, OC étant déterminé à partir de OM, nous avons supprimé l'attribut OC.

Après la suppréssion des lignes dupliquées et l'attribut OC, X_train et Y_train contient 706 lignes et 12 colonnes.

NB : Dans le chapitre où l'on va étudier les modèles non supervisés, nous allons ignorer la classe (l'attribut "Fertility").

1.3.4 Transformation des Données

Comme nos données présentent des échelles différentes, nous avons effectué une normalisation min-max.

1.4 Conclusion

À la fin de ce chapitre, nous avons rappelé notre dataset, l'avons prétraité, les rendant aptes à être utilisées aussi bien pour la classification que pour le clustering

Chapitre 2

Analyse supervisée

2.1 Introduction

L'apprentissage supervisé est une approche du machine learning où l'on entraîne un modèle à partir de données étiquetées afin de généraliser et de faire des prédictions sur des exemples qu'il n'a jamais rencontrés. Dans ce chapitre, nous allons voir trois types d'algorithmes supervisés, les évaluer et les comparer.

2.2 K plus proches voisins

2.2.1 Principe

Le principe de l'algorithme K plus proches voisins (KNN) consiste à classer une nouvelle instance en fonction de la majorité des classes parmi ses k voisins les plus proches dans un ensemble d'entraînement, en utilisant une mesure de distance préalablement définie.

2.2.2 Pseudo-code

Input : dataset étiquetée , K , InstanceTest
Output: prediction

for *instance in dataset* **do**
  Calculer la distance entre InstanceTest et instnace du dataset ;
end
Trier les distances calculées par ordre croissant;
Sélectionner les k voisins les plus proches ;
Compter le nombre d'instances appartenant à chaque classe parmi les k voisins ;
Affecter à InstanceTest la classe majoritaire parmi les k voisins ;

return la classe prédite pour InstanceTest;

Algorithm 1: Pseudo code de l'Algorithme K plus proches voisins.

2.2.3 Application de l'Algorithme sur le dataset

Nous avons implémenté l'algorithme KNN sous forme d'une classe qui prend en compte les attributs tels que la distance, permettant le choix entre la distance euclidienne, la distance de Manhattan, la distnace de minokiski, etc, ainsi que le paramètre k représentant le nombre de voisins.

Nous avons appliqué l'algorithme KNN aux instances du dataset en sélectionnant la distance euclidienne avec k=2. Le résultat obtenu est un tableau de 177 lignes contenant les classes prédites pour X_test.

```
array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0], dtype=int64)
```

FIGURE 2.1 – Prédictions du modèle KNN sur dataset 1

2.2.4 Exemple de déroulement de l'Algorithme

Afin de saisir pleinement le fonctionnement de l'algorithme d'Arbre de décision, nous procédons à une exploration sur une instance spécifique du jeu de données. Prenons, à titre d'exemple :

N	P	k	PH	EC	S	Zn	Fe	Cu	Mn	B	OM
0.516	0.048	0.319	0.125	0.441	0.618	0.318	0.057	0.693	0.198	0.851	0.794

TABLE 2.1 – Instance de déroulement KNN

Étape 1 : Calculer les distances

Dans cette étape on doit calculer la distance entre l'instance et chaque ligne de X_Train, le résultat est sous forme un tableau contenant 706 rows.

0	1	2	...	703	704	705
1.429	1.385	1.284	...	1.257	1.419	1.448

TABLE 2.2 – Distances instance KNN

Étape 2 : Trier les distances par ordre croissant

Le trie des distance se fait sur les indices du tableau des distances calculé dans l'étape 1.

0	**1**	**2**	**...**	**703**	**704**	**705**
150	598	37	...	391	140	439

TABLE 2.3 – Indices triés KNN

Étape 3 : Sélectionner les K plus proches voisins

Dans notre cas, K = 2, les deux plus proches voisins pour l'instance choisie sont : 150 et 598.

Étape 4 : Sélectionner la classe dominante

La classe des deux voisins de l'instance est 0, donc l'instance appartient à la classe 0.

2.2.5 Évaluation du model

La matrice de confusion est un tableau qui résume les performances d'un modèle de classification en comparant les prédictions du modèle avec les véritables résultats. Elle montre le nombre de prédictions correctes et incorrectes, divisé en quatre catégories : vrais positifs (VP), vrais négatifs (VN), faux positifs (FP) et faux négatifs (FN). Cette matrice est une importante mesure d'évaluation pour comprendre la qualité des prédictions d'un modèle et calculer des métriques telles que l'exactitude, la précision, le rappel et le F-score.
Ci-dessous la matrice de confusion de l'algorithme KNN :

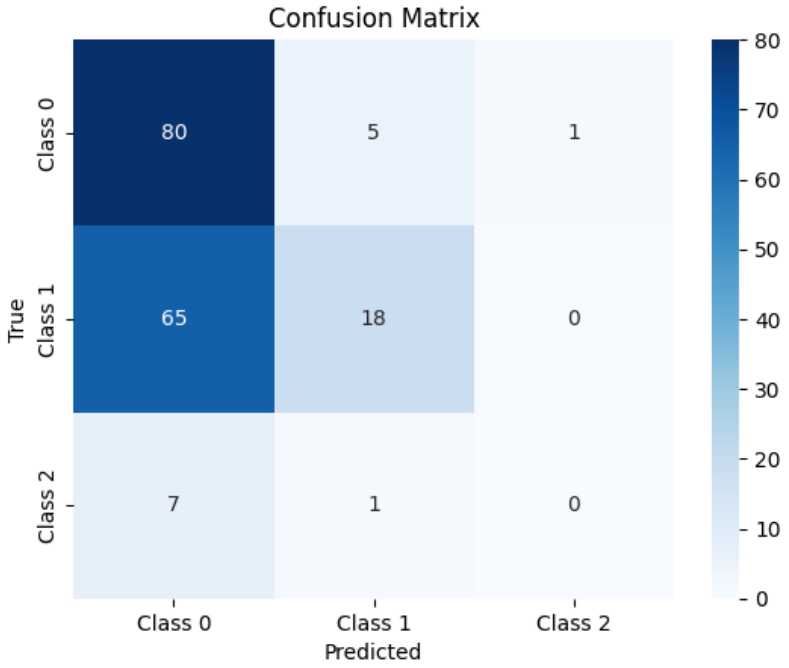


FIGURE 2.2 – Matrice de Confusion pour l'algorithme KNN avec le Dataset 1

Les mesures de performances, telles que l'exactitude, la précision, la spécificité, le rappel, et le score F1, sont essentielles dans l'évaluation des modèles d'apprentissage automatique. Elles permettent d'analyser la qualité des prédictions d'un modèle en comparant ses résultats avec les véritables valeurs de classe.
On calcule ces metriques comme suit :

$$EXACTITUDE = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

$$SPCIFICIT = \frac{TN}{TN + FP}$$

$$PRCISION = \frac{TP}{TP + FP}$$

$$RAPPEL = \frac{TP}{TP + FN}$$

$$F - SCORE = \frac{2 \cdot P \cdot R}{P + R}$$

Le tableau renferme les indicateurs de performance associés à l'algorithme :

EXACTITUDE	SPÉCIFICITÉ	PRÉCISION	RAPPEL	F-SCORE
0.702	0.776	0.553	0.553	0.553

Table 2.4 – Mesures de performances KNN

2.2.6 Expérimentation

Nous avons entrepris une série d'expérimentations dans le but d'évaluer les performances de l'algorithme KNN sur notre jeu de données. L'objectif principal était d'analyser la capacité du modèle à généraliser à de nouvelles données et de déterminer la configuration optimale du paramètre K pour résoudre notre problème.

A. Choix du paramètre k :

Nous avons commencé par tester différentes valeurs de k pour identifier celle qui offre les meilleures performances. Nous avons exploré des valeurs allant de 1 à 10.
k_values = [2, 3, 5, 7, 9]

B. Entraînement et évaluation :

Pour chaque valeur de k, nous avons entraîné le modèle k-NN sur l'ensemble d'entraînement. Le modèle utilise toujours la distance euclidienne pour mesurer la proximité entre les points du jeu de données. Et en fure à mesure on calcule l'exactitude de l'algorithme KNN pour chaque K.

K	2	3	5	7	9
EXACTITUDE	0.702	0.789	0.800	0.841	0.807

Table 2.5 – Exactitude KNN

Nous avons visualisé les résultats finaux en utilisant un graphique montrant l'évolution de l'exactitude en fonction des valeurs de k.

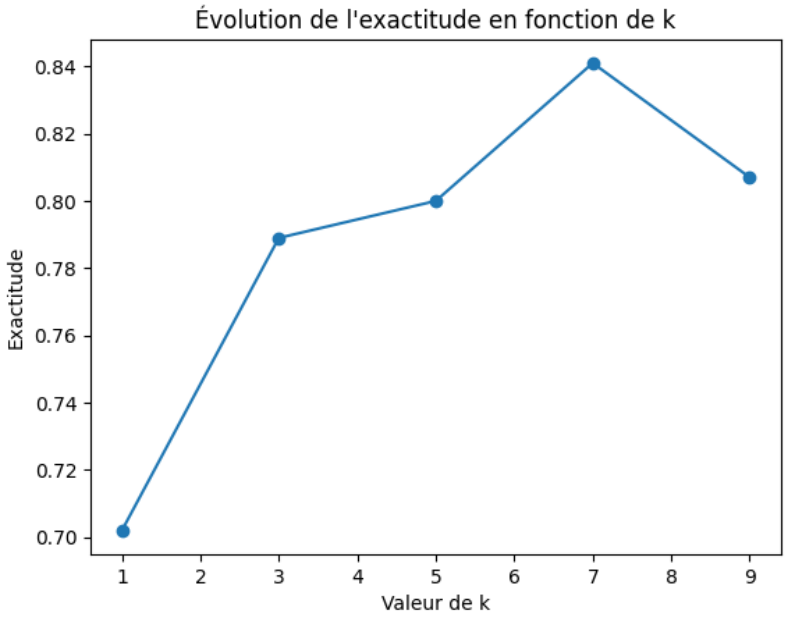


FIGURE 2.3 – Evolution de l'exactitude en fonction de K pourl'algorithme KNN

C. Sélection du meilleur k :

En se basant sur l'exactitude, nous avons identifié le meilleur k parmi l'ensemble des valeurs qu'on a choisi. Il s'agit de la valeur qui maximise l'exactitude sur l'ensemble de test (K = 7).

2.3 Arbre de décision

2.3.1 Principe

L'algorithme d'arbre de décision partitionne de manière récursive un ensemble de données en sous-groupes homogènes en utilisant des règles basées sur les caractéristiques, afin de créer une structure arborescente permettant de classifier ou de prédire de nouvelles instances.

2.3.2 Pseudo-code

Input : dataset étiquetée , attributCible, attributs
Output: Node

```
if dataset est vide then
    return un nœud Erreur ;
    else
        if attributs est vide then
            return un nœud ayant la classe C la plus représentée pour attributCible ;
            else
                if tout le dataset a la même classe C de l'attributCible then
                    return un nœud ayant cette classe C ;
                    else
                        attributSélectionné = attribut maximisant le gain d'information parmi
                         attributs ;
                        attributsRestants = suppressionListe(attributs, attributSélectionné) ;
                        newNode = nœud étiqueté avec attributSélectionné ;
                        for chaque valeur V de attributSélectionné do
                            dataFiltrésV = getAttributValeur(dataset, attributSélectionné, V) ;
                            newNode->fils(V) = ID3(dataFiltrésV, attributCible, attributsRestants) ;
                        end
                        return newNode ;
            end
    end
end
```

Algorithm 2: Pseudo code de l'Algorithme Arbre de décision.

2.3.3 Application de l'Algorithme sur le dataset

Au sein de notre projet, nous avons inclus une classe dédiée à l'algorithme Arbre de Décision. Cette classe est équipée de fonctions essentielles telles que "build_tree", qui construit l'arbre en utilisant des critères appropriés, "information_gain" pour évaluer la qualité des attributs, et "make_prediction" pour effectuer des prédictions sur de nouvelles données.

On a appliqué l'algorithme arbre de décision sur l'ensemble de dataset avec max_depth = 3 et min_samples_split = 3, la figure ci_dessous montre l'arbre contruite :

```
X_0 <= 0.6657824933687002 ? 0.06421514255099903
 left:X_1 <= 0.06797706797706797 ? 0.02065586419753085
  left:X_1 <= 0.01228501228501229 ? 0.009104534893369709
    left:1.0
    right:X_6 <= 0.011447650846649178 ? 0.00812278721078119
        left:0.0
        right:0.0
  right:X_0 <= 0.5013262599469496 ? 0.0893367763772176
    left:X_2 <= 0.26533247256294384 ? 0.17006802721088446
        left:1.0
        right:0.0
    right:X_4 <= 0.26666666666666666 ? 0.07926851955203318
        left:0.0
        right:1.0
 right:X_9 <= 0.0831984460990612 ? 0.009767598530658617
  left:X_3 <= 0.6391752577319587 ? 0.03080847723704852
    left:1.0
    right:X_0 <= 0.9655172413793104 ? 0.022682445759368883
        left:1.0
        right:1.0
  right:X_1 <= 0.019656019656019656 ? 0.011855191531174536
    left:X_8 <= 0.26279863481228666 ? 0.19970414201183428
        left:1.0
        right:0.0
    right:X_8 <= 0.04778156996587031 ? 0.009474815351630417
        left:0.0
        right:1.0
```

FIGURE 2.4 – Arbre de Décision pour dataset 1

2.3.4 Exemple de déroulement de l'Algorithme

Pour bien comprendre le principe de l'algorithme Arbre de décision, on déroule sur une instance de dataset. on prend l'exemple :

N	P	k	PH	EC	S	Zn	Fe	Cu	Mn	B	OM
0.516	0.048	0.319	0.125	0.441	0.618	0.318	0.057	0.693	0.198	0.851	0.794

TABLE 2.6 – Instance de déroulement pour l'Arbre de décision

En utilisant l'arbre construite précédamment on obtient :

- X_0 (N) <= 0.6657824933687002 : Oui (left)
- X_1 (P) <= 0.06797706797706797 : Oui (left)
- X_1 (P) <= 0.01228501228501229 : Non (right)
- X_6 (Zn) <= 0.011447650846649178 : Non (right)

La classe prédite est 0 (nœud terminal)

2.3.5 Évaluation du model

La matrice de confusion de l'algorithme Arbre de Décision permet d'évaluer ses performances en comparant les prédictions du modèle avec les valeurs réelles, mettant en évidence le nombre de vrais positifs, vrais négatifs, faux positifs et faux négatifs.

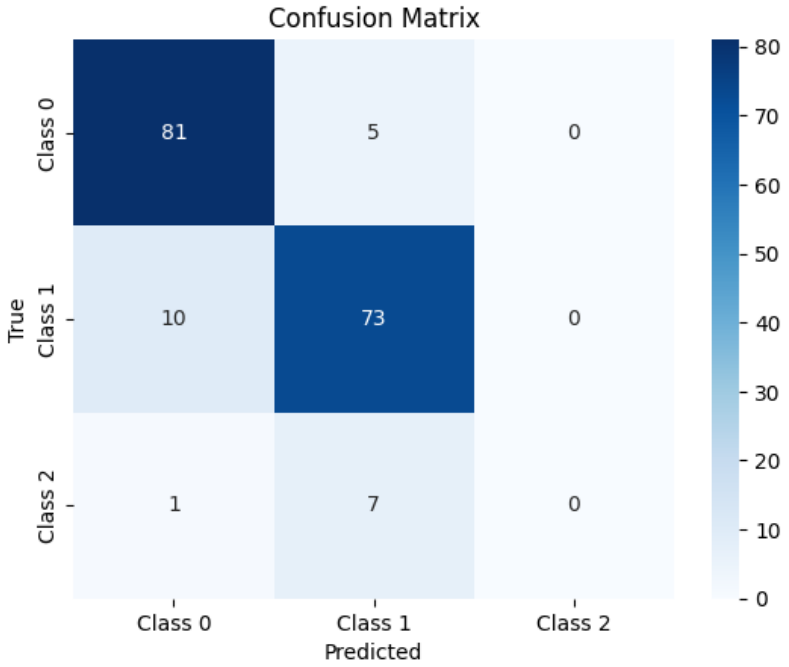


FIGURE 2.5 – Matrice de Confusion pour l'Arbre de Décision avec le Dataset 1

Le tableau présenté ci-dessous illustre les performances de l'algorithme de l'arbre de décision.

EXACTITUDE	SPÉCIFICITÉ	PRÉCISION	RAPPEL	F-SCORE
0.913	0.935	0.870	0.870	0.870

TABLE 2.7 – Mesures de performances de l'Arbre de décision

2.3.6 Expérimentation

Nous avons mené une série d'expériences visant à évaluer les performances de l'algorithme de l'arbre de décision sur notre ensemble de données. L'objectif était d'analyser la capacité du modèle à généraliser à de nouvelles données et à identifier la meilleure configuration d'hyperparamètres pour notre problème.

A. Résultat initiaux :

Dans la phase initiale de notre expérimentation, l'algorithme de l'arbre de décision a été configuré avec un seuil de division minimal (min_samples_split) égal à 3 et une profondeur maximale (max_depth) de 3.
Sur l'ensemble de test, le modèle a démontré des performances encourageantes. L'exactitude a atteint 91.3%, indiquant la proportion de prédictions correctes par rapport au nombre total d'observations. La spécificité, mesurant la capacité du modèle à identifier les vrais négatifs, s'est

élevée à 93.5%. La précision, représentant la proportion de vrais positifs parmi les prédictions positives, a été calculée à 87%. Le rappel, mesurant la capacité du modèle à identifier tous les exemples positifs, a également atteint 87%. Enfin, le F-score, une mesure équilibrée entre la précision et le rappel, s'est établi à 87%.
Ces résultats constituent notre point de départ pour l'évaluation des performances du modèle et guident la recherche ultérieure d'optimisation des hyperparamètres.

B. Optimisation des Hyperparamètres :

Dans le but d'optimiser les performances, nous avons entrepris une recherche d'hyperparamètres. En explorant différentes combinaisons de profondeur maximale et de seuil de division minimal (aléatoirement), nous avons identifié la configuration optimale du modèle.

— min_samples_split=4, max_depth=6

EXACTITUDE	SPÉCIFICITÉ	PRÉCISION	RAPPEL	F-SCORE
0.770	0.827	0.655	0.655	0.655

TABLE 2.8 – Évaluation de l'Arbre de Décision avec min_samples_split=4 et max_depth=6

— min_samples_split=2, max_depth=5

EXACTITUDE	SPÉCIFICITÉ	PRÉCISION	RAPPEL	F-SCORE
0.698	0.774	0.548	0.548	0.548

TABLE 2.9 – Évaluation de l'Arbre de Décision avec min_samples_split=2 et max_depth=5

— min_samples_split=5, max_depth=5

EXACTITUDE	SPÉCIFICITÉ	PRÉCISION	RAPPEL	F-SCORE
0.709	0.782	0.564	0.564	0.564

TABLE 2.10 – Évaluation de l'Arbre de Décision avec min_samples_split=5 et max_depth=5

C. Résultats optimisés :

Lors de l'optimisation des hyperparamètres, nous avons exploré différentes configurations, y compris une profondeur maximale plus élevée et un seuil de division minimal plus restrictif. Cependant, de manière intéressante, nous avons constaté que la combinaison initiale de min_samples_split=3 et max_depth=3 a généré des performances globales supérieures par rapport aux autres configurations testées.
Ces résultats suggèrent que, dans le contexte de notre problème spécifique, une approche moins complexe de l'arbre de décision avec une profondeur limitée et un seuil de division minimal modéré offre une meilleure généralisation aux nouvelles données par rapport aux configurations plus complexes que nous avons explorées.

2.4 Forêts aléatoires

2.4.1 Principe

Le principe de Random Forest repose sur la construction d'un ensemble (forêt) de plusieurs arbres de décision. Chaque arbre est construit sur un sous-ensemble aléatoire des données d'entraînement (bootstrap) et à chaque division d'un nœud, un sous-ensemble aléatoire de caractéristiques est considéré. Ensuite, les prédictions des arbres individuels sont combinées par un vote majoritaire (classification) ou une moyenne (régression) pour obtenir la prédiction finale du modèle. Cette approche réduit le surajustement (overfitting) en introduisant de l'aléa et améliore la performance globale du modèle.

2.4.2 Pseudo-code

Input : dataset étiquetée , InstanceTest, max_tree
Output: prediction

while *max_tree* **do**
    Bootstrapping data.
    Appliquer l'algorithme Arbre de décision.
    Prédections += predection.
end
return Prédections

2.4.3 Application de l'Algorithme sur le dataset

◦ **Fontionnement du code :**

Le code de la classe RandomForest vise à implémenter un modèle de forêt aléatoire en utilisant des arbres de décision. Lors de l'initialisation de la classe, les paramètres tels que le nombre maximal d'arbres (max_tree), le nombre minimal d'échantillons pour la division (min_samples_split), et la profondeur maximale des arbres (max_depth) sont définis.

La méthode bootstrapping est responsable de générer des sous-ensembles de données en utilisant l'échantillonnage bootstrap. Pour chaque arbre dans la forêt, un sous-ensemble aléatoire est créé en sélectionnant aléatoirement des lignes et des colonnes du jeu de données initial.

La méthode make_forest construit la forêt aléatoire en utilisant les sous-ensembles générés par le bootstrapping. Pour chaque sous-ensemble, un arbre de décision est formé à l'aide de la classe DecisionTree, dont la mise en œuvre n'est pas fournie ici. Chaque arbre est ensuite ajouté à la forêt.

La méthode statique prediction fait des prédictions sur de nouvelles données en agrégeant les prédictions de chaque arbre dans la forêt. Les prédictions sont obtenues en utilisant la modalité (mode) des prédictions individuelles de chaque arbre.

◦ **Application :**

Dans ce projet, on appliqué l'algorithme Random Forest sur l'ensemble de dataset avec max_tree = 10, min_samples_split = 2, max_depth = 2 et le résultat est un tableau de 177 lignes contient les classes prédites de X_test.

2.4.4 Exemple de déroulement de l'Algorithme

Pour bien comprendre le principe de l'algorithme Random Forest, on déroule sur une instance de dataset. on prend l'exemple :

N	P	k	PH	EC	S	Zn	Fe	Cu	Mn	B	OM
0.516	0.048	0.319	0.125	0.441	0.618	0.318	0.057	0.693	0.198	0.851	0.794

TABLE 2.11 – Instance de déroulement Random Forest

Le modèle prend l'instance en entrée, effectue une prédiction en se basant sur chaque arbre précédemment construit, puis renvoie un tableau de prédictions de taille égale au nombre de max_tree. On peut voir ci dessous le tableau de prédictions de notre cas :

Trees	1	2	3	4	5	6	7	8	9	10
Prédictions	0	0	0	0	0	1	0	0	1	1

TABLE 2.12 – Prédictions de l'instance Random Forest

En utilisant le tableau de prédictions, le modèle détermine le mode de ce tableau et l'assigne comme résultat pour l'instance.
Avec notre exemple, le mode = 0 alors :

La classe de l'instance est 0

2.4.5 Évaluation du model

La matrice de confusion de l'algorithme Random Forest offre une évaluation de ses performances en confrontant les prédictions du modèle aux valeurs réelles, mettant en lumière le nombre de vrais positifs, de vrais négatifs, de faux positifs et de faux négatifs.

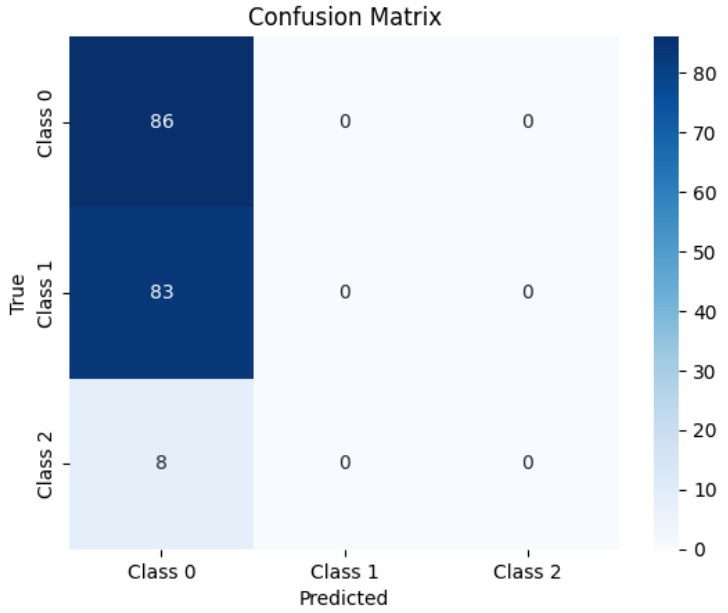


FIGURE 2.6 – Matrice de Confusion pour l'algorithme Random Forest avec le Dataset 1

Le tableau présenté ci-dessous illustre les performances de l'algorithme de Random Forest.

EXACTITUDE	SPÉCIFICITÉ	PRÉCISION	RAPPEL	F-SCORE
0.657	0.742	0.485	0.485	0.485

TABLE 2.13 – Mesures de performances Random Forest

2.4.6 Expérimentation

Nous avons mené une série d'expériences visant à évaluer l'efficacité de l'algorithme Random Forest sur notre ensemble de données. L'objectif principal était d'analyser la capacité du modèle à améliorer la généralisation et à fournir des performances robustes, en modifiant le paramètre max_tree et déterminer la configuration optimale du ce paramètre.

A. Choix du paramètre max_tree :

Nous avons commencé par tester différentes valeurs de max_tree pour identifier celle qui offre les meilleures performances. Nous avons exploré des valeurs allant de 10 à 100.
max_tree_values = [10, 30, 50, 70, 100]

B. Entraînement et évaluation :

Pour chaque valeur de max_tree, nous avons entraîné le modèle Random Forest sur l'ensemble d'entraînement. Et en fure à mesure on calcule l'exactitude de l'algorithme Random Forest pour chaque valeur de max_tree.

max_tree	10	30	50	70	100
EXACTITUDE	0.657	0.691	0.736	0.755	0.801

TABLE 2.14 – Exactitude Random Forest

Nous avons visualisé les résultats finaux en utilisant un graphique montrant l'évolution de l'exactitude en fonction des valeurs de max_tree.

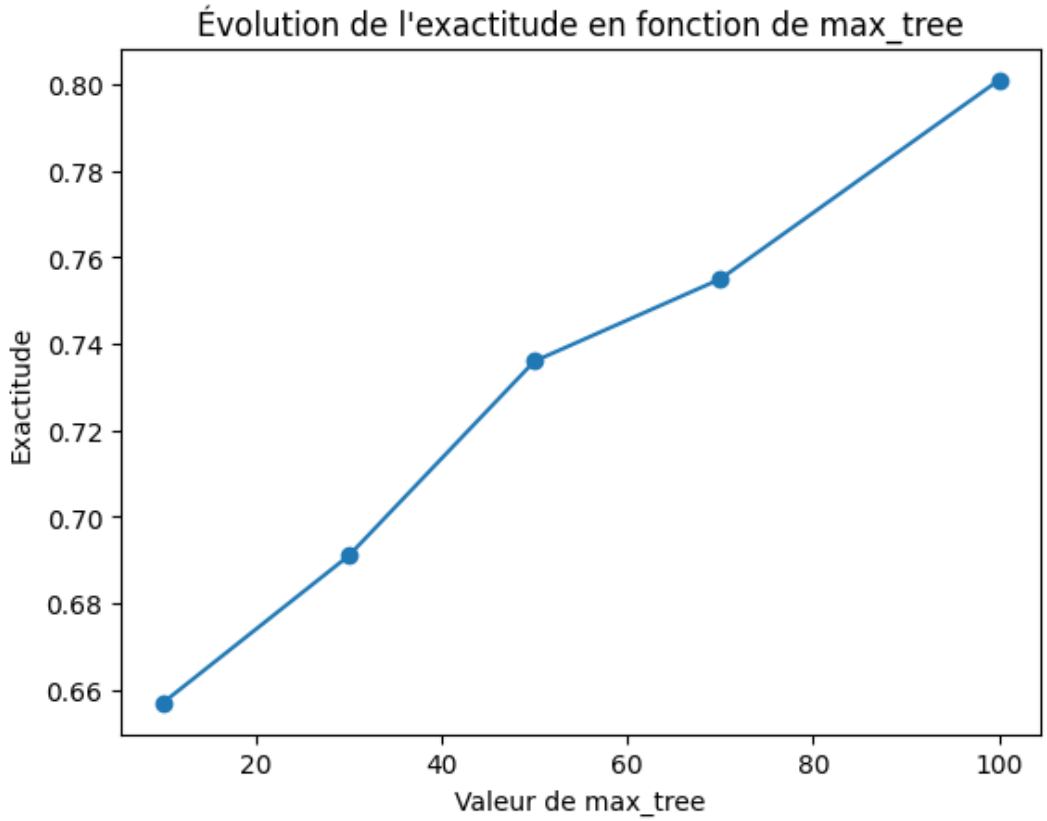


FIGURE 2.7 – Evolution de l'exactitude en fonction de max_tree pour Random Forest

C. Sélection du meilleur max_tree :

En analysant l'exactitude, nous avons observé une corrélation directe : à mesure que la valeur de max_tree augmente, la performance du modèle s'améliore. Par conséquent, nous optons pour la valeur maximale possible de max_tree.

2.5 Comparaison des Modèles

Après avoir mis en œuvre et expérimenté les algorithmes KNN, Arbre de décision et Random Forest en vue de les comparer, nous avons fixé les valeurs des paramètres suivants : K = 2 , max_depth = 3, min_samples_split = 3 et max_tree=10.
Nous avons opté pour l'exactitude en tant que mesure de performance principale et le temps d'exécution de chaque modèle comme références de comparaison.
Le tableau présent ci-dessous affiche l'exactitude de chaque modèle pour les paramètres sélectionnés.

Modèle	KNN	Arbre de décision	Random Forest
EXACTITUDE	0.702	0.913	0.657

TABLE 2.15 – Comparaison de l'exactitude

Le graphe ci déssous montrant l'exactitude des trois modèles pour les même paramètres :

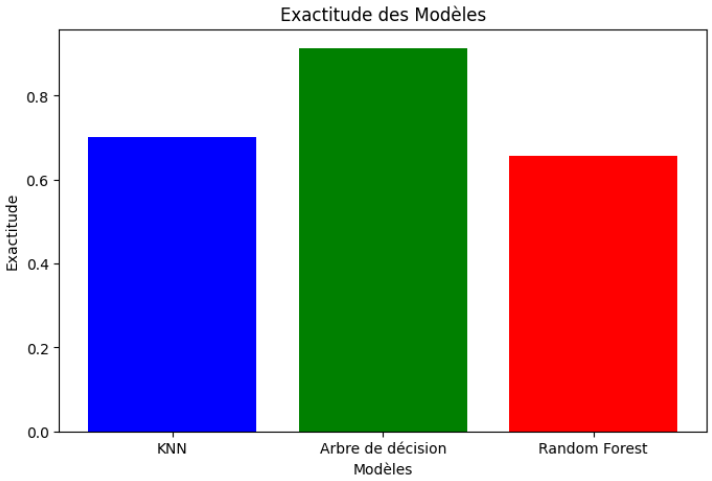


FIGURE 2.8 – Exactitude des trois Modèles

Observation : En observant les résultats d'exactitude pour les différents modèles, on remarque que l'Arbre de Décision affiche la plus haute exactitude avec 91.3%, suivi de près par le modèle KNN avec 70.2%, et enfin la Forêt Aléatoire avec 65.7%. Cette observation suggère que, pour la tâche spécifique considérée, l'Arbre de Décision est le plus performant en termes d'exactitude.
Le tableau présent ci-dessous affiche le temps d'exécution de chaque modèle pour les même paramètres

Modèle	**KNN**	**Arbre de décision**	**Random Forest**
Temps d'exécution	0.830	5.147	34.799

TABLE 2.16 – Comparaison du temps d'exécution

Le graphe ci déssous montrant le temps d'exécution des trois modèles pour les même paramètres :

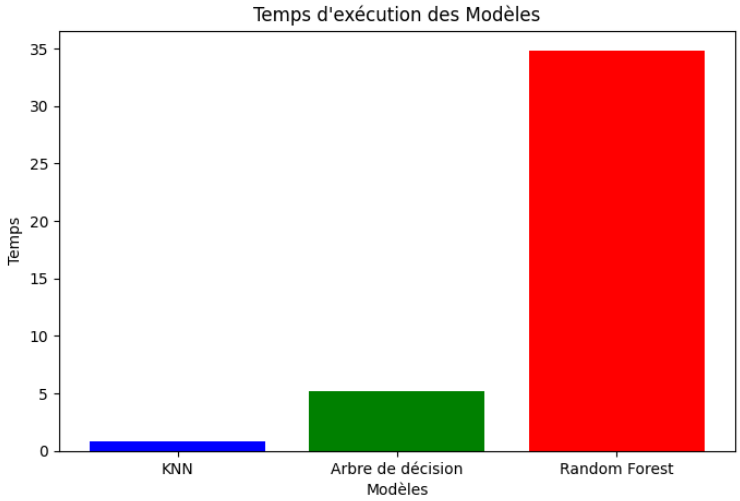


FIGURE 2.9 – Temps d'exécution des trois Modèles

Observation :
En examinant les temps d'exécution des différents modèles, on constate que le modèle KNN affiche le temps le plus court avec 0.830 secondes, suivi de l'Arbre de Décision avec 5.147 secondes, et enfin la Forêt Aléatoire avec 34.799 secondes. Cette observation souligne la rapidité d'exécution du modèle KNN par rapport aux autres.

Conclusion :

En conclusion, l'analyse comparative des modèles KNN, Arbre de Décision, et Random Forest a révélé des résultats intéressants. L'Arbre de Décision a démontré la plus haute exactitude, ce qui suggère une meilleure capacité de prédiction pour la tâche spécifique. Cependant, il est crucial de prendre en compte le temps d'exécution, où le modèle KNN a montré une performance remarquablement rapide par rapport aux autres. Le choix entre ces modèles dépend donc d'un compromis entre la précision de la prédiction et la rapidité d'exécution, en fonction des exigences spécifiques du problème et des contraintes de temps. Une évaluation plus approfondie, notamment à l'aide d'autres métriques de performance, est recommandée pour prendre une décision éclairée sur le modèle optimal.

2.6 Conclusion

En conclusion, l'apprentissage supervisé représente une méthode fondamentale du machine learning, permettant à un modèle de s'entraîner sur des données étiquetées en vue de généraliser et de faire des prédictions sur de nouveaux exemples.
Dans ce chapitre, nous avons exploré trois types d'algorithmes supervisés, les avons évalués et comparés. Cette analyse approfondie nous offre un aperçu significatif des performances et des caractéristiques spécifiques de chaque algorithme, éclairant ainsi le choix potentiel du modèle le mieux adapté à une tâche donnée.

Chapitre 3

Analyse non supervisée

3.1 Introduction

L'apprentissage non supervisé représente l'inverse de l'apprentissage supervisé. Dans ce type d'apprentissage, les données ne sont pas étiquetées, permettant au modèle d'explorer la structure inhérente aux données et de rechercher des motifs, des clusters ou des relations. Ce chapitre se concentrera sur l'examen, l'évaluation et la comparaison de deux types d'algorithmes non supervisés.

3.2 K-Means

3.2.1 Principe

K-means cherche à regrouper un ensemble de données en k clusters de manière à minimiser la variance intra-cluster, en attribuant chaque point au cluster dont le centroïde est le plus proche et en ajustant itérativement les positions des centroïdes.

3.2.2 Pseudo-code

Input : dataset , K
Output: clusters

Sélectionnez k points au hasard dans le dataset en tant que centroïdes.;
for *chaque point P in dataset* **do**
    selectionnez le centroid le plus proche C_j;
    Ajouter P au clusterS_j
end
Mise a jour les centroids.;
Répétez la boucle à l'étape 2, arrêtez lorsque les centroïdes ne changent plus.

return Clusters;

Algorithm 3: Pseudo code de l'Algorithme K plus proches voisins.

◦ **Initialisation intelligente des centroïdes :**

Pour aider l'algorithme K-means à converger rapidement, nous avons utilisé un processus de sélection de centroïde plus stratégique, la méthode K-means++, qui consiste à :

1. Sélectionnez un premier centroïde au hasard parmi les points de données.
2. Calculez la probabilité pour chaque point d'être choisi comme prochain centroïde. Cette probabilité est basée sur la distance au carré par rapport au centroïde déjà choisi.
3. Choisissez le prochain centroïde en fonction des probabilités calculées. Plus un point est éloigné des centroïdes déjà choisis, plus il a de chances d'être sélectionné.
4. Répétez les étapes 2 et 3 jusqu'à ce que le nombre requis de centroïdes soit atteint.

Pour notre cas, nous allons implémenter les deux méthodes d'initialisation, aléatoire et intelligente, et nous allons expérimenter afin de trouver le meilleur choix.

3.2.3 Application de l'Algorithme KMeans sur dataset 1

Nous avons bien effectué l'implémentation de notre algorithme K-Means, qui prend les paramètres suivants :

— n_clusters : représente le nombre de clusters.

— max_iter : représente le nombre maximal d'itérations en cas de non-convergence.

— n_init : détermine le nombre d'initialisations des nouveaux centroids.

— init : représente le type de génération des centroids : K-Means++ pour une initialisation intelligente et random pour une génération aléatoire.

— metric : représente le type de distance utilisé : Euclidienne, Manhattan, cosinus, etc.

Nous avons appliqué notre algorithme sur le dataset que nous avons prétraité précédemment (en n'oubliant pas d'ignorer la classe attribut "fertilité") avec les paramètres suivants : un nombre de clusters égal à 3, une génération de centroids intelligente avec n_init égal à 10, et une mesure de distance euclidienne.
Les centroïdes que nous avons obtenus sont les suivants :

Attribut	N	P	K	pH	EC	S	Zn	Fe	Cu	Mn	B	OM
C1	0.447	0.037	0.321	0.643	0.526	0.472	0.007	0.014	0.374	0.110	0.707	0.027
C2	0.486	0.153	0.308	0.641	0.519	0.194	0.007	0.106	0.334	0.307	0.110	0.019
C3	0.788	0.063	0.321	0.647	0.523	0.201	0.011	0.092	0.246	0.288	0.147	0.022

TABLE 3.1 – Centroids finaux de l'algorithme K-means.

Pour une meilleure visualisation, nous avons réalisé une représentation graphique des clusters à l'aide de la méthode PCA (Analyse en Composantes Principales), qui réduit les dimensions pour nous permettre de visualiser les clusters de manière plus concise et compréhensible. Pour obtenir une compréhension rapide de la méthode PCA (Analyse en Composantes Principales), veuillez vous référer à l'**Annexe : Brève Définition de la PCA.**
La représentation des clusters obtenue est illustrée dans la figure suivante :

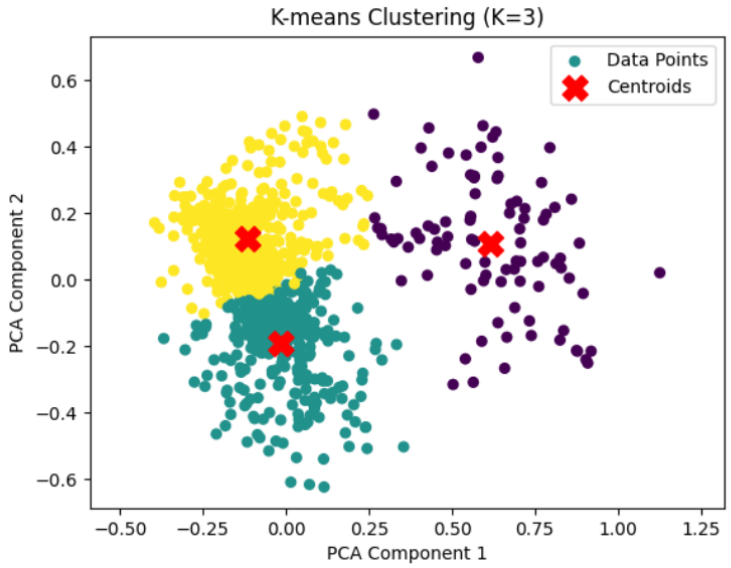


FIGURE 3.1 – Visualisation des Clusters Obtenu avec K-means (K = 3)

Chaque point sur le graphique représente une instance du Dataset 1, et les couleurs distinctes indiquent les clusters formés par notre algorithme K-means avec k=3. Les symboles "X" en rouge représentent le centre de chaque cluster.

3.2.4 Exemple de déroulement de l'Algorithme

Pour bien comprendre l'algorithme, nous allons décrire son déroulement étape par étape en utilisant des graphes illustratifs. Cela nous permettra de visualiser comment l'algorithme évolue au fil de son exécution.

— **Etape 1 :** Sélectionnez k points au hasard dans le dataset en tant que centroïdes.

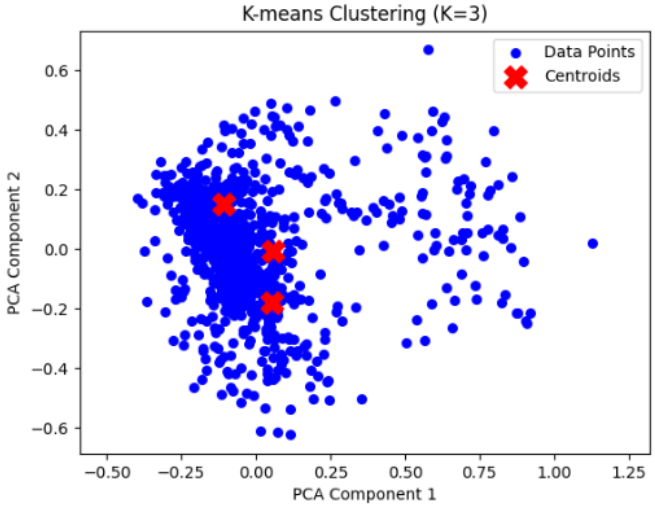


FIGURE 3.2 – Centroïdes Initiaux Générés Aléatoirement - Étape 1 de l'Algorithme K-means

— **Etape 2 :** Attribuer chaque instance au centroïde le plus proche.

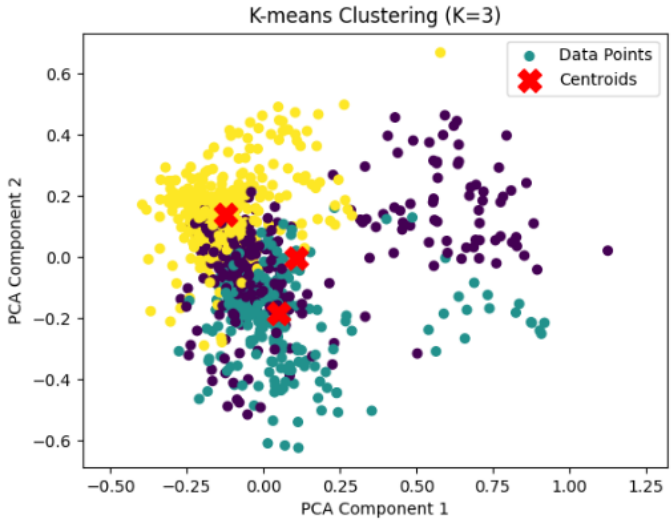


FIGURE 3.3 – Attribution des Instances aux Clusters Proches - Étape 2 de l'Algorithme K-means

— **Etape 3 :** Mise a jour les centroids.

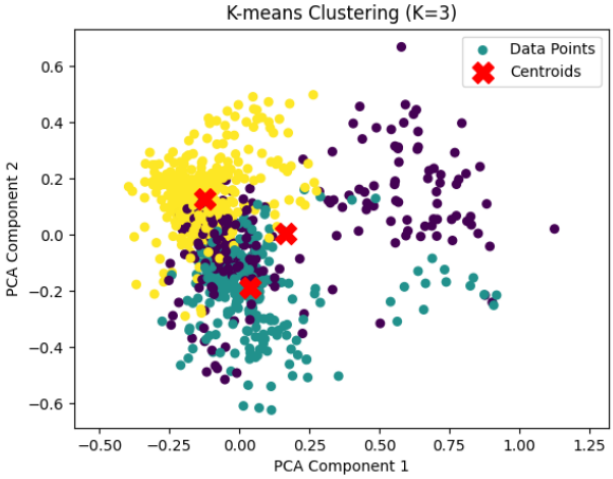


FIGURE 3.4 – Mise à jour des Centroides par Moyenne des Clusters - Étape 3 de K-means

Nous avons détaillé l'exécution de l'algorithme K-Means pour une seule itération.
Pour un aperçu complet du déroulement jusqu'à la convergence, veuillez consulter :
l'Annexe B : Déroulement de K-Means jusqu'à Convergence.

3.2.5 Évaluation du model

Après la construction du modèle, la méthode pour évaluer la performance de notre modèle consiste à utiliser deux quantités :

- **La distance intra-cluster :** qui calcule la distance au sein du cluster afin de la minimiser.

$$\text{Intra-cluster distance} = \sum_{i=1}^{K} \sum_{j=1}^{|C_i|} d(x_j, c_i) \quad \text{où :}$$

  - K est le nombre de cluster
  - $|C_i|$ est le nombre de points dans le cluster C_i .
  - $d(x_j, c_i)$ est la distance entre le point $\mathbf{x}_i$ dans le cluster et le centre de cluster.

- **La distance inter-cluster :** qui calcule la distance entre les clusters pour la maximiser, est généralement représentée par :

$$\text{Inter-cluster distance} = \sum_{j \neq i}^{K} d(C_i, C_j) \quad \text{où :}$$

  - K est le nombre de cluster
  - $d(c_i, c_j)$ est la distance entre le centre de deux cluster différents.

Et on trouve aussi **le coefficient de silhouette** qui meseure a quel point les objets du même cluster sont similaires entre eux et différents des objets des autres clusters. Le coefficient de silhouette (S) pour un point est donné par la formule :

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

tel que :

- **a** c'est la distance moyenne du point à son groupe, définie par :

$$a_i = \frac{1}{n-1} \cdot \sum_{i \neq j, j \in \text{Cluster}} d(i, j) \quad \text{où :}$$

  - d(i, j) est la distance entre les points $\mathbf{p}_i$ et $\mathbf{p}_j$,
  - n est le nombre de points dans le cluster

- **b** c'est la distance moyenne du point à son groupe voisin, définie par :

$$b_i = \min_{k \neq \text{Cluster}} \left(\frac{1}{|\text{Cluster}|} \sum_{i \in \text{Cluster}} d(i, k) \right) \quad \text{où :}$$

  - d(i, k) est la distance entre le point $\mathbf{p}_i$ dans le cluster et un point $\mathbf{p}_k$ dans le cluster voisin le plus proche
  - |Cluster| est le nombre de points dans le cluster.

Le coefficient de silhouette global pour l'ensemble des données est :

$$S = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|I_k|} \sum_{i \in I_k} s_i \quad \text{où :}$$

— K est le nombre total de clusters.

— $|I_k|$ est le nombre de points dans le cluster k.

— s_i est le coefficient de silhouette individuel pour le point i dans le cluster k

Un coefficient de silhouette proche de 1 indique une bonne séparation entre les clusters, -1 indique une mauvaise séparation, et 0 indique un chevauchement significatif entre les clusters.

Nous avons bien implémenté ces métriques, nous avons évalué notre modèle et les résultats obtenus sont :

- **Coefficient de silhouette** égal à : 0.17668013728367943
- **intra-cluster** égale à : 321.8053387952193
- **inter-cluster** égale à : 1.7732250691584412

3.2.6 Expérimentation du K-means

Dans le but d'optimiser les performances du modèle, nous avons mené des expérimentations en utilisant notre datset. Ces expérimentations nous ont permis d'ajuster et de fixer les paramètres du modèle afin d'obtenir des résultats plus efficaces et pertinents.

NB : Dans cette expérimentation, nous utilisons la métrique de distance euclidienne.

A. Déterminer la meilleure valeur pour K :

Dans le processus de fixation du nombre de clusters, nous avons utilisé la méthode du silhouette. Tout d'abord, nous avons calculé la moyenne des coefficients de silhouette pour l'ensemble des points du dataset, à chaque valeur de K. Ensuite, nous avons tracé deux graphiques illustrant l'évolution du coefficient en fonction du nombre de clusters, l'un pour une initialisation aléatoire et l'autre pour une initialisation intelligente.

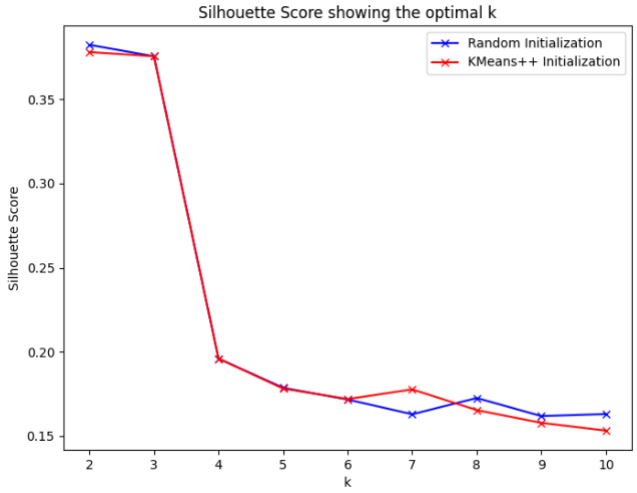


FIGURE 3.5 – Évolution du Coefficient de Silhouette en Fonction du Nombre de Clusters

Analyse et observations : D'après les deux graphe, on constate que :

- Les deux courbes sont presque identiques.
- A chaque fois que le nombre de clusters augmente, le coefficient de silhouette diminuet.
- Le coefficient de silhouette atteint son maximum lorsque K = 2.
- Pour un nombre de clusters égal à 3, le coefficient de silhouette atteint une valeur proche de son maximum, indiquant ainsi une bonne adéquation des clusters.

Conclusion : La valeur optimale de K est égale à 2.

Si vous souhaitez visualiser graphiquement les clusters en fonction du nombre de clusters, voir l'**Annexe C : Analyse Visuelle des Clusters en Fonction de K**

B. Identifier la meilleure méthode d'initialisation des centroids et le nombre maximal d'itérations :

Pour choisir la meilleure méthode d'initialisation des centroids, nous avons exécuté l'algorithme plusieurs fois pour les deux méthodes, calculant à chaque fois le coefficient de silhouette. Nous avons également déterminé le nombre d'itérations atteint jusqu'à convergence.
Les tableaux suivants résument les résultats que nous avons obtenus.

	Execution 1	**Execution 2**	**Execution 3**	**Execution 4**
Nombre d'itérations	13	5	12	14
Score de silhouette	0.382333670	0.378056722	0.37805672	0.381109880

TABLE 3.2 – Résultats obtenus en utilisant l'initialisation aléatoire des centroids

	Execution 1	**Execution 2**	**Execution 3**	**Execution 4**
Nombre d'itérations	2	2	7	7
Score de silhouette	0.3779169541	0.38110988	0.38110988	0.38110988

TABLE 3.3 – Résultats obtenus en utilisant l'initialisation intelligente des centroids

En observant les tableaux, on peut clairement constater que :

— L'initialisation aléatoire nécessite plus d'itérations pour converger que l'initialisation intelligente, de 2 à 7 itérations pour une initialisation intelligente et de plus de 10 pour une initialisation aléatoire.

— Les deux méthodes présentent presque le même coefficient de silhouette.

Pour mieux visualiser cela, nous allons calculer la moyenne du nombre d'itérations et du coefficient de silhouette pour chaque méthode.

Méthode d'Initialisation	**Aléatoire**	**Intelligente**
Moyenne de nombre d'iterations	11	4
Moyenne de coefficient dee silhoutte	0.379889248	0.3803116487

TABLE 3.4 – Comparaison des Résultats entre Initialisation Aléatoire et Intelligente.

D'après le tableau, on peut bien confirmer que l'initialisation intelligente a une performance supérieure, avec une moyenne de coefficient élevée et un nombre d'itérations faible par rapport à l'aléatoire.

Conclusion :

— La méthode que nous allons choisir, c'est l'initialisation intelligente.

— Le nombre maximal d'itérations est de 10.

C. Déterminer le nombre d'initialisations :

Dans le processus de fixation du nombre d'initialisations, nous avons fixer les paramètres que nous avions expérimentés précédemment : nombre de clusters = 2, méthode d'initialisation intelligente avec un maximum d'itérations égal à 10. Nous allons maintenant faire varier n_init pour obtenir le graphe suivant :

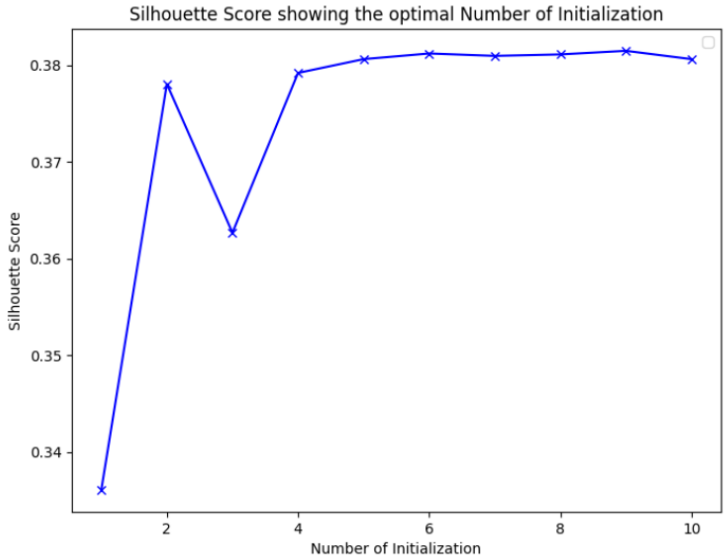


FIGURE 3.6 – Évolution du Coefficient de Silhouette en Fonction du nombre d'initialisation

Analyse et observations : D'après le graphe, on observe que :

— Pour n_init égal à 1, le coefficient de silhouette atteint une valeur minimale avant de commencer à augmenter.

— Pour n_init = 3, le coefficient de silhouette diminue, car il pourrait tomber sur un optimum local.

— Pour n_init supérieur à 4, on n'observe pas de changement significatif dans le coefficient de silhouette.

Conclusion : Pour éviter l'optimum local, nous effectuons plusieurs initialisations. Alors, un n_init d'environ 5 à 10 pourrait être un meilleur choix. Dans notre cas, nous optons pour 5 initialisations.

Evaluations : Une fois que nous avons fixé nos paramètres, les clusters que nous avons obtenus sont :

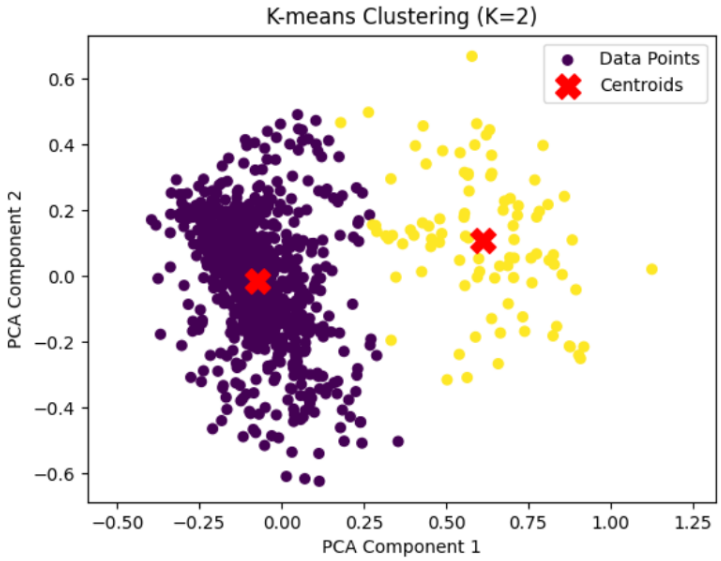


FIGURE 3.7 – Clusters finaux obtenus par l'algorithme K-means après l'ajustement des paramètres

L'évaluation finale de notre modèle :

— **Coefficient de silhouette** égal à : 0.3811098802176154

— **intra-cluster** égale à : 347.02860678240035

— **inter-cluster** égale à : 0.7016258781059354

En conclusion de nos expérimentations avec l'algorithme K-Means, les résultats obtenus ont fourni une amélioration significative de la qualité du clustering. En fixant les paramètres, nous avons observé une meilleure séparation des clusters dans nos données, soutenue par des métriques de performance.

3.3 DBSCAN

3.3.1 Principe

Le DBSCAN est un algorithme de clustering qui crée des groupes de points dans un espace en fonction de leur densité relative. Il identifie les régions denses de points comme des clusters, tout en isolant les points dispersés en tant que bruit.

3.3.2 Pseudo-code

Input : dataset , eps , minPts
Output: clusters

```
for chaque point P on dataset do
    if P est déjà visité then
        Passer au point suivant
    end
    Marquer P comme visité;
    Trouvez les voisins de P dans un rayon epsilon.;
    if le nombre de voisins de P est inférieur à minPts then
        Marquez P comme un point de bruit.
    end
    else
        Former clutser;
        for chaque voisin Q in Voision du P do
            if Q n'est pas visité then
                Marquer Q comme visité;
                if le nombre de voisins de Q est supperieur à minPts then
                    Ajouter voisions de Q a voisins de P
                end
                Assigner Q au cluster du celle de P
            end
        end
    end
end

return Clustres;
```

Algorithm 4: Pseudo code de l'Algorithme DBSCAN

3.3.3 Application de l'Algorithme DBSCAN sur dataset 1

Nous avons implémenté l'algorithme DBSCAN qui prend les paramètres suivants :

— eps : c'est la distance maximale entre deux points pour qu'ils soient considérés comme voisins.

— min_samples : Nombre minimum de points pour former un cluster.

— metric : représente le type de distance utilisé : Euclidienne, Manhattan, cosinus, etc

Nous avons appliqué l'algorithme sur notre dataset1 avec epsilon (eps) égal à 0.3 et un nombre minimum de points (min_samples) égal à 3. Les clusters obtenus sont présentés dans la figure suivante.

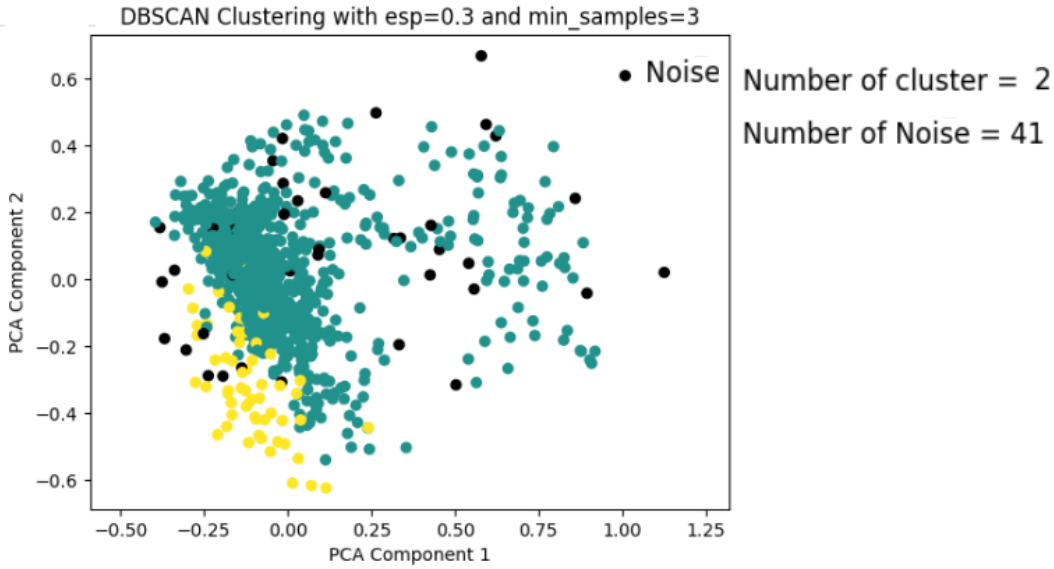


FIGURE 3.8 – Visualisation des Clusters Obtenu avec DBSCAN

Chaque point dans la figure représente une instance du dataset, les couleurs verte et jaune représentent les clusters formés, tandis que les points en noir représentent les valeurs aberrantes (outliers).

3.3.4 Exemple de déroulement de l'Algorithme

Dans cette section, nous allons décrire le déroulement de notre algorithme DBSCAN :

— **Étape 1 : Tous les points sont du bruit (Noise)**

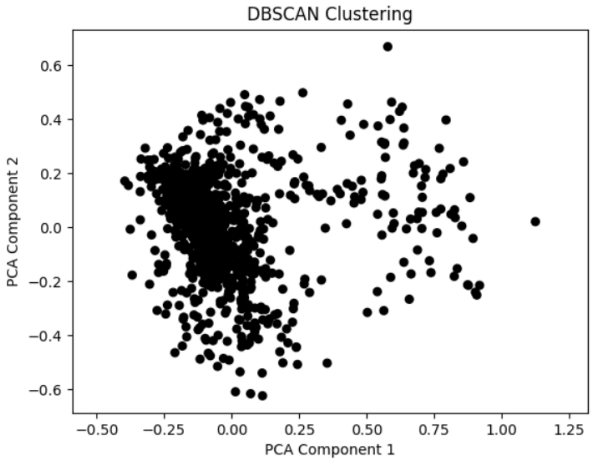


FIGURE 3.9 – Tous les points sont du bruit - Étape 1 de l'algorithme DBSCAN

— **Étape 2 : Sélectionnez un point au hasard et trouvez ses voisins**

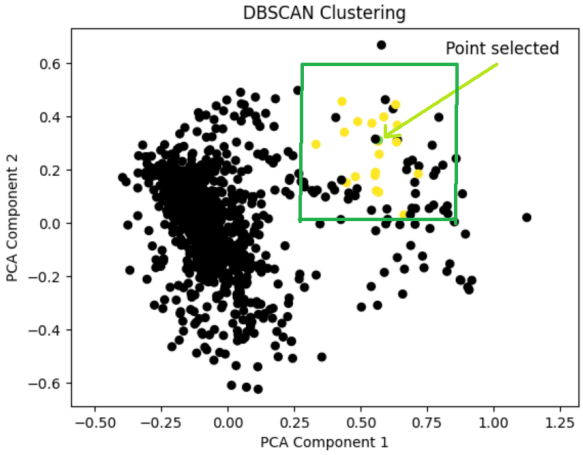


FIGURE 3.10 – Identification des voisins d'un point - Étape 2 de l'algorithme DBSCAN

— **Étape 3 : Formation des Clusters - Étape 3 de l'algorithme DBSCAN**
Dans cette étape, nous marquons P comme visité et vérifions le nombre de voisins de P. Si ce nombre est supérieur ou égal à min_samples, alors nous formons un cluster et ajoutons tous ses voisins à ce cluster. Ensuite, nous itérons sur ces voisins. Sinon, nous passons à un autre point.
Dans notre cas, le nombre de voisins de p est de 20, égal à min_samples, le cluster est formé. Ensuite, nous procédons à la suite du processus :

1. Sélectionnez un point voisin Q non marqué de P, puis calculez les voisins de ce point.

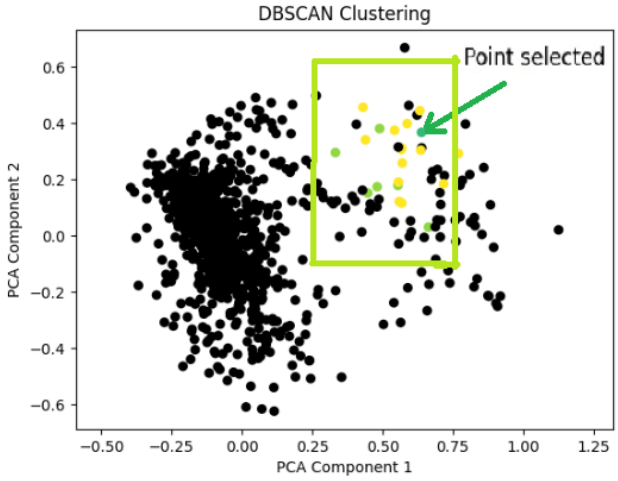


FIGURE 3.11 – Voisins d'un voisin d'un point - Étape 3.1 de l'algorithme DBSCAN

Nous avons marqué Q comme visité et vérifié si le nombre de points est supérieur ou égal à min_samples pour les ajouter aux voisins de P. Sinon, nous passons à un autre voisin. Et ainsi de suite jusqu'à la fin du traitement de tous les voisins de P, formant ainsi le premier cluster.

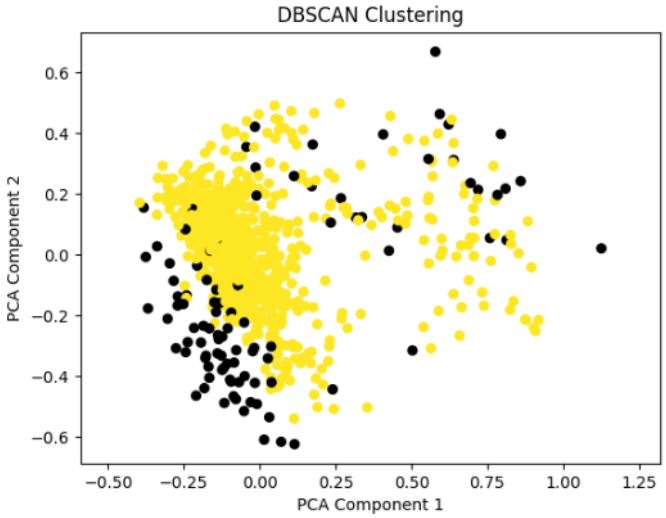


FIGURE 3.12 – Fin de la Formation du 1er Cluster - Étape 3.1 de l'algorithme DBSCAN

2. Si aucun voisin n'est resté, nous sélectionnons un autre point au hasard dans notre dataset non marqué et réitérons les étapes 2 et 3, jusqu'à ce que tous les points soient marqués.

3.3.5 Évaluation du model

Nous avons évalué notre modèle en utilisant les mêmes métriques que celles employées dans l'algorithme K-means : le coefficient de silhouette, qui varie entre -1 et 1, et les mesures inter-cluster et intra-cluster. les reulstats obenus sont les suivantes :

- **Coefficient de silhouette** égal à : 0.3289547900536501
- **intra-cluster** égale à : 329.19400488941136
- **inter-cluster** égale à : 0.6591503136798922

Afin d'optimiser les performances de ce modèle, nous explorerons dans la section suivante l'expérimentation de l'algorithme

3.3.6 Expérimentation du DBSCAN

Nous suivons la même approche que celle utilisée pour l'algorithme K-means. Nous réalisons des expérimentations en utilisant nos données afin d'ajuster les deux paramètres de DBSCAN, à savoir eps et min_samples.

NB : Dans ces expérimentations, nous avons également utilisé la distance euclidienne.

A. Expérimentations sur le paramètre eps :

Dans le processus de fixation du paramètre eps, nous avons fixé min_samples à 4, puis nous allons modifier les valeurs d'eps et évaluer le modèle pour chaque valeur afin de tracer la courbe. La figure suivante illustre la variation du coefficient de silhouette en fonction du paramètre eps :

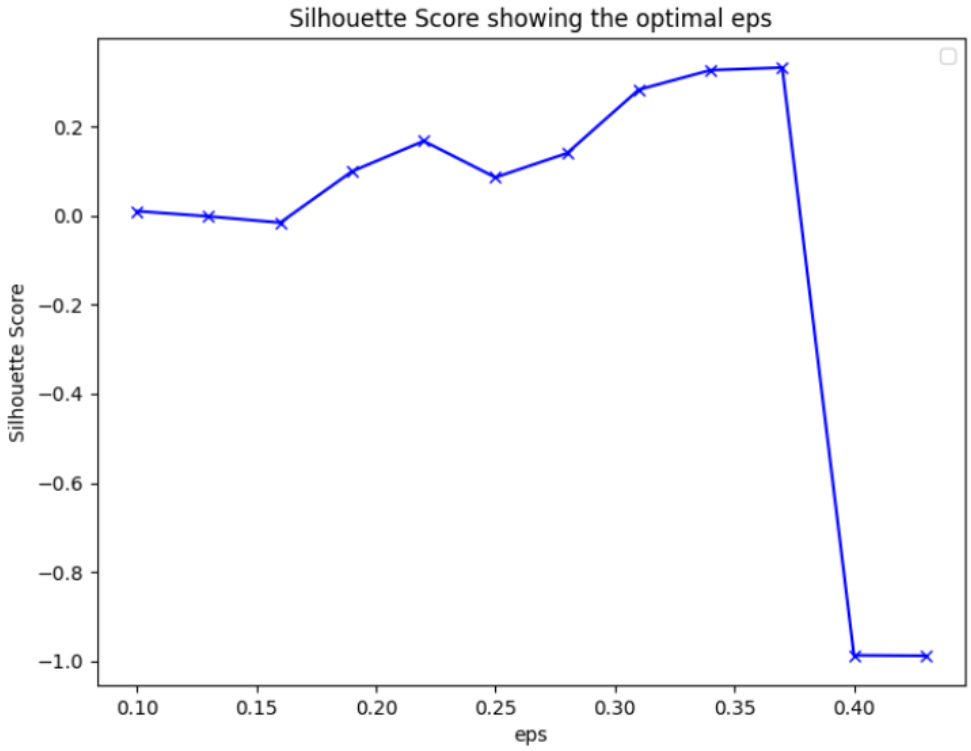


FIGURE 3.13 – Variation du Coefficient de Silhouette en Fonction du Paramètre eps

D'après le graphe, on constate qu'à chaque augmentation du paramètre eps, le coefficient de silhouette augmente, indiquant une amélioration du modèle jusqu'à atteindre son maximum à 0.38. Ensuite, on observe un début de dégradation des performances.

Conclusion : Le meilleur choix pour le paramètre eps est 0.38.

B. Expérimentations sur le paramètre min_samples :

Après avoir trouvé le meilleur choix pour le paramètre eps, nous allons faire de même pour min_samples. Nous allons fixer eps à 0.38 et faire varier le paramètre min_samples.
La figure suivante illustre les variations du coefficient de silhouette en fonction du paramètre min_samples.

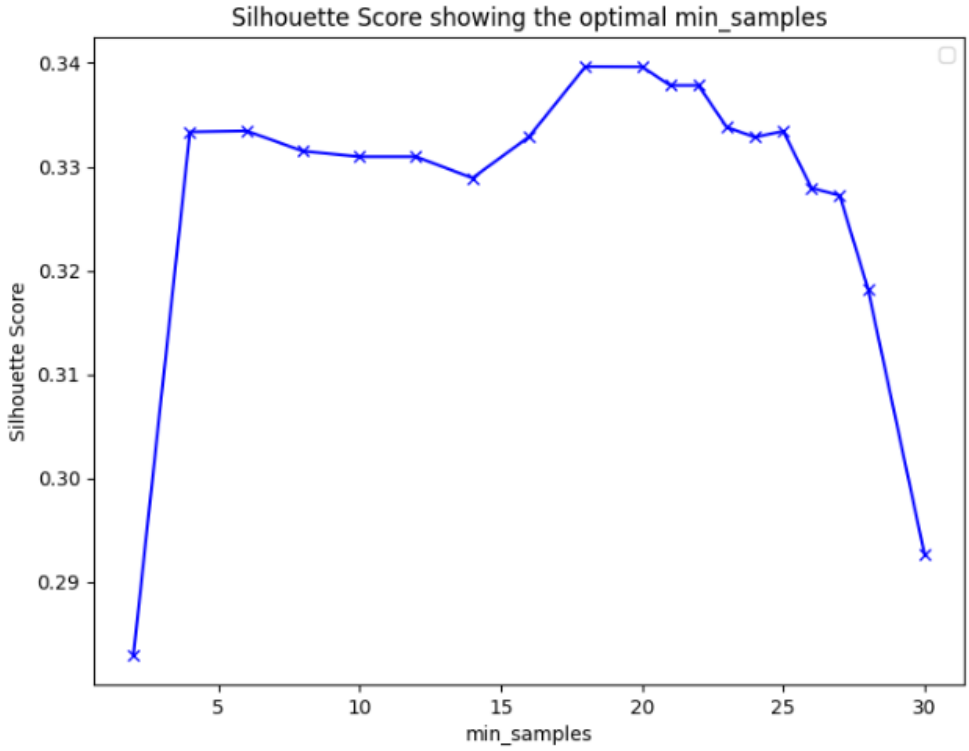


FIGURE 3.14 – Variation du Coefficient de Silhouette en Fonction du Paramètre min_samples

En observant le graphe, nous constatons que lorsque min_samples est égal à 20, le coefficient de silhouette atteint son maximum, ce qui indique que c'est la meilleure valeur.

Conclusion : min_samples = 20 est le choix optimal.

Evaluations : Une fois que nous avons fixé nos paramètres, les clusters que nous avons obtenus sont :

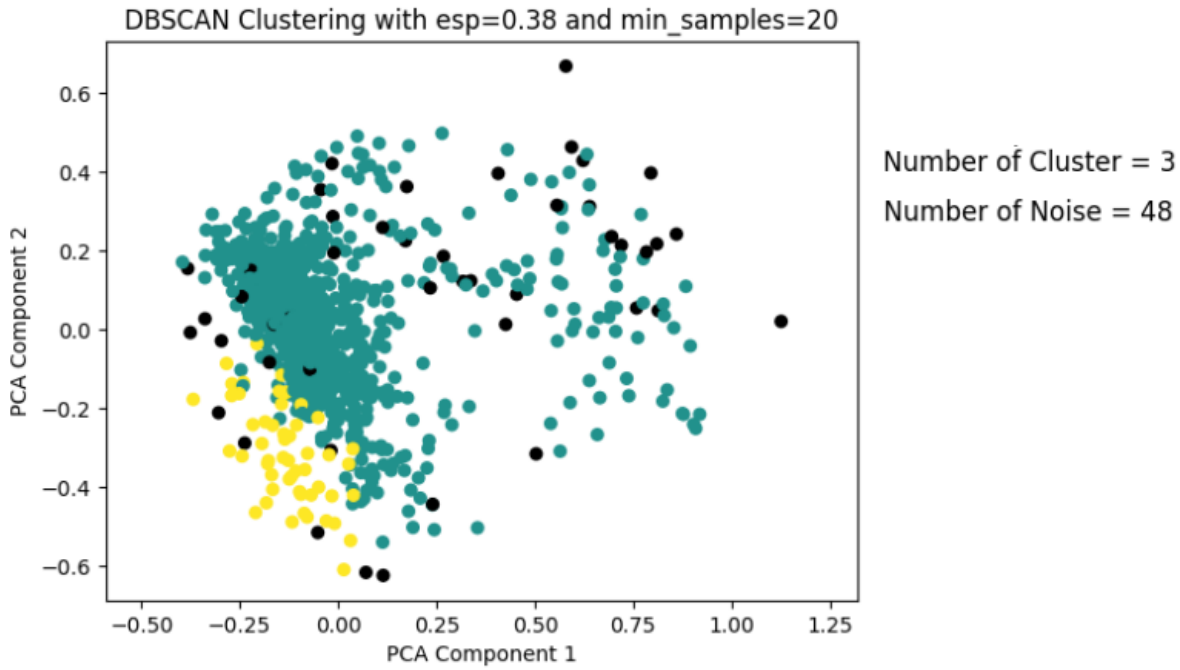


FIGURE 3.15 – Clusters finaux obtenus par l'algorithme DBSCAN après l'ajustement des paramètres

L'évaluation finale de notre modèle :

- **Coefficient de silhouette** égal à : 0.33962966232188185
- **intra-cluster** égale à : 321.8419770283451
- **inter-cluster** égale à : 0.6797049886180755

En conclusion de nos expérimentations avec l'algorithme DBSCAN, les résultats obtenus ont démontré une amélioration de la qualité du clustering, soutenue par des métriques de performance.

3.4 Comparaison des Modèles

Après l'implémentation et l'expérimentation des algorithmes K-means et DBSCAN, afin de faire la comparaison, nous avons présenté les clusters formés par chacun des algorithmes :

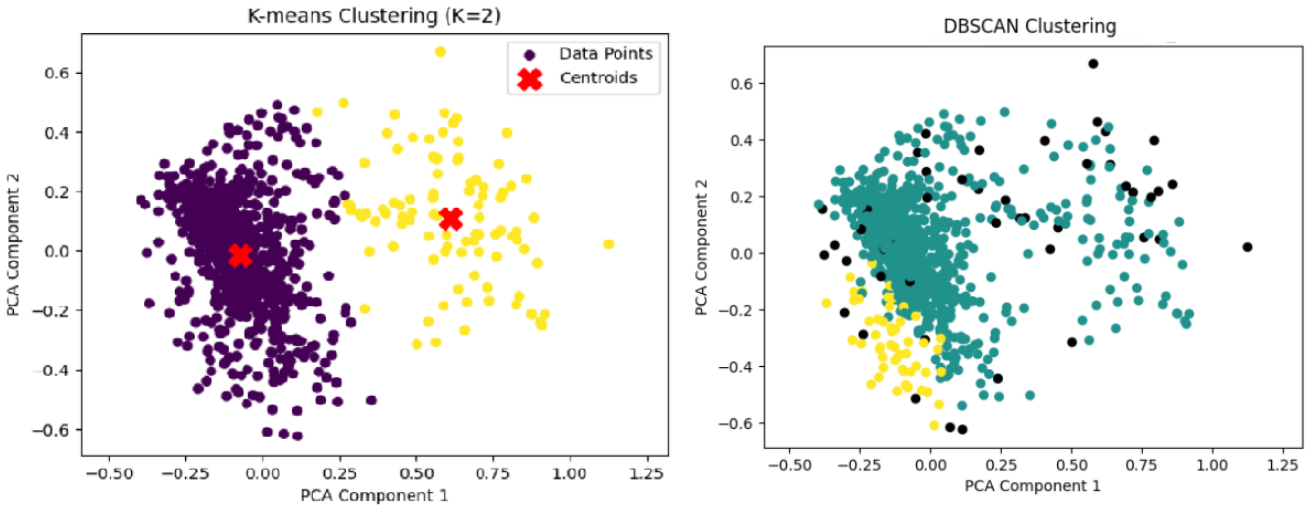


FIGURE 3.16 – Comparaison visuelle des Clusters entre K-means et DBSCAN

Visuellement, il est clair que les clusters formés par l'algorithme K-means sont meilleurs que ceux formés par DBSCAN. Pour confirmer cela, nous trouvons dans le tableau suivant une comparaison des métriques de performance de chaque algorithme :

Metric	k-means	DBSCAN
Coefficient de Silhoutte	0.3811098802176154	0.33962966232188185
Inertia	347.02860678240035	321.8419770283451
Inertie	0.7016258781059354	0.6797049886180755

TABLE 3.5 – Comparaison des Évaluations des Modèles K-means et DBSCAN

Les résultats du tableau confirment de manière significative la performance supérieure de l'algorithme K-means par rapport à DBSCAN sur notre dataset, selon les métriques telles que le coefficient de silhouette et l'inter-cluster. Cela souligne la capacité cohérente de K-means à produire des clusters plus compacts et mieux définis.

Cependant, en ce qui concerne la métrique intra-cluster, DBSCAN peut surpasser K-means en raison de sa capacité à gérer efficacement les outliers. DBSCAN isole les points aberrants, favorisant ainsi des clusters plus compacts et homogènes. Bien que K-means excelle dans certaines métriques, la robustesse de DBSCAN face aux outliers peut conduire à une meilleure métrique intra-cluster dans certaines situations.

3.5 Conclusion

En conclusion, le clustering s'avère être une méthode particulièrement efficace pour l'analyse de jeux de données non étiquetés, permettant la découverte de structures et de similarités sans recours à des annotations préalables.

Conclusion générale

En conclusion, après avoir exploré les modèles supervisés et non supervisés dans ce rapport, on constate que les premiers excellent quand on a des exemples étiquetés, fournissant des réponses précises. En revanche, les modèles non supervisés se démarquent en identifiant des schémas cachés dans des données non marquées.En combinant ces deux approches, on élargit notre compréhension et nos possibilités dans le domaine de l'apprentissage automatique.

Bibliographie

[1] Data Mining Concepts and Techniques Jiawei Han Jian Pei Hanghang Tong 4th Edition

[2] Serie Exercice 6 Algorithmes de Clustering

[3] Data Mining : Concepts and Techniques - Chapter 4 —

[4] Data Mining : Concepts and Techniques — Chapter 5 —

Annexe

ANNEXE A : Brève Définition de la PCA (Analyse en Composantes Principales)

La PCA, ou Analyse en Composantes Principales, est une méthode de réduction de dimensionnalité largement utilisée dans l'analyse de données. Son objectif est de transformer un ensemble de variables interdépendantes en un ensemble de variables non corrélées, appelées composantes principales.

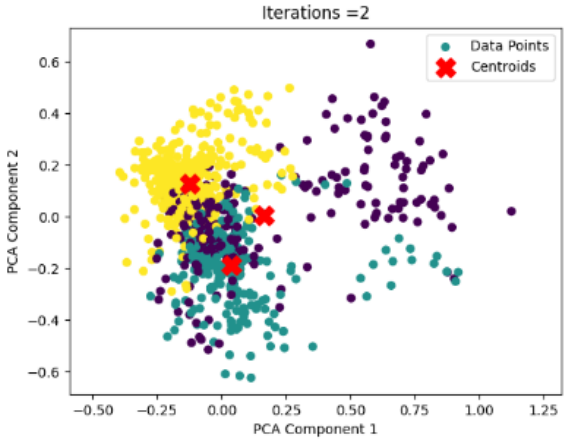
Utilisation dans la Visualisation des Clusters : En contexte de clustering, la PCA est souvent employée pour réduire la complexité des données et faciliter la visualisation des clusters. En projetant les données dans un espace de dimensions réduites, la PCA permet une représentation graphique des clusters.

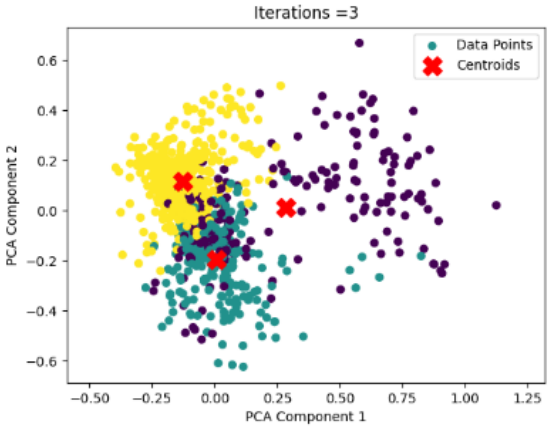
Voici le code qui nous permet de visualiser les clusters à l'aide de PCA.

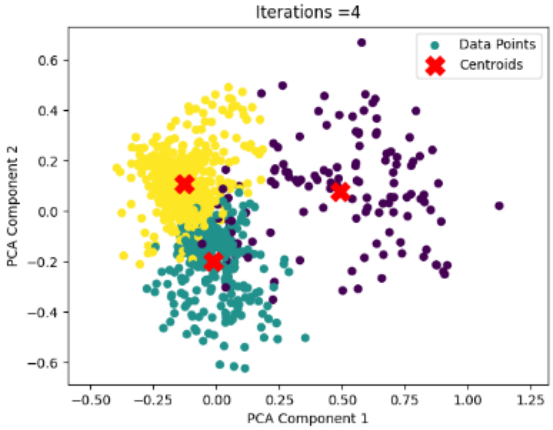
```
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

def visualizeClusters(X, labels, centroids):
    n_clusters = len(np.unique(labels))
    pca = PCA(n_components=2)
    data_pca = pca.fit_transform(X)
    centroids_pca = pca.transform(centroids)
    # Visualize the clusters using the first two principal components
    plt.scatter(data_pca[:, 0], data_pca[:, 1], c=labels, cmap='viridis', s=30, label='Data Points')
    # Add the centroid points in PCA space
    plt.scatter(centroids_pca[:, 0], centroids_pca[:, 1], c='red', marker='X', s=200, label='Centroids')
    plt.title(f'K-means Clustering (K={n_clusters})')
    plt.legend()
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.axis('equal')
    plt.show()
```

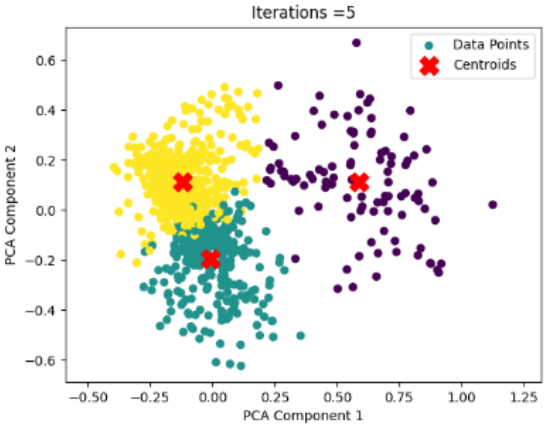
ANNEXE B : Déroulement de K-Means jusqu'à Convergence

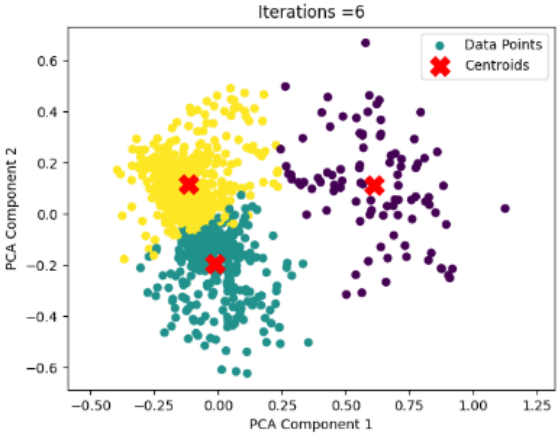
L'annexe suivante présente la suite du déroulement de l'algorithme K-Means en effectuant des mises à jour des centroïdes et des clusters jusqu'à la fin de l'algorithme

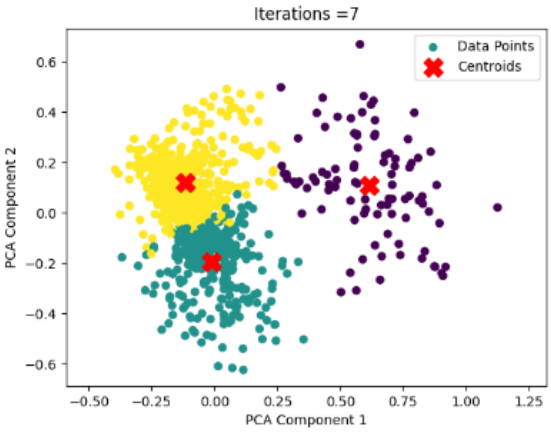












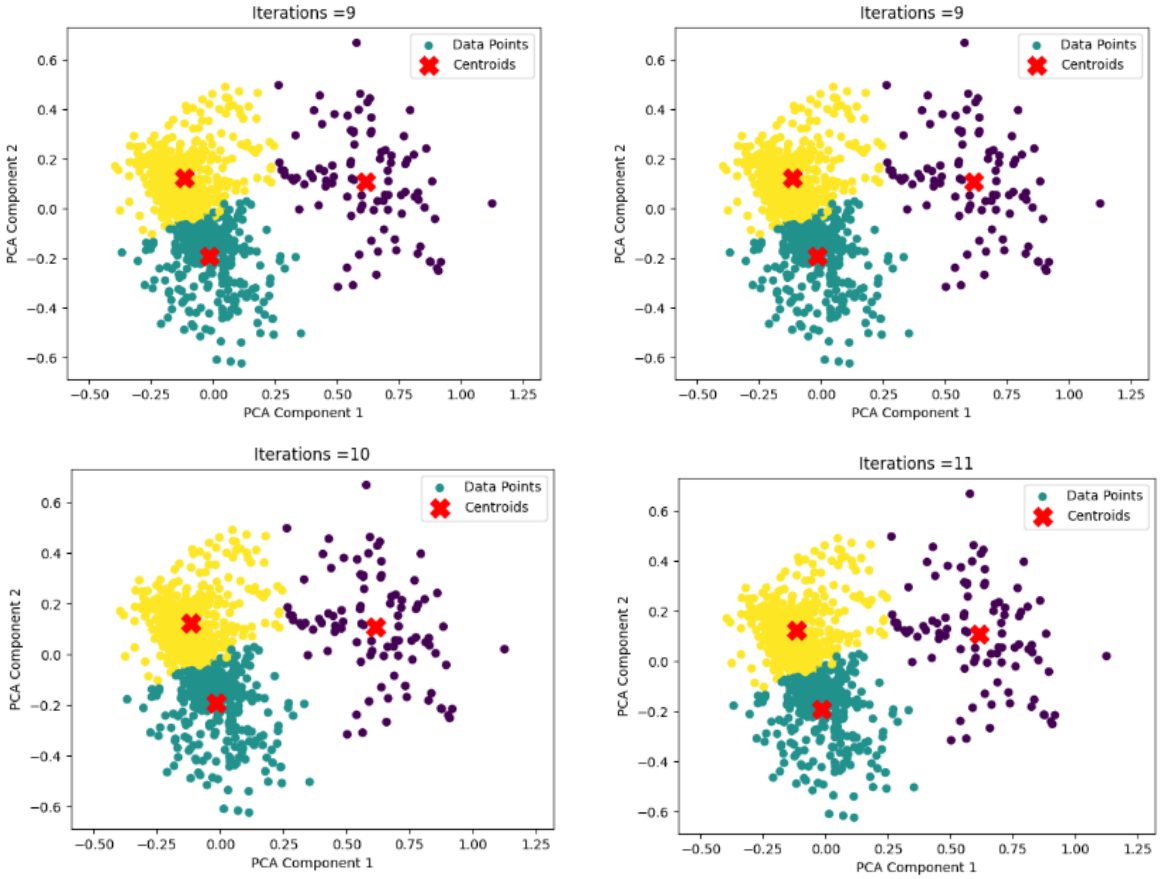


FIGURE 3.17 – Évolution de l'algorithme K-Means jusqu'à convergence

Arrêtez l'algorithme avec max_iterations = 11 car il a atteint une condition d'arrêt (les centroïdes ne changent pas).

ANNEXE C : Analyse Visuelle des Clusters en Fonction de K

La figure suivante montre l'évolution des clusters en fonction du nombre de clusters choisi :

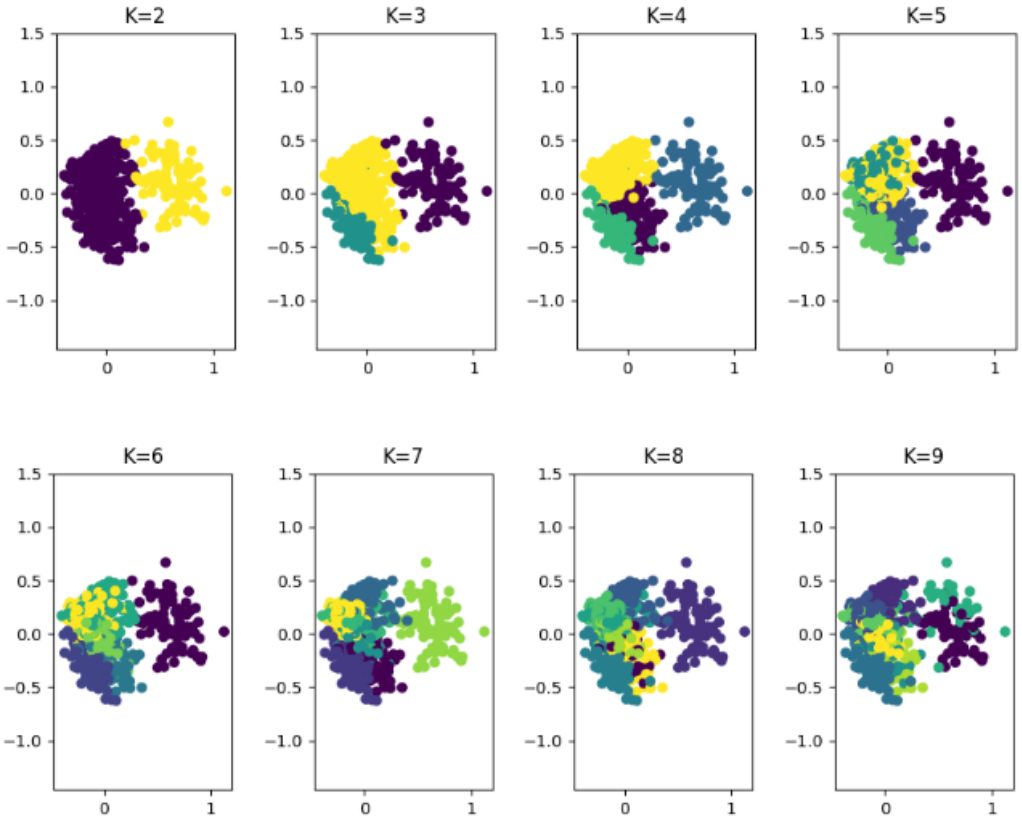


FIGURE 3.18 – Visualisation des Clusters en Fonction du Nombre de Clusters (K)