Université des Sciences et de la Technologie
Houari Boumediene Faculté Informatique

TP DATA MINING

Apprentissage supervisé et non supervisé

Étudiants :
HOUACINE Maya
OUCHAR Manel
SII G02

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Précedemment dans la partie 1, nous avons entre autres analyser un dataset sur les types de sols et appliquer plusieurs types de prétraitements dessus.

Maintenant que nos donnees sont pretes nous allons implementer 3 methode de classification supervisés à savoir KNN Decision Trees ainsi que Random Forest, pour ensuite les comparer entre elles à l'aides des mesures de performances connus tel que la precision , rappel .. Ensuite dans un second temps, nous allons cette fois developpé 2 methodes de clustering Kmeans et DBSCAN et les comparés egalement apres experimentations

Des exemples et deroulements seront presentés pour ces algorithmes afin de bien visualiser le fonctionnement et les resultat.

Chapitre 1

Description et prétraitement des données

1.1 Description du dataset

Le dataset que nous explorons dans cette première partie du projet, comprend **14** caractéristiques concernant le sol, notamment des éléments nutritifs tels que l'azote et le potassium, des indicateurs de santé comme le pH et la matière organique, ainsi que des métaux essentiels.

Ces données fournissent une vue globale sur les propriétés des echantillons de **884** sols (instances), ainsi que leurs fertilités respectives comprises entre 0 et 2, offrant une base pour des analyses approfondies en agriculture et en gestion environnementale.

1.2 Analyse des caractéristiques des attributs

1.2.1 Calcul des mesures de tendance centrale et déduction Des symétries

En comparant entre les valeurs de moyenne, mode et médiane de chaque attribut, nous pouvons déduire la nature de leurs distributions.

PH, EC, Zn et OC se distinguent par des distributions symetriques/légérement asymétrique, car leurs mode = médiane = moyenne. Tandis que les attributs K, Mn et B ont

une Moyenne > Médiane >Mode ce qui correspond à une asymétrie positive. Ils indiquent que les valeurs de ses caractéristiques sont généralement faibles dans les sols, mais cela peut aussi être dû à la présence de valeurs aberrantes. Quant au reste des attributs, ils sont soit asymétriques simples soit suivent une distribution asymétriques négative comme Fertility

En ce qui concerne les écart-types, elles varient entre 0.14 et 129.03 indiquant que certains attributs dénotent d'une grande variabilité dans les données alors que d'autres sont denses dans leurs distributions.

À noter que tous les attributs sont unimodale à l'exception d'EC et S qui sont bimodale

Nom	**Mean**	**Mode**	**Min**	**Q1**	**Q2**	**Q3**	**Max**	**Ecart type**
N	246.997	[207.0]	6.0	201.0	257.0	307.0	383.0	77.315
P	14.555	[8.3]	2.9	6.8	8.1	10.7	125.0	21.918
K	501.338	[444.0]	11.0	412.0	475.0	581.0	1560.0	129.031
Ph	7.511	[7.5]	0.9	7.35	7.5	7.63	11.15	0.4643
EC	0.5439	[0.53, 0.62]	0.1	0.43	0.55	0.64	0.95	0.1412
OC	0.617	[0.88]	0.1	0.38	0.59	0.78	24.0	0.84064
S	7.545	[4.22, 5.13]	0.64	4.7	6.64	8.75	31.0	4.415
Zn	0.468	[0.28]	0.07	0.28	0.36	0.47	42.0	1.887
Fe	4.126	[6.32]	0.21	2.05	3.56	6.31	44.0	3.10
Cu	0.952	[1.25]	0.09	0.63	0.93	1.25	3.02	0.52
Mn	8.6536	[7.54]	0.11	6.21	8.34	11.44	31.0	4.298
B	0.593	[0.34]	0.06	0.27	0.41	0.61	2.82	0.5744
OM	1.0637	[1.51]	0.17	0.65	1.0148	1.34	41.28	1.445
fertility	0.592	[1.0]	0.0	0.0	1.0	1.0	2.0	0.578

TABLE 1.1 – Tendances centrales, quartiles et ecart type des attributs du Dataset1

1.2.2 Boîtes à moustache et données aberrantes.

la figure suivante englobante les boxplots des 14 attributs de notre dataset sur une échelle logarithmique afin de les comparer entre eux met en évidence les faits suivants :

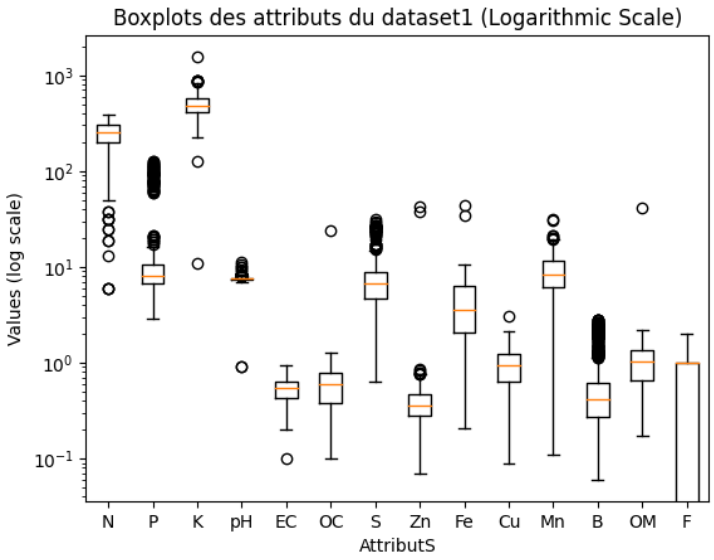

FIGURE 1.1 – Boxplots des 14 attributs du dataset1 avec échelle logarithmique

- Les caractéristiques de nos sols appartiennent à des échelles distinctes allant de 10∧(-1) à 10∧3
- Nous pouvons visuellement constater les différentes distributions des attributs et confirmer qu'elles concordent avec les observations et deductions du tableau precedant
- Pour ce qui est des valeurs aberrantes, nous remarquons que le phosphore, le Bore, le Souffre et le PH sont sensible aux Outliers de par leurs nombres conséquents. À l'inverse OC, EC, le Cuivre et la matière organique n'en contiennent que très peu.

1.2.3 Histogrammes et distribution des données

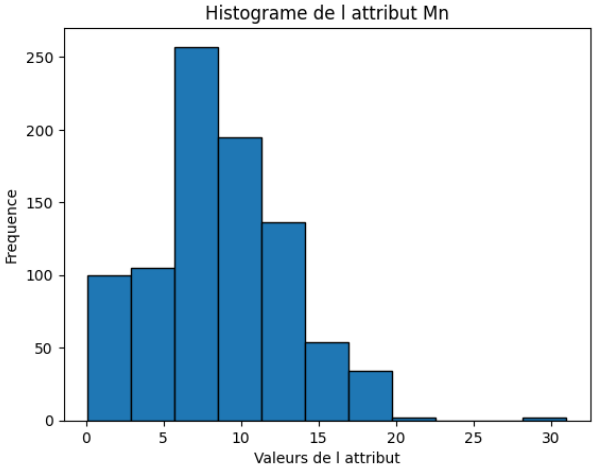

FIGURE 1.2 – Histogramme de l'attribut Mn (Manganèse) avant remplacement des valeurs abberantes

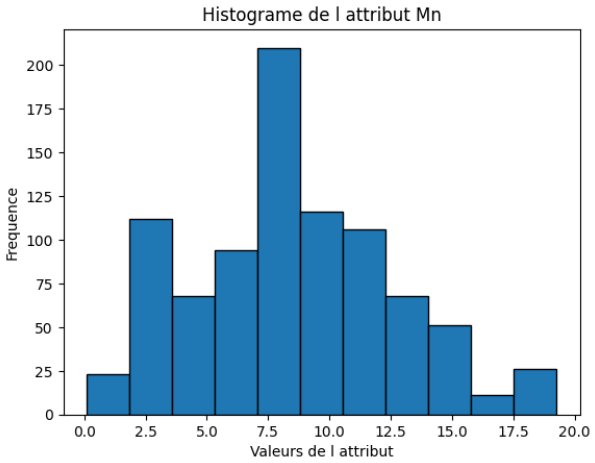

FIGURE 1.3 – Histogramme de l'attribut Mn (Manganèse) après remplacement des valeurs abberantes

L'observation initiale de l'histogramme de fréquence du manganèse (Mn) montre l'existance de valeurs abberantes ,causant une classe vide et affectant la représentation globale de la distribution (Figure 1.2). Afin d'assurer une interprétation plus fiable, nous avons traité ces valeurs aberrantes et généré un nouvel histogramme dépourvu de ces anomalies (Figure 1.3).

Ce deuxième histogramme montre une distribution legerement asymétrique positive, car la majorité des échantillons ont des concentrations modérées en manganèse. Ceci met en évidence la tendance des sols, à ne pas accumuler beaucoup de manganèse. Cette petite inclination peut être due entre autres à la nature géologique du sol qui limite l'absorption du manganèse.

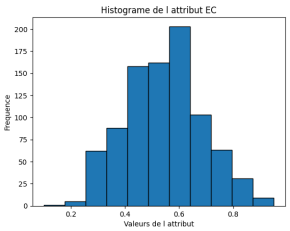

FIGURE 1.4 – Histogramme de l'attribut EC

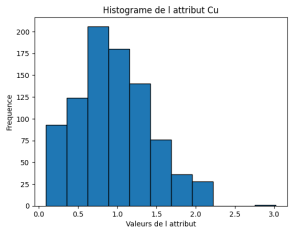

FIGURE 1.5 – Histogramme de l'attribut Cu

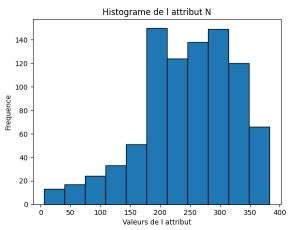

FIGURE 1.6 – Histogramme de l'attribut N

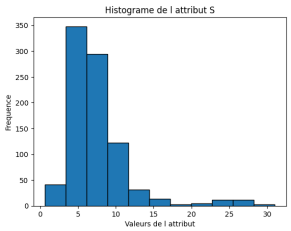

FIGURE 1.7 – Histogramme de l'attribut S .

L'histogramme de la conductivité électrique (EC) des sols, dévoile une distribution symétrique, car la plupart des sols ont des concentrations moyennes. Les fréquences les

plus élevées se situent entre 0,2 et 0,6, ce qui indique qu'il y a généralement une tendance vers des niveaux modérés comme le confirme sa valeur faible d'ecart type.

L'observation des l'histogrammes représentant la fréquence du soufre (S) et "Cuivre" (Cu) sur les sols, révèle une distribution asymétrique. Cette asymétrie est positive car la majorité des données sont concentrées à gauche Les raisons de cette tendance peuvent être multiples, allant des propriétés géologiques du sol aux pratiques agricoles spécifiques et meme l'activité humaine et d'autres facteurs.

Tandis que l'histogramme indiquant la fréquence de l'Azote (N) sur les sols, révèle une distribution asymétrique négative etant donnée que ses valeurs on tendance à etre grandes et effectivement ce qui etait moins facile à deduire à partir du tableau des tendances centrales.

1.2.4 Diagrammes de dispersion des données

Les diagrammes de dispersions nous fournissent un aperçu sur les corrélations existantes entre les attributs ce qui est primordial, que se soit lors du prétraitement dans la réduction de dimensions, la découverte de nouvelles relations implicite entre attributs et l'amélioration des modèles utilisé sur la dataset par la suite

Étant donné qu'il y a 91 possibles scatter plots, nous avons calculé la matrice de corrélation afin de choisir judicieusement nos graphes.

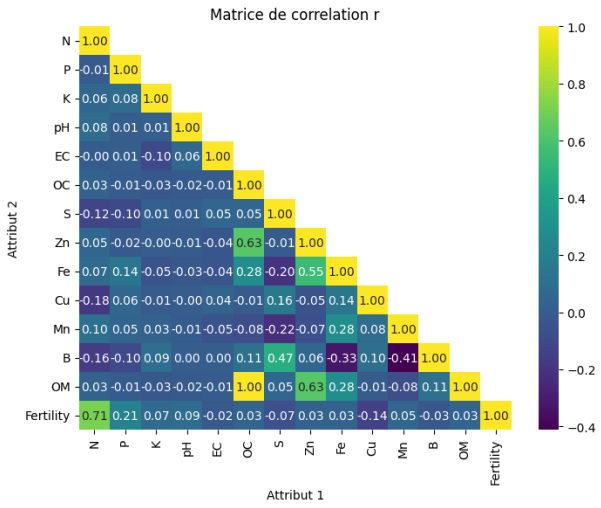

FIGURE 1.8 – Matrice de corrélation des attributs du Dataset 1.

Nous pouvons observer à partir de cette matrice que la plupart des attributs sont indépendants, et que ceux corrélés négativement relativement fortement sont presque inexistant. Cependant, il y a plusieurs propriétés du sol qui sont corrélées positivement avec EC et OM atteignant une valeur de 1.0 (Selon Pearson corrélation).

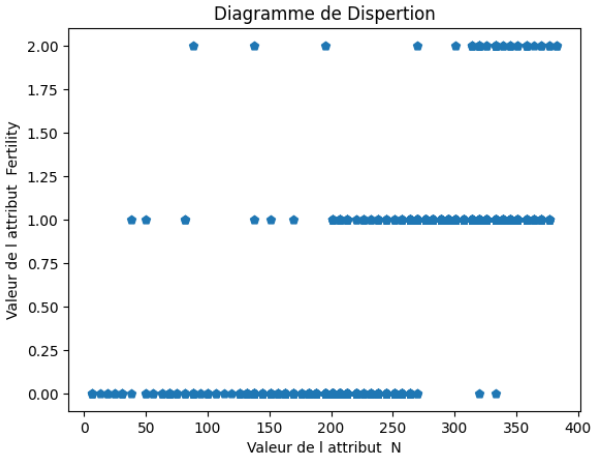

FIGURE 1.9 – Scatter plot des deux attributs N et Fertility.

Une autre paire correlée positivement est la paire Fertility et N, comme l'attribut Fertility est de type categorique les points s'accumulent a 0,1 et 2 seulement. Donc le Nitrogene est un bon indicateur de la fertilité du sol.

Pour ce qui est des graphes concerant la paire d'attribut OM et OC apres avoir remplacer les valeurs abberantes par regression lineaire, nous constatons qu'elle correspond à la fonction x=y indiquant une claire correlation positive entre les 2 (r=1). Ceci s'explique par le fait que le carbone organique constitue 58% de la matiere organique.

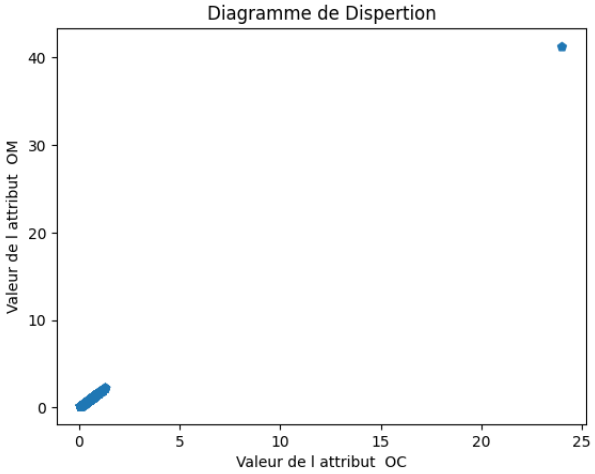

FIGURE 1.10 – Scatter plot des deux attributs OC et OM avant traitement des valeurs aberrantes.

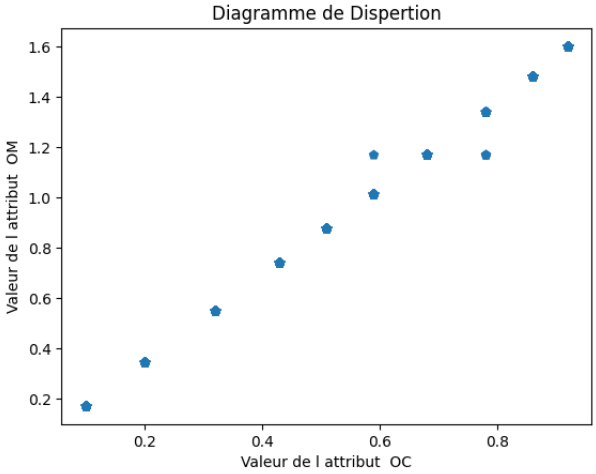

FIGURE 1.11 – Scatter plot des deux attributs OC et OM après traitement des valeurs aberrantes.

Malgré le remplacement des valeurs aberrantes, ainsi que l'obtension de r<0 pour la paire (B,Mn) nous ne voyons pas de corrélations négative calire entre les 2 caractéristiques, cela est attribuable au fait que r=-0.4 est plus proche de 0 (aucune corrélation) que de -1

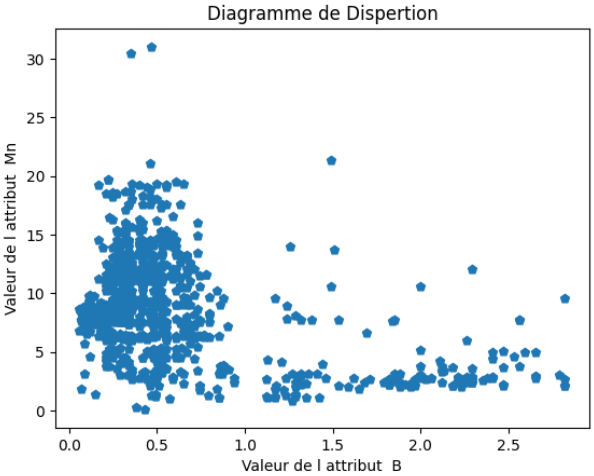

FIGURE 1.12 – Scatter plot des deux attributs K et Zn avant traitement des valeurs aberrantes.

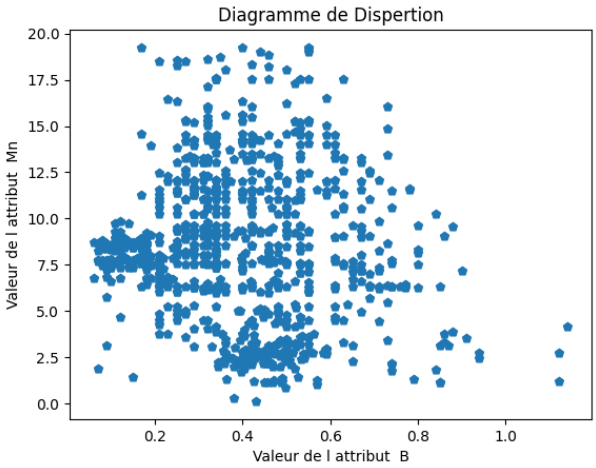

FIGURE 1.13 – Scatter plot des deux attributs K et Zn après traitement des valeurs aberrantes.

Pour ce qui est des 2 paires (Cu,Ph) et (K,Zn) leurs points sont dispersées de manière aléatoire prouvant l 'indépendance entre ces propriétés.

1.3 Prétraitement

1.3.1 Traitement des valeurs manquantes et aberrantes :

1.3.1.1 Choix de la méthode de remplacement des valeurs manquantes.

Après avoir analysé nos données, il est temps de nettoyer notre dataset. Pour les valeurs manquantes, nous en avons détecté 4 (Qui sont " " , " ?“ , " ?" , "NA") en utilisant une expression régulière. Cela en retournant les valeurs qui ne respectent pas le pattern correspondant à des valeurs numériques (étant donné que tous nos attributs sont de ce type)

Pour le remplacement 2 techniques fut implémenter

1. **Remplacement par Mode** : Cette méthode consiste à remplacer chaque valeur manquante par la valeur la plus fréquemment observée de sa colonne. Nous avons identifié le mode en comptabilisant la fréquence de chaque valeur non manquante dans la colonne respective. En cas de plusieurs modes, nous avons choisi aléatoirement l'une de ces valeurs.

2. **Remplacement par Moyenne** : Adaptée aux données numériques, cette approche remplace les valeurs manquantes par la moyenne des valeurs non manquantes de la même classe (valeur de Fertility) assurant un remplacement contextuel des valeurs manquantes. Cette opération préserve la cohérence des données.

1.3.1.2 Choix de la méthode de traitement des valeurs aberrantes

Les valeurs aberrantes quant à elles sont détectées si elles verifient la condition suivante :

$$\text{si } x > Q3 + 1.5 \times IQR \text{ ou } x < Q1 - 1.5 \times IQR]$$

L'IQR est la différence entre le troisième quartile (Q3) et le premier quartile (Q1) des données.

$$IQR = Q3 - Q1$$

— **Remplacement par regression lineaire** : Nous avons créer un modèle de régression linéaire qui nous permet d'estimer la valeur aberrante trouvée en utilisant les autres variables du groupe. Cette méthode permet un remplacement plus contextuel en tenant compte des relations entre la variable cible et les autres variables.

— **Remplacement par discretisation par frequence** : Apres avoir etablit les intervalles des classes de l'attribut contenant des valeurs abberantes, Nous avons choisis de remplacer la valeur aberrante trouvée par la mediane de l'intervalle ou il se trouve. Cette méthode de discrétisation est insensible au outliers contrairement à Equal Width.

— **Remplacement par winorisation modifiée** :Une nouvelle methode de remplacement de valeurs abberantes a été ajoutée afin d'etre utilisé plus tard pour kmeans (plus de details la dessus dans la partie k-means) elle remplace les outliers depassant un seuil (quantille choisis avec alpha) et celles inferieur au seuil symetrique par ces seuils, respectivement. Quant aux autres outliers ils restent inchangé mais nous avons choisis de les remplacer par la mediane (meilleur que la moyenne car fonctionne meme pour les attribut dont la distribution n'est pas normale). voici ci-dessous la formule correspondante à la winorisation :

$$\text{Winorized xi} = \begin{cases} \text{Lower threshold} & \text{si xi} < \text{quantille(alpha)} \\ \text{mediane} & \text{si quantille(alpha)} \leq \text{xi} \leq \text{quantille(1-alpha)} \\ \text{Upper threshold} & \text{si xi} > \text{quantille(1-alpha)} \end{cases}$$

1.3.2 Réduction des données horizontales / verticales.

Réduire les dimensions de notre dataset a une influence considérablement sur les performances des modèles d'apprentissage automatique de plusieurs manières : en réduisant le temps d'exécution et la mémoire consommée, améliorant les résultats en prévenant l'overfitting et en réduisant les redondances.

Pour ce qui est de la réduction horizontale, nous avons éliminé les lignes redondantes ainsi, 2 instances ont été supprimées. Concernant la réduction verticale, nous avons calculé la corrélation de Pearson ente chaque paire d'attributs et parmi celles dont la valeur depassent le seuil (paramètre empirique à expérimenter en partie 2), l'une des deux est retirée.

1.3.3 Normalisation des données :

La dernière étape de notre prétraitement consiste à normaliser nos données. Elle implique de ramener à une échelle commune tous les attributs de notre dataset afin de s'assurer qu'ils contribuent de manière équitable lors de l'exécution des algorithmes de clustering. Sans cela, certaines domineront plus que d'autres simplement en raison de la disparité entre ses valeurs et ceux des autres attributs.

Nous avons appliqué les 2 méthodes de normalisation suivantes :

1.3.3.1 Méthode Min-Max

Nous permet de normaliser les données au sein d'un intervalle dont les bornes sont choisies en entrée, voici sa formule :

$$Valeur_{(i,new)} = \frac{Valeur_{(i,old)} - Valeur_{(min,old)}}{Valeur_{(max,old)} - Valeur_{(min,old)}} \left(Valeur_{(max,new)} - Valeur_{(min,new)}\right) + Valeur_{(min,new)} \quad (1.1)$$

1.3.3.2 Méthode z-score.

Elle quantifie à quel point chaque valeur est éloignée de la moyenne de l'attribut courant en termes d'unités d'écart-type comme le montre la formule suivante

$$Valeur_{(i,new)} = \frac{Valeur_{(i,old)} - Valeur_{(mean,old)}}{\sqrt{\frac{1}{N}\sum_{i=1}^{N} \left|Valeur_{(i,old)} - Valeur_{(mean,old)}\right|^2}} \quad (1.2)$$

Chapitre 2

Analyse supervisée

2.1 Algorithmes de classification

Pour assurer la robustesse de nos modèles de classification, il est essentiel de séparer judicieusement notre ensemble de données en données d'apprentissage et de test. Cette étape est essentielle pour évaluer la capacité des modèles à être généralisé sur des données non vues. Nous avons choisi une répartition équilibrée, allouant 80 % des instances par classe pour l'apprentissage et 20 % pour l'évaluation, garantissant ainsi une représentation adéquate de chaque catégorie de fertilité du sol.

2.1.1 KNN

L'algorithme KNN (K plus proches voisins) est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression.[1] Il fonctionne en suivant les étapes suivantes :

1. Entrée : Un nouvel exemple est présenté auquel il faut déterminer la classe ou la valeur.
2. Calcul de la distance : Pour chaque voisin, la distance entre le nouvel exemple et chaque voisin est calculée en utilisant une fonction de distance, généralement la distance euclidienne ou la distance de cosine.
3. Sélection des k voisins : Les k plus proches voisins sont sélectionnés en fonction de la distance calculée.
4. Détermination de la classe : La classe ou la valeur du nouvel exemple est déterminée en fonction des classes ou des valeurs majoritaires parmi les k voisins sélectionnés.

2.1.1.1 Pseudo code de l'algorithme KNN

Algorithm 1 KNN Algorithm

```
procedure KNN(k, methode_distance, xt, yt)
    self.k ← k
    self.methode_distance ← methode_distance
    self.Xtrain ← xt
    self.Ytrain ← yt
    function _PREDICT(Xtest)
        for each instance in self.Xtest do
            dist[i] ← DISTANCE(instance, Xtest, self.methode_distance)
        end for
        ind ← argsort(dist)
        knn ← self.Ytrain[ind[: self.k]]
        Y ← info_gain_method(knn)
        return Y
    end function
end procedure
```

2.1.1.2 Expérimentations en variant les paramètres de KNN

2.1.1.2.a Prétraitement

Étant donné que nous avons implémenté plusieurs méthodes de prétraitement dans la partie 1, nous pouvons à présent choisir les plus adaptées pour nos algorithmes. Les expérimentations sur les différentes méthodes de prétraitement pour KNN, ont révélé que la combinaison la plus efficace était d'utiliser la méthode de remplacement par la moyenne pour traiter les valeurs manquantes car les données du dataset ont une distribution normale, la régression linéaire pour gérer les valeurs aberrantes, et la normalisation avec Min-Max (vmin, vmax). Cette combinaison spécifique a été utilisé pour obtenir tout les résultats de KNN.

2.1.1.2.b Hyperparametres

Pour le modèle KNN, l'ajustement des hyperparamètres nous a montré que le meilleur choix pour le nombre de voisins (k) était 3, optimisant ainsi la précision globale du modèle comme le montre le Tableau ... en annexe. De plus, les métriques de distance euclidienne et de Minkowski ont démontré des performances similaires, et donc nous avons choisi ces valeurs optimisées d'hyperparamètres pour maximiser l'efficacité du modèle KNN dans la tâche de classification.

2.1.1.3 Exemple de déroulement de l'algorithme KNN

Nous avons choisi une instance spécifique de l'ensemble de test du dataset (X_test), représentée par le vecteur [0.01801802, 0.36641221, 0.3627451, 0.33027523, 0.62666667, 0.74576271, 0.11841283, 0.07156673, 0.39303483, 0.1322228, 0.27280741, 0.03066867], pour illustrer le déroulement de l'algorithme KNN.

Étapes du Déroulement de l'Algorithme KNN (voir Figure 2.1) :

1. **Instance de Test** : Nous débutons avec une instance spécifique que nous cherchons à classifier. Pour cela nous avons choisi l'instance caractérisée par le vecteur [0.01801802, 0.36641221, 0.3627451, 0.33027523, 0.62666667, 0.74576271, 0.11841283, 0.07156673, 0.39303483, 0.1322228, 0.27280741, 0.03066867].
2. **Calcul des Distances** : Les distances entre cette instance de test et toutes les autres instances de l'ensemble d'entraînement (X_train) sont calculées à l'aide d'une mesure de distance spécifiée (Euclidienne, Manhattan, etc.). Dans cet exemple, nous avons opté pour la distance Euclidienne. Ces distances sont ensuite triées par ordre croissant à l'aide des indices.
3. **Identification des classes des k Plus Proches Voisins** : Les k instances les plus proches de l'instance de test sont identifiées en fonction des distances calculées. Les classes correspondantes aux k plus proches voisins sont extraites. Dans notre exemple, les classes sont [0.0, 1.0, 0.0] comme le montre la Figure 2.1.
4. **Vote Majoritaire et Classe Prédite** : La classe prédite pour l'instance de test est déterminée par un vote majoritaire parmi les k plus proches voisins. Chaque voisin "vote" pour sa classe respective, et la classe prédominante est attribuée à l'instance de test. Dans cet exemple, la classe prédite est 0, car elle apparaît deux fois parmi les voisins les plus proches comme le montre la Figure 2.1.

```
Test Instance: [0.01801802, 0.36641221, 0.3627451, 0.33027523, 0.62666667, 0.74576271, 0.11841283, 0.07156673, 0.39303483, 0.1322228, 0.27280741, 0.03066867]
Distances: [1.29557887 0.78846168 1.23216653 0.99025482 1.09471399 0.63917327
 1.2475573  0.7328497  0.79497161 1.26087691 0.98585356 1.28694245
 0.83669261 1.19919011 1.0086793  1.23499745 1.17194247 0.51317291
 1.25734381 1.173298   1.29264212 1.03114911 0.91347758 0.88194464
 1.29820398 1.01274877 0.83316811 1.23812598 0.90134974 0.88590727
 1.2109094  1.15203498 1.0843811  1.31992562 0.9632161  0.842769
 1.0070103  1.08139027 1.22529194 0.87961603 1.15191716 1.02281271
 1.0591741  1.10782584 0.8000839  1.45113205 0.83418214 0.92361714
 0.84738279 1.06204056 1.33528569 1.29130917 0.94037112 0.51638823
 0.69551014 1.15269317 0.79493787 1.16897285 0.9730315  1.24909937
 0.97580832 0.91408868 0.93422881 1.01640165 1.24661526 1.05144335
 0.26187891 1.16186393 1.08207478 1.22443075 1.10002017 1.2132994
 1.20791106 0.82309667 0.98297161 1.09210318 0.9107175  0.83481539
 1.18934211 1.0246506  0.95901949 1.17913765 1.20485947 1.04628251
 1.09504671 1.19122941 1.17601166 1.16061087 1.13359632 0.68507365
 0.71649304 0.53972921 0.89948568 0.86817504 0.67982503 1.16357666
 1.05562746 0.89987195 0.90060947 1.0852762  1.10068158 1.08695926
 1.40406279 1.18666418 1.13956413 0.95892688 1.30262892 1.38894647
 0.82676915 1.12812455 1.09845457 1.09141946 1.3957549  0.98804651
 0.83373714 0.59147768 1.20433026 1.24043895 1.32530602 1.0918981
 1.43795682 1.01785667 0.95229382 1.19479155 1.07415961 0.98176158
 1.26190204 1.19711978 1.33990153 1.27454578 1.14625433 0.80088059
 1.3553016  1.12770995 1.21854887 1.31694977 0.97193765 1.19294913
 1.18358584 0.84204046 0.57884773 0.82364598 0.53025074 1.44368265
...
K Nearest Neighbors Indices: [468  66 386]
K Nearest Neighbors: [0. 1. 0.]
Predicted Class: 0.0
```

FIGURE 2.1 – Exemple de déroulement de l'algorithme KNN.

2.1.2 Decision Trees

Les décision trees, sont une méthode d'apprentissage supervisé utilisée pour la classification et la régression. ce modèle fonctionne en créant une structure arborescente où chaque nœud interne représente un attribut, chaque branche représente une valeur possible de cet attribut, et chaque feuille représente une classe ou une valeur. L'arbre de décision divise récursivement l'ensemble de données en sous-ensembles plus petits en fonction des valeurs des attributs, jusqu'à ce que les feuilles représentent de manière homogène la classe. ce modèle fonction de la manière suivante [2] :

1. Entrée : Un nouvel exemple est présenté auquel il faut déterminer la classe ou la valeur.
2. Calcul de la valeur de qualité : Pour chaque attribut, la valeur de qualité (par exemple, la valeur de la classe ou de la valeur) est calculée pour chaque sous-ensemble de données formé par les valeurs possibles de cet attribut.
3. Sélection de l'attribut : L'attribut ayant la valeur de qualité la plus importante est sélectionné.
4. Création d'un nœud : Un nœud est créé pour l'attribut sélectionné, et les données sont divisées en sous-ensembles plus petits en fonction des valeurs de cet attribut.
5. Création d'un nœud : Un nœud est créé pour l'attribut sélectionné, et les données sont divisées en sous-ensembles plus petits en fonction des valeurs de cet attribut.
6. Répétition : Les étapes 2 à 4 sont répétées pour chaque sous-ensemble de données formé par les valeurs possibles de l'attribut sélectionné jusqu'à ce que les feuilles représentent de manière homogène la classe ou la valeur.

Les Decision Trees sont appréciés pour leur facilité d'interprétation et peuvent être combinés à d'autres techniques pour former des ensembles d'arbres, tels que les forêts aléatoires (Random Forest) que nous entamerons juste après.

2.1.2.1 Pseudo code de l'algorithme Decision trees

2.1.2.2 Pseudo code de la fonction de gain d'information

Nous avons mis en œuvre deux méthodes distinctes pour le gain d'information des noeuds, à savoir Gini et l'entropie. La méthode Gini évaluant l'impureté en se basant sur la probabilité de mauvaise classification, tandis que l'entropie quantifie l'incertitude au sein d'un ensemble à travers les probabilités de classe. Ci-dessous leurs implémentations respectives :

2.1.2.3 Expérimentations en variant les paramètres de Decision Trees

2.1.2.3.a Prétraitement

Les expérimentations sur les différentes méthodes de prétraitement pour Decision Trees, ont révélé que la combinaison la plus efficace était d'utiliser la méthode de remplacement par la moyenne pour traiter les valeurs manquantes car les données du dataset ont une distribution normale, la régression linéaire pour gérer les valeurs aberrantes, et la normalisation avec Min-Max (vmin, vmax). Cette combinaison spécifique a été utilisé pour obtenir tout les résultats de Decision trees.

2.1.2.3.b Hyperparametres

Pour l'algorithme Decision Trees, nous avons effectué un ajustement des hyperparamètres en considérant la profondeur maximale (max_depth) et le nombre minimal d'échantillons requis pour diviser un nœud (min_samples_split). Les résultats de l'expérimentation montrent que le modèle atteint la meilleure performance lorsque max_depth est égale à 5 et min_samples_split est égal à 2. Cela correspond à une précision de test maximale de 90.30% comme le montre le tableau A.4 en annexe.

Concernant la méthode de gain d'information, nous avons comparé l'utilisation de l'indice de Gini et de l'entropie. Les résultats ont montré que l'indice de Gini était plus rapide en termes de temps d'execution, car il évite le calcul du logarithme. Ainsi, nous

Algorithm 2 Decision Tree

```
procedure DecisionTree(min_samples_split, max_depth, info_gain_method, n_features, dat
    function BuildTree(dataset, curr_depth=0)
        X, Y ← FeaturesAndLabels(dataset)
        if NotEnoughSamplesOrMaxDepth(curr_depth) then
            return LeafNode(MajorityClass(Y))
        end if
        best_split ← FindBestSplit(dataset)
        if NoBestSplitOrNoInfoGain(best_split) then
            return LeafNode(MajorityClass(Y))
        end if
        left_subtree ← BuildTree(best_split["dataset_left"], curr_depth + 1)
        right_subtree ← BuildTree(best_split["dataset_right"], curr_depth + 1)
        if PostPruningBeneficial(best_split) then
            return LeafNode(MajorityClass(Y))
        end if
        return DecisionNode( feature_index ← best_split["feature_index"], thre-
shold ← best_split["threshold"], left ← left_subtree, right ← right_subtree,
info_gain ← best_split["info_gain"])
    end function
    function FindBestSplit(dataset)
        best_split ← {}
        max_info_gain ← −∞
        for Each Feature in RandomFeatures(dataset, n_features) do
            possible_thresholds ← UniqueValues(dataset[ :, Feature])
            for Each Threshold in possible_thresholds do
                dataset_left, dataset_right ← Split(dataset, Feature, Threshold)
                info_gain ← INFORMATIONGAIN(dataset, dataset_left, dataset_right, info_gain
                if info_gain > max_info_gain then
                    max_info_gain ← info_gain
                    best_split ← {"feature_index" ← Feature, "threshold" ←
Threshold, "dataset_left" ← dataset_left, "dataset_right" ←
dataset_right, "info_gain" ← info_gain}
                end if
            end for
        end for
        return best_split
    end function
end procedure
```

Algorithm 3 Fonction Gain d'Information

function INFORMATIONGAIN($parent, l\_child, r\_child, info\_gain\_method$)
  if l_child is None or r_child is None **then**
    return 0
  end if
  $weight\_l \leftarrow \text{len}(l\_child)/\text{len}(parent)$
  $weight\_r \leftarrow \text{len}(r\_child)/\text{len}(parent)$
  if info_gain_method = "Gini" **then**
    $gain \leftarrow \text{GINI\_INDEX}(parent) - (weight\_l \times \text{GINI\_INDEX}(l\_child) + weight\_r \times \text{GINI\_INDEX}(r\_child))$
  end if
  if info_gain_method = "Entropy" **then**
    $gain \leftarrow \text{ENTROPY}(parent) - (weight\_l \times \text{ENTROPY}(l\_child) + weight\_r \times \text{ENTROPY}(r\_child))$
  end if
  return $gain$
end function

Algorithm 4 Fonction Entropie

function ENTROPY(y)
  $class\_labels \leftarrow \text{UniqueValues}(y)$
  $entropy \leftarrow 0$
  for each cls in $class\_labels$ **do**
    $p\_cls \leftarrow \text{len}(y[y == cls])/\text{len}(y)$
    $entropy \leftarrow entropy - p\_cls \cdot \log_2(p\_cls)$
  end for
  return $entropy$
end function

Algorithm 5 Fonction Gini Index

function GINI_INDEX(y)
  $class\_labels \leftarrow \text{UniqueValues}(y)$
  $gini \leftarrow 0$
  for each cls in $class\_labels$ **do**
    $p\_cls \leftarrow \text{len}(y[y == cls])/\text{len}(y)$
    $gini \leftarrow gini + p\_cls^2$
  end for
  return $1 - gini$
end function

avons choisi l'indice de Gini comme méthode de gain d'information pour ce modèle.

2.1.2.4 Exemple de déroulement de l'algorithme Decision Trees

Nous avons choisi une instance de l'ensemble de test du dataset (X_test), représentée par le vecteur [0.01801802, 0.36641221, 0.3627451, 0.33027523, 0.62666667, 0.74576271, 0.11841283, 0.07156673, 0.39303483, 0.1322228, 0.27280741, 0.03066867], pour illustrer le processus de l'algorithme d'arbre de décision.

Étapes du déroulement de l'Algorithme Arbre de Décision (voir Figure 2.1) :

1. **Construction de l'Arbre de Décision** : L'algorithme construit un arbre en évaluant les caractéristiques de l'ensemble d'entraînement. À chaque étape, la meilleure division est choisie en maximisant les gains d'information.
2. **Calcul des Gains d'Information** : À chaque nœud, l'algorithme calcule les gains d'information à l'aide de gini index ou l'entropie pour chaque caractéristique et seuil possibles, choisissant celui qui maximise le gain pour diviser les données.
3. **Division de l'Ensemble de Données** : L'ensemble de données est divisé récursivement en deux sous-ensembles selon la caractéristique et le seuil sélectionnés.
4. **Évaluation du Nœud Feuille** : Lorsqu'un nœud feuille est atteint, la classe majoritaire parmi les exemples d'entraînement dans ce nœud est attribuée à l'instance de test.
5. **Instance de Test** : nous avons choisi l'instance spécifique [0.01801802, 0.36641221, 0.3627451, 0.33027523, 0.62666667, 0.74576271, 0.11841283, 0.07156673, 0.39303483, 0.1322228, 0.27280741, 0.03066867] que nous cherchons à classer.
6. **Prédiction de la Classe** : La classe prédite pour l'instance de test est déterminée en traversant l'arbre construit en évaluant séquentiellement les caractéristiques. À chaque nœud, une condition basée sur la valeur d'une caractéristique spécifique et un seuil associé est vérifiée (Figure 2.2). Si la condition est satisfaite, l'instance se déplace vers le sous-arbre gauche ; sinon, elle se dirige vers le sous-arbre droit. Ainsi de suite jusqu'à ce que l'instance atteigne un nœud feuille, où la classe prédite est attribuée. Ceci est montré par notre exemple sur la Figure 2.2, la classe prédite est 0 ainsi que la Figure 2.3 qui montre le chemin emprunté sur l'arbre (en rouge) construit précédemment.

```
At Node: Feature 0 <= Threshold 0.6216216216216216, Move to Left Subtree
At Node: Feature 1 <= Threshold 0.48091603053435106, Move to Left Subtree
At Node: Feature 7 > Threshold 0.02514506769825919, Move to Right Subtree
At Node: Feature 0 <= Threshold 0.5645645645645646, Move to Left Subtree
At Node: Feature 9 > Threshold 0.11703239289446186, Move to Right Subtree
Reached leaf node: Predicted Value 0.0
```

FIGURE 2.2 – Exemple de déroulement de l'algorithme Decision Trees.

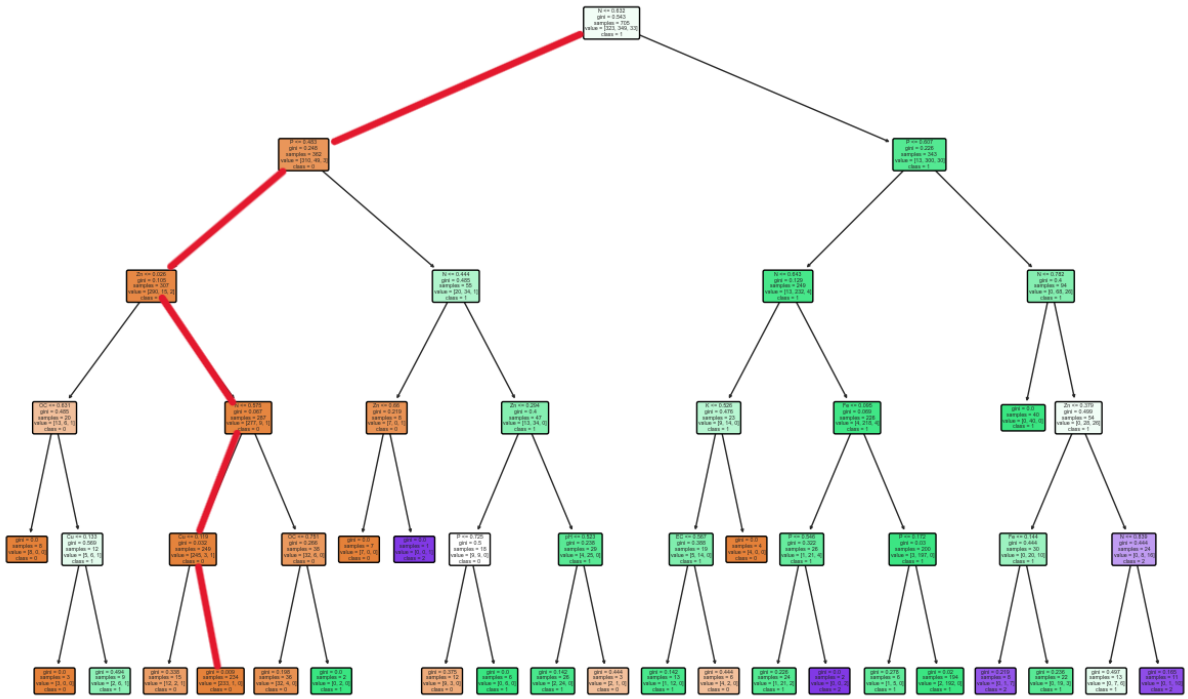

FIGURE 2.3 – Arbre construit a partir de l'ensemble d'entrainement du dataset1 avec l'algorithme Decision Trees.

2.1.3 Random Forest

Le Random Forest (RF) est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. Il fonctionne en créant plusieurs Decision Trees et en combinant leurs prédictions pour obtenir une meilleure précision. Voici comment fonctionne l'algorithme Random Forest :

1. Sélection d'attributs : L'algorithme choisit aléatoirement un ensemble d'attributs pour créer un arbre de décision.
2. Création d'Decision Trees : Pour chaque arbre, l'algorithme crée un ensemble d'Decision Trees en divisant l'ensemble de données en sous-ensembles plus petits

en fonction des valeurs des attributs sélectionnés.

3. Ensemble d'arbres : Les Decision Trees créés sont combinés en un ensemble d'arbres qui vote pour prédire la classe ou la valeur d'un nouvel exemple.
4. Vote : Les prédictions des Decision Trees sont combinées pour déterminer la classe ou la valeur majoritaire.

Le Random Forest est reconnu pour sa facilité d'interprétation, sa capacité à gérer les données imbalgées et ses performances élevées en termes de précision et de réduction de dimensionnalité.[3]

2.1.3.1 Pseudo code de l'algorithme Random Forest

Algorithm 6 Algorithme Random Forest

```
procedure RandomForest(n_trees, max_depth, min_samples_split, max_features, criterio
    trees ← EmptyList()
    for i ← 1 to n_trees do
        subset_indices ← RandomSampleIndices(X_train)
        subset_X ← Subset(X_train, subset_indices)
        subset_Y ← Subset(Y_train, subset_indices)
        tree ← DecisionTree(min_samples_split, max_depth, info_gain_method, max_fea
        tree.fit(subset_X, subset_Y)
        trees ←tree     end for

        tree_predictions ← EmptyList()
        for each tree in trees do
            tree_predictions.append(tree.predict(X_test))
        end for
        predictions ← EmptyList()
        for each row in tree_predictions do
            predictions.append(MajorityVote(row))
        end for
        Return predictions
    end procedure
```

2.1.3.2 Expérimentations en variant les paramètres de Random Forest

2.1.3.2.a Prétraitement

Les expérimentations sur les différentes méthodes de prétraitement pour Random Forest, ont révélé que la combinaison la plus efficace était d'utiliser la méthode de remplacement par la moyenne pour traiter les valeurs manquantes car les données du dataset

ont une distribution normale, la régression linéaire pour gérer les valeurs aberrantes, et la normalisation avec Min-Max (vmin, vmax). Cette combinaison spécifique a été utilisé pour obtenir tout les résultats de Random Forest.

2.1.3.2.b Hyperparametres

Pour l'algorithme Random Forest, nous avons effectué un ajustement des hyper-paramètres en considérant la profondeur maximale (max-depth) et le nombre minimal d'échantillons requis pour diviser un nœud (min-samples-split), le nombre d'attributs des sous-arbre (n-features) et le nombre d'arbres de la foret (n-trees). Les résultats de l'expérimentation montrent que le modèle atteint la meilleure performance lorsque max_depth est égale à 6 et min_samples_split est égal à 2, n_features est égal à 6 et n_trees est égal à 30. Cela correspond à une précision de test maximale de 92.09%.

Concernant la méthode de gain d'information, nous avons choisi l'indice de Gini comme méthode de gain d'information pour ce modèle pour les memes raisons que pour Decision Trees.

2.1.3.3 Exemple de déroulement de l'algorithme Random Forest

Nous avons choisi une instance de l'ensemble de test du dataset (X_test), représentée par le vecteur [0.01801802, 0.36641221, 0.3627451, 0.33027523, 0.62666667, 0.74576271, 0.11841283, 0.07156673, 0.39303483, 0.1322228, 0.27280741, 0.03066867], pour illustrer le processus de l'algorithme Random Forest.

1. **Création du Random Forest** : L'algorithme commence par générer un ensemble d'Decision Trees comme vu précédemment, chacun formé sur un sous-ensemble aléatoire du dataset. Pour cet exemple, nous avons utilisé un ensemble de 30 arbres contenant 6 attributs.
2. **Entraînement des Arbres** : Chaque arbre est entraîné sur un ensemble de données unique, obtenu par échantillonnage aléatoire avec remplacement.
3. **Prédictions des Arbres Individuels** : L'instance spécifique est ensuite soumise à chaque arbre individuel de la forêt. Chaque arbre génère une prédiction indépendante basée sur ses caractéristiques et seuils spécifiques. Dans notre exemple, 30 prédictions ont été retourner.
4. **Prédictions Finales par Vote Majoritaire** : Les prédictions de chaque arbre sont agrégées par un vote majoritaire. La classe attribuée à l'instance est déterminée par la classe qui recueille le plus grand nombre de votes parmi l'ensemble

des arbres. Dans cet exemple, la classe prédite est 0 comme le montre la Figure 2.4.

FIGURE 2.4 – Exemple de déroulement de l'algorithme Random Forest.

2.2 Comparaison des résultats obtenus par chaque algorithme

Après avoir expliqué les principes de chaque algorithme, nous allons faire une comparaison des résultats obtenus par les trois algorithmes de classification, à savoir K-Nearest Neighbors (KNN), Decision Trees, et Random Forest, en examinant les classes prédites pour une vingtaine d'instances. (voir Tableau 2.1 dans l 'annexe)

Les prédictions des trois algorithmes sont généralement cohérentes avec les classes réelles, indiquant une performance globalement bonne. Cependant, quelques variations, notamment pour les instances 7 et 14, mettent en évidence des différences de capacité de généralisation. En comparaison, Random Forest et Decision Tree semblent plus stables que KNN.

2.3 Matrices de confusion

Une matrice de confusion est une matrice qui mesure la qualité d'un algorithme de classification. Chaque ligne correspond à une classe réelle, chaque colonne correspond à une classe estimée. La cellule ligne L, colonne C contient le nombre d'éléments de la classe réelle L qui ont été estimés comme appartenant à la classe C1. Elle permet de mesurer la précision, la rappel et d'autres indicateurs de qualité du classificateur en comparant les résultats de classification avec un ensemble de données de référence[4]. C'est pour cela que

nous avons calculé une matrice de confusion pour chacun des algorithmes de classification KNN, DT, RF, montrées sur les figures ci-dessous :

FIGURE 2.5 – Matrice de confusion de KNN

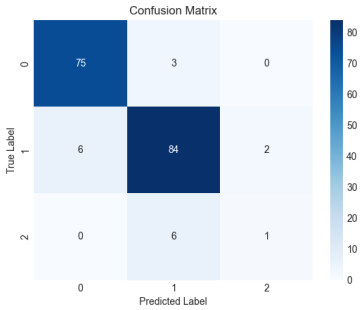

FIGURE 2.6 – Matrice de confusion de Decision Trees

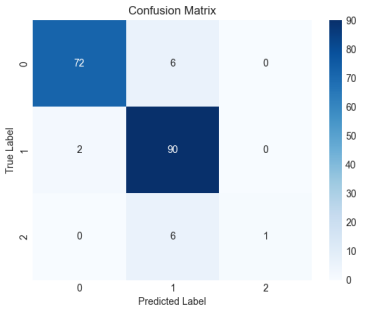

FIGURE 2.7 – Matrice de confusion de Random Forest

D'après les matrices de confusion des Figures 2.5, 2.6, 2.7, nous remarquons que KNN présente des résultats équilibrés avec une répartition relativement uniforme des prédictions dans les différentes classes. Cependant, il montre des difficultés à distinguer la classe 1.

Decision Tree offre une performance élevée dans la classification des classes 0 et 1, avec un rappel parfait pour la classe 2. Cependant, il éprouve des difficultés à identifier la classe 2, comme en témoigne le rappel 1 sur 6 pour cette classe.

Random Forest surpasse les autres en offrant une performance équilibrée et élevée pour toutes les classes. Il présente une précision et un rappel significativement supérieurs pour les classes 1 et 2 par rapport à KNN et DT.

Dans l'ensemble, le Random Forest se distingue comme le choix optimal, démontrant une capacité à maintenir une performance élevée pour chaque classe. Cete tendance est attribuée à la nature de Random Forest, qui atténue le overfitting en combinant plusieurs arbres, ce qui améliore la généralisation, contrairement au Decision Trees, qui peut souffrir de overfitting, et à KNN, qui est sensible aux valeurs aberrantes.

2.4 Évaluation et Comparaison les modèles de classification

2.5 Comparaison des algorithmes de classification KNN, Decision Trees et Random Forest avec exemples

Les mesures de performance implémentées pour l'évaluation de ces algorithmes de classification sont : l'Exactitude (Accuracy), la Spécificité, la Précision, le Rappel, le F-Score pour chaque classe et globale en plus du temps moyen d'exécution.

2.5.1 Exactitude (Accuracy)

L'exactitude mesure la proportion d'instances correctement classées par le modèle.

$$\text{Accuracy} = \frac{\text{Nombre d'instances correctement classées}}{\text{Nombre total d'instances}}$$

2.5.2 Spécificité (Specificity)

La spécificité mesure la capacité du modèle à identifier correctement les instances de la classe négative.

$$\text{Specificity} = \frac{\text{Vrais négatifs}}{\text{Vrais négatifs} + \text{Faux positifs}}$$

2.5.3 Précision (Precision)

La précision mesure la proportion d'instances correctement classées comme positives parmi toutes les instances classées comme positives.

$$\text{Precision} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux positifs}}$$

2.5.4 Rappel (Recall)

Le rappel mesure la proportion d'instances positives correctement identifiées par le modèle parmi toutes les instances réellement positives.

$$\text{Recall} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux négatifs}}$$

2.5.5 F-Score

Le F-score est une moyenne pondérée de la précision et du rappel, offrant une mesure équilibrée entre les deux.

$$\text{F-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.5.6 Analyse comparative

Dans cette partie du rapport, nous explorons et analysons les performances des trois modèles de classification éxpliqués auparavant : K-Nearest Neighbors (KNN), Decision Trees, et Random Forest. L'évaluation de ces modèles se fera sur un ensemble de métriques, notamment l'exactitude, la précision, le rappel, la spécificité, le F-Score ainsi que le temps moyen d'execution. Ces métriques fournissent des indications précieuses sur la capacité de chaque algorithme à effectuer des classifications précises et équilibrées.

Model	Exactitude	Précision	Rappel	Spécificité	F Score
KNN	0.84181	0.64094	0.62937	0.90358	0.63352
Decision Trees	0.90395	0.72083	0.67248	0.94058	0.68383
Random Forest	0.92090	0.95178	0.68140	0.94620	0.70840

TABLE 2.1 – Comparison des métriques de performances pour les algorithmes KNN, Decision Tree, and Random Forest

Les résultats de KNN, Decision Trees et Random Forest révèlent qu'en termes d'exactitude, Random Forest affiche la performance la plus élevée, dépassant celle de Decision Trees, tandis que KNN obtient la valeur la plus basse.

En ce qui concerne de précision, Random Forest présente la meilleure capacité à minimiser les faux positifs, suivie par Decision Trees, tandis que KNN obtient le résultat le moins élevé.

Pour le rappel, c'est encore Random forest qui surpassent les autres modèles, indiquant une meilleure capacité à identifier les instances positives, suivi de près par Decision Trees, tandis que KNN affiche le rappel le plus bas.

En ce qui concerne la spécificité, Random Forest et Decision Trees présentent des valeurs similaires, démontrant leur aptitude à minimiser les faux négatifs, tandis que KNN est legerement inférieur dans cette mesure.

Enfin, le F-Score, qui équilibre précision et rappel, confirme la supériorité de Random Forest, suivie par Decision Trees, et KNN affiche le score le plus bas.

En conclusion, Random Forest se démarque globalement en atteignant de bons résultats pour toutes les métriques. Cette supériorité est attribuée à sa nature, qui combine les prédictions de multiples arbres. Cette approche atténue le overfitting et améliore la généralisation du modèle, lui permettant de maintenir ces performances élevées. Decision Trees, bien que présentant des performances solides, peuvent être plus sensibles au overfitting en raison de leur tendance à s'adapter trop aux données d'entraînement. En revanche, KNN malgré ses performances légèrement inférieures, demeure acces pertinent.

Pour obtenir une évaluation robuste, nous avons pris en compte le temps moyen d'exécution, résultant de 10 exécutions indépendantes de chaque algorithme. Les résultats sont représentés sur le Tableau 2.3.

Model	**Temps d'execution moyen (s)**
KNN	0.0
Decision Trees	10.99509
Random Forest	168.98212

TABLE 2.2 – Comparaison du temps d'execution moyen des algorithmes KNN, Decision Tree, and Random Forest

Algorithme	Entraînement	Prédiction
K-Nearest Neighbors (KNN)	$O(1)$	$O(n \cdot m)$
Arbre de Décision (DT)	$O(n \cdot m \cdot \log(m))$	$O(\log(m))$
Random Forest (RF)	$O(T \cdot n \cdot \log(n) \cdot m)$	$O(T \cdot \log(n) \cdot m)$

TABLE 2.3 – Complexités temporelles des algorithmes (n=nbr instances,m=nbr colonnes,t=nbr iterations)

L'analyse comparative en termes de temps d'exécution entre les modèles K-Nearest Neighbors (KNN), Decision Trees, et Random Forest révèle des tendances explicables par leurs complexités temporelles respectives. (voir Tableau 2.4)

Tout d'abord, KNN, avec un temps d'exécution nul, resultant de sa complexité constante lors de l'entraînement, bien que la prédiction devienne plus coûteuse à mesure que le nombre d'échantillons et de caractéristiques augmente.

Pour Decision Trees, bien que plus lents que KNN, affichent un temps d'exécution moyen d'environ 11 secondes, en ligne avec leur complexité logarithmique pendant l'entraînement. La prédiction est plus efficace grâce à une complexité logarithmique également.

En revanche, Random Forest, malgré ses performances élevées, présente le temps d'exécution le plus long, résultant de la construction et de l'agrégation de plusieur arbres, entraînant une complexité plus élevée.

Ainsi, l'analyse met en lumière le compromis entre la précision des 3 algorithmes et le temps d'exécution.

Chapitre 3

Analyse non supervisée

K-means et DBSCAN sont tous les deux des algorithmes de clustering visant à diviser un ensemble de données en groupes homogènes en fonction de leur similarité, mais ils utilisent à cet effet des méthodes différentes que nous allons détailler dans les sections qui suivent.

Mais d'abord, il faut savoir que pour juger de la similitude entre les instances en question, ces algorithmes emploient l'une des nombreuses mesures de distance existantes parmi lesquelles nous avons implémenté :

3.0.1 Distance de Minkowski

$$D(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \tag{3.1}$$

Comme montre ci-dessus, l'equation 3.1 a comme hyperparamètre la variable p et contient deux cas particuliers : si p=2, cela correspond à la distance euclidienne, tandis que si p=1, il s'agit de la distance de Manhattan.

3.0.2 Distance de cosinus

Une autre approche, dont la formule 3.2 est la suivante

$$\text{Cosine Similarity} = \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{3.2}$$

3.1 Application d'algorithme de clustering basé partitionnement

3.1.1 K-means

K-means est une méthode de classification non supervisée basée sur le partitionnement. Elle répartit les données en k clusters (k étant défini en entrée). Sa complexité est de O(n*k*t) où n est le nombre de points , k le nombre de cluster et t le nombre d'iterations

Nous allons à présent dérouler notre algorithme pour bien comprendre son fonctionnement. K-means se déroule en 4 étapes :

1. **Initialisation** : Choix de k (en l'occurrence 2) centres de cluster initiaux, appelés centroids. Nous avons développé deux méthodes de sélection. La première consiste à les choisir de manière aléatoire. La deuxième approche vise une sélection plus judicieuse en optant pour une initialisation arbitraire du premier centroid, suivi du choix des suivants de sorte qu'ils soient loin les uns des autres. Cette approche évite le cas de centroids adjacents, ce qui retardera la convergence. Le pseudocode de cette partie est detaillé dans l'algorithme 7 et le reste des etapes dans l'algorithme 8
2. **Affectation** : Association de chaque point de notre dataset au centroid le plus proche selon la mesure de distance choisie citée précédemment.
3. **Mise à jour du centre** : Les nouveaux centres sont calculés en prenant la moyenne des points affectés à leurs groupes respectifs.

Ce processus est répété jusqu'à convergence, c'est-à-dire la stabilisation des clusters. Dans le cas où cela n'arrive pas, nous interrompons la boucle lorsque le nombre d'itérations maximales est atteint.

3.1.2 Experimentation en variant les paramètres de k-means

3.1.2.1 Prétraitement

Étant donné que nous avons implémenté plusieurs méthodes de prétraitement dans la partie 1, nous pouvons à présent choisir les plus adaptées pour nos algorithmes.

Notre hypothèse était qu'au vu de la sensibilité de k-means aux outliers, la normalisation Z-score, qui est robuste face à cela, conviendrait mieux, ainsi que l'ajout de la

Algorithm 7 K-Means Clustering

```
1: procedure KMeans(k, methode_d, methode_c, max_iterations)
2:     self.k ← k
3:     self.centroid ← listeVide
4:     self.dataset_letiqu ← dataset[ :, -1]
5:     self.methode_c ← methode_c
6:     self.methode_d ← methode_d
7:     self.max_iterations ← max_iterations
8:     self.Xtrain ← xt
9:     function centroid_selection(methode)
10:        if methode == 0 then
11:            self.centroid = random.sample(Xtrain,k)
12:        else if methode == 1 then
13:            self.centroid.ajouter(random.element(Xtrain))
14:            for x in Xtrain do
15:                dist ←  distance(centroid[0],x))
16:            end for
17:            ind ← np.argsort(dist)
18:            for i self.k to 0 step -1 do
19:                self.centroid.append(Xtrain[ ind[(len(ind)/self.k*i )-1], :])
20:            end for
21:        end if
22:    end function
23: end procedure
```

Algorithm 8 K-Means Clustering suite

```
function _CLUSTER
    self.centroid_selection(self.methode_c)
    change ← True
    nbr_iteration ← 0
    while change do
        for j 0 to longueur(Xtrain) do
            distances ← listeVide
            for i 0 to self.k do
                distances.append(distance(self.centroid[i], self.Xtrain[j, :], self.methode_d))
            end for
            c ← np.argmin(distances)
            self.dataset_letiqu[j,-1] ← np.argmin(distances)
        end for
        oldcentroid ← self.centroid.copy()
        for i0 to self.k do
            cluster ← [row[ :-1] for row in self.dataset_letiqu if row[-1]==i
            self.centroid[i] ← np.array([np.average(cluster[ :,j]) for j in longueur(cluster) ] )
        end for
        if self.centroid != oldcentroid or nbr_iteration > self.max_iterations then
            change ← False
        end if
        nbr_iteration ← nbr_iteration + 1
    end while
    return self.dataset_letiqu
end function
function _PREDICTION(instance)
    distances ←listeVide     for i ← 0 self.k do
    distances.append(distance(self.centroid[i], instance, self.methode_d))

    return [row[ :-1] for row in self.dataset_letiqu if row[-1]==np.argmin(distances)]
end function
```

méthode de remplacement de valeurs aberrantes par winsorisation, qui permet d'éviter le phénomène de "outlier injection", c'est-à-dire l'apparition de nouvelles valeurs aberrantes due au changement de Q1 et Q3 après les remplacements.

Cependant, après avoir testé les différentes combinaisons de techniques de prétraitement, la meilleure combinaison en utilisant la silhouette comme mesure de performance s'avère être le remplacement de valeurs manquantes par la moyenne par classe, le remplacement des outliers par discrétisation et la normalisation par min-max (0-1). Cependant, cela est le cas uniquement parce que nous avons intégré la méthode PCA (Analyse en Composantes Principales) pour réduire les données à deux colonnes.

Il faut noter quand même que l'absence d'outliers dans les attributs de notre dataset n'implique pas l'absence d'outliers de l'entité instance.

3.1.2.2 Hyperparametres

En ce qui concerne les hyperparamètres, le tableau A.1 dans l'annexe indique les résultats des expérimentations avec la combinaison de prétraitement choisie. Nous remarquons que les silhouettes les plus pertinentes sont obtenues avec un k=2, la meilleure étant 0.45 pour la méthode de centroid intelligente et la distance de Minkowski.

Le choix de la méthode de sélection de centroid n'a pas eu d'influence majeure (très légère amélioration), de même pour les méthodes de distance euclidienne, de Manhattan et de Minkowski. Quant à la méthode cosine, elle donne de moins bonnes silhouettes, ce qui la rend moins adaptée pour nos données.

3.1.2.3 Analyse

FIGURE 3.1 – Graphique de l'evolution du temps d'execution en fonction de k (nbr cluster)

Ce graphe 3.1 illustre une corrélation positive entre le nombre de clusters et le temps d'exécution. En effet, l'augmentation de k entraîne une hausse du temps d'exécution, phénomène pouvant s'expliquer par le fait qu'un plus grand k implique plus de calculs lors de l'affectation des points aux centroids et de leur mise à jour. Cela mène aussi à une stabilisation tardive des clusters.

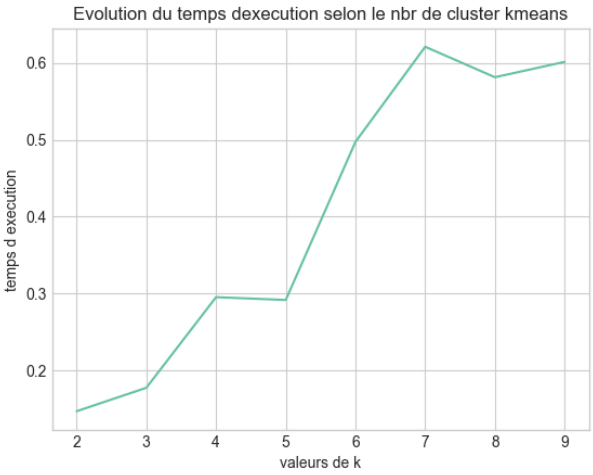

FIGURE 3.2 – Graphique de l'evolution de la silhouette en fonction de k (nbr cluster)

Nous constatons à partir de cette courbe 3.2 que la silhouette de l'algorithme k-means est inversement proportionnelle à la valeur de k. Elle passe de 0.45 pour k=2 à 0.37 pour k=9. Cela est dû à la nature de nos données, suggérant que 2 est le nombre de clusters à utiliser.

3.2 Application d'algorithme de clustering basé densité

3.3 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) identifie des clusters basés sur la densité des points. Il est capable de traiter des données aberrantes

en les attribuant à une catégorie spéciale. Tout ca avec une complexité de l'ordre de O(n2) avec n etant nombre de point

Cet algorithme prend en entrée 2 paramètres : le rayon "radius" et le nombre minimum de points requis pour former un cluster, appelé "MinPts".

L'algorithme commence par choisir une instance arbitraire et examine son voisinage dans un rayon défini par "radius" selon la mesure de distance sélectionnée. Si le nombre de voisins est inférieur au seuil MinPts, il sera temporairement considéré comme un outlier, sinon un nouveau cluster est créé en incluant ces points et en étendant la recherche à leurs voisins. Dans cette phase, un point précédemment mis dans la catégorie outlier peut être réaffecté dans le nouveau cluster s'il satisfait les conditions nécessaires, comme détaillé dans le pseudo-algorithme.

Le processus global se répète jusqu'à ce que tous les points fassent partie soit d'un cluster, soit de la catégorie outlier. Les details sont presentés dans les pseudo codes 9 et 10

Algorithm 9 DBSCAN Clustering

```
procedure (Point)
    function __INIT__(instance)
        self.instance ← instance
        self.marked ← False
        self.cluster ← False
    end function
end procedure
function VOISINAGE(P, radius, methode_d)
    voisins ← listeVide
    for i 0 to longueur(dataset1) do
        if distance(dataset1[i, :], P.instance, methode_d) ≤ radius then
            voisins.ajouer(i)
        end if
    end for
    return voisins
end function
```

3.3.1 Experimentation des paramètres de DBSCAN

3.3.1.1 Prétraitement

Les résultats des tests de méthodes de prétraitement ont montré que la meilleure combinaison pour DBSCAN est : le remplacement de valeurs manquantes par la moyenne

Algorithm 10 DBSCAN Clustering suite

```
function DB_Scan(radius, min_points, methode_d)
    C ← 0
    Outlier ← listeVide
    dataset_labeled ← listeVide
    listeP ← [Point(instance) for instance in dataset]
    for P in listeP do
        if not P.marked then
            P.marked ← True
            PtsVoisins ← Voisinage(P, radius, methode_d)
            if len(PtsVoisins) < min_points then
                Outlier.append(P)
                dataset_labeled.append(np.append(P.instance, −1))
            else
                C ← C + 1                                   ▷ new cluster
                P.instance.ajoute(C)
                P.cluster ← True
                dataset_labeled.append(P.instance)
                for i in PtsVoisins do
                    if not listeP[i].marked then
                        listeP[i].marked ← True
                        v ← Voisinage(listeP[i], radius, methode_d)
                        if len(v) ≥ min_points then
                            PtsVoisins.extend(v)
                        end if
                    end if
                    if not listeP[i].cluster then
                        listeP[i].cluster ← True
                        if listeP[i] in Outlier then
                            Outlier.remove(listeP[i])
                            for j in range(len(dataset_labeled)) do
                                if dataset_labeled[j][ :-1]==listeP[i].instance) then
                                    listeP[i].ajoute(C)
                                    dataset_labeled[j][-1] ← C
                                    break
                                end if
                            end for
                        else
                            listeP[i].instance.ajoute(C)
                            dataset_labeled.ajoute(listeP[i].instance)
                        end if
                    end if
                end for
            end if
        end if
    end for
    return dataset_labeled
end function
```

par classe, le remplacement des outliers par régression linéaire, et la normalisation par min-max (0-1), et ceci sans PCA.

3.3.1.2 Hyperparametres

Pour le tuning des hyperparamètres, il a été effectué sur toutes les combinaisons de prétraitement et sur une plage de [0.5-1.2] pour le rayon et [5-25] pour minpoints. Pour comparer nos résultats, nous avons choisi comme mesure de performance le davies bouldin score (à minimiser) et non pas la silhouette, car cette dernière n'est pas la plus adaptée pour les algorithmes de clustering basés sur la densité (pouvant avoir des clusters emboîtés).

Le tableau A.2 montre que le rayon est le paramètre le plus influençable pour la performance de notre algorithme, atteignant 0.41 (à interpréter comme un très bon résultat selon cette mesure) en privilégiant des rayons supérieurs ou égaux à 0.8, tandis que le nombre de minpoints dans l'intervalle choisi n'affecte pas le résultat, qui est la formation d'un cluster avec ses outliers.

3.3.1.3 Analyse

3.4 Comparaison des deux algorithmes de clustering k-means et DBSCAN avec exemples

Les mesures de performance implémentées au niveau du clustering sont la silhouette, la distance intra-cluster ainsi que la distance inter-cluster. Nous avons également utilisé le davies bouldin score pour le tuning de DBSCAN.

La silhouette est une mesure qui évalue la cohésion intra-cluster (à quel point les points d'un même cluster sont proches) par rapport à la séparation inter-cluster (à quel point les points d'un cluster sont éloignés des points d'autres clusters). Ses valeurs appartiennent à une intervalle de -1 à 1 (à maximiser). Sa formule est la suivante 3.3 :

$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{3.3}$$

- $a(i)$ est la distance moyenne du point i aux autres points du même cluster (cohésion intra-cluster).

K-means	DBSCAN
basé Partitionnement	basé Densité
nombre de cluster doit être spécifié à l'avance	Peut détecter automatiquement le nombre de clusters
simple et rapide	moins rapide et simple
forme de cluster generalement spherique	Capable de gérer des formes arbitraites et complexe
Sensible à l'echelle des données (normalisation)	Moins sensible grâce à la densité
Sensible aux bruits, attribue chaque point à un cluster	identifier les bruits dans une classe à part
fonctionne qu'avec des données numerique (a cause de la moyenne)	fonctionne avec tout type de donnée en adaptant la mesure de distance
choix des centroides peut affecter les performance	donnée peuvent sembler unifome avec plusieurs dimensions

TABLE 3.1 – Comparaison entre les methodes Kmeans et DBSCAN

- $b(i)$ est la plus petite distance moyenne du point i aux points d'un cluster différent, minimisé sur les clusters voisins (séparation inter-cluster).

Le score silhouette pour l'ensemble du clustering est la moyenne des scores silhouette de tous les points dans l'ensemble de données.

3.4.1 Analyse

K-means et DBSCAN ont tous les deux atteint 0.45 comme meilleure silhouette, avec 2 clusters pour K-means et 1 cluster avec outliers pour DBSCAN. Selon les besoins, une méthode peut être privilégiée par rapport à l'autre en considérant les différences entre les deux techniques, comme indiqué dans le tableau 3.1

3.4.2 Exemple

Voici ci-dessous des exemples de clustering par K-means et DBSCAN en utilisant

PCA pour réduire les données à un nombre de dimensions plotable.

Dans les deux graphes suivants 3.3 et 3.4, nous avons utilisé les hyperparamètres optimaux pour K-means, mais pas pour DBSCAN, car sa meilleure combinaison n'inclut pas de PCA.

Nous remarquons directement la différence de méthodologie entre les deux algorithmes, notamment par la forme des clusters obtenus : ceux de K-means étant arrondis, alors que DBSCAN, selon les hyperparamètres choisis, a créé qu'une seule classe avec des outliers affectés au reste des points.

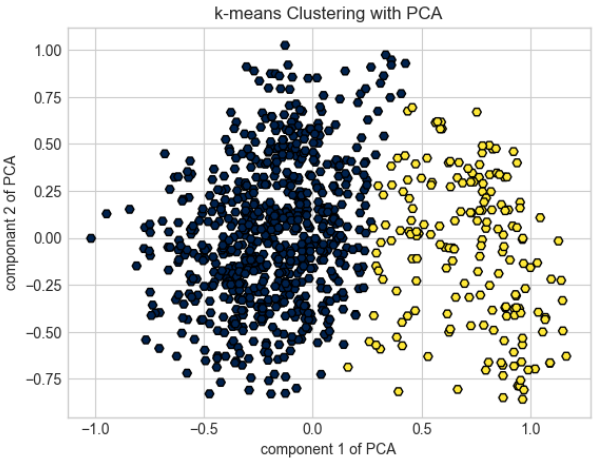

FIGURE 3.3 – Graphique d'un exemple de clustering avec K-means

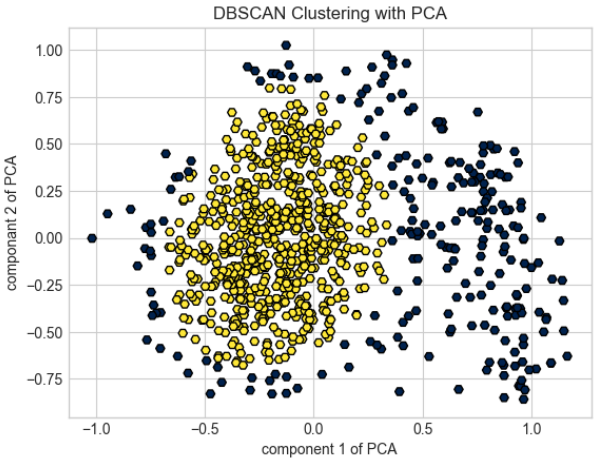

FIGURE 3.4 – Graphique d'un exemple de clustering avec DBSCAN

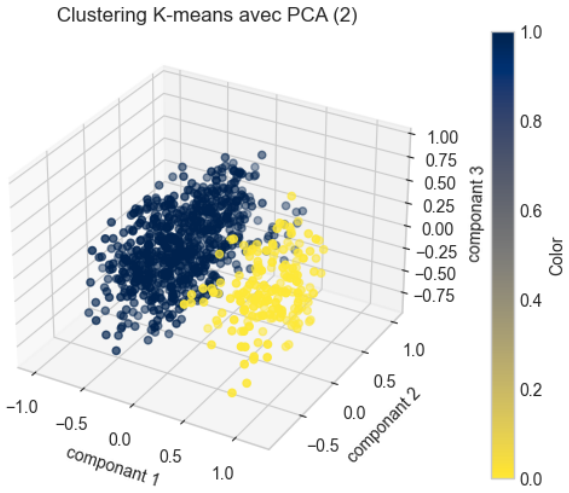

FIGURE 3.5 – Graphique 3D d'un exemple de clustering avec K-means

Nous avons également visualisé le clustering K-means avec un graphe 3D 3.5(toujours PCA avec 3 composants cette fois), constatant une haute performance de part sa capacité à regrouper et à distinguer entre les clusters

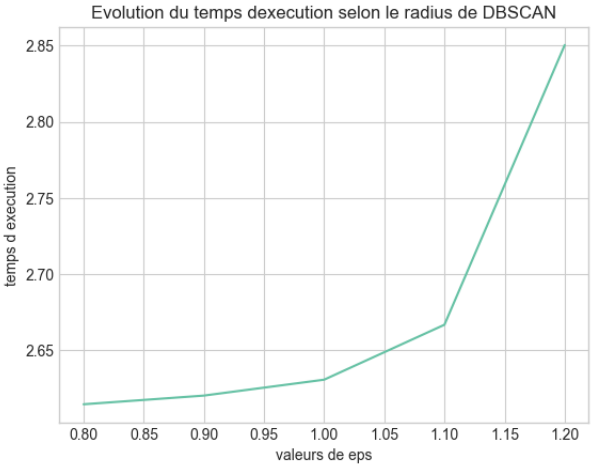

FIGURE 3.6 – Graphique de l'evolution du temps d'execution selon les valeur du rayon

La courbe 3.6 décrivant le temps d'exécution de DBSCAN en fonction de la valeur du rayon montre une augmentation exponentielle du temps lorsque l'on agrandit le rayon, vraisemblablement parce qu'un plus grand rayon mène à un voisinage plus large et, par conséquent, à plus d'itérations lors de l'extension des clusters.

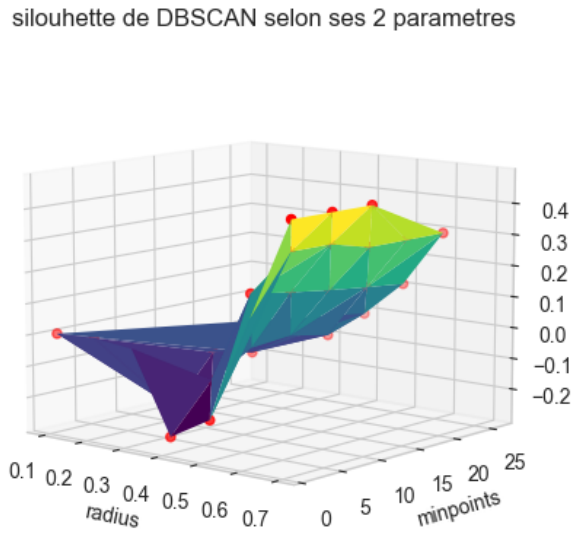

FIGURE 3.7 – Graphique de l'evolution de la silhouette en fonction du rayon et du nombre de minpoints

Le graphe 3D suivant 3.7 indique la variation de la silhouette en fonction du rayon et de la valeur de minpoints. L'agrandissement du rayon mène à une amélioration nette du score de silhouette, intensifiée par un minpoint entre 5 et 15.

Chapitre 4

Conclusion et perspectives

Au cours de ce projet, nous avons analyser puis préparer notre dataset par differents type de prétraitements, afin d'extraire des informations implicites et non trivial à partir de ces données. Cela en appliquant des 3 methodes de classification supervisé et 2 non supervisé. Le choix de la meilleur methode depend du but souhaité et des performances. Nous en avons tiré les conclusions suivantes :

- Les hyperparametres ne sont pas les seuls elements qui impactent la performance du clustering ou de la classification,meme les methodes de pretraitement et le dataset ainsi que l'utilisation ou non d'une reduction de données (PCA) influent sur les resultats
- Le clustering à pour but le regroupement de donnes similaires alors que la classification vise a predire la classe d'un nouveau point
- kmeans est certe rapide est simple, mais est sensible aux outliers,l'echelle et type de données, nombre de cluster donnée en entrée et choix des centroids
- DBSCAN peut detecter des fromes abstraites de cluster , gerer les bruits et les differents types de données mais à tendence à regrouper les données en un cluster selon le dataset et ses dimensions
- Meme si la silhouette est une mesure populaire de performance de clustering, elle nest pas toujours la plus adapté (ex : dbscan avec formes abstraites de cluster)
- KNN est une methode naive et féneante et depend de k mais rapide,simple et non sensible aux outlier
- Decision trees offre une interprétabilité aisée des décisions prises, mais peut être sujette à l'overfitting sur des ensembles de données complexes. c'est pourquoi on emploie le pre-prunning ou le postpruning alors que Random forest, en agrégeant plusieurs arbres de décision, meem si elle prend plus de temps elle améliore la robustesse du modèle en réduisant l'overfitting.

Références

1. Boutet, A. (2017) KIFF : un algorithme de construction de graphe KNN générique, rapide et évolutif. https://www.semanticscholar.org/paper/KIFF%3A-un-algorithme-de-construction-de-graphe-KNN-Boutet-Kermarrec/d8941e089da23229adaa80c7b641e6a0be601af4.

2. Dorgo, G. (2021) Decision trees for informative process alarm definition and alarm-based fault classification. https://www.semanticscholar.org/paper/Decision-trees-for-informative-process-alarm-and-Dorgo-Dorgo/41e9c29da68675a21c89a0438408b96134587889.

3. Schonlau, M. (2020) The random forest algorithm for statistical learning. https://www.semanticscholar.org/paper/The-random-forest-algorithm-for-statistical-Schonlau-Zou/e3eaf3c461114bc34675b0aa33e48ac0be003451.

4. Contributeurs aux projets Wikimedia (2023) Matrice de confusion. https://fr.wikipedia.org/wiki/Matrice_de_confusion.

Annexe

Dans l'annexe suivant nous retrouverons different tableaux ainsi qu'un graphe de la partie 1 (Tous cité dans leurs sections respectives)

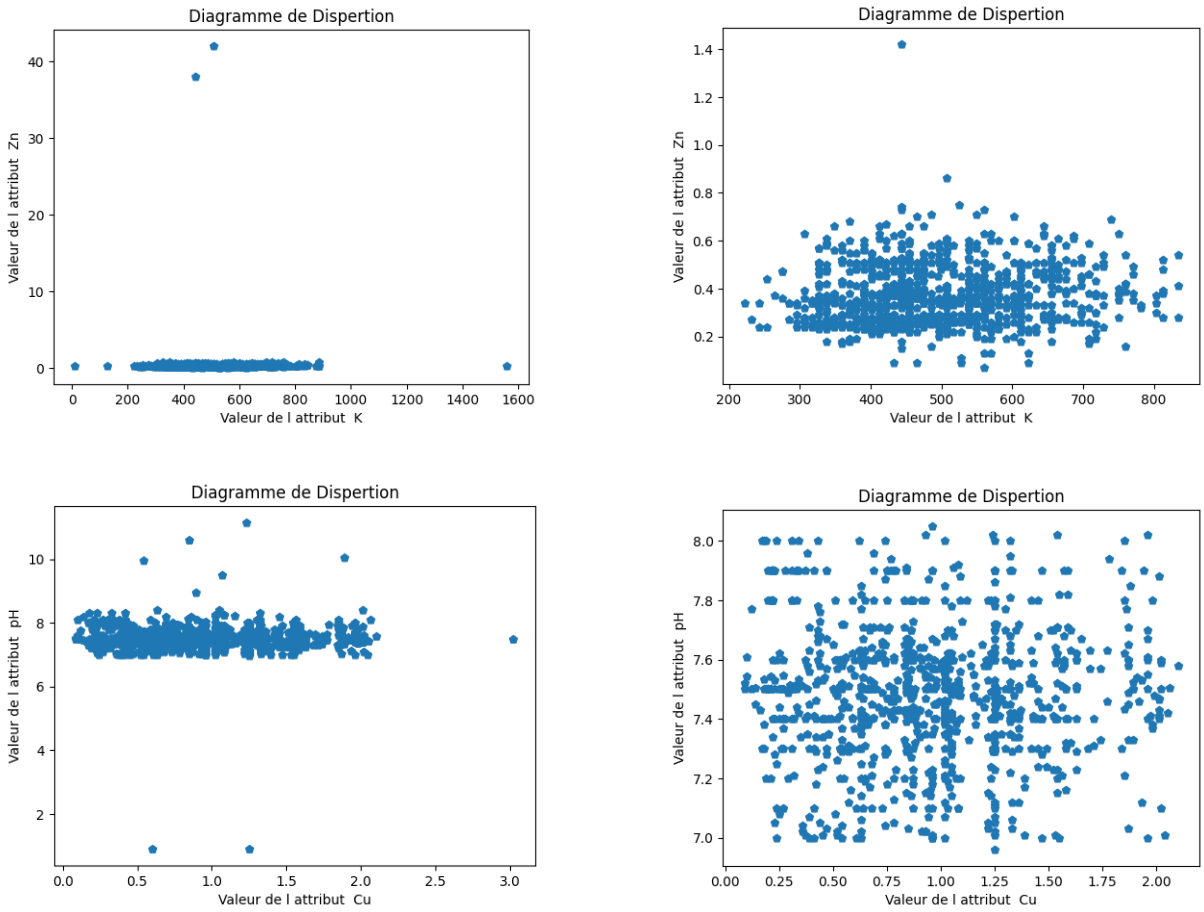

FIGURE 4.1 – Scatter plots des deux attributs K et Zn, Cu et pH, avant et après traitement des valeurs aberrantes.

mesure distance	**methode centroid**	**k**	**silhouette**
0	0	2	0.347479364
0	1	2	0.347479364
1	0	2	0.453945318
1	1	2	0.45474061
2	0	2	0.457001951
2	1	2	0.457001951
3	0	2	0.457274085
3	1	2	0.457821508
4	0	2	0.360483813
4	1	2	0.456858414
0	0	3	0.382237512
0	1	3	0.382237512
1	0	3	0.400709172
1	1	3	0.400709172
2	0	3	0.406325218
2	1	3	0.406325218
3	0	3	0.405202409
3	1	3	0.405202409
4	0	3	0.404304627
4	1	3	0.404430325

0	0	4	0.332101533
0	1	4	0.32529674
1	0	4	0.334944626
1	1	4	0.335037967
2	0	4	0.347459188
2	1	4	0.347545554
3	0	4	0.347808862
3	1	4	0.348345255
4	0	4	0.394462593
4	1	4	0.330645116
0	0	5	0.303396276
0	1	5	0.302740746
1	0	5	0.339531729
1	1	5	0.34651698
2	0	5	0.348342584
2	1	5	0.349189888
3	0	5	0.344699098
3	1	5	0.353491726
4	0	5	0.3505817
4	1	5	0.35285783
0	0	6	0.271203863

0	1	6	0.258077089
1	0	6	0.322154054
1	1	6	0.357513656
2	0	6	0.371314943
2	1	6	0.374743603
3	0	6	0.373855188
3	1	6	0.37350323
4	0	6	0.372910141
4	1	6	0.344990821
0	0	7	0.242158361
0	1	7	0.252546453
1	0	7	0.357442937
1	1	7	0.343067722
2	0	7	0.358458887
2	1	7	0.352783317
3	0	7	0.309856674
3	1	7	0.34986425
4	0	7	0.34712235
4	1	7	0.348981402
0	0	8	0.224592704
0	1	8	0.230258435

1	0	8	0.3269148
1	1	8	0.351962506
2	0	8	0.339810448
2	1	8	0.337437177
3	0	8	0.347875231
3	1	8	0.342980514
4	0	8	0.333578063
4	1	8	0.339707119
0	0	9	0.185929786
0	1	9	0.200743996
1	0	9	0.325446962
1	1	9	0.347215887
2	0	9	0.34877294
2	1	9	0.347126355
3	0	9	0.324259653
3	1	9	0.341256602
4	0	9	0.329008757
4	1	9	0.34158352

TABLE A.1 – Resultats du tuning des hyperparametres de kmeans

rayon	Minpoints	davies bouldin score
0.5	5	5.217599625
0.5	10	5.596099084
0.5	15	5.863837322
0.5	20	5.50224815
0.8	5	0.414992206
0.8	10	0.414992206
0.8	15	0.414992206
0.8	20	0.414992206
1.1	5	0.414992206
1.1	10	0.414992206
1.1	15	0.414992206
1.1	20	0.414992206

TABLE A.2 – Resultats du tuning des hyperparametres de DBSCAN

k	Accuracy
2	0.7966
3	0.8418
4	0.8475
5	0.8588
6	0.8475

7	0.8249
8	0.8249
9	0.8249
10	0.8249

TABLE A.3 – Resultats du tuning des hyperparamètres de KNN.

max_depth	**min_samples_split**	**Accuracy**
2	3	0.8983
2	4	0.8983
2	5	0.8983
2	6	0.8983
5	2	0.9040
5	3	0.9040
5	4	0.9040
5	5	0.9040
5	6	0.9040
10	2	0.8701
10	3	0.8701
10	4	0.8757
10	5	0.8757
10	6	0.8701

15	2	0.8701
15	3	0.8701
15	4	0.8757
15	5	0.8757
15	6	0.8701
20	2	0.8701
20	3	0.8701
20	4	0.8757
20	5	0.8757
20	6	0.8701

TABLE A.4 – Resultats du Tuning des hyperparamètres de Decision Tree.

Instance	KNN	Decision Tree	Random Forest	Actual Class
0	1.0	1.0	1	1.0
1	1.0	1.0	1	1.0
2	1.0	0.0	0	0.0
3	1.0	1.0	1	1.0
4	0.0	0.0	0	0.0
5	1.0	1.0	1	0.0
6	1.0	1.0	1	1.0
7	1.0	1.0	1	2.0
8	1.0	1.0	1	1.0
9	1.0	1.0	1	1.0
10	1.0	1.0	1	1.0
11	1.0	1.0	1	1.0
12	1.0	1.0	1	1.0
13	0.0	0.0	0	0.0
14	2.0	1.0	1	1.0
15	1.0	0.0	0	0.0
16	1.0	1.0	1	1.0
17	0.0	0.0	0	0.0
18	1.0	1.0	1	1.0
19	1.0	1.0	1	1.0

TABLE A.5 – Comparison des classes prédites par les algorithmes KNN, Decision Tree, and Random Forest

Instance	KNN	Decision Tree	Random Forest	Actual Class
0	1.0	1.0	1	1.0
1	1.0	1.0	1	1.0
2	1.0	0.0	0	0.0
3	1.0	1.0	1	1.0
4	0.0	0.0	0	0.0
5	1.0	1.0	1	0.0
6	1.0	1.0	1	1.0
7	1.0	1.0	1	2.0
8	1.0	1.0	1	1.0
9	1.0	1.0	1	1.0
10	1.0	1.0	1	1.0
11	1.0	1.0	1	1.0
12	1.0	1.0	1	1.0
13	0.0	0.0	0	0.0
14	2.0	1.0	1	1.0
15	1.0	0.0	0	0.0
16	1.0	1.0	1	1.0
17	0.0	0.0	0	0.0
18	1.0	1.0	1	1.0
19	1.0	1.0	1	1.0

TABLE A.6 – Comparison des classes prédites par les algorithmes KNN, Decision Tree, and Random Forest