UNIVERSITÉ PARIS DESCARTES

MEMBRE DE

USPC Université Sorbonne Paris Cité

UFR DE MATHEMATIQUES ET INFORMATIQUE

MASTER 2 MLDS 2017 – 2018

RAPPORT DE PROJET

APPRENTISSAGE NON SUPERVISE

Professeur : Allou SAME

Etudiant : NGUYEN Ngoc Tu

N° étudiant : 21710268

1. Problématique

L'objectif de ce projet est d'étudier une méthode de clustering qui recherche des classes ordonnées dans le temps : la méthode de programmation dynamique de Fisher. Je le compare avec des méthodes classiques (K-means et CAH-Ward) en les appliquant à des données simulées et des données réelles (*aiguillage*) et en évaluant les résultats obtenus.

2. L'Algorithme de Programmation Dynamique de Fisher

L'algorithme de programmation dynamique de Fisher cherche à partitionner une séquence de données $(x_1, \ldots, x_n)$ d´écrites par p variables en K classes qui apparaissent l'une après l'autre. Compte-tenu de la contrainte d'ordre, ce problème diffère de celui (classique) visant à trouver une partition minimisant par exemple l'inertie intra-classe. Dans cette nouvelle situation, l'objectif est de déterminer K intervalles temporel $[1;\ t_1[, [t_1;\ t_2[, \cdots, [t_{K-2};\ t_{K-1}[, [t_{K-1};\ n]$s, ou, de manière équivalente, $K-1$ instants de changement $(t_1, \ldots, t_{K-1})$ *minimisant le critère (de type inertie intra-classe)* suivant :

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

avec

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

D est une matrice (triangulaire supérieure) de taille (n, n) appelée souvent matrice des diamètres des classes. Le cœur de cet algorithme est la relation entre les partitions optimales en K classes et les partitions optimales en $K-1$ classes. Supposant que $M_1[i, k]$ est l'inertie intra-classe optimale en divisant i premiers individus en k classes ($[1;\ t_1[, [t_1;\ t_2[, \cdots, [t_{k-2};\ t_{k-1}[, [t_{k-1};\ i]$), donc $M_1[t_{k-1} - 1, k-1]$ doit-être optimale (partitionnement $t_{k-1} - 1$ premiers individus en $k-1$ classes). En utilisant cette relation, cet algorithme calcule successivement des partitions optimales en $1, 2, \ldots, K$ classes, et construit sur $K-1$ partitions pour trouver K partitions.

3. Implémentation

3.1. Implémentation de l'algorithme de programmation dynamique de Fisher

Voilà toutes mes fonctions que j'ai implémentées pour ce projet (vous pourriez trouver le code détaillé dans le script joint R) :

a. *diam(X, a ,b)* :

Calculer le diamètre selon (2). Je l'ai implémentée avec des fonctions intégrées de R pour optimiser le temps d'exécution au lieu de l'implémentation naïve avec des boucles.

Inputs :

- X : matrice de données, avec la taille : n lignes, p colonnes.
- a : nombre entier
- b : nombre entier (b >= a)

Outputs :

- Valeur réelle calculée selon la formule (2)

b. diam_matrice(X)

Dans les étapes suivantes, il faut mettre en œuvre la méthode du coude en faisant le clustering de Fisher plusieurs fois, puis analyser des inerties intra-classes obtenues. Après de nombreux tests, je trouve que cette étape (calculer la matrice des diamètres) prend la plupart de temps d'exécution. Pour éviter de calculer la matrice des diamètres des classes nombreux fois, j'ai implémenté la fonction *clustfisher* en même structure de *hclust*, elle prend la matrice distance en tant qu'argument au lieu de matrice de données. Donc c'est la raison pour laquelle j'ai créé cette fonction.

Inputs :

- *X* : matrice de données

Outputs :

- Matrice de diamètres des classes

c. clustfisher(D, K)

Cette fonction utilise l'algorithme de programmation dynamique de Fisher pour classifier des données représentées par la matrice des diamètres des classes *D* en *K* classes.

Inputs :

- *D* : matrice des diamètres des classes
- *K* : nombre de classes

Outputs :

- *cluster* : vecteur des labels des individus
- *t* : vecteur des instants de changement
- *tot.withinss* : valeur d'inertie intra-classes

d. clustering(X, K)

L'objectif de cette fonction est de résoudre les questions 2) et 3). Car il y a deux jeux des données à travailler, je l'ai implémentée pour éviter le redoublement du code. Cette fonction va prendre les données X et les classifier en K classes, en utilisant trois algorithmes de classification : Fisher, K-means et CAH-Ward.

Inputs :

- *X* : matrice des données
- *K* : nombre de classes voulu. Si K se présente, cette fonction va classifier X en K classes. Si non, elle va classifier X plusieurs fois avec le K varie (en utilisant l'algorithme de Fisher). Ensuite, elle va montrer un graphe des inerties intra-classes et vous laisser choisir K manuellement.

Outputs :

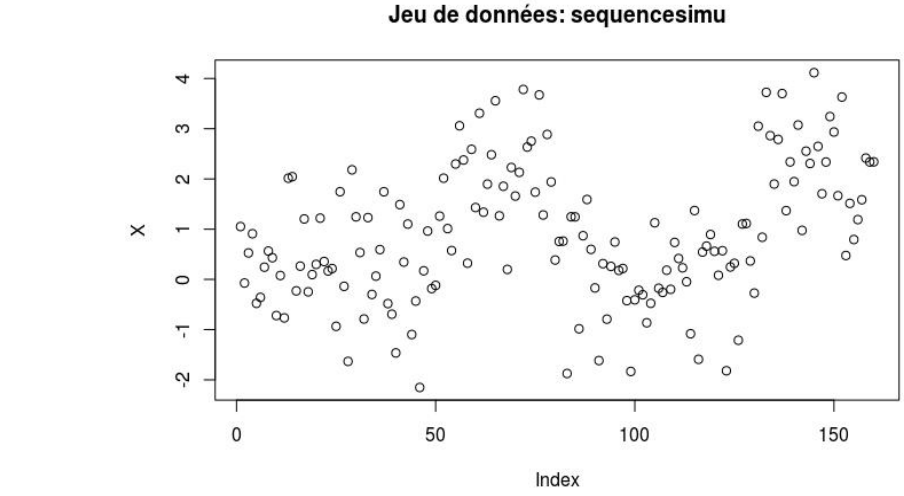
- *clust_fisher* : résultat obtenu avec l'algorithme de programmation dynamique de Fisher
- *clust_kmeans* : résultat obtenu avec l'algorithme de K-means
- *clust_cah_ward* : résultat obtenu avec l'algorithme de CAH-Ward

3.2. Résultats

3.2.1. Application a des données simulées : *sequencesimu*

a. Analyse

- Le jeu de données *sequencesimu* est uni varié.




```
> summary(sequencesimu)
       V1
 Min.   :-2.15190
 1st Qu.:-0.05362
 Median : 0.74983
 Mean   : 0.89217
 3rd Qu.: 1.86473
 Max.   : 4.11444
```

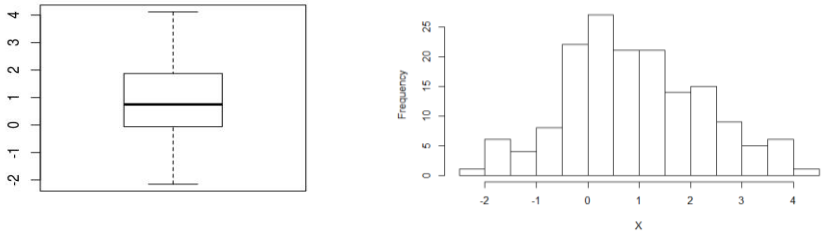
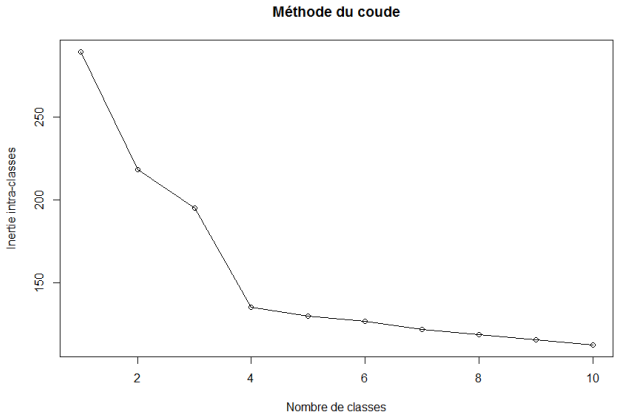


FIGURE 1 – Données sequencesimu et leur analyse

b. Clustering avec des algorithmes : Fisher, K-means, CAH-Ward

En exécutant la fonction *clustering(sequencesimu)*, j'obtiens le graphe suivant :




```
> sequencesimu_clustering = clustering(sequencesimu)
Calculer inertie intra-classes pour la methode du coude...
avec K = 2 ...
avec K = 3 ...
avec K = 4 ...
avec K = 5 ...
avec K = 6 ...
avec K = 7 ...
avec K = 8 ...
avec K = 9 ...
avec K = 10 ...
K = 4
```

FIGURE 2 – Méthode du coude

En analysant ce graph, je trouve que K = 4 est le nombre de classes optimal. Après avoir choisi K = 4, j'obtiens des plots des résultats obtenus :

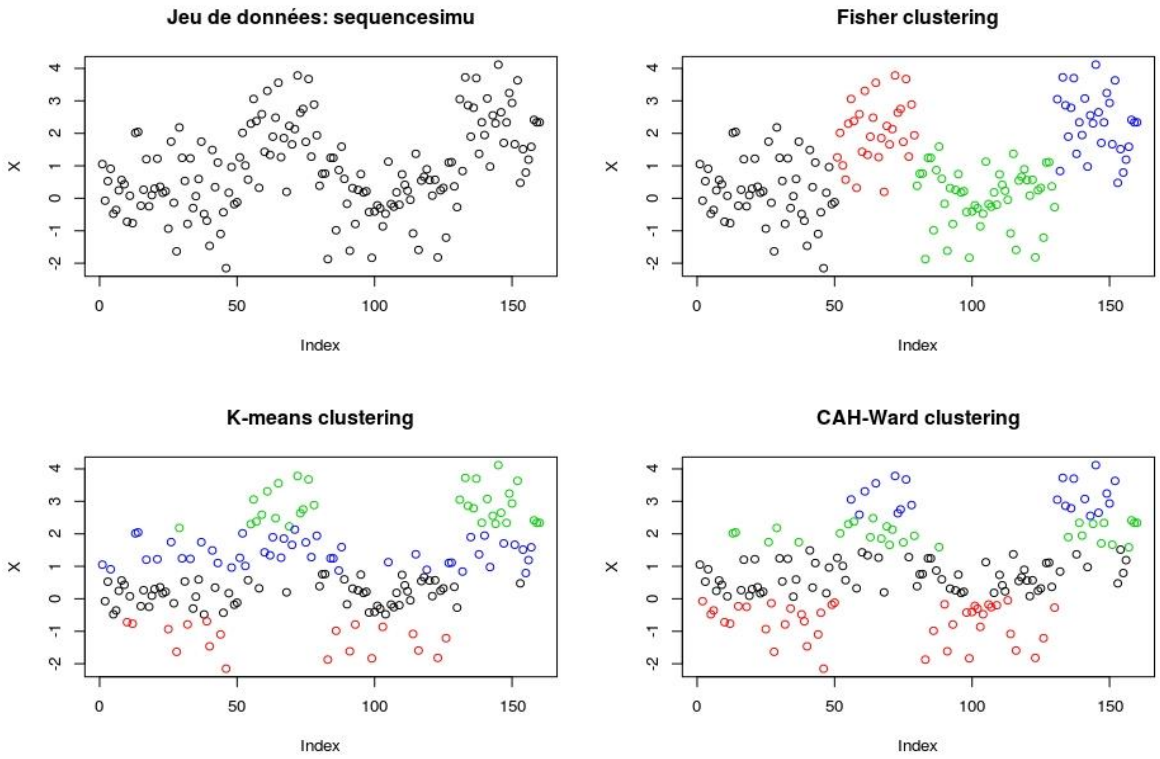


FIGURE 3 – Résultats obtenus avec les données sequencesimu

Car les données *sequencesimu* est uni varié, les résultats de l'algorithme de K-means et l'algorithme de CAH-Ward sont ressemblés, et ils sont différents que le résultat obtenu par l'algorithme de Fisher, qui est le vainqueur dans cette situation (il fait la segmentation/ détection de changement dans une séquence de données).

3.2.2. Application a des données réelles : *aiguillage*

a. Analyse

- Le jeu de données *Aiguillage* comprend 140 séries temporelles d´écrites par 553 variables : les 552 premières variables correspondent 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 - noir, 2 : défaut mineur - rouge, 3 : défaut critique - vert, 4 : panne - bleu).

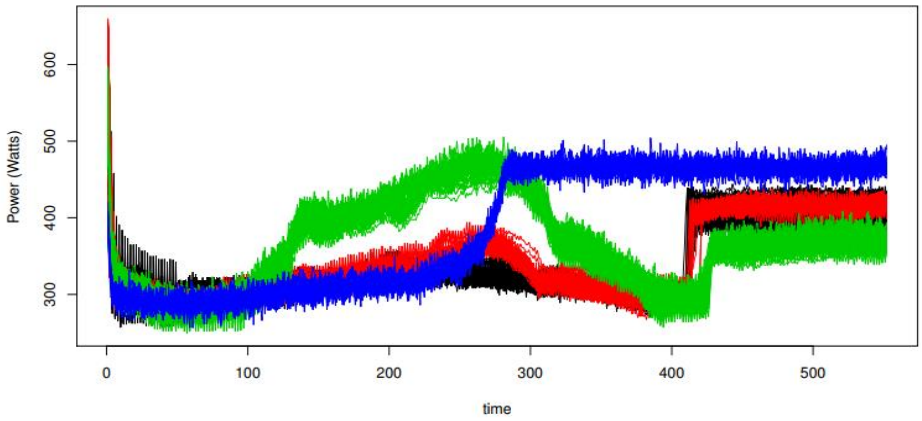


FIGURE 4 - Séries temporelles d'énergie consommée au cours de 140 mouvements d'un système d'aiguillage ferroviaire

b. Résultats (K = 4)

Voici les résultats obtenus :

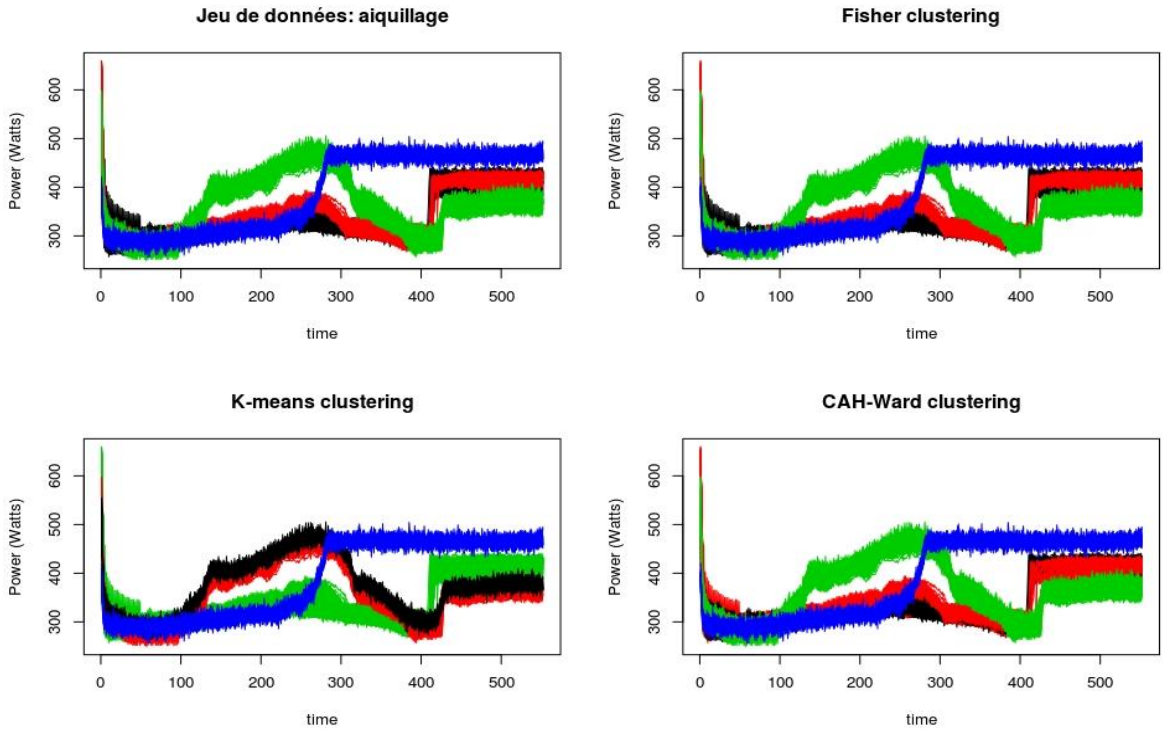


FIGURE 5 – Résultats obtenus avec les données aiguillage

L'algorithmes de Fisher et CAH-Ward bien segmentent les données *aiguillage*, leurs résultats sont ressemblés avec l'origine. Pour le résultat de l'algorithme de K-means, je le classifierai plusieurs fois :

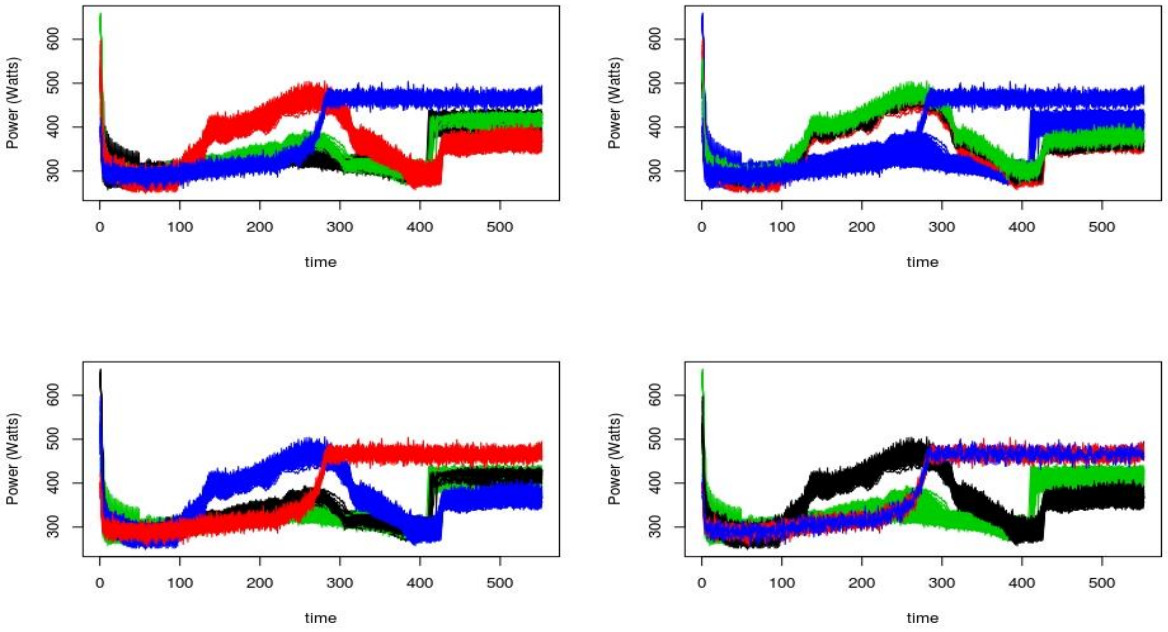


FIGURE 6 – Résultats des aiguillage (K-means)

Les résultats obtenus avec l'algorithme de K-means sont différents car l'initialisation des centres des classes est différente à chaque fois je fais la classification avec l'algorithme de K-means.

Pour mieux comparer les résultats, j'utilise l'Indice de Rand (Adjusted Rand Index) pour évaluer les performances de chaque l'algorithme (librairie *mclust*). C'est une mesure de similarité entre deux partitions d'un ensemble (*0.0* : pas de similarité, *1.0* : identique). Son principe est de mesurer la consistance (le taux d'accord) entre deux partitions [1]. Voici les Indices de Rand entre chaque résultat obtenu avec des labels vrais :

Algorithmes	**Indice de Rand**
Fisher	*1*
CAH-Ward	*0.8589*
K-means (le meilleur résultat)	*0.9426*

TABLE 1 : Indices de Rand des algorithmes sur les données aiguillage

L'algorithme de Fisher est le meilleur dans cette situation, il classifie les données aiguillage exactement comme les labels vrais. L'algorithme de CAH-Ward prend la deuxième place, car l'algorithme de K-means est instable, donc il n'est pas mieux que l'algorithme CAH-Ward sur les données *aiguillage*.

4. Conclusion

Après avoir analysé et implémenté l'algorithme de programmation dynamique de Fisher en appliquant sur des données simulées et réelles, je trouve que cet algorithme donne des bons résultats sur des données temporelles. L'inconvénient de cet algorithme : il prendra du temps si les données sont grandes, car l'étape de calcul la matrice des diamètres des classes est basé sur les boucles. La fonction *diam_matrice(aiguillage)* prend ***4.52*** **secondes**, au lieu que la fonction *clustering(aiguillage, 4)* (classification des données *aiguillage* en 4 classes, avec 3 algorithmes : Fisher, CAH-Ward, K-means) prend ***4.56*** **secondes** (Core i7 3740QM @ 2.7GHz, Ubuntu 16.04). Une idée possible pour améliorer l'algorithme de Fisher, c'est la vectorisation l'étape de calcul la matrice des diamètres des classes.

5. Références

[1] https://en.wikipedia.org/wiki/Rand_index