IV

Algorithme des k moyennes (ou *k-means algorithm*) : utilisation pour le partitionnement en apprentissage automatique non supervisé

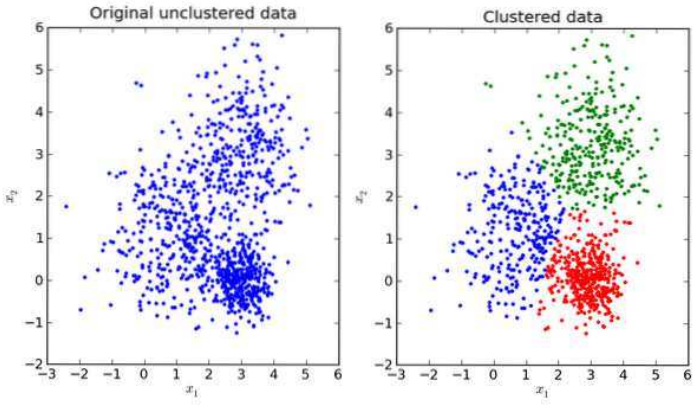


FIGURE IV.1 – Comment peut-on regrouper ces données en 3 "grappes" ou clusters ? (Source : fr.linuxteaching.com)

Le terme « k-means » a été utilisé pour la première fois par James MacQueen en 1967, bien que l'idée originale ait été proposée par Hugo Steinhaus en 1957. L'algorithme classique a été proposé par Stuart Lloyd en 1957 à des fins de modulation d'impulsion codée, mais il n'a pas été publié en dehors des Bell Labs avant 1982. En 1965, E. W. Forgy publia une méthode essentiellement similaire, raison pour laquelle elle est parfois appelée « méthode de Lloyd-Forgy ». Une version plus efficace, codée en Fortran, a été publiée par Hartigan et Wong en 1975/1979 (Source : Wikipedia*)*

PLAN DU CHAPITRE

I Position du problème et idée de l'algorithme 2
I.1 Principe de la classification non supervisée 2
I.2 Idée générale de l'algorithme des k-moyennes - premier exemple illustratif 2
I.3 Que peut-on faire avec l'apprentissage automatique non supervisé ? 3
II Formalisation du problème 4
II.1 Barycentre de classe, inertie intra-classe, inertie totale de partition 4
II.2 Algorithme des K-moyennes 6
III Exemple complet de mise en œuvre : classification d'un ensemble de vins 8

I Position du problème et idée de l'algorithme

I.1 Principe de la classification non supervisée

Dans le chapitre précédent, consacré à l'apprentissage supervisé, nous avons appris à exploiter un `dataset`, dont les étiquettes des éléments étaient toutes connues (on parle de **typologie** connue), pour affecter à un nouveau candidat l'étiquette qui lui «convenait le mieux» compte tenu de ses caractéristiques.

Nous allons ici nous intéresser à la `classification dite non supervisée` qui consiste en un ensemble de méthodes permettant d'établir ou de retrouver une typologie existante entre les N éléments d'un set connaissant simplement les n caractéristiques de chaque élément ; il s'agira de placer chaque élément dans une `classe` ou `groupe` appelé "cluster" ; on parle aussi de technique de `clustering` ou `classement en grappes`.

Une difficulté émerge immédiatement : le nombre de **clusters** n'étant pas connu à l'avance, il va falloir le poser, ou bien tester le résultat obtenu pour plusieurs valeurs de celui-ci et choisir celle qui assure le meilleur partitionnement.

EN RÉSUMÉ, UN ALGORITHME DE CLASSIFICATION NON SUPERVISÉ :

- exploite un `dataset` **non étiqueté** et...
- ...doit "inventer" les classes en regroupant les données dans k sous-ensembles (clusters) les plus homogènes possible, avec une finesse qui n'est souvent pas accessible à l'œil humain, d'où son intérêt.

I.2 Idée générale de l'algorithme des k-moyennes - premier exemple illustratif

On se donne un ensemble de données, chacune possédant n caractéristiques, sur lesquelles on souhaite réaliser un partitionnement en k clusters.
Pour cela, on va exploiter l'algorithme des $k-moyennes$ qui est l'une des méthodes les plus simples et qui est couramment utilisée ; son principe général est exposé sur l'exemple ci-dessous pour des données à deux caractéristiques, donc représentées par des points du plan (abscisse et ordonnée) :

1. on suppose qu'il existe $k=3$ classes.
2. on déclare arbitrairement k points comme étant les "barycentres" des k classes : c'est l'initialisation (*Initialization*).
3. l'algorithme itératif débute ensuite :
   (a) chaque point de donnée est affecté à la classe possédant le barycentre qui lui est le plus proche : c'est la première assignation des points. (*Assign Points (1)*)
   (b) les positions des barycentres des classes sont à nouveau calculées : c'est la première réassignation des centres (*Recompute Centers (1)*)
4. on itère ensuite ce processus (dans notre exemple la procédure est exécutée 3 fois) jusqu'à la validation d'un critère d'arrêt qui peut être :
   - nombre maximum d'itérations atteint
   - stabilité de l'état des classes : les points ne changent plus de classes, ou les positions des points ne varient plus de manière significative (quasi-stabilité de **l'inertie** intra-classes (la notion d'inertie est définie plus bas.))

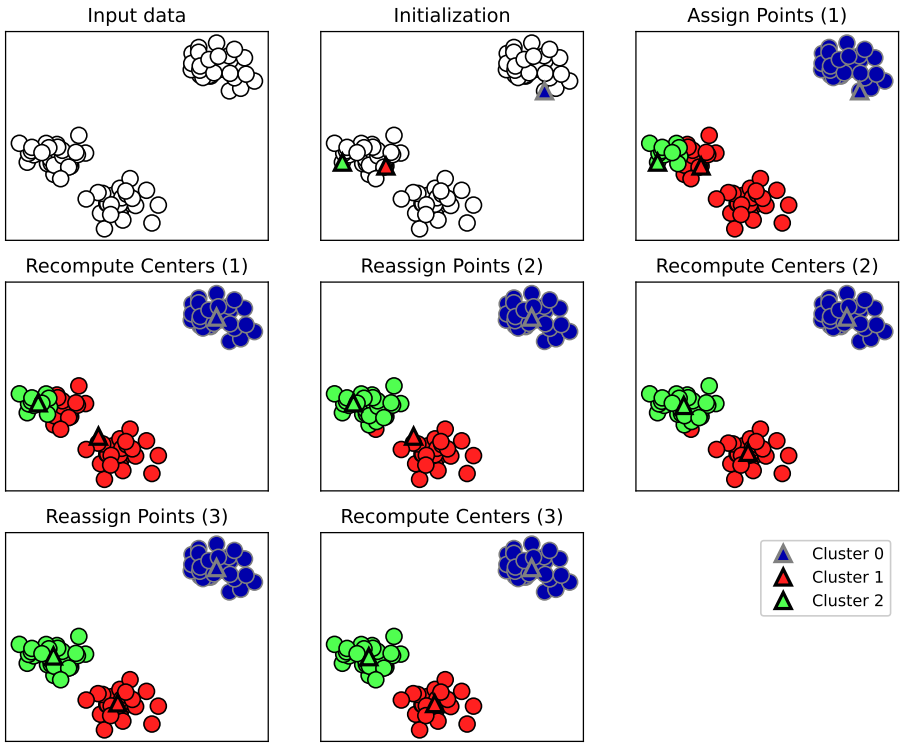
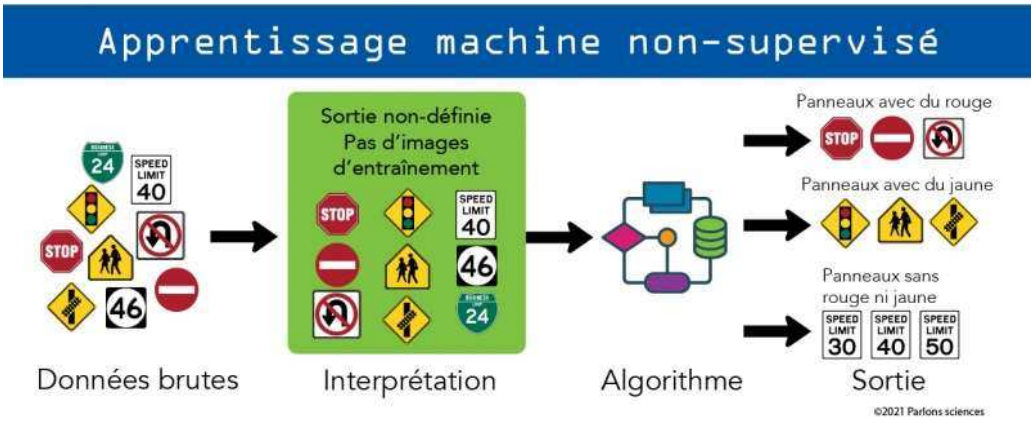


FIGURE IV.2 – Les différentes étapes du clustering (Source : `parlonssciences.ca`)

I.3 Que peut-on faire avec l'apprentissage automatique non supervisé ?

L'apprentissage automatique non supervisé peut être mis à profit dans deux nombreuses situations nécessitant de partitionner **finement** des données :

- faire de la segmentation de clientèle : par exemple un société de location de véhicules cherche à rapprocher les loueurs potentiels aux habitudes similaires (utilisation trajets maison-travail en semaine, et pas de trajets le week-end) dans un groupe et éloigner les autres clients aux habitudes différentes (utilisation véhicule uniquement le week-end) pour les placer dans un autre groupe ; cela permet alors de construire des offres bien ciblées : offre «semaine» ou offre «week-end» chacune présentant des tarifs étudiés pour leurs clients cibles...
- Les véhicules modernes possèdent un système de reconnaissance des panneaux de signalisation afin d'informer le conducteur d'un éventuel danger :
  - les panneaux comportant du rouge signale un danger potentiel immédiat et requièrent donc une attention de tout premier plan.
  - ceux comportant du jaune, signale un danger potentiel et sont également importants mais à un degré moindre.
  - enfin les autres panneaux qui doivent absolument être lus mais sans prévenir d'un danger potentiel.



II Formalisation du problème

II.1 Barycentre de classe, inertie intra-classe, inertie totale de partition

L'évaluation des algorithmes de classification nécessite de définir les notions de barycentre de classe/cluster et d'inertie intra-classe.

Considérons un sous-ensemble ou classe ou cluster A_p d'une partition $(A_p)_p$ d'une ensemble de données E. On notera $m_p = |A_p| = card(A_p)$.

Définition II-1: BARYCENTRE D'UNE CLASSE ET INERTIE INTRA-CLASSE

Si $|A_p| = m_p$, et X_j un élément de A_p avec $j \in [0, m_p - 1]$ possédant n caractéristiques, alors le barycentre G_p de cette dernière est défini par le vecteur colonne :

$$G_p = \frac{1}{m_p} \sum_{j=0}^{m_p-1} X_j = \frac{1}{m_p} \begin{pmatrix} \sum_{j=0}^{m_p-1} x_{0,j} \\ \sum_{j=0}^{m_p-1} x_{1,j} \\ \dots \\ \sum_{j=0}^{m_p-1} x_{n-1,j} \end{pmatrix}$$

On définit par ailleurs l'inertie intra-classe qui témoigne du niveau de dissimilarité des éléments de la classe par :

$$I_p = \sum_{j=0}^{m_p-1} ||X_j - G_p||^2 = \sum_{j=0}^{m_p-1} \sum_{i=0}^{n-1} (x_{i,j} - G_{p_i})^2$$

et de même la variance intra-classe :

$$v = \frac{I_A}{m_p}$$

Enfin, on définit également l'inertie totale de la partition E comme la somme des inertie intra-classe des classes de cette partition :

Définition II-2: INERTIE TOTALE D'UNE PARTITION

$$I = \sum_{p=0}^{k-1} I_p = \sum_{p=0}^{k-1} \sum_{X_j \in A_p} ||X_j - G_p||^2$$

Exercice de cours: **(II.1) - n° 1.** *On suppose que les données du* `dataset` *E sont implémentées dans une matrice X de dimensions (n, N) ; chaque colonne X_j de X représente donc un élément du set possédant n caractéristiques, avec N le cardinal du set.* **Attention :** *pour raison de commodité, cette implémentation est différente de celle adoptée dans le chapitre sur l'apprentissage supervisé dans lequel chaque ligne de la matrice X représentait un élément du set.*

1. *Ecrire une fonction* `barycentre(X:array)` $\longrightarrow$ `array` *qui renvoie la matrice colonne G dont le terme $G[i,0]$ est la moyenne des termes $X[i,:]$. On rappelle que la fonction* `np.sum(M)` *renvoie la somme des termes de M.*
2. *Ecrire une fonction* `inertie(P,X)` *qui prend en argument P, une liste de listes d'entiers et X, une matrice de flottants.*

STRUCTURE DES ARGUMENTS/SORTIE :

- *P est une partition des indices des colonnes de X. En d'autres termes, chaque sous-liste de P représente les indices des éléments de X qui constituent une famille/classe.*
- `inertie(P,X)` *renvoie l'inertie de la partition P de l'ensemble E.*

NB : *on rappelle que l'on peut extraire, par exemple les colonnes 1 et 2 d'une matrice X par slicing avec : $X[:,[1,2]]$*

RÉPONSE :

1. La première fonction donne :

Listing IV.1 – Fonction barycentre(X :array)

```
1 def barycentre(X):
2   n,N=X.shape
3   G=np.zeros((n,1),dtype=float)
4   for i in range(n): #itération sur lignes donc sur caractéristiques de chaque élément
5     Gi=0 #initialisation de la i-ième composante du barycentre
6     for j in range(N): #itération sur le nombre d'éléments du set
7       Gi+=X[i,j] #construction de la somme
8     G[i,0]=Gi/N #calcul de la i-ième composante du barycentre
9   return G
```

Autre proposition de code, un peu plus compacte (attention : les coûts cachés du slicing n'en font pas pour autant une solution plus efficace en terme de complexité temporelle) :

Listing IV.2 – fonction barycentre(X :array) version 2

```
def barycentre(X):
  n,N=X.shape
  G=np.zeros((n,1),dtype=float)
  for i in range(n):
    G[i,0]=np.sum(X[i,:])/N
  return G
```

❷ On obtient pour la seconde fonction :

Listing IV.3 – Fonction inertie(P :list,X :array)

```
def inertie(P,X):
  n,N=X.shape
  k=len(P) #nombre de partitions
  I=0
  for pk in P: # itération sur les classes de la partition P
    Ip=0 # initialisation de l'inertie intra-classe de la classe pk
    G=barycentre(X[:,pk]) # calcul du barycentre de la classe pk
    for j in pk: # boucle de calcul de l'inertie de la classe pk
      for i in range(n):
        Ip+=(X[i,j]-G[i,0])**2
    I+=Ip
return I
```

II.2 Algorithme des K-moyennes

On rappelle les étapes de l'algorithme des K-moyennes :

On dispose donc d'un `dataset` non étiqueté de N données possédant chacune n caractéristiques sous forme d'une matrice X (n,N) ; ces données peuvent être de toute nature, pourvu que l'on soit capable d'évaluer une notion de "similarité" ou bien la distance les séparant deux à deux (dans un espace de dimension n).

- on choisit le nombre de classes k
- **Initialisation** :
  - on initialise une liste P de k listes vides.
  - on désigne au hasard k données de X comme étant les barycentres Gp des k classes. On les place dans une liste $C = [G_0, G_1, \ldots, G_{k-1}]$.
- **Itération** :
  - pour chaque point de donnée $X_j = X[:,j]$, on calcule les k distances $d(X_j, G_p)$ avec $p = 0, \ldots, k-1$. Si la distance $d(X_j, G_{p_{min}})$ est minimale alors on place la donnée X_j dans la classe p_{min}, c'est à dire que **l'on place l'indice j de la donnée X_j dans la liste $P[p_{min}]$**.
  - on dispose alors d'une première partition P de l'ensemble des données, et on réévalue les k barycentres compte tenu des modifications des classes ; on met alors à jour également la liste C.
  - on itère ce processus jusqu'à stabilité des classes/modification non significative de l'inertie intra-classe.

SCRIPT PYTHON :

On va construire la fonction `k_moyennes(X:array,k:int)` ⟶ `(P,C)` recevant comme argument la matrice

de données X et le nombre de classe(s) $k \geq 1$, et renvoyant le tuple (P,C) où P est une liste de k liste(s) formant une partition P du `dataset`, et C la liste des k centre(s) de gravité de cette partition (liste de k vecteur(s) colonne).

Listing IV.4 – fonction k_moyennes(X :array,k :int)

```
def k_moyennes(X,k):
  n,N=X.shape
  ##### Initialisation #####
  P=[[] for i in range(k)] #initialisation de la partition (une liste de k listes vides)
  C=[] #initialisation liste vide des barycentres de classes
  assert N>=k # vérification que le nombre de classes est inférieur au nombre d'éléments du dataset
  indC=rd.sample(range(0,N),k) #indice des k éléments choisis au hasard comme barycentres
  for p in indC:
    C.append(X[:,p])

  ##### Début de la procédure itérative #####
  modifclasses=True # on initialise un booleen à True pour vérifier la stabilité des classes
  while modifclasses:
    Pancien=P
    P=[[] for i in range(k)] #réinitialisation de la partition avant reconstruction
    for j in range(N):
      dmin=math.inf #on initialise la distance minimale (infinie)
      for p in range(k): # p indice d'itération sur les barycentres des k classes
        dp=dist(X[:,j],C[p])
        if dp<dmin:
          dmin=dp
          pmin=p
      P[pmin].append(j)
    #### Recalcul des k barycentres ####
    C=[] #réinitialisation de la liste des barycentres
    for p in range(k):
      Xp=X[:,[P[p]]] # on forme une matrice de données Xp constituée uniquement de la classe p
      Gp=barycentre(Xp)
      C.append(Gp)
    if Pancien==P:
      modifclasses=False
  return (P,C)
```

Listing IV.5 – fonction dist(Xj :array, Cp :array)

```
def dist(Xj,Cp):
    n=Cp.shape[0]
    dst2=0.0
    for i in range(n):
        dst2=dst2+(Xj[i]-Cp[i])**2
    return np.sqrt(dst2)
```

Remarque II-1: PROBLÈME DES CONVERGENCES LOCALES

En exécutant plusieurs fois l'algorithme des k-moyennes sur un cas concret (cf TD), on remarquera que pour un même jeu de données, on peut avoir des partitionnements différents. En effet, l'initialisation des tous premiers k-barycentres est totalement aléatoire, et l'algorithme pourra dégager des contenus de classes différents suivant cette première initialisation ; la configuration des classes trouvée alors peut ne pas être optimale. On parle de **minimum local**.
Afin de palier aux problèmes des minima locaux, il suffit de lancer K-means plusieurs fois sur le jeu de données (avec le même nombre K de clusters et des initialisations des barycentres différentes) et voir la composition des clusters qui se forment afin de les comparer. On procèdera par "vote" en gardant la configuration la plus fréquente.

III Exemple complet de mise en œuvre : classification d'un ensemble de vins

cf TD/TP.