- Université des Sciences et de la Technologie Houari Boumediene -

Data Mining

Rapport de mini projet

Réalisé par :

MEGHNI Mohamed EL Amine

LARABA Oussama

Spécialité : Master 2, SII

Groupe : 02

Année universitaire : 2020 / 2021

Introduction

Le data mining, connu aussi sous le nom découverte de connaissances dans les bases de données (KDD), est un processus permettant la découverte d'informations nouvelles et potentiellement utiles à partir d'une grande masse de données. Le data mining s'applique dans différents domaines, notamment la vente au détail, la bio-informatique, en médecine, en industrie, en analyse financière, etc.

Problématique

Dans ce projet, nous allons appliquer le processus de data mining sur un dataset récupéré à travers cinq laboratoires médicaux, le dataset contient des tests sur des patients ayant présenté ou non, des symptômes de la fonction de la glande thyroïde, les symptômes remarqués sont, une augmentation ou d'une diminution des concentrations plasmatiques d'hormones thyroïdiennes (hyperthyroïdie ou hypothyroïdie, respectivement).

1 Etudes statistiques des données

Le benchmark « Thyroid_Dataset.txt » contient 215 instances, les instances ont été récupérées depuis cinq laboratoires différents, chaque instance de ce dataset représente un test sur un patient, représentant ou non, des symptômes de disfonctionnement de la glande thyroïde, les symptômes remarqués sont, une augmentation ou d'une diminution des concentrations plasmatiques d'hormones thyroïdiennes (hyperthyroïdie ou hypothyroïdie, respectivement). Chaque instance est constituée de 6 attributs, à savoir :

Attribut N°	Description	Type	Intervalle de valeurs
1	Cet attribut classifie l'état de la glande thyroïde du patient	Entier	[1, 3]
2	Test d'absorption de la résine T3	Entier	[65, 144]
3	La valeur du taux sanguin de FT4	Réel	[0.5, 25.3]
4	Triiodothyronine sérique totale mesurée par radio -immunologique	Réel	[0.2, 10]
5	Triiodothyronine sérique totale mesurée par radio -immunologique	Réel	[0.1, 56.4]
6	Différence absolue maximale de la valeur TSH après injection de 200 micro grammes d'hormone de libération de la thyrotropine par rapport à la valeur basale.	Réel	[-0.7, 56.3]

La figure 1 représente une description du dataset implémentée dans notre interface graphique :

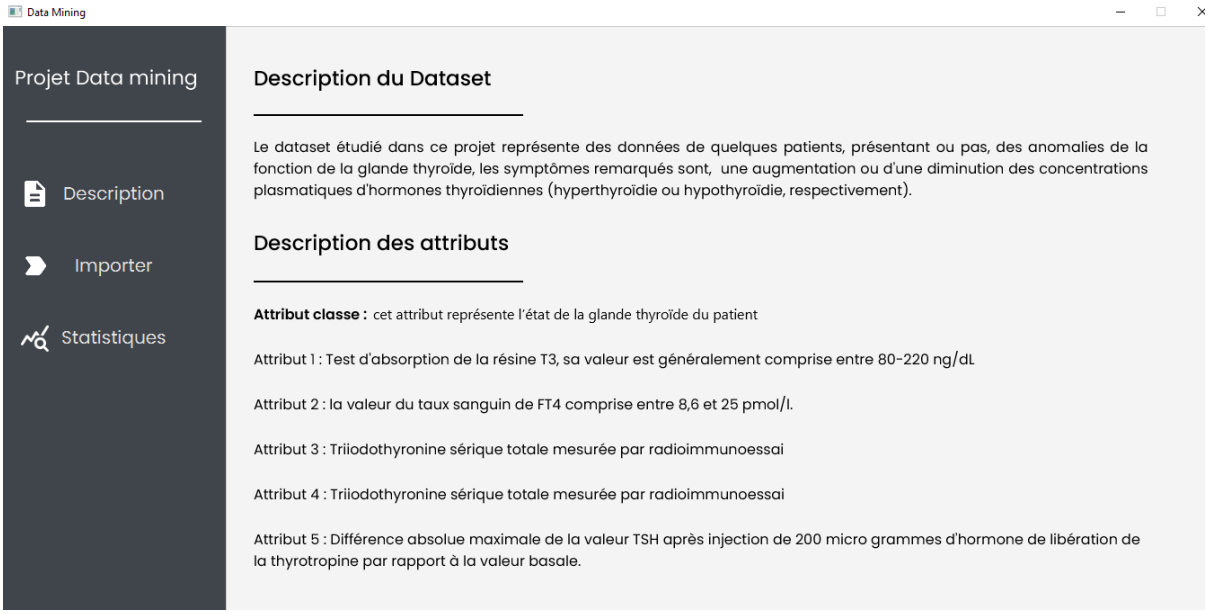
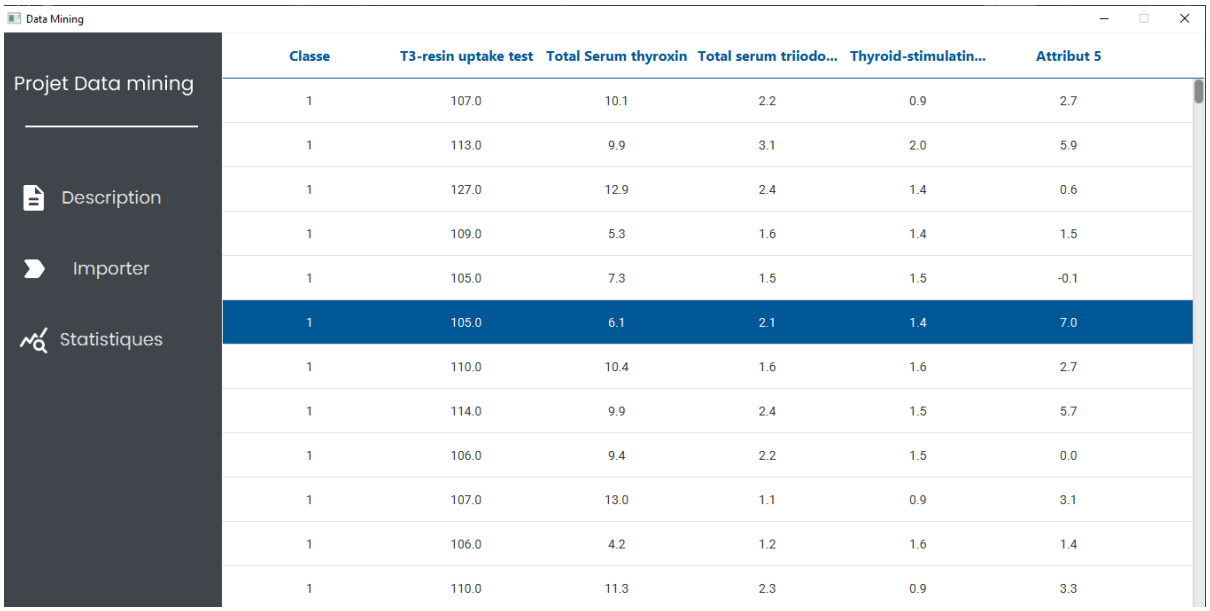


Figure 1 : description de dataset implémentée dans l'interface graphique

Manipulation du dataset

A cette étape, notre application doit être capable de lire chaque instance du benchmark, et de l'afficher à l'utilisateur.

Chaque instance du fichier est considérée comme un objet de type patient, en récupérant toutes les instances du fichier, notre application disposera à la fin d'une liste de patients pour faciliter les calculs statistiques par la suite de ce chapitre. La figure 2 représente les instances importées à l'interface graphique :



Classe	T3-resin uptake test	Total Serum thyroxin	Total serum triiodo...	Thyroid-stimulatin...	Attribut 5
1	107.0	10.1	2.2	0.9	2.7
1	113.0	9.9	3.1	2.0	5.9
1	127.0	12.9	2.4	1.4	0.6
1	109.0	5.3	1.6	1.4	1.5
1	105.0	7.3	1.5	1.5	-0.1
1	105.0	6.1	2.1	1.4	7.0
1	110.0	10.4	1.6	1.6	2.7
1	114.0	9.9	2.4	1.5	5.7
1	106.0	9.4	2.2	1.5	0.0
1	107.0	13.0	1.1	0.9	3.1
1	106.0	4.2	1.2	1.6	1.4
1	110.0	11.3	2.3	0.9	3.3

Figure 2 : Affichage des instances dans l'interface graphique

Etude exploratoire des caractéristiques descriptives des données

Dans cette section, nous allons calculer le moyenne, mode, et la médiane, des attributs 'T3-resin uptake test', 'Total Serum thyroxin' et 'Total serum triiodothyronine', on déduire par la suite la distribution gaussienne de ces données, puis nous allons construire une boite à moustache pour chaque attribut, nous déduirons par la suite les outliers (valeurs aberrantes), ensuite, nous allons construire des histogrammes pour chaque attribut, nous fournirons à la fin des diagrammes de dispersion pour savoir si des corrélations existent ou non entre les attributs.

Moyenne

Pour les trois attributs, le calcul se fait de la même manière, on somme les valeurs de la colonne de l'attribut souhaité, et on divise cette somme sur le nombre d'instances total N.

La formule de la moyenne est donnée par :

$$X = \frac{\sum_{i=1}^{n} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_n}{N}$$

Médiane

Pour calculer la médiane d'un attribut, on trie d'abord les valeurs de cet attribut par ordre croissant, si le nombre d'élément est impair, alors la valeur au milieu de la liste sera la médiane, sinon, dans le cas où le nombre d'élément est impair, il n'y a pas de valeur unique en milieu de la liste, dans ce cas, la médiane est considérée comme la médiane est calculée par la formule :

$$median(x) = \frac{x_{\frac{N}{2}} + x_{\frac{N}{2}+1}}{2}$$

Mode

Le mode d'un attribut est considéré comme la valeur la plus fréquente pour cet attribut.

La figure 3 illustre les valeurs de la moyenne, mode, et la médiane calculés :

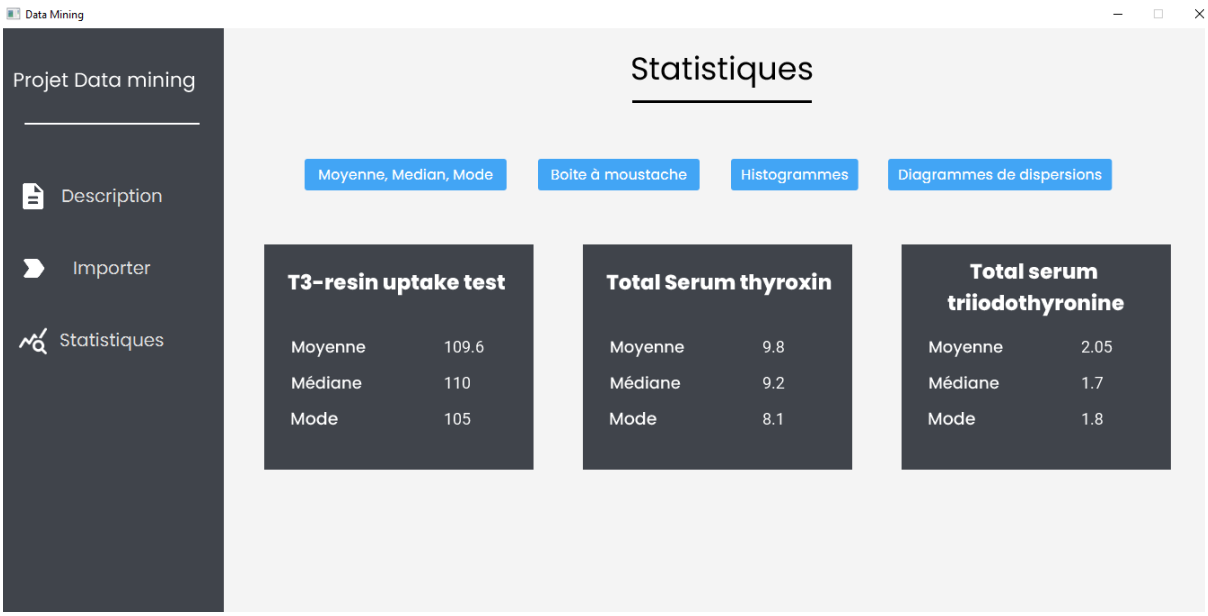


Figure 3 : affichage des valeurs du mode, médiane et la moyenne

Découverte de la symétrie

Une distribution est dite symétrique si les valeurs observées se répartissent de façon uniforme autour des trois valeurs centrales : la moyenne, le mode et la médiane. On distingue trois types de symétrie :

$$\begin{cases} symétrique \Rightarrow mode = médiane = moyenne \\ Asymétrie\ positive\ (droite) \Rightarrow mode < médiane < moyenne \\ Asymétrie\ négative\ (gauche) \Rightarrow mode > médiane > moyenne \end{cases}$$

Pour l'attribut T3-resin uptake test, la distribution est presque symétrique.

Pour l'attribut Total Serum thyroxin, on a une asymétrie positive.

Pour l'attribut Total serum triiodothyronine, on a une asymétrie positive.

Boites à moustaches

Les boites à moustache sont un outil très puissant pour détecter les outliers (valeurs aberrantes) dans notre dataset, pour chaque attribut, nous allons construire une boite à moustache, nous déduirons par la suite les outliers dans l'interface graphique.

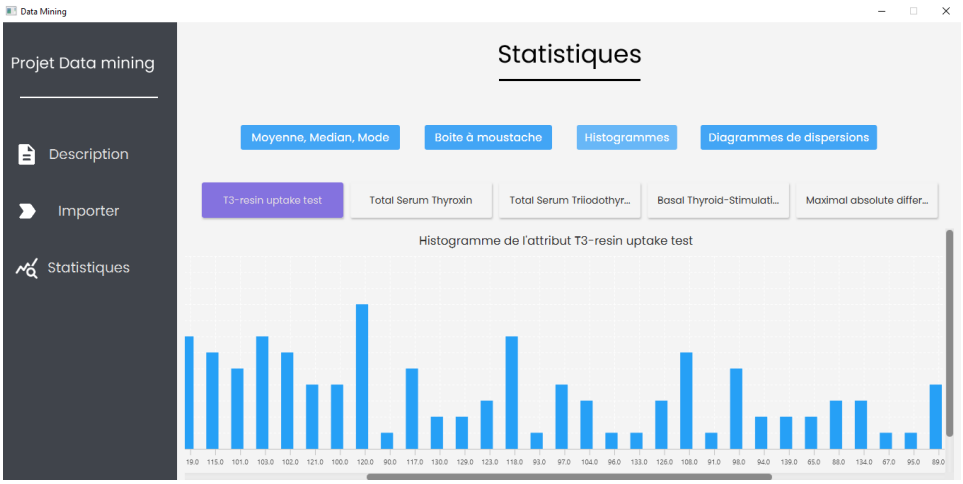
La figure 4 représente une boite à moustache pour l'attribut « T3-resin uptake test » :

Figure 4 : boite à moustache de l'attribut T3-resin uptake

Histogrammes

Les Histogramme sont très utiles dans l'analyse des données, ils permettent de visualiser la fréquence des valeurs des attributs, ainsi que leurs distributions ce qui nous permet d'extraire des informations comme l'asymétrie par exemple.

La figure 5 représente l'histogramme de l'attribut « T3-resin uptake test » :



La figure 5 : histogramme de l'attribut T3-resin uptake test

Diagrammes de dispersions

Le diagramme de dispersion, ou de corrélation, est un outil qui permet de vérifier l'existence de corrélation, ou d'une relation entre les variables, remarquer des regroupements de données, …

La corrélation la plus forte qu'on a trouvé est de coefficient 0.72 entre les deux attributs « Total Serum Triiodothyronine » et « Total Serum thyroxin »

La figure 6 illustre cette corrélation :

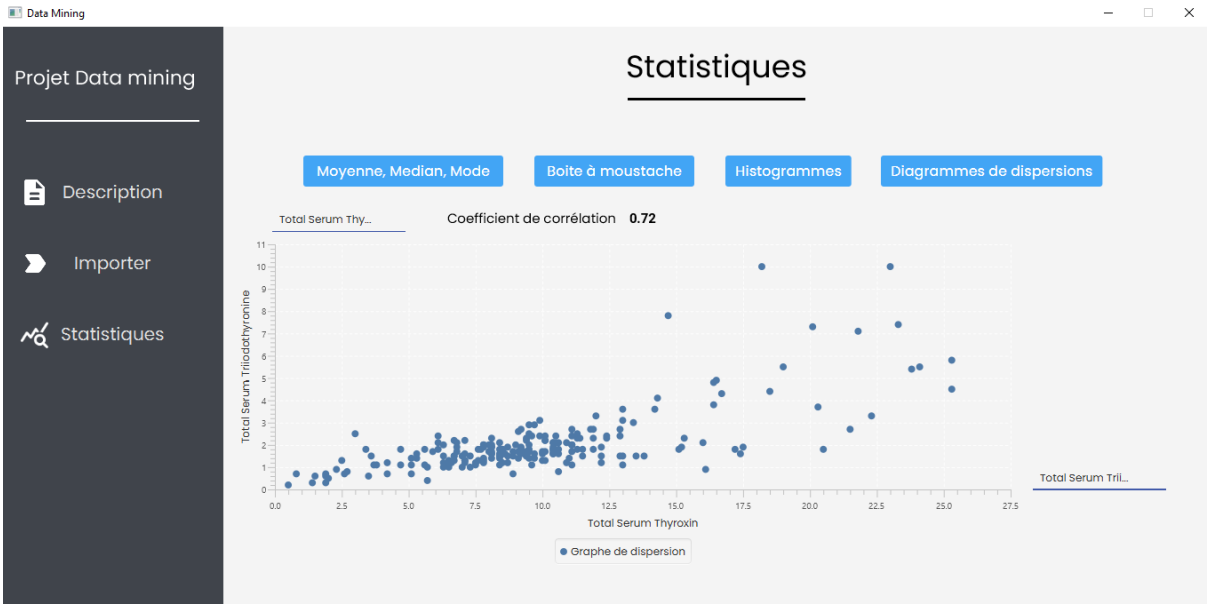


figure 6 : diagramme de dispersion entre l'attribut Total Serum Triiodothyronine et Total Serum thyroxin

Conclusion

Dans cette première partie, nous avons acquis beaucoup d'information en visualisant les données, nous avons appris à comprendre comment nos données se comporte schématisant sous forme de différents graphes et diagrammes.

2. Techniques du data mining

Discrétisation

La discrétisation est une technique qui permet de convertir les valeurs réelles dans notre dataset en valeur discrète avant de procéder à l'utilisation des techniques du data mining, en effet, certaines techniques nécessitent des valeurs qualitatives d'une part, d'autre part, les valeurs discrètes ont tendance à produire de bons résultats lors d'utilisation des techniques de DM.

Apriori

Afin d'implémenter l'algorithme apriori, nous avons utilisé les structures suivantes :

- **Motifs fréquents** : nous avons utilisé la structure Hashmap, qui prend comme clé, une chaine de caractère (cette chaine de caractère représente un motif), et comme valeur la fréquence de ce motif dans le dataset
- **Règles d'associations :** pour les règles d'associations, nous avons considéré la partie gauche d'une règle comme une chaine de caractères (représentant le motif), et la partie droite une Arraylist de chaine de caractères.
- **Dataset :** l'algorithme apriori utilise le dataset discrétisé, qui est structuré sous forme d'une Arraylist de Patient (l'objet patient contient tous les attributs d'une instance du dataset)

Algorithme Apriori
Entrée : dataset discrétisé, support, confiance **Sortie :** frequent itemsets et les règles d'association
Var **Liste :** liste de patient **L1, combinaison, fréquents** : hashmap<chaine de caractères, entier> **Temp :** chaine de charactères **K** : entier

Début
// Récupérer les données du dataset
Pour chaque instance du Dataset Faire
  Ajouter instance à liste ;
Fait ;
// Construction du premier niveau L1
Pour chaque instance dans liste Faire
  Pour chaque item dans instance Faire
  Si item ∉ L1 alors L1[item] = 1
  Sinon L1[item] = L1[item] + 1
  Fait ;
Fait ;
// Supprimer de L1 tous les items qui ont une fréquence inferieure au support
Pour chaque item dans L1 Faire

```
   Si L1[item] < support alors supprimer item de L1 ;
Fait ;
// Construction des niveaux suivants
K := 2
Tant que (L1 ! = ∅) Faire
   Initialiser combinaison
   Pour chaque item_1 dans L1 Faire
      Pour chaque item_2 dans L1 différent de L1 Faire
         Temp := concaténer(item_1, item_2) ;
         Trier Temp ;
         Si (Temp.taille = K) alors
            combinaison[Temp] := 0 ;
      fait
   fait
   // Calculer la fréquence de chaque itemset dans combinaison
   Pour chaque instance dans liste Faire
      Pour chaque itemset dans combinaison Faire
         Si itemset ⊂ liste alors
            combinaison[itemset] = combinaison[itemset] + 1
      fait
   fait
   // Supprimer de combinaison les itemset dont la fréquence < support
   Pour chaque itemset dans combinaison Faire
      Si combinaison [itemset] < support alors supprimer itemset de combinaison ;
   Fait ;
   L1 := combinaison
   Si (L1 != ∅) alors fréquents := L1
   K++ ;
Fait ;
```

// Génération des règles

On répète même traitement pour construire les itemsets sauf qu'on supprime la partie de calcul des supports et on varie la longueur des parties gauches de 1 à longueur (motif fréquent) -1. Une fois toutes les possibilités construites et ajoutées à fréquents.

```
Pour chaque élément dans fréquents Faire
   On crée une nouvelle règle ;
   On met élément dans la partie gauche de la règle ;
   Pour chaque item dans fréquents Faire
      Si item n'est pas dan élément alors ajouter item à la partie droite de la règle ;
   Fait ;
   Ajouter règle à l'ensemble des règles
Fait ;
```

```
// Calcul des confiances
Pour chaque règle dans l'ensemble des règles Faire
  Support_1 := support de partie gauche + droit ;
  Support_2 := support partie gauche ;
  cnf := (support1/support2)*100 ;
  si (cnf >= confiance) alors on sélectionna cette règle
fait
Fin.
```

K-means

Afin d'implémenter l'algorithme k-means, nous avons utilisé les structures suivantes :

- **Cluster** : hashmap, dont la clé est de type Patient, et la valeur est une ArrayList de Paitent.

```
Algorithme K-means
Entrée : dataset, K
Sortie :  K-clusters
Var
Liste : liste de patient
Cluster : hashmap (Patient, liste de patient)
Début
Récupérer les instances depuis dataset et les mettre dans liste
Shuffle(liste) ;
Prendre K points aléatoirement de liste et les mettre dans clusters
Tant que non convergence Faire
  Pour chaque poids de liste qui n'est pas un centroid Faire
    Calculer distance de ce point avec chaque centroid ;
    Affecter ce point au cluster de centroid le plus proche ;
  Pour chaque cluster dans clusters Faire
    Mettre à jour le centroid du cluster par la moyenne
  Fait
  Si aucun cluster n'a changé alors l'algorithme a convergé
Fait ;
Fin
```

K-medoid

Afin d'implémenter l'algorithme k-medoid, nous avons utilisé les structures suivantes :

- **Cluster** : hashmap, dont la clé est de type Patient, et la valeur est une ArrayList de Paitent.

```
Algorithme K-medoid
Entrée : dataset, K
Sortie :  K-clusters
```

```
Var
Liste : liste de patient
Cluster : hashmap (Patient, liste de patient)
Début
Récupérer les instances depuis dataset et les mettre dans liste
Shuffle(liste) ;
//initialiser les clusters.
Pour i allant de 1 à k :
    On crée un nouveau cluster.
    On sélectionne aléatoirement un entier entre 0 et longueur du DataSet.
    On vérifie s'il n'a pas été déjà choisi comme centre.
    On l'affecte au centre du cluster créé et On ajoute le cluster à Clusters.
    Fait

Pour chaque poids de liste qui n'est pas un centroid Faire
    Calculer distance de ce point avec chaque centroid ;
    Affecter ce point au cluster de centroid le plus proche ;
    fait

Tant que non convergence Faire
[2]  Pour chaque cluster recupure centroid Faire
    [3]  Pour chaque objet de cluster faire
            On créer new_clusters qui a comme centroid les ancienes centres  et en remplace
             Le centroid actuelle par l'objet.
            Pour chaque objet
                on affecte au cluster de new_clusters le plus proche

             cost 1 = Calculé le cout d'anciene clusters
            cost 2 = Calculé le cout de new_clusters
            si ( cost 2 < cost 1) alors
                remplacé le cluster actuelle par new_clusters
                sortir des deux boucle [2] [3]
        fait
    fait
     si (centroids d'ancienne cluster = centroids new clusters)alors
          l'algorithme est converge.
Fait ;
Fin
```

Clarans

Afin d'implémenter l'algorithme clarans, nous avons utilisé les structures suivantes :

- **Cluster** : hashmap, dont la clé est de type Patient, et la valeur est une ArrayList de Paitent.

```
Algorithme Clarans
Entrée : voisin, itération, k, dataset
Sortie : cluster
Début
  i := 1 ;
  coût := infini ;
  courant := k médoïdes pris aléatoirement ;
  j := 1 ;
  Tant que (i <= itération) Faire
    S := voisin de courrant ;
    Si (cout(S) < Cout(courant) alors
      Courrant := S ;
      J := 1 ;
    Sinon
      J++ ;
      Si (j >= voisin) alors
        Si (cout(courant) < coût) alors
          Coût := coût(courant) ;
          Meilleur := courant ;
        I++ ;
        courant := k médoïdes pris aléatoirement ;
      Fsi ;
    Fsi ;
  Fait ;
Retourner Meilleur ; // meilleur médoïdes
```

Comparaison entre les algorithmes :

Pour la comparaison on a prend k = 3 que nous a permit de calcule le f-measure et pour chaque algorithmes on a fait l'execution 8 fois et calculé la moyenne.

Tableau de Distances (Costs)

Algorithmes	Costs								Moy
K means	1963.14	1978.29	1967.41	1952.82	1972.32	1974.27	1963.06	1969.011	1967.540
K medoid	1920.32	1885.43	1885.43	1920.32	1885.43	1920.32	1885.43	1920.32	1902.875
Clarans	403.36	312.53	351	417	368.19	329.36	333.51	416.86	366.476

Graph associe au tableau

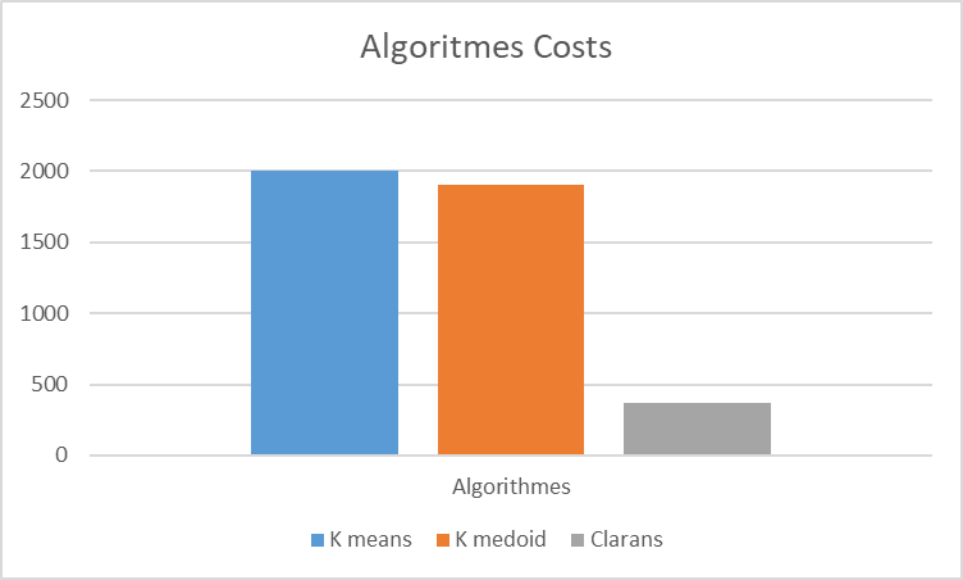


Tableau de f-measure

Algorithmes	F-meas								Moy
K means	0.861	0.84	0.86	0.853	0.859	0.859	0.858	0.851	0.855125
K medoid	0.688	0.642	0.642	0.688	0.642	0.688	0.642	0.688	0.665
Clarans	0.6	0.618	0.669	0.651	0.586	0.744	0.75	0.623	0.655125

Graph associe au tableau

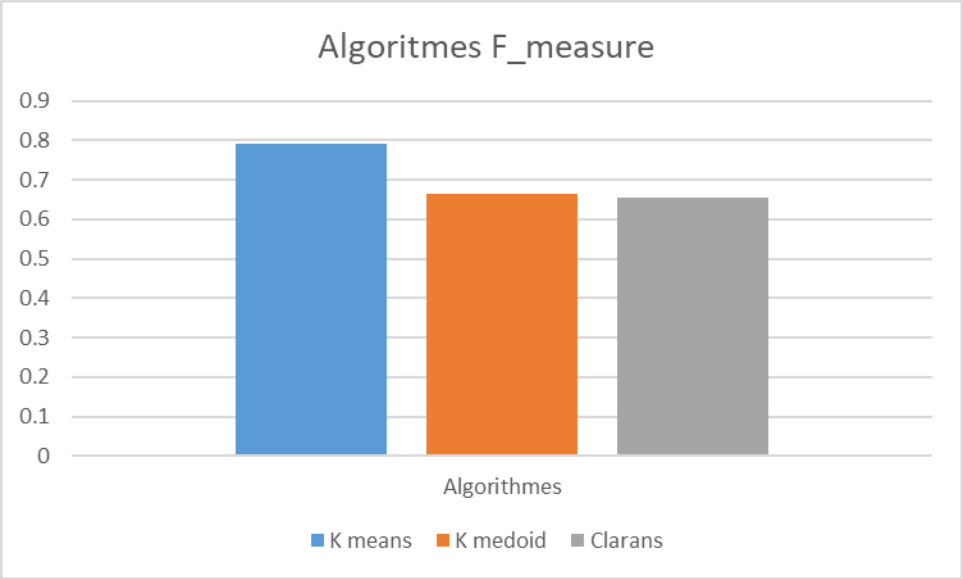
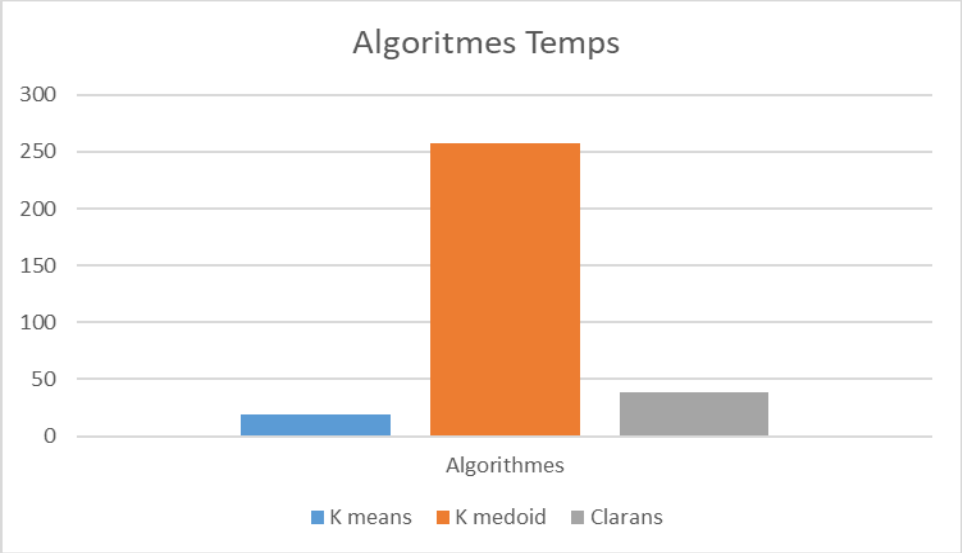


Tableau de Temps

Algorithmes	F-meas								Moy
K means	24	26	19	17	17	30	7	13	19.125
K medoid	556	349	152	131	296	297	135	140	257
Clarans	77	45	30	33	28	24	34	40	38.875

Graph associe au tableau



Pour le même K = 3 on remarque que l'algorithme Clarans est le meilleure Algorithme sans aucune doute si on prends le facteur de distance on considération,et on a K medoid il minimse le cout mieux que K means dans ce case là, mais au contraire il prend beaucoup de temps,par rapport à les 2 autres il n'est pas un grande diffrence de temps mais k means il est mieux, maintenant pour le f-measure on a que le K means il est meilleur que les 2 autres par 20% et il n'est pas une grande diffrence entre K medoid et clarans donc il sont égale.

Maintenant pour les différentes valeur de K

Tableau de costs

K	4	5	6	7	8	15	20	50
K means	1648.57	1426.4	1317.49	1215.76	1171.67	842.07	714.58	418.51
K medoid	1635.93	1478.07	1337.71	1239.74	1187.25	894.42	773.32	517.59
Clarans	319.13	365.89	396.23	397.87	336.79	367.26	393.54	354.78

Graph associe au tableau

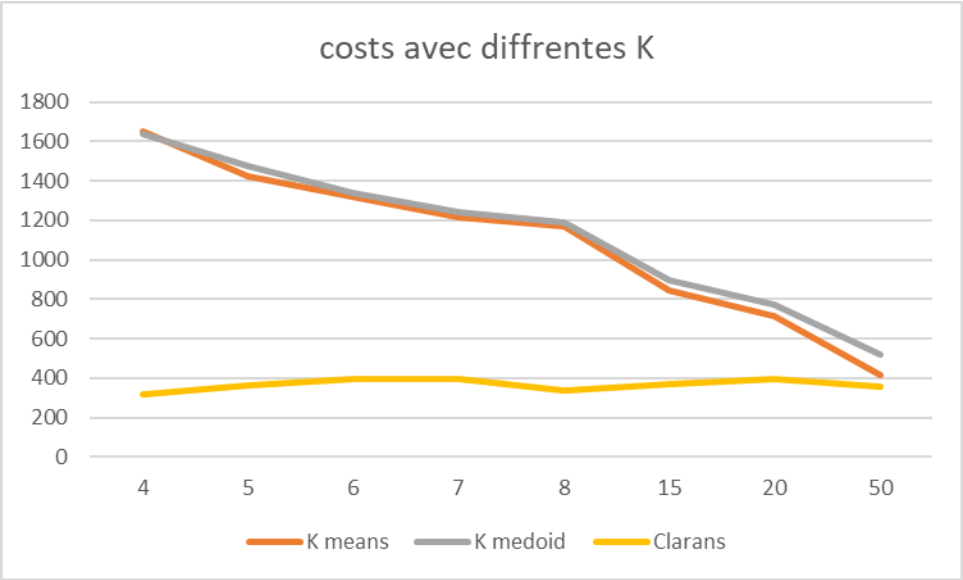
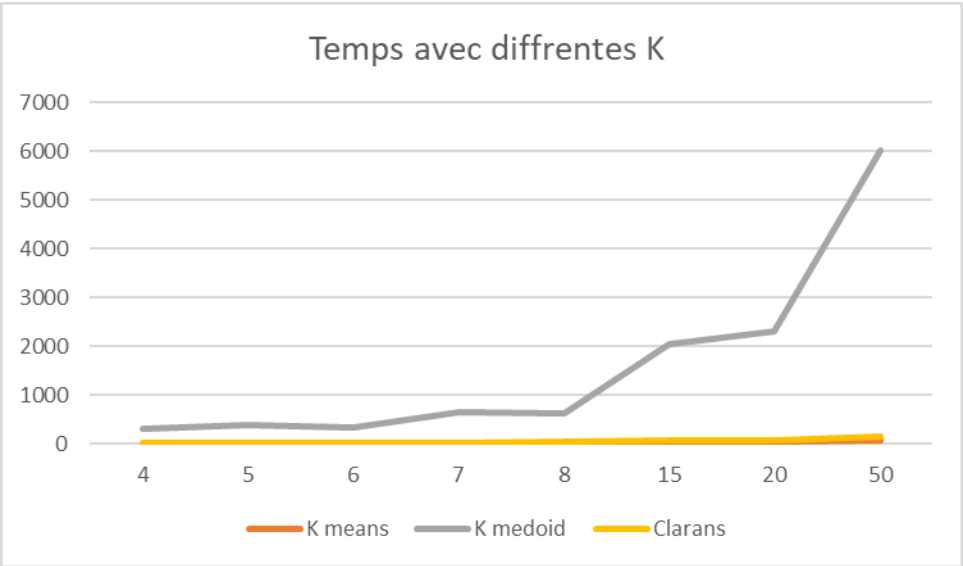


Tableau de temp

K	4	5	6	7	8	15	20	50
K means	8	9	17	12	19	30	47	67
K medoid	321	399	345	659	634	2040	2324	6029
Clarans	29	32	31	32	37	64	80	147

Graph associe au tableau



On remarque que le cots de l'algorithme clarance reste fix malgré le changment de k au contraire au 2 autre algorithme il sont diminués lorsque le k est grand, par rapport au temps d'execution on a que les 3 algorithme le d'execution il sera plus lorsque on augmente la valeur de k mais k medoid est très grand a cause de leur complexité $O(n^2)$.

Conclusion

- ✓ K-means est un algorithme de classification très rapide au vu de son implémentation simple comme on a pu le voir. Mais il manque de précision et il est sensibles aux données aberrantes, car les centres sont changés par des valeurs qui n'existent pas dans le DataSet et les données aberrantes comptent dans le calcul de la moyenne, mais aussi le coût n'est pas pris en compte pour le changement du centre ce qui fait que les clusters finaux dépendront uniquement d'une base aléatoire (répartition de départ).

- ✓ K-medoids, en revanche, il palie au problème de précision du K-means. Les coûts sont pris en compte dans le changement des centres, ce qui fait qu'on ne change de centre que pour un meilleur centre. Donc, on obtient un meilleur clustering. Aussi, les centres sont des membres du DataSet donc notre base de comparaison est un élément du DataSet, ce qui donne un clustering plus fiable. Cependant, comme on a pu le constater le temps d'exécution du K-medoids est bien plus conséquent que celui du K-means pour un même nombre de clusters et le même dataSet

- ✓ Clarans, c'est l'algorithme qui combine entre les avantages des deux algorithmes de classification précédents (K-means et K-medoids) le premier en rapidité et le deuxième en qualité du clustering.