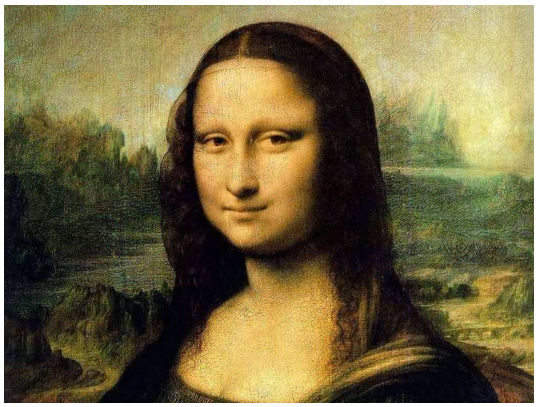
Master MLSD, Paris 5 2019-2020

TP - Classification automatique

1 Exercice 1 : compression image

Dans cette partie, on souhaite réduire grâce à la classification automatique le nombre de couleurs qui codent l'image ci-dessous.



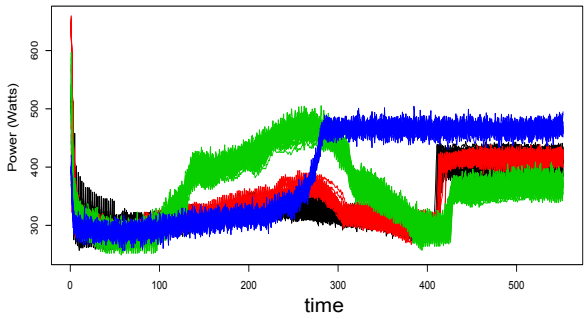
— Télécharger le fichier `image.ppm` et charger le package `pixmap`.
— Charger l'image dans une variable `I` en utilisant la commande `I=read.pnm(`"image.ppm").
— Taper la commande `plot(I)`.
— Convertir l'image `I` en un tableau de données `x` donnant le pourcentage de rouge/vert/bleu pour chaque pixel de l'image. On pourra par exemple utiliser la commande `as.vector(attr(I,"red"))`
— Inversement, pour convertir un tableau de données `y` (pourcentage RGB) en une image `J` de dimensions 327×435 pixels, on pourra utiliser la commande `J = pixmapRGB(as.matrix(y),nrow=327,ncol=435)`

1. Effectuer une étude descriptive des données contenues dans le tableau `x`.
2. Etudier la répartition des couleurs rouge/vert/bleu dans l'image.
3. Appliquer la CAH-Ward et l'algorithme des k-means, pour réduire le nombre de couleurs codant l'image.
   Dans le nouveau tableau de données, les points seront remplacés par le centre de leur classe.
4. Interpréter les classes obtenues.
5. Représenter la nouvelle image ainsi obtenue.

Exercice 2 : Données aiguillage

Le jeu de données Aiguillage comprend 140 objets (séries temporelles) décrites par 553 variables : les 552 premières variables correpondent à l'énergie consommée au cours des

mouvements mécaniques d'un système d'aiguillage ferroviaire et la 553e variable correspond à la classe (1 : sans défaut, 2 : défaut mineur, 3 : défaut critique, 4 : panne).



L'objectif de cet exercice est d'utiliser des algorithmes de classification automatique pour résumer ces données.

1. Classifier les données `aiguillage` en utilisant l'algorithme CAH avec les critères d'agrégation de Ward et de lien minimum.
2. Comparer les classes obtenues avec les vraies classes.
3. Proposer des représentations graphiques des classes obtenues.

Exercice 3 : Données Vélib

L'objectif de cet exercice est de regrouper en classes homogènes les stations du réseau Vélib parisien et de tenter d'interpréter les classes obtenues, ceci de manière non supervisée. La principale caractéristique utilisée pour cela est le nombre de vélos disponibles dans chaque station (taux de charge), qui est disponible toutes les heures. On se focalisera sur la semaine du lundi 1er septembre 2014 au dimanche 7 septembre 2014, c'est-à-dire sur 168 heures qui constitueront nos variables. Plus spécifiquement, nous utiliserons une version lissée des données du package `funFEM`.

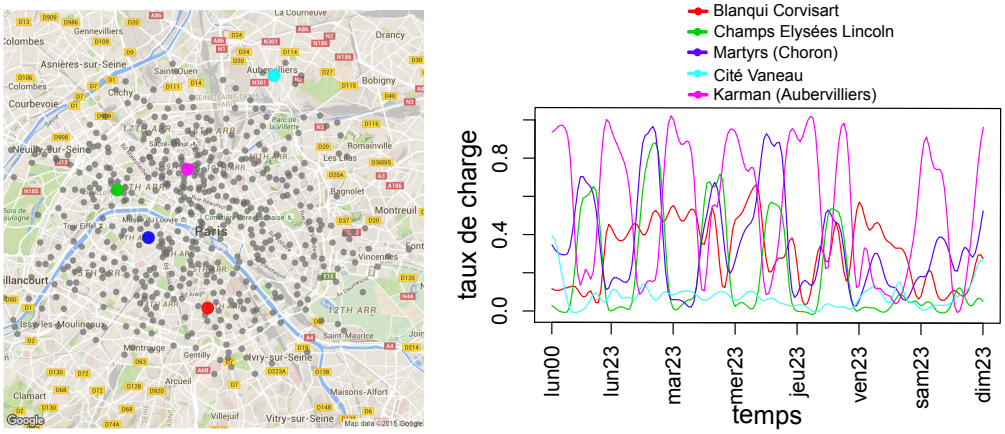


FIGURE 1 – Données Vélib

1. Télécharger les données `velib` puis créer une matrice de donnée, constituée uniquement des taux de charge. Il s'agira ici d'une matrice à 1189 lignes et 168 colonnes, sur laquelle les différents algorithmes de clustering seront appliqués.

2. Représenter graphiquement, pour quelques stations choisies au hasard, les courbes de charge. Observez-vous des différences sur celles-ci ?
3. Réaliser un clustering en 6 classes des stations Vélib en utilisant les algorithmes CAH-Ward et k-means.
4. Pour chacune des méthodes, représenter graphiquement (sous la forme de courbes) les centres des clusters obtenus. Commenter les profils obtenus.
5. Représenter graphiquement les partitions obtenues en utilisant les coordonnées géographiques des stations. Proposer une interprétation des clusters en fonction de leur localisation spatiale.

Exercice 4 : algorithme de programmation dynamique de Fisher

Cet exercice vise à mettre en œuvre une méthode de clustering qui recherche des classes ordonnées dans le temps : la méthode de programmation dynamique de Fisher [1][2]. Celle-ci est utilisée pour la segmentation de signaux temporels, ou pour la détection de changement dans une séquence de données. Dans les systèmes de reconnaissance de la parole par exemple, on exploite ce type de méthode pour partitionner des signaux audio en plages temporelles associées à des locuteurs différents qui sont ensuite identifiés. De manière plus générale, l'organisation d'une séquence de données en segments homogènes est un processus qui aide à mieux les organiser.

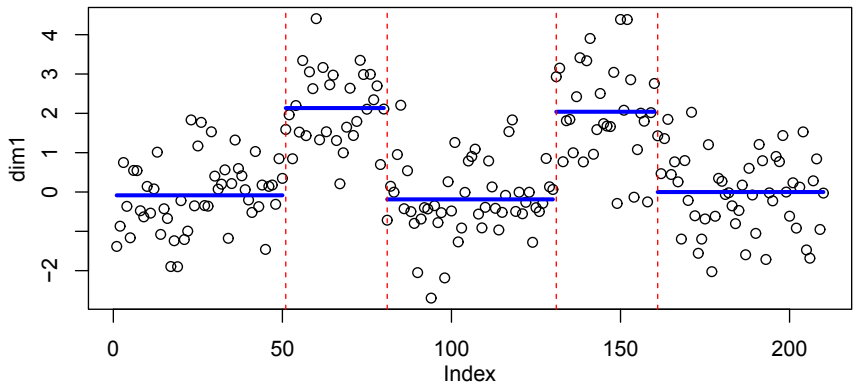


FIGURE 2 – Partitionnement de données avec contraintes d'ordre sur les classes (segmentation)

Description de l'algorithme

L'algorithme de programmation dynamique de Fisher cherche à partitionner une séquence de données $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ décrites par p variables en K classes qui apparaissent l'une après l'autre. Compte-tenu de la contrainte d'ordre imposée sur les classes, ce problème diffère de celui (classique) visant à trouver une partition minimisant par exemple l'inertie intra-classe. L'objectif est désormais de déterminer K intervalles temporels $[1; t_1[$, $[t_1; t_2[$, $\cdots$, $[t_{K-2}; t_{K-1}[$, $[t_{K-1}; n]$, ou, de manière équivalente, $K-1$ instants de changement $(t_k)_{k=1,\ldots,K-1}$ minimisant le critère (de type inertie intra-classe) suivant :

$$\left\{ \begin{array}{lcll} C(t_1,\ldots,t_{K-1}) & = & D(1,t_1-1)+\left(\sum_{k=2}^{K-1} D(t_{k-1},t_k-1)\right)+D(t_{K-1},n) & \text{si } K\geq 3 \\ C(t_1) & = & D(1,t_1-1)+D(t_1,n) & \text{si } K=2, \end{array} \right. \tag{1}$$

avec

$$D(a,b)=\sum_{i=a}^{b} \parallel \boldsymbol{x}_i-\boldsymbol{\mu}_{ab} \parallel^2 \qquad \text{et} \qquad \boldsymbol{\mu}_{ab}=\frac{\sum_{i=a}^{b}\boldsymbol{x}_i}{b-a+1}. \tag{2}$$

On notera ici que D est une matrice (triangulaire supérieure) de taille (n,n) appelée souvent **matrice des diamètres des classes**.

Contrairement au critère d'inertie intra-classe qui a été étudié en cours (dans lequel aucune contrainte d'ordre n'est imposée sur les classes), le critère C peut ici être **minimisé de manière exacte (optimum global)** grâce à l'algorithme de programmation dynamique de Fisher qui est basé sur le principe selon lequel « toute sous-partition d'une partition optimale est nécessairement optimale ». L'algorithme est décrit ci-dessous.

Questions

1) Implémenter sous R l'algorithme de programmation dynamique de Fisher (voir ci-après l'Algorithme 1). Créer pour cela deux fonctions : une fonction `diam` calculant le diamètre $D(a,b)$ suivant la formule (2) et une fonction `clustfisher` pour l'algorithme lui-même. Pour le calcul de l'argument minimum (argmin) d'un vecteur numérique, vous pourrez utiliser la fonction `which.min` de R.

2) Application à des données simulées : en utilisant l'algorithme de Fisher, segmenter la séquence de données simulées `simu` (à télécharger) et régler le nombre de classes (segments) à l'aide de la méthode du coude appliquée au critère C. Comparer cette partition à celle issue de l'algorithme des kmeans.

3) Application à des données réelles : segmenter en 4 classes la séquence $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_{140})$ des données `aiguillage` (on rappelle que dans ce cas, chaque observation $\boldsymbol{x}_i$ appartient à $\mathbb{R}^{552}$. Comparer la partition obtenue par l'algorithme de Fisher avec la vraie partition ainsi qu'avec la partition issue de l'algorithme des kmeans.

Algorithme 1 : Algorithme de programmation dynamique de Fisher

Entrées : séquence $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, nombre de segments souhaité ($K \geq 2$)

Étape 0

Initialiser les matrices D, M_1 et M_2, ainsi que le vecteur $\mathbf{t}$ des instants de changement et le vecteur **cluster** des classes (sous R, on pourra utiliser des matrices et des vecteurs nuls)

Étape 1 : calcul de la matrice triangulaire supérieure des diamètres

pour $a = 1, \ldots, n,\ b = a, \ldots, n$ **faire**

| Calculer $D[a, b]$ selon la formule (2)

fin

Étape 2 : calcul récursif des critères optimaux

pour $i = 1, \ldots, n$ **faire**

| $M[i, 1] \leftarrow D[1, i]$

fin

pour $k = 2, \ldots, K$, $i = k, \ldots, n$ **faire**

| $M[i, k] \leftarrow \min\left\{M[t-1, k-1] + D[t, i] \quad \text{avec} \quad t = k, \ldots, i\right\}$

| $A[i, k] \leftarrow \text{argmin}\left\{M[t-1, k-1] + D[t, i] \quad \text{avec} \quad t = k, \ldots, i\right\}$

fin

Étape 3 : calcul récursif des instants de changement optimaux

$k \leftarrow K - 1$

$m \leftarrow n$

tant que $k \geq 1$ **faire**

| $t_k \leftarrow A[m, k+1]$

| $m \leftarrow t_k - 1$

| $k \leftarrow k - 1$

fin

Étape 5 : inertie intra-classe et labels des classes issus des instants de changement

$InertieIntra \leftarrow M[n, K]$

pour $i = 1, \ldots, t_1 - 1$ **faire**

| $cluster[i] \leftarrow 1$

fin

pour $k = 2, \ldots, K-1$, $i = t_{k-1}, \ldots, t_k - 1$ **faire**

| $cluster[i] \leftarrow k$

fin

pour $i = t_{K-1}, \ldots, n$ **faire**

| $cluster[i] \leftarrow K$

fin

Sorties : labels **cluster** et instants de changement $\mathbf{t} = (t_1, \ldots, t_{K-1})$