Compte-Rendu TP1

Machine Learning – Master 2 Informatique

Auteur(s) :
Romain CHARPENTIER
Morgane THIELEMANN
romain.charpentier@etu.univ-poitiers.fr
morgane.thielemann@etu.univ-poitiers.fr

6 novembre 2018

à rendre le Mardi 6 Novembre 8H, par mail en ZIP

I. Introduction

L'objectif de ce TP était de mettre en œuvre un algorithme K-mean sur divers jeux de données afin d'observer les forces et les limites de cet algorithme. Pour réaliser cet exercice, nous avons travaillé sur en Matlab.

L'algorithme Kmean en lui-même est codé dans le fichier « *coalescence.m* ». Le fichier « *generateData.m* » contient une fonction pour générer les données du premier exercice. « *erreur_classif.m* », et « *affiche_class.m* » sont des fonctions annexes fournies pour calculer le nombre d'erreurs dans la classification et afficher les résultats de l'algorithme de façon visuelle.

II. Réalisation de la fonction K_mean

Algorithme K-Mean : Fonction « kmean.m »

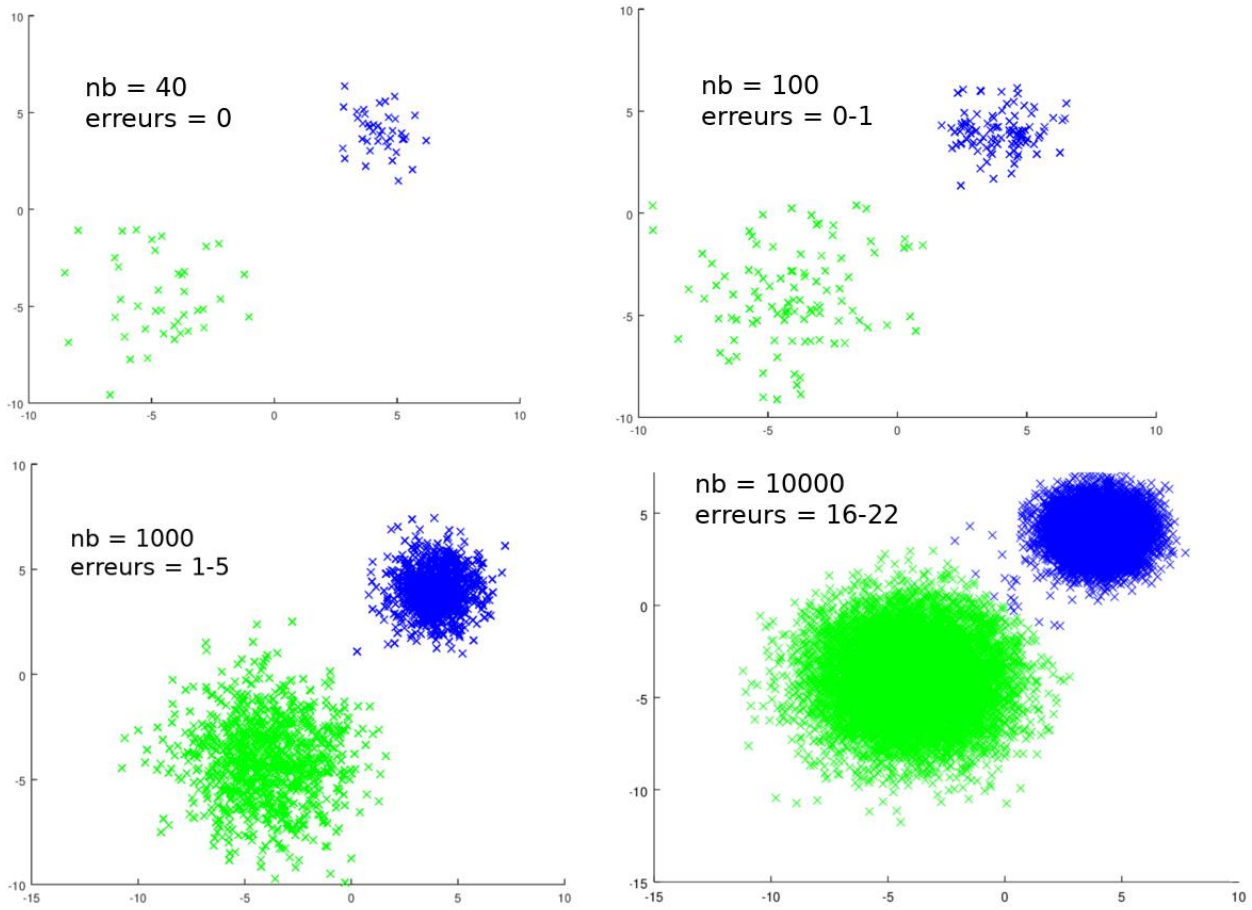
Le but de l'algorithme est de classer les colonnes d'une matrice x en K groupes, chaque colonne contient ici 2 informations (2 lignes). Chaque colonne de la matrice correspond donc à un point car elle possède 2 valeurs (x,y).

On va d'abord commencer par choisir K points au hasard (variable g). Ces points serviront d'initialisation à l'algorithme. Il faut ensuite pour chaque point de la matrice les réunir autour du point le plus proche. On peut pour cela calculer la norme entre les 2 points et on ne garde ainsi que le point avec lequel la distance est minimale.

Pour finir, nous allons calculer pour chaque groupe la moyenne des coordonnées. Ainsi les points de g se déplaceront vers l'endroit où les points sont le plus nombreux dans le groupe. Les points les plus éloignés pourront donc être classés dans des groupes différents lors de l'itération suivante. En effet, on rejoue l'algorithme jusqu'à ce que les points centraux des groupes (variable g) ne bougent plus.

```
function [clas,g2] = coalescence(x,K,M,g)
  dim = size(x);
  clas = 1:dim(2);
  g2 = zeros(2,K);
  g2_nb = zeros(1,K);
  for i = 1:dim(2)
    if K>0
      ## Pour chaque point, on le place dans le groupe dont la distance
      ## avec le point central est minimale
      res = 1;
      norm_min = (g(:,1)-x(:,i))'*M*(g(:,1)-x(:,i));
      for j = 2:K
        norm = (g(:,j)-x(:,i))'*M*(g(:,j)-x(:,i));
        if norm<norm_min
          ## On a trouvé une distance plus petite, donc on remplace le groupe
          res = j;
          norm_min = norm;
        end
      end
      g2(:,res) = g2(:,res) + x(:,i);
      g2_nb(res) = g2_nb(res) + 1;
      clas(i) = res;
    end
  end
  ## Calcul de la moyenne des groupes
  for i = 1:K
    if g2_nb(i)==0
      g2(:,i) = 0; ## Erreur : pas normal
    else
      g2(:,i) = g2(:,i) / g2_nb(i);
    end
  end
  ## Si les points centraux ont bougé, on relance l'algorithme
  if (g2!=g)
    [clas,g2] = coalescence(x,K,M,g2);
  end
end
```

En utilisant, la formule donnée en TP qui concatène deux tableaux de nombres aléatoires, on obtient les résultats présentés ci-dessous. Cette formule crée deux groupes de points, nous pouvons donc aisément créer l'oracle et observer le nombre d'erreurs (« erreurs » sur le schéma) pour un nombre « nb » de points.

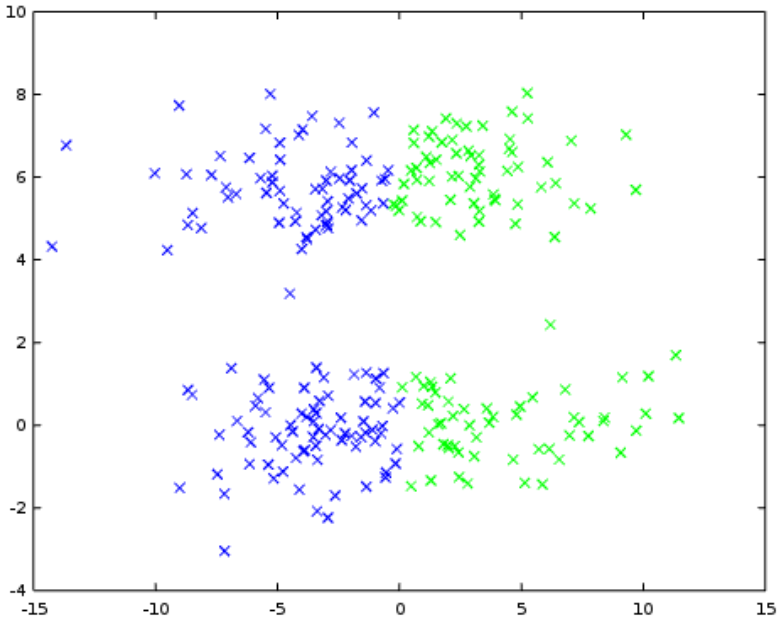


A l'œil nu, nous ne pouvons observer aucune erreur, en effet les points en erreur sont à la limite entre les deux groupes. Il est donc compréhensible que l'algorithme puisse se tromper. On peut donc en conclure que l'algorithme est fonctionnel.

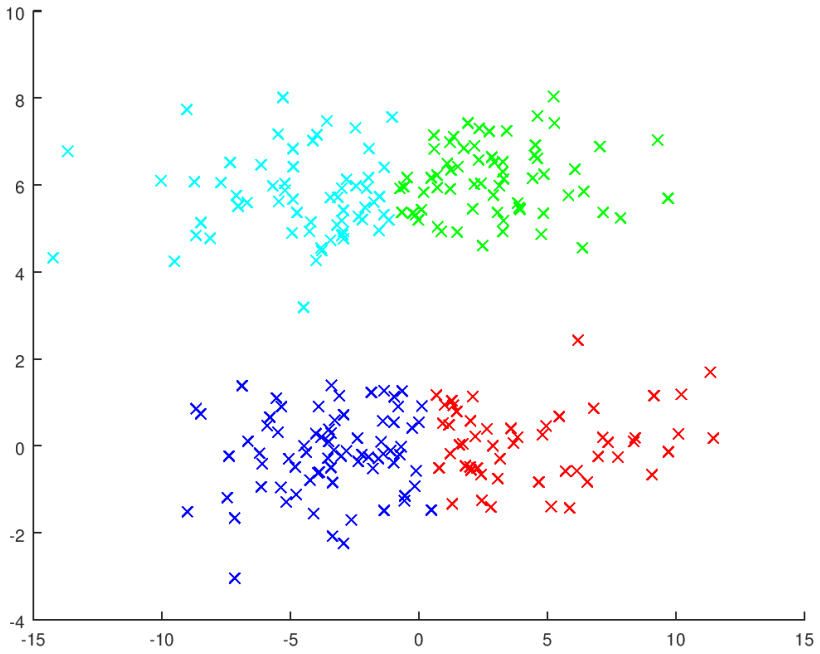
III. L'influence des centres initiaux

Grâce au fichier « *kmean_file_random.m* », on charge le premier jeu de données « td2_d1.txt ». On peut observer que les points ont l'air de se regrouper autour de 2 groupes horizontaux.

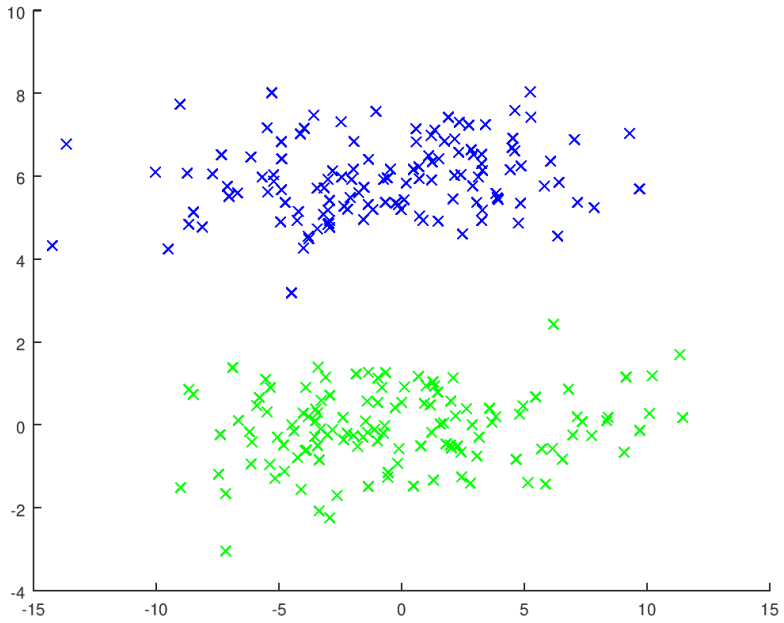
On joue donc l'algorithme avec des points g choisis au hasard et pour K=2 groupes. Le résultat obtenu est le suivant :



Malheureusement, l'algorithme a du mal à séparer les points de manière horizontale. En essayant avec d'autres points initiaux (choisis dans x de manière aléatoire), on peut observer que le résultat reste sensiblement le même, il est donc compliqué de trouver des points corrects.
En essayant avec K=4 groupes, on obtient ce rendu :



Le résultat est donc mieux car les groupes sont séparés également verticalement dans cet exemple. Cependant, il est impossible de savoir combien de groupes choisir avant de lancer l'algorithme et donc il devrait fonctionner de même manière qu'importe le nombre de groupes, cela nous montre les limites de cet algorithme.

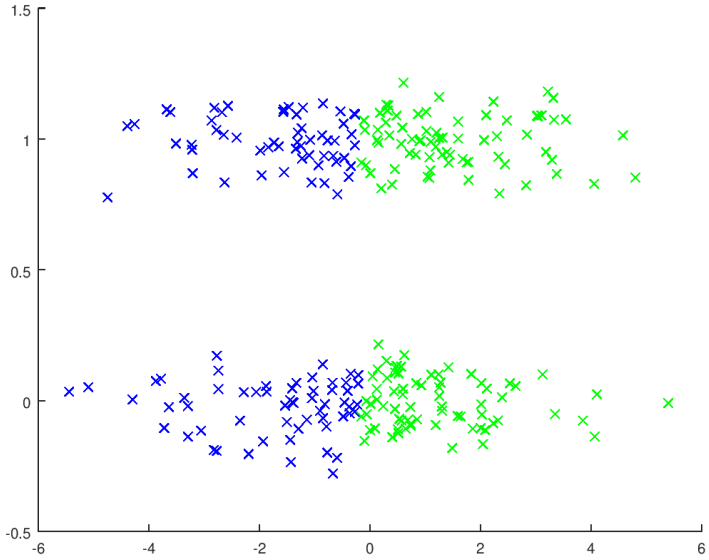


En choisissant les points, tel que *g = [0,0;6,0];* (donc choisis par l'utilisateur). On retrouve une séparation plus nette. Cela montre bien que le choix des points de départ a un impact fort sur le résultat que l'on obtient.

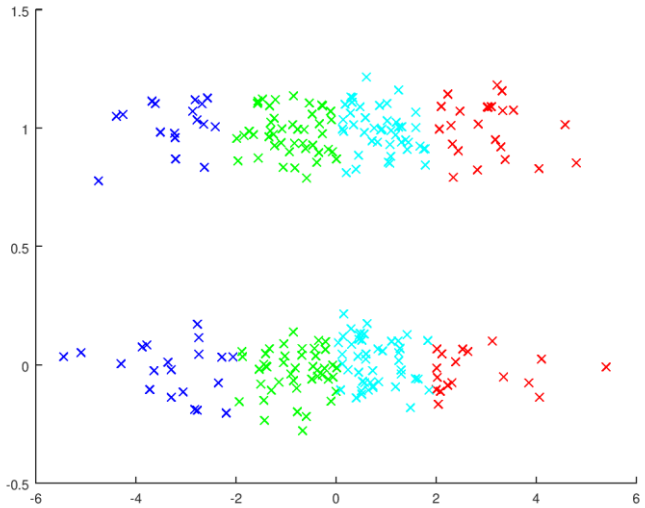
Par la suite, on utilisera donc un algorithme de choix des points initiaux légèrement différent de l'aléatoire de base. En effet, dans notre algorithme, nous choisirons un premier point au hasard. Nous calculerons ensuite les distances de chaque point avec ce premier centre. On lancera ensuite un deuxième aléatoire pondéré par les distances précédemment calculées. Nous avons ainsi plus de chance que nos centres initiaux soient clairement éloignés les uns des autres. Cet algorithme se trouve dans le fichier « *initialize_kmean.m* ».

IV.L’influence de la définition de la distance

On charge le fichier « td2_d2.txt ». On teste de nouveau, pour K=2 avec des points initiaux aléatoires.

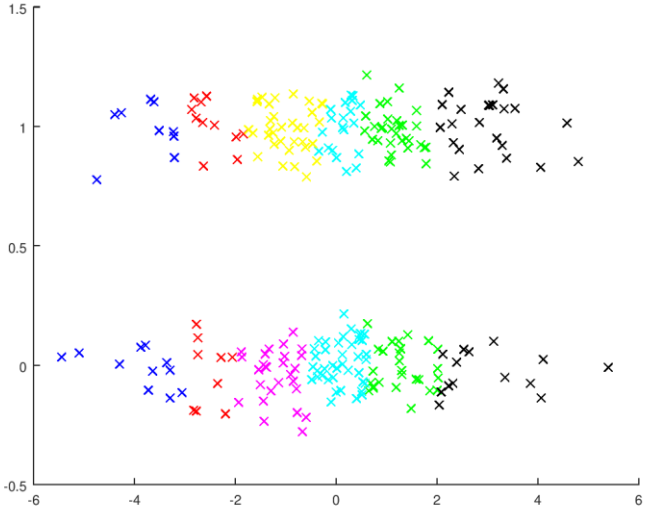


Le résultat est visuellement le même que pour le précédent jeu de données. On peut voir le même souci rencontré précédemment, à savoir que les groupes sont séparés de manière verticale et non horizontale comme on pourrait le penser.



K=2

Cependant, on peut observer que pour K=4 (image ci-dessus) que les résultats diffèrent. En effet, le jeu de données précédent donnait de meilleurs résultats (séparation horizontale et verticale) alors qu'ici on voit très clairement que l'algorithme ne sépare que verticalement. Ce qui démontre d'autant plus la faiblesse de l'algorithme qui peut ne pas fonctionner pour certains jeux de données. Il ne suffit pas de varier K pour corriger le souci. (Voir image ci-dessous pour K=7).



K=7

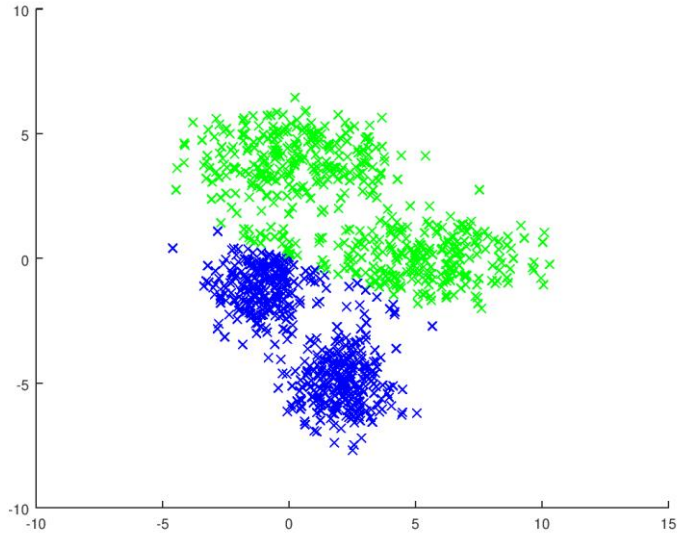
On essaie donc comme précédemment de choisir des points particuliers et non au hasard. Pour cela, on repère les coordonnées des centres des groupes de points. La méthode avait fonctionné auparavant mais pas cette fois-ci où l'on obtient le même résultat que pour des points choisis au hasard.

Cela s'explique sûrement par le peu d'écart de valeurs entre les points des 2 groupes. Sur le schéma, nous pouvons bien voir que les points se dispersent moins sur l'axe Y. Les mesures (X et Y) ne sont pas à la même échelle.

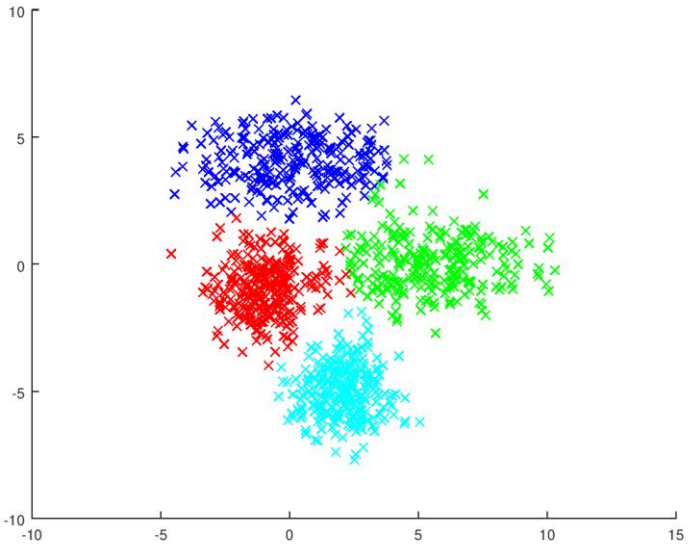
La solution serait donc de changer la valeur de la matrice de normalisation « M » qui est de base à (1,1;1,1) pour normaliser les données à la même échelle.

V. Le choix du nombre de classe

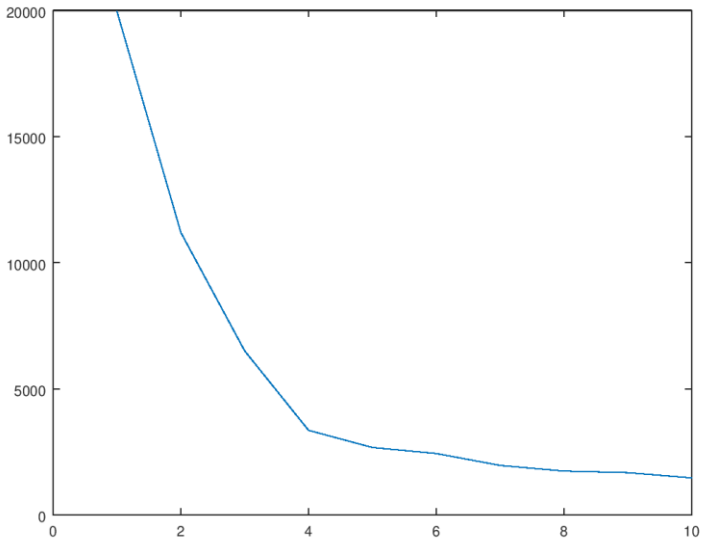
Pour cet exercice, on charge le troisième jeu de données. On obtient le résultat montré ci-dessous :



Qu'importe le placement des points initiaux de l'algorithme, on retrouve toujours un résultat semblable à celui-ci-dessus. Cela peut s'expliquer par le fait que l'on peut observer 4 pôles de points et non seulement 2. La séparation des groupes serait plus nette avec 4 groupes, on peut donc observer ce résultat pour K=4 (schéma ci-contre)

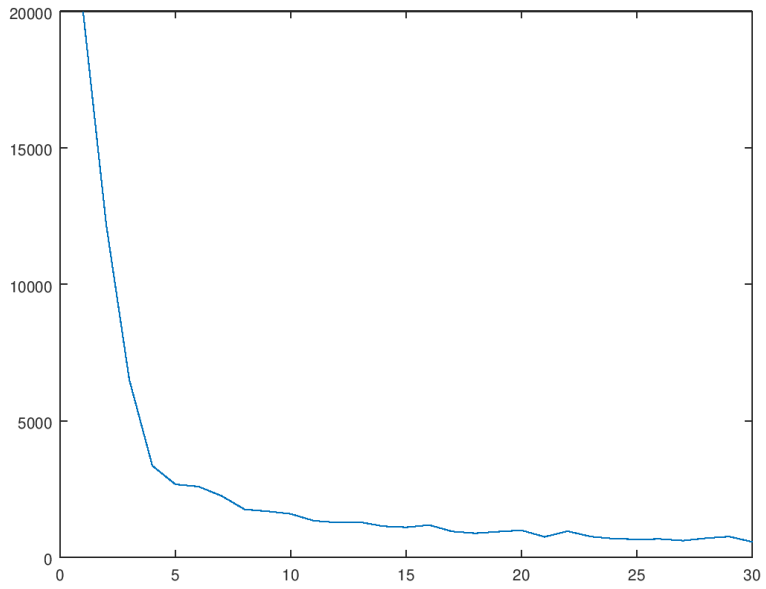


Afin de déterminer un nombre optimal de classe pour chaque jeu de données, nous décidons de coder une fonction de coût. Le principe de cette fonction est assez simple : on lance l'algorithme kmean un nombre k de fois (k étant un paramètre de la fonction). A chaque lancement de l'algorithme, on fait une moyenne des distances entre un point et son centre. Ceci nous permet de définir un « coût » pour chaque point. On lance donc cette fonction sur notre algorithme une première fois avec k=10. On obtient alors la courbe suivante :



On peut observer qu'il y a une nette diminution du coût quand k est entre 0 et 4 puis une diminution beaucoup moins importante ensuite. On peut donc déduire que le nombre de classe optimal pour ce jeu de données semble est situé autour de 4. Des valeurs supérieures donnent certes de résultats un tout petit peu meilleures mais la différence n'est pas suffisamment significative pour qu'elles soient vraiment optimales.

Afin de s'assurer qu'un résultat plus net n'est pas trouvable plus loin encore, on fait tourner l'algorithme 30 fois. On obtient une courbe qui nous indique clairement que la tendance vers 4 semble se confirmer.



VI. Conclusion

L'algorithme Kmean est un algorithme simple qui permet de discriminer plusieurs classes selon certains critères (ici la distance). Cependant cet algorithme n'est pas sans faille et présente des limites : différences de résultats selon le choix des centres initiaux, selon la définition de la distance, selon le nombre de classe…

VII. Annexes

Pour la fonction d'initialisation des centres, nous utilisons une bibliothèque permettant de faire de l'aléatoire pondéré (randsample). Voici les manipulations à effectuer pour la faire fonctionner :

- installer octave-statistics (« sudo apt-get install octave-statistics »)

- lancer la commande « pkg load statistics » dans Octave