K-Means Clustering

-Partitionnement en k moyennes-

Rapport du Projet

Réalisation

Ouadie EL FAROUKI

Année Universitaire 2016/2017

Table des matières

I. INTRODUCTION

La POO (Programmation Orientée Objet) consiste en la définition et l'interaction de briques logicielles dites « objets » ; un objet représente un concept, une idée ou toute entité du monde physique (ex. véhicule, personne, portefeuille financier...). Il possède une structure interne et un comportement, et il sait interagir avec les autres objets. Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre certains problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle. Pour ce fait, plusieurs outils sont utilisés, notamment Le langage de Modélisation Unifié ou l'UML (Unified Modeling Language).

La POO n'a donc pas de particularité technique. En effet, il s'agit plutôt d'une manière de concevoir/modéliser que d'une façon de coder. A ce titre, on pourrait envisager de "Coder objet" avec presque n'importe quel langage, et inversement, utiliser un langage objet sans avoir besoin de l'exploiter comme tel.

II. K-Means Clustering

1 - Analyse des Données

L'**analyse des données** est une famille de méthodes statistiques dont les principales caractéristiques sont d'être multidimensionnelles et descriptives et dont l'outil mathématique majeur est l'algèbre matricielle, et qui s'exprime sans supposer a priori un modèle probabiliste.

Ces méthodes trouvent leurs applications, qui s'avèrent en fait efficaces, dans pas mal de domaines, notamment en sociologie, en économie, en biologie, en microfinance, en marketing…

2 - Analyse par Segmentation

L'analyse par segmentation est un type d'analyse de données servant à partitionner les données. Étant donnés des points et un entier *k*, le problème est de diviser les points en *k* groupes, souvent appelés *clusters*, de façon à minimiser une certaine fonction. On considère la distance d'un point à la moyenne des points de son cluster, la fonction à minimiser est la somme des carrés de ces distances.

La segmentation fait partie des algorithmes dits non supervisés (unsupervised) qui signifie que les segments à obtenir ne sont pas définis à l'avance, autrement dit, la méthode cherche seulement à répartir les données selon des segments d'individus par critère de proximité. Les algorithmes dits supervisés (supervised) cherchent plutôt à joindre les individus à des classes prédéfinies et les regrouper ainsi, et ce par critère de proximité ou similitudes.

Parmi les algorithmes de segmentation les plus connus, on trouve la méthode dite « méthode *des* k-*moyennes* » (K-means clustering).

3 - K-means Clustering

1. Définition :

Étant donné un ensemble de points ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_n$), on cherche à partitionner les *n* points en *k* ensembles $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ ($k \leq n$) en minimisant la distance entre les points à l'intérieur de chaque partition :

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

μi : étant la moyenne (barycentre) des points dans le cluster Sj

2. Algorithme :

L'algorithme classique pour le problème de segmentation par la méthode des k-moyennes peut être décrit en 3 étapes :

a. Choix des du nombre de clusters (segments) à considérer
b. Positionnement à priori des 'Centroids' (centre des clusters)
c. Calcul des distances Centroids-Points du plan

d. Groupement des points par critère de proximité à leurs Centroids
e. Calcul/correction des nouveaux Centroids des groupements (Clusters) obtenus
f. Itération en retournant à l'étape c ; l'algorithme est arrêté une fois les Centroids ne bougent plus (équilibre atteint).

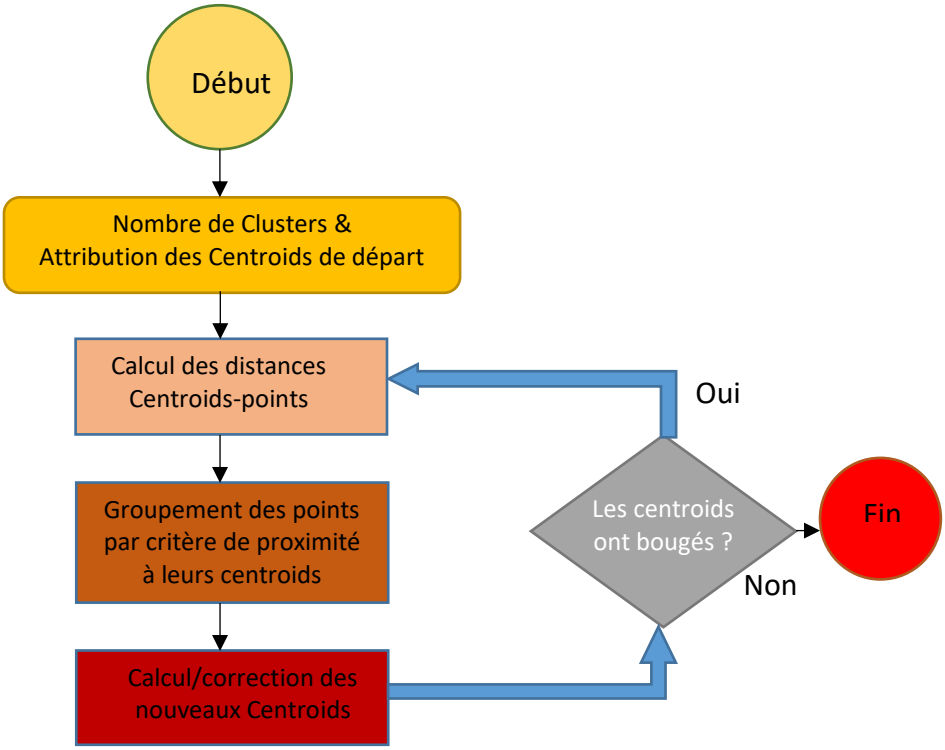

Figure 1 : Schéma d'un algorithme K-means Classique

Le résultat après un nombre fourni d'itérations (ou un arrêt automatique si l'équilibre est atteint) est un ensemble de clusters (nuage de points proches) centrés autour de leurs Centroids, et qui ne sont autre que les barycentres de chaque cluster.

Les groupes d'individus (points) obtenus formeront ainsi des segments homogènes de la population, une sorte de catégories rassemblant les points avec des similitudes.

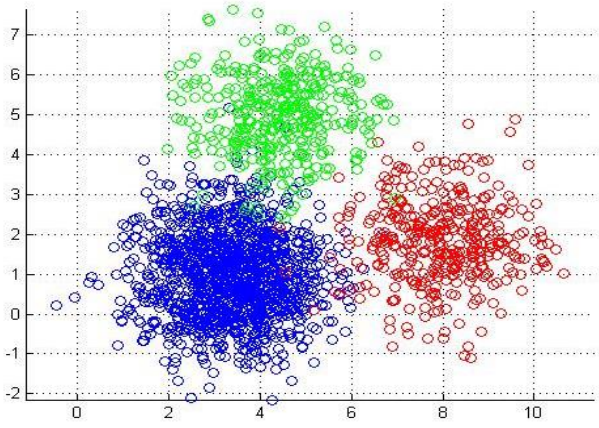

Figure 2 : représentation graphique de résultat d'un *k-Means clustering* avec 3 clusters

III. Programme JAVA

Implémentation

La classe Point

1. Les variables d'instances et les champs de classe :
   - x et y de types double, représentent les coordonnées du point
   - Id de type int, renvoi à l'Identifiant du point, il est positif pour les points ordinaires et négatif pour les centroids
   - Cluster de type int, représente le numéro du cluster auquel le point appartient
   - nombreP de type int, renvoi le nombre d'objets de type Point crées, c'est un champ statique car il dépend de la classe et non pas de chaque objet

2. Les Constructeurs :
   - Il y'en a 3, un constructeur sans argument, ordinaire, et par recopie

3. Les méthodes usuelles
   - set..() Méthodes d'accès aux variables (x, y, Id et Cluster)
   - get..() Méthodes d'altération/initialisation des variables (x, y, Id et Cluster)
   - affiche() Méthode d'affichage pour afficher les valeurs des variables, elle dépend de l'Id pour distinguer entre point et centroid

4. Les méthodes de calcul (fonctions)
   - distance(Point p) renvoi la distance entre l'objet actuel (argument implicite) et le point passé en argument explicite
   - listDist(Point…points) renvoi un tableau contenant la distance entre le point actuel et chaque point de l'ellipse (Point…) ou tableau(Point[]) passé en argument explicite.
   - coincide(Point p) renvoi un boolean indiquant la coincidence ou pas de p avec le point actuel

5. Les méthodes de classe (statiques)
   - distance(Point p1,Point p2) renvoi la distance entre les 2 points passés en argument
   - echange(Point p1, Point p2) échange les 2 points passés en argument
   - Mean(int Cluster, Point...points) renvoi le barycentre (nouveau centroid, donc un objet de type Point) des points appartenant au même cluster, ces derniers ont en effet une valeur int de Cluster unique, d'où le passage de celle-ci en argument pour préciser le cluster.
   - genP(int Xmax, int Ymax, int NB) génère un nombre NB de points aléatoires dont les bornes en x sont comprises entre (-Xmax,+Xmax) et les bornes en y entre (-Ymax,+Ymax)

Code de la classe Point

```
import java.util.Random;
class Point {
        //Déclaration des champs (propres aux objets) & champs de classe (propres aux classes)
        private double x,y;
        private int Id=0;
        private int Cluster=0;
        private static int nombreP=0;

        //Constructeurs
        //1-Constructeur sans argument
        public Point(){x=0.; y=0.; Id=0;}

        //2-Constructeur à arguments explicites
        public Point(double x, double y, int Id){
                this.x=x ; this.y=y ; this.Id=Id ;
                if (Id>0) {        Point.nombreP++ ; }
        }

        //3-Constructeur à argument de classe Point
        public Point(Point p){
                this.x=p.x;
                this.y=p.y;
                this.Id=p.Id;
        }
        //-----------------------------------------------------

        //Méthodes ordinaires (non statiques)
        //1-Méthodes d'accès (get) et Méthodes d'Altération (set)
        public double getX() {return(this.x); }
        public double getY() {return(this.y); }
        public void setY(double y) {  this.y = y; }
        public void setX(double x) {  this.x = x; }
        public void setCluster(int cl) {          this.Cluster=cl; }
        public int getCluster() { return(  this.Cluster); }
        public void setId(int id) {  this.Id=id; }
        public int getId() {       return(this.Id); }
        public int getNB(){ return(nombreP); }

        //2-Méthodes d'affichage et méthodes de calcul (fonctions)
        //Affichage
        public void affiche(){
                if (this.Id>0){
```

```
                    System.out.println("les coordonnées du point "+this.getId()+" sont
("+x+","+y+")");
                }
                if (this.Id<0){
                    System.out.println("les coordonnées du centroid "+(-this.getId())
+"sont ("+x+","+y+")");
                }
        }

        //Calcul de distance
        public double distance(Point p){

                double carré=(this.x-p.x)*(this.x-p.x)+(this.y-p.y)*(this.y-p.y);
                return(Math.sqrt(carré));
        }

        //Calcul de distance entre le point et un ensemble de points, retourne un tableau
unidimensionnel
        public double[] listDist(Point...points){
                double distances[]=new double[points.length];
                int i=0;

                for (Point p:points){
                        distances[i]=this.distance(p);
                        i++;
                }
                return(distances);
        }

        //Comparaison entre points, test de coincidence
        public boolean coincide(Point p){
                return((p.x==this.x) && (p.y==this.y));
        }

        //-----------------------------------------------------

        //Méthodes de classe (statiques)

        //Distance entre 2 points
        public static double distance(Point p1,Point p2){
                double carré=(p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y);
                return(Math.sqrt(Math.abs(carré)));
        }
        //Echange de 2 points
        public static void echange(Point p1, Point p2){
                double X=p1.x;
                double Y=p1.y;
```

```
            int ID=p1.Id;
            p1.x=p2.x;
            p1.y=p2.y;
            p1.Id=p2.Id;
            p2.x=X ; p2.y=Y ; p2.Id=ID ;
        }
        //Calcul du barycentre (de Type Point) des points appartenant au cluster (passé en argument) parmi les points (passés en argument)
        public static Point Mean(int Cluster, Point...points){
            double Xg=0,Yg=0;
            int n=0;
            for (Point p:points){
                if (Cluster==p.getCluster()){
                    n++;
                    Xg=Xg+p.getX();
                    Yg=Yg+p.getY();
                }
            }
            Point Centre=new Point(Xg/n,Yg/n, Cluster);
            return(Centre);
        }
        //Générateur aléatoire de points
        public static Point[] genP(int Xmax, int Ymax, int NB){
            Random randX= new Random();
            Random randY= new Random();
            Random randSigne= new Random();
            Point[] points=new Point[NB];
            for (int i = 0; i < NB; ++i){
                int signe=randSigne.nextInt(2);
                if (signe==0) signe=-1;
              double randomx = signe*randX.nextInt(Xmax);
              signe=randSigne.nextInt(2);
                if (signe==0) signe=-1;
              double randomy = signe*randY.nextInt(Ymax);
               points[i]=new Point(randomx,randomy,i+1);
              }
            return(points);
    }
}
```

La classe *MatriceManip*

La classe MatriceManip contient une seule méthode et qui est statique :

minCol(double[][] mat, int col) : elle prend en argument une matrice MN,P de 'double', calcule le min(M[i][j])i∈[1 :N] pour chaque j dans [1 :P] et renvoi l'indice i de celui-ci pour le stocker ensuite dans un tableau de longueur P. Autrement dit, elle renvoi un tableau d'indices des minimums des colonnes.

En effet, cette fonction servira à exploiter la matrice des distances Centroids-Points, une matrice K*NP dont les lignes sont les tableaux des distances Centroids-points obtenus par la méthode statique de la classe Point listDist(Point [] points) (voir Classe Point).

MatriceManip est utilisée ainsi dans l'étape d : « Groupement des points par critère de proximité à leurs centroids », en effet les indices renvoyés ne sont autre que les Id des Centroids (avec un signe moins, on rappelle que les Centroids ont des Id<0), ces Id négatifs serviront à réinitialiser les valeurs des variables d'instance 'clusterId' des points de la population, et donc regrouper les points en clusters : les points appartenant au même cluster ont la même valeur dans leur variable d'instance 'clusterId'.

Clarification :

Le point **P** le plus proche du Centroid **C** se voit attribuer l'Id de ce Centroid (un Id négatif) non pas dans son propre Id (qui est toujours positif !), mais dans son clusterId.

Code de la classe MatriceManip

```
public class MatriceManip {
	public static int minCol(double[][] mat, int col){
		int i=0,j=1;
		int indice=0;
		bloc :
		while(i<(mat.length-1)){
			while(j<mat.length && mat[i][col]<mat[j][col]){
				if (j+1==mat.length) break bloc;
					else j++;
			}
			i=j;
			indice=j;
			j++;
		}
		return(indice);
	}	}
```

Le programme Principal

Les opérations effectuées au sein de la fonction main() sont les suivantes :

1) Saisie* du nombre NB de points de la population à utiliser. *: *(Une classe 'Clavier', semblable à la classe 'Saisie' sera exploitée pour gérer les input)*
2) Choix en (Y/N) de la génération aléatoire de ces points ou bien la saisie manuelle
   -Si c'est aléatoire, on utilise la fonction genP de la classe Point
   -Si c'est manuel, on fait entrer les coordonnées des points
3) Affichage des points par la méthode affiche() de la classe Point
4) Saisie du nombre des clusters à considérer et des coordonnées des centroids à placer en premier lieu. Les centroids se voient attribuer des Id négatif pour les distinguer des points ordinaires.
5) Affichage des Id des centroids
6) Début de la boucle while, (ré)initialisation de la Matrice des distances en parcourant pour chaque centroid tous les points ordinaires, la fonction listDist s'occupe de cette opération
7) Attribution des Id des centroids aux clusterId des points proches, et création ainsi de segmentations caractérisées par un clusterId et un centroid, la fonction minCol de la classe MatriceManip d'occupe de cette opération
8) Calcul des nouveaux barycentres des clusters et déplacement des anciens centroids vers ces nouveaux centroids à la fonction Mean de la classe Point. Il existe un test d'arrêt dans cette étape : si les centroids ne bougent plus (c'est-à-dire que les anciens et les nouveaux coïncident), il y'a sortie de la boucle while
9) Affichage des centroids actuels à l'aide de la fonction affiche de la classe Point
10) Affichage pour chaque point du cluster auquel il appartient et ce en accédant à son clusterId

Code du programme Principal

```
//Ecrit dans le fichier kMeans.java
public class kMeans {
	public static void main(String[] stringss){
		//Saisie des points à utiliser
		double x,y;
		System.out.println("entrez le nombre de points à utiliser");
		int NB=Clavier.lireInt();
		Point[] points=new Point[NB];
```

```
		System.out.println("Voulez-vous entre manuellement les coordonnées des points Y/N?");
		char réponse=Clavier.lireChar();
		if (réponse=='Y'){
			int j=1;
			for (int i=0;i<NB;i++){
				System.out.println("entrez x du P"+(1+i));
				x=Clavier.lireDouble();
				System.out.println("entrez y du P"+(1+i));
				y=Clavier.lireDouble();
				points[i]=new Point(x,y,j);
				j++;
			}
		} else if (réponse=='N') {
			System.out.println("Les points seront générés aléatoirement");
			System.out.println("Entrez la borne max comme entier de vos X");
			int Xmax=Clavier.lireInt();
			System.out.println("Entrez la borne max comme entier de vos Y");
			int Ymax=Clavier.lireInt();
			points=Point.genP(Xmax, Ymax, NB);
		}

		System.out.println("Voici mes points");
		for (int i=0;i<NB;i++){
			points[i].affiche();
		}

		//Choix du nombre de clusters à considérer
		System.out.println("Entrez le nombre de Clusters à considérer, il ne doit pas dépasser le nombre des points choisis");
		int K=Clavier.lireInt();

		//Choix du nombre d'itérations
		System.out.println("Entrez le nombre max d'itérations");
		int iterations=Clavier.lireInt();


		//Saisie des coordonnées des centroids
		Point[] centroids=new Point[K];
		int f=0;
		System.out.println("*****entrée des coordonnées des centroids*****");
		for (int i=0;i<K;i++){
			System.out.println("entrez x");
			x=Clavier.lireDouble();
			System.out.println("entrez y");
			y=Clavier.lireDouble();
			f=f-1;
```

```
                centroids[i]=new Point(x,y,f);
            }

            //Affichage des Id des centroids
        System.out.println("voyons voir les Ids de nos centroids");
            for (int k=0;k<K;k++){
                System.out.println("Id[C"+(k+1)+"]= "+centroids[k].getId());
            }

            //Itérations
            double[][] Matrice=new double[K][];
            int j=0;
            int déplacés=K;
            while (j<iterations && déplacés!=0){
                for (int i=0;i<K;i++){
                    Matrice[i]=(centroids[i]).listDist(points);
                }

                //Attribution des numéro de clusters aux points
                for (int i=0;i<points.length;i++){
                    int Cluster=centroids[MatriceManip.minCol(Matrice,i)].getId();
                    points[i].setCluster(Cluster);
                }

                //Calcul des nouveaux centres des clusters et correction des anciens
clusters par ces nouveaux
                for (int i=0;i<K;i++){
                    if (centroids[i].getX()==(Point.Mean(centroids[i].getId(),
points)).getX() && centroids[i].getY()==(Point.Mean(centroids[i].getId(), points)).getY()){
                        déplacés=déplacés-1;
                    }
                    centroids[i].setX((Point.Mean(centroids[i].getId(),
points)).getX());
                    centroids[i].setY((Point.Mean(centroids[i].getId(),
points)).getY());
                }

                //Affichage des coordonnées des centroids après (j+1)ème itération
                System.out.println("après la "+(j+1)+" ème correction");
                for (int i=0;i<K;i++){
                    centroids[i].affiche();
                }
                //Affichage des clusters auxquels les points appartiennent après
(j+1)ème itération
                System.out.println("Clusters des points après "+(j+1)+" itérations");
                for (int k=0;k<NB;k++){
```

```
                                System.out.println("Cluster[P"+(k+1)+"]=
"+points[k].getCluster());
                        }
                        if (déplacés==0){
                                System.out.println("Vous avez atteint l'équilibre des
centroids");
                        }
                        j++;
                }
        }
}
```

Difficultés rencontrées

Bien que l'algorithme des K-Moyennes soit une méthode plutôt simple à saisir vu qu'elle ne présente pas vraiment de complexité mathématique, l'implémentation d'une telle méthode en utilisant un langage de programmation orienté Objet tel que Java fait relever certaines difficultés. Voici les 2 difficultés principales rencontrés ainsi que les approches proposées et qui ont servi de solutions :

Problème : L'algorithme est censé 'joindre' des points ordinaires à des points dits 'centroids', les groupes des points liés au même centroid forment un cluster, il fallait donc trouver une manière pour distinguer entre un Point ordinaire et un centroid en sachant que tous les 2 sont de classe Point, et en même temps créer ce lien entre points du cluster à partir du centroid à proximité.

Solution : Création d'une variable d'instance int Id positive pour les points ordinaires et négative pour les centroid. Création d'une autre variable clusterId qui sert à créer le point commun entre les points du même cluster, elle se voit attribuer en effet l'Id du centroid qui rassemble ce cluster, une sorte de signature du centroid qui touche les points les plus proches.

Problème : Pour chaque point ordinaire, l'algorithme doit trouver le centroid le plus proche. Le point reçoit l'Id de ce centroid qui sera, comme on a déjà vu, l'étiquette du cluster auquel il appartient, ce qui le lie aux autres points du même cluster.

Solution : Après calcul des distances centroid-points, il fallait non pas retourner la valeur minimale pour chaque colonne (distance point P-centroids), mais l'indice de la ligne où se trouve cette valeur, ce qui revient à dire l'Id x(-1) de ce centroid.

Test du programme

1) Cas où les points sont entrés manuellement

```
entrez le nombre de points à utiliser
10
Voulez-vous entre manuellement les coordonnées des points Y/N?
Y
entrez x du P1
3
entrez y du P1
2
entrez x du P2
4
entrez y du P2
0
entrez x du P3
6
entrez y du P3
8
entrez x du P4
0
entrez y du P4
6
entrez x du P5
-4
entrez y du P5
-2
entrez x du P6
-5
entrez y du P6
3
entrez x du P7
-2
entrez y du P7
7
entrez x du P8
4
entrez y du P8
-3
entrez x du P9
10
entrez y du P9
5
entrez x du P10
9
```

```
entrez y du P10
-7
Voici mes points
les coordonnées du point 1 sont (3.0,2.0)
les coordonnées du point 2 sont (4.0,0.0)
les coordonnées du point 3 sont (6.0,8.0)
les coordonnées du point 4 sont (0.0,6.0)
les coordonnées du point 5 sont (-4.0,-2.0)
les coordonnées du point 6 sont (-5.0,3.0)
les coordonnées du point 7 sont (-2.0,7.0)
les coordonnées du point 8 sont (4.0,-3.0)
les coordonnées du point 9 sont (10.0,5.0)
les coordonnées du point 10 sont (9.0,-7.0)
Entrez le nombre de Clusters à considérer, il ne doit pas dépasser le nombre des points choisis
2
Entrez le nombre max d'itérations
6
*****entrée des coordonnées des centroids*****
entrez x
5
entrez y
5
entrez x
-1
entrez y
1
voyons voir les Ids de nos centroids
Id[C1]= -1
Id[C2]= -2
```

```
après la 1 ème correction
les coordonnées du centroid 1sont (7.0,2.0)
les coordonnées du centroid 2sont (-0.5,1.8333333333333333)
Clusters des points après 1 itérations
Cluster[P1]= -1
Cluster[P2]= -2
Cluster[P3]= -1
Cluster[P4]= -2
Cluster[P5]= -2
Cluster[P6]= -2
Cluster[P7]= -2
Cluster[P8]= -2
Cluster[P9]= -1
Cluster[P10]= -1
après la 2 ème correction
les coordonnées du centroid 1sont (6.6,0.6)
les coordonnées du centroid 2sont (-1.6,3.2)
Clusters des points après 2 itérations
Cluster[P1]= -2
Cluster[P2]= -1
Cluster[P3]= -1
Cluster[P4]= -2
Cluster[P5]= -2
Cluster[P6]= -2
Cluster[P7]= -2
Cluster[P8]= -1
Cluster[P9]= -1
Cluster[P10]= -1
après la 3 ème correction
les coordonnées du centroid 1sont (6.0,0.8333333333333334)
les coordonnées du centroid 2sont (-2.75,3.5)
Clusters des points après 3 itérations
Cluster[P1]= -1
Cluster[P2]= -1
Cluster[P3]= -1
Cluster[P4]= -2
Cluster[P5]= -2
Cluster[P6]= -2
Cluster[P7]= -2
Cluster[P8]= -1
Cluster[P9]= -1
Cluster[P10]= -1
```

```
après la 4 ème correction
les coordonnées du centroid 1sont (6.0,0.8333333333333334)
les coordonnées du centroid 2sont (-2.75,3.5)
Clusters des points après 4 itérations
Cluster[P1]= -1
Cluster[P2]= -1
Cluster[P3]= -1
Cluster[P4]= -2
Cluster[P5]= -2
Cluster[P6]= -2
Cluster[P7]= -2
Cluster[P8]= -1
Cluster[P9]= -1
Cluster[P10]= -1
Vous avez atteint l'équilibre des centroids
```

Le test de saisie manuel comprend les opérations suivantes :

- Saisie du nombre de points à utiliser et leurs coordonnées. (ici NB=10)
- Affichage des coordonnées de ces points
- Saisie du nombre d'itérations maximales (ici itérations=6)
- Saisie du nombre de Clusters à former (ici K=2)
- Saisie des coordonnées des Centroids de départ (ici (5,5) et (-1,1))
- Affichage des Id des Centroids (ces Id seront attribués aux champ clusterId des points appartenant au cluster du Centroid
- Itérations (max=j) : Affichage des coordonnées des Centroids après la jème itération
    - Affichage des clusterId des points du plan (ce qui revient à dire leurs Clusters)
    - Affichage du message d'équilibre en cas de sortie de la boucle avec j<itérations

On remarque à partir de cet exemple comment les points quittent un cluster pour passer à un autre au fur et à mesure que les distances centroids-points sont recalculés et les anciens Centroids déplacés (exemple du [P1] : -1, puis -2, puis -1 et stabilisation dans ce cluster). On remarque également comment les Centroids bougent (exemple de centroid1 : (7.0,2.0)->(6.6,0.6)->(6.0,0.83) et stabilisation en ce point).

L'algorithme s'arrête après 4 itérations, le nombre max d'itération étant saisi égale à 6.

2) Cas où les points sont générés aléatoirement

```
entrez le nombre de points à utiliser
50
Voulez-vous entre manuellement les coordonnées des points Y/N?
N
Les points seront générés aléatoirement
Entrez la borne max comme entier de vos X
15
Entrez la borne max comme entier de vos Y
20
Voici mes points
les coordonnées du point 1 sont (-5.0,-6.0)
les coordonnées du point 2 sont (0.0,-2.0)
les coordonnées du point 3 sont (-2.0,8.0)
les coordonnées du point 4 sont (3.0,-9.0)
les coordonnées du point 5 sont (6.0,18.0)
les coordonnées du point 6 sont (-1.0,17.0)
les coordonnées du point 7 sont (9.0,13.0)
les coordonnées du point 8 sont (-6.0,-7.0)
les coordonnées du point 9 sont (14.0,17.0)
les coordonnées du point 10 sont (3.0,-6.0)
les coordonnées du point 11 sont (7.0,-1.0)
les coordonnées du point 12 sont (-5.0,-2.0)
les coordonnées du point 13 sont (-6.0,-4.0)
les coordonnées du point 14 sont (10.0,18.0)
les coordonnées du point 15 sont (-9.0,-17.0)
les coordonnées du point 16 sont (3.0,10.0)
les coordonnées du point 17 sont (5.0,0.0)
les coordonnées du point 18 sont (-6.0,-19.0)
les coordonnées du point 19 sont (-8.0,-16.0)
les coordonnées du point 20 sont (2.0,11.0)
les coordonnées du point 21 sont (-12.0,-19.0)
les coordonnées du point 22 sont (6.0,-18.0)
les coordonnées du point 23 sont (0.0,15.0)
les coordonnées du point 24 sont (9.0,12.0)
les coordonnées du point 25 sont (-1.0,-11.0)
les coordonnées du point 26 sont (14.0,17.0)
les coordonnées du point 27 sont (0.0,17.0)
les coordonnées du point 28 sont (-11.0,3.0)
les coordonnées du point 29 sont (-5.0,-2.0)
les coordonnées du point 30 sont (-5.0,2.0)
les coordonnées du point 31 sont (14.0,14.0)
les coordonnées du point 32 sont (-12.0,7.0)
```

```
les coordonnées du point 33 sont (-8.0,-19.0)
les coordonnées du point 34 sont (5.0,12.0)
les coordonnées du point 35 sont (13.0,-18.0)
les coordonnées du point 36 sont (13.0,4.0)
les coordonnées du point 37 sont (-9.0,-9.0)
les coordonnées du point 38 sont (10.0,-10.0)
les coordonnées du point 39 sont (5.0,4.0)
les coordonnées du point 40 sont (-12.0,18.0)
les coordonnées du point 41 sont (9.0,-5.0)
les coordonnées du point 42 sont (-11.0,8.0)
les coordonnées du point 43 sont (7.0,13.0)
les coordonnées du point 44 sont (-11.0,9.0)
les coordonnées du point 45 sont (-2.0,-12.0)
les coordonnées du point 46 sont (-14.0,14.0)
les coordonnées du point 47 sont (2.0,-8.0)
les coordonnées du point 48 sont (-7.0,-11.0)
les coordonnées du point 49 sont (9.0,-12.0)
les coordonnées du point 50 sont (-7.0,0.0)
Entrez le nombre de Clusters à considérer, il ne doit pas dépasser le nombre des points choisis
3
Entrez le nombre max d'itérations
8
*****entrée des coordonnées des centroids*****
entrez x
5
entrez y
3
entrez x
-10
entrez y
-4
entrez x
10
entrez y
8
voyons voir les Ids de nos centroids
Id[C1]= -1
Id[C2]= -2
Id[C3]= -3
```

```
après la 1 ème correction
les coordonnées du centroid 1sont (5.384615384615385,-5.923076923076923)
les coordonnées du centroid 2sont (-7.818181818181818,-4.2272727272727275)
les coordonnées du centroid 3sont (7.0,13.866666666666667)
Clusters des points après 1 itérations
Cluster[P1]= -2
Cluster[P2]= -1
Cluster[P3]= -1
Cluster[P4]= -1
Cluster[P5]= -3
Cluster[P6]= -3
Cluster[P7]= -3
Cluster[P8]= -2
Cluster[P9]= -3
Cluster[P10]= -1
Cluster[P11]= -1
Cluster[P12]= -2
Cluster[P13]= -2
Cluster[P14]= -3
Cluster[P15]= -2
Cluster[P16]= -3
Cluster[P17]= -1
Cluster[P18]= -2
Cluster[P19]= -2
Cluster[P20]= -3
Cluster[P21]= -2
Cluster[P22]= -1
Cluster[P23]= -3
Cluster[P24]= -3
Cluster[P25]= -2
Cluster[P26]= -3
Cluster[P27]= -3
Cluster[P28]= -2
Cluster[P29]= -2
Cluster[P30]= -2
Cluster[P31]= -3
Cluster[P32]= -2
Cluster[P33]= -2
Cluster[P34]= -3
Cluster[P35]= -1
Cluster[P36]= -3
Cluster[P37]= -2
```

```
Cluster[P38]= -1
Cluster[P39]= -1
Cluster[P40]= -2
Cluster[P41]= -1
Cluster[P42]= -2
Cluster[P43]= -3
Cluster[P44]= -2
Cluster[P45]= -2
Cluster[P46]= -2
Cluster[P47]= -1
Cluster[P48]= -2
Cluster[P49]= -1
Cluster[P50]= -2
après la 2 ème correction
les coordonnées du centroid 1sont (4.928571428571429,-7.714285714285714)
les coordonnées du centroid 2sont (-8.263157894736842,-4.631578947368421)
les coordonnées du centroid 3sont (5.352941176470588,13.764705882352942)
Clusters des points après 2 itérations
Cluster[P1]= -2
Cluster[P2]= -1
Cluster[P3]= -3
Cluster[P4]= -1
Cluster[P5]= -3
Cluster[P6]= -3
Cluster[P7]= -3
Cluster[P8]= -2
Cluster[P9]= -3
Cluster[P10]= -1
Cluster[P11]= -1
Cluster[P12]= -2
Cluster[P13]= -2
Cluster[P14]= -3
Cluster[P15]= -2
Cluster[P16]= -3
Cluster[P17]= -1
Cluster[P18]= -2
Cluster[P19]= -2
Cluster[P20]= -3
Cluster[P21]= -2
Cluster[P22]= -1
Cluster[P23]= -3
Cluster[P24]= -3
```

```
Cluster[P25]= -1
Cluster[P26]= -3
Cluster[P27]= -3
Cluster[P28]= -2
Cluster[P29]= -2
Cluster[P30]= -2
Cluster[P31]= -3
Cluster[P32]= -2
Cluster[P33]= -2
Cluster[P34]= -3
Cluster[P35]= -1
Cluster[P36]= -3
Cluster[P37]= -2
Cluster[P38]= -1
Cluster[P39]= -1
Cluster[P40]= -3
Cluster[P41]= -1
Cluster[P42]= -2
Cluster[P43]= -3
Cluster[P44]= -2
Cluster[P45]= -1
Cluster[P46]= -2
Cluster[P47]= -1
Cluster[P48]= -2
Cluster[P49]= -1
Cluster[P50]= -2
après la 3 ème correction
les coordonnées du centroid 1sont (4.923076923076923,-8.615384615384615)
les coordonnées du centroid 2sont (-7.944444444444445,-5.666666666666667)
les coordonnées du centroid 3sont (4.315789473684211,13.263157894736842)
Clusters des points après 3 itérations
Cluster[P1]= -2
Cluster[P2]= -1
Cluster[P3]= -3
Cluster[P4]= -1
Cluster[P5]= -3
Cluster[P6]= -3
Cluster[P7]= -3
Cluster[P8]= -2
Cluster[P9]= -3
Cluster[P10]= -1
```

```
Cluster[P11]= -1
Cluster[P12]= -2
Cluster[P13]= -2
Cluster[P14]= -3
Cluster[P15]= -2
Cluster[P16]= -3
Cluster[P17]= -1
Cluster[P18]= -2
Cluster[P19]= -2
Cluster[P20]= -3
Cluster[P21]= -2
Cluster[P22]= -1
Cluster[P23]= -3
Cluster[P24]= -3
Cluster[P25]= -1
Cluster[P26]= -3
Cluster[P27]= -3
Cluster[P28]= -2
Cluster[P29]= -2
Cluster[P30]= -2
Cluster[P31]= -3
Cluster[P32]= -2
Cluster[P33]= -2
Cluster[P34]= -3
Cluster[P35]= -1
Cluster[P36]= -3
Cluster[P37]= -2
Cluster[P38]= -1
Cluster[P39]= -3
Cluster[P40]= -3
Cluster[P41]= -1
Cluster[P42]= -2
Cluster[P43]= -3
Cluster[P44]= -2
Cluster[P45]= -1
Cluster[P46]= -3
Cluster[P47]= -1
Cluster[P48]= -2
Cluster[P49]= -1
Cluster[P50]= -2
```

```
après la 4 ème correction
les coordonnées du centroid 1sont (4.923076923076923,-8.615384615384615)
les coordonnées du centroid 2sont (-7.944444444444445,-5.666666666666667)
les coordonnées du centroid 3sont (4.315789473684211,13.263157894736842)
Clusters des points après 4 itérations
Cluster[P1]= -2
Cluster[P2]= -1
Cluster[P3]= -3
Cluster[P4]= -1
Cluster[P5]= -3
Cluster[P6]= -3
Cluster[P7]= -3
Cluster[P8]= -2
Cluster[P9]= -3
Cluster[P10]= -1
Cluster[P11]= -1
Cluster[P12]= -2
Cluster[P13]= -2
Cluster[P14]= -3
Cluster[P15]= -2
Cluster[P16]= -3
Cluster[P17]= -1
Cluster[P18]= -2
Cluster[P19]= -2
Cluster[P20]= -3
Cluster[P21]= -2
Cluster[P22]= -1
Cluster[P23]= -3
Cluster[P24]= -3
Cluster[P25]= -1
Cluster[P26]= -3
Cluster[P27]= -3
Cluster[P28]= -2
Cluster[P29]= -2
Cluster[P30]= -2
Cluster[P31]= -3
Cluster[P32]= -2
Cluster[P33]= -2
Cluster[P34]= -3
Cluster[P35]= -1
Cluster[P36]= -3
Cluster[P37]= -2
```

```
Cluster[P38]= -1
Cluster[P39]= -3
Cluster[P40]= -3
Cluster[P41]= -1
Cluster[P42]= -2
Cluster[P43]= -3
Cluster[P44]= -2
Cluster[P45]= -1
Cluster[P46]= -3
Cluster[P47]= -1
Cluster[P48]= -2
Cluster[P49]= -1
Cluster[P50]= -2
Vous avez atteint l'équilibre des centroids
```

Le test de générateur aléatoire comprend les opérations suivantes :

- Saisie du nombre de points à utiliser (ici NB=50)
- Saisie des bornes max des abscisses et des ordonnées des points (x dans [15,15] et y dans [-20,20])
- Affichage des coordonnées de ces points
- Saisie du nombre d'itérations maximales (ici itérations=8)
- Saisie du nombre de Clusters à former (ici K=3)
- Saisie des coordonnées des Centroids de départ (ici (5,3) et (-10,-4))
- Affichage des Id des Centroids (ces Id seront attribués aux champ clusterId des points appartenant au cluster du Centroid

- Itérations (max=j) : Affichage des coordonnées des Centroids après la jème itération
    - Affichage des clusterId des points du plan (ce qui revient à dire leurs Clusters)
    - Affichage du message d'équilibre en cas de sortie de la boucle avec j<itérations

L'équilibre ici est atteint à la 4ème itération, donc bien avant le max permis d'itérations 8. On voit comment les Centroids évoluent d'une itération à la suivante ainsi que les clusters auxquels appartient chaque point.

IV. Conclusion

L'approche adoptée dans ce projet, dite approche distance, n'est qu'un cas particulier dans les solutions K-means, à côté de l'approche probabiliste (vraisemblance), l'approche prototype (un prototype/fonction qui dépend du type des individus de la population).

Le langage R qui est surtout dédié au statistiques et à l'analyse des données est doté d'une multitudes de fonctions d'analyse quantitative prête-à-utiliser, notamment le K-means clustering , Hierarchical Clustering… R est également doté de systèmes de représentations graphiques très riches et performants (packages comme ggplot2, lattice..) qu'on peut utiliser en parallèle avec les fonctions statistiques ce qui permet de fournir une meilleure visibilité du processus et des résultats de ce genre d'algorithmes.

La méthode des K-Means est jusqu'à aujourd'hui un outil connu pour son efficacité en analyse non-supervisée de données, surtout utilisé dans les premières étapes d'analyse à savoir l'exploitation des données, le Data Mining…