Développement d'un nouvel algorithme de régression clusterwise

Encadrants :
Niang Ndeye
Thiria Sylvie
Stagiaire :
Bernhard Rémi

CNRS/LOCEAN

Mars-Juin 2018

Remerciements

Je tiens à remercier toutes les personnes qui m'ont accompagné durant ce stage et qui ont contribué à sa réussite.

Je remercie vivement mes encadrantes Mesdames Ndeye Niang et Sylvie Thiria, toujours à l'écoute et très disponibles. Leurs nombreux conseils, tant relatifs au sujet du stage en lui-même qu'à la méthodologie, m'ont permis de mener à bien mes recherches et de progresser quant à ma méthode de travail. Leur accompagnement tant au niveau académique qu'humain a été une composante forte de ce stage.

Je remercie aussi les membres du laboratoire CNRS/LOCEAN ainsi que les autres stagiaires pour leur accueil et leur sympathie. J'ai eu la chance d'effectuer ce stage dans un environnement de travail agréable et propice à la découverte d'autres thèmes de recherche passionnants.

Sommaire

1 Introduction

1.1 Contexte du stage

Ce stage de six mois a été effectué dans le laboratoire LOCEAN (Laboratoire d'Océanographie et du Climat : Expérimentations et approches numériques) de l'UPMC (Université Pierre et Marie Curie), qui est un laboratoire de recherche rattaché au CNRS (Centre National de la Recherche Scientifique). Le laboratoire LOCEAN est une unité mixe de recherche qui couvre des domaines allant de l'apprentissage automatique et des méthodes statistiques aux processus biogéochimiques océaniques et à l'étude de la variabilité climatique.
J'ai effectué ce stage sous l'encadrement de Madame Sylvie Thiria (laboratoire LOCEAN) et de Madame Ndeye Niang (laboratoire CEDRIC : Centre d'Etudes et de Recherche en Informatique et Communications).

1.2 Motivation

Dans de nombreux domaines d'application tels que la sociologie statistique et l'analyse des consommateurs, il est courant que tous les individus étudiés ne soient pas issus de la même population, et alors une régression sur l'ensemble des individus peut se révéler très infructueuse. Lorsque l'on souhaite effectuer une régression tout en supposant qu'il existe au sein des individus un certain nombre de groupes tels qu'au sein de chaque groupe la relation entre la variable réponse et les variables explicatives est particulière à ce groupe, une méthode couramment utilisée est la régression clusterwise. Deux principales méthodes existent pour effectuer de la régression clusterwise. La première, dite probabiliste, repose sur la maximisation de la vraisemblance du modèle (*cf.* [DeSarbo and Cron, 1988]). Cette méthode est paramétrique et oblige à choisir une loi de densité (ce choix est nécessaire pour le calcul de la vraisemblance). De plus, elle ne permet pas de facilement effectuer la prédiction de la valeur de la variable réponse pour un nouvel individu. La deuxième méthode, dite géométrique, repose sur le principe des moindres carrés (*cf.* [Späth, 1979]). Cette méthode peut cependant échouer lorsque le nombre d'individus au sein d'un groupe est inférieur au nombre de variables explicatives.
Nous proposons ici dans un premier temps un nouvel algorithme reprenant le principe de l'algorithme décrit dans [Späth, 1979] en y ajoutant les cartes topologiques de Kohonen par le biais d'une modification de la fonction de coût de l'algorithme décrit dans [Späth, 1979]. Nous définissons notamment certains critères qui l'empêchent d'échouer comme cela peut arriver à l'algorithme présenté dans [Späth, 1979], lorsque le nombre de variables explicatifs est supérieur au nombre d'individus au sein d'un groupe, et le rendent plus performants en prédiction en généralisation que l'algorithme présenté dans [Späth, 1979]. Bien que cet algorithme soit peu coûteux en temps, il reste moins bon en généralisation que l'algorithme de régression clusterwise du *package R mbclusterwise* (*cf.* [Bougeard et al., 2017]), qui reprend le principe présenté dans [Späth, 1979] en y introduisant la régression PLS. Nous développons alors une autre version de notre algorithme qui reprend le principe de celui présenté dans en [Bougeard et al., 2017] en y ajoutant les cartes topologiques de Kohonen. Nous montrons alors que notre algorithme obtient de meilleures performances sur données simulées dans certains cas précis que l'algorithme présenté dans [Bougeard et al., 2017]. Nous comparons finalement ces algorithmes sur des données réelles.

1.3 Démarche

Le but de ce stage a été de développer un nouvel algorithme permettant d'effectuer de la régression clusterwise, qui hérite des propriétés des cartes topologiques de Kohonen. La régression clusterwise est une méthode d'apprentissage automatique qui permet d'effectuer de l'apprentissage de modèles de régression en même temps que de la classification.
Les enjeux particuliers de ce stage ont été de développer et implémenter un algorithme pertinent dans le sens où il ne présente pas les défauts des algorithmes existants tout en conservant de bonnes performances. Il a fallu notamment définir des critères afin qu'il n'échoue pas quand le nombre de groupes et de variables explicatives est trop grand en comparaison du nombre d'individus, et montrer sa supériorité à d'autres algorithmes existants en termes de prédiction en généralisation.
Nous avons travaillé sur un jeu de données réelles issues d'une étude de la pollution de l'air, fourni par M. François Kaly, présenté dans [Kali, 2016].

Dans une première partie, nous présentons les différents algorithmes existants que nous avons étudiés. Nous expliquons notamment le principe de la régression PLS, différents algorithmes relatifs à la régression clusterwise, et l'algorithme des cartes de Kohonen. Nous illustrons notamment leur fonctionnement sur des données simulées.
Dans une seconde partie, nous présentons les deux algorithmes que nous avons développés dans le but de faire de la régression clusterwise, et qui héritent des propriétés topologiques des cartes de Kohonen : l'algorithme SOM-clusterwise et l'algorithme SOM-PLS-Clusterwise. Nous discutons notamment de notre apport concernant leur implémentation, leur convergence, et des critères destinés à améliorer la robustesse et limiter le sur-apprentissage pour l'algorithme SOM-Clusterwise.
Dans une troisième partie, nous effectuons des comparaisons sur des données simulées et sur le jeu de données réelles issues d'une étude de la pollution de l'air. Nous commençons par comparer l'algorithme SOM-Clusterwise à un autre algorithme de référence (*cf.* [Späth, 1979]) permettant d'effectuer de la régression clusterwise. Nous comparons ensuite les algorithmes SOM-PLS-Clusterwise et SOM-Clusterwise, puis les algorithmes mbpls et PLS-SOM-Clusterwise. Nous nous intéressons surtout aux performances en généralisation mais aussi à la qualité de la classification.

2 Etat de l'art

Dans cette partie, nous nous attachons à expliquer le fonctionnement de certains algorithmes permettant de réaliser une régression clusterwise. Nous commençons par détailler la régression PLS (*cf.* [Abdi, 2010] et [Tenenhaus et al., 1995]), puis nous présentons deux méthodes permettant de réaliser une régression clusterwise (*cf*[Späth, 1979] et [DeSarbo and Cron, 1988]).

2.1 Régression PLS

On considère un problème avec n individus décrits par p variables quantitatives où on cherche à prédire la valeur d'une autre variable quantitative. On note X la matrice centrée de dimension $n \times p$ qui contient la valeur de chacune des p variables explicatives pour chacun des n individus. On note X^j la j^e colonne de la matrice X $\forall j \in [\![1, p]\!]$. On note y le vecteur centré de taille n qui contient la valeur de la variable réponse pour chaque individu, et y_i le i^e élément du vecteur y.
Si l'on désire effectuer la régression linéaire de y par X, mais que la matrice $X^T X$ n'est pas inversible car des colonnes de X sont des combinaisons linéaires d'autres colonnes de X (c'est-à-dire que la dimension de l'espace engendré par les colonnes de X est inférieur à p), la régression linéaire n'est alors plus applicable. De même, si les variables X^j présentent des corrélations linéaires fortes entre elles, les coefficients de la régression linéaire peuvent biaisés. On aimerait trouver une matrice T de dimension $n \times q$ avec $q \leq p$ construite à partir de la matrice X dont les colonnes seraient orthogonales entre elles (au sens du produit scalaire défini par $< A, B >= E(AB)$, qu'on appelera par la suite produit scalaire issu de la covariance).
Pour obtenir une telle matrice T, une méthode très utilisée est l'ACP (Analyse en Composantes Principales). Cependant, le principe de l'ACP étant uniquement de maximiser l'inertie du nuage projeté dans l'espace des nouveaux axes, si on utilisait l'ACP ici, on ne prendrait pas en compte la variable y, et les nouvelles variables (les axes de l'ACP) obtenues ne seraient donc pas forcément les plus explicatives vis-à-vis de la variable y.
La méthode que l'on va expliquer ici pour apprendre un modèle linéaire entre la variable y et la matrice X en passant par les colonnes d'une nouvelle matrice T telle que définie précédemment, s'appelle la régression PLS simple.
On note T^j la j^e colonne de la matrice T $\forall j \in [\![1, q]\!]$.
Le principe de la régression PLS est de déterminer de nouvelles variables T^j ($\forall j \in [\![1, q]\!]$) orthogonales entre elles (au sens du produit scalaire issu de la covariance) qui sont des combinaisons linéaires des "anciennes" variables X^j ($\forall j \in [\![1, p]\!]$) et qui maximisent la covariance avec la variable réponse y et d'autres variables qui seront détaillées par la suite. On considère le modèle :

$$Y = XB + E$$

$$X = TW + R_x$$

$$Y = TV + R_y$$

avec :
$T : n \times q$
$W : q \times p$
$R_x : n \times p$ avec $\forall i \in [\![1, n]\!] R_{X,i} \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma_X^2)$ (où $R_{X,i}$ correspond à la ligne i de la matrice R_x)
$V : q \times 1$
$R_y : n \times 1$ avec $\forall i \in [\![1, n]\!] R_{Y,i} \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma_Y^2)$ (où $R_{Y,i}$ correspond à la ligne i de la matrice R_y)
$E : n \times 1$ avec $\forall i \in [\![1, n]\!] E_i \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma_3^2)$ (où E_i correspond à la ligne i de la matrice E)
$\sigma_X^2 \in \mathcal{R}$, $\sigma_Y^2 \in \mathcal{R}$ et $\sigma_3^2 \in \mathcal{R}$

On détaille les deux premières étapes de l'algorithme, puis on le détaillera formellement.
Etape 1 :
On détermine T^1, combinaison linéaire des X^i qui maximise la covariance avec la variable réponse y.
On cherche donc $C^1 = (c_1^1, c_2^1, ..., c_p^1) \in \mathbb{R}^p$ tel que : $cov(\sum_{j=1}^p c_j^1 X^j, y)$ soit maximale et $||C^1|| = 1$ avec $||.||$ la norme euclidienne classique, et on aura :

$$T^1 = \sum_{j=1}^{p} c_j^1 X^j$$

On montre que l'on trouve :

$$\forall j \in [\![1,p]\!], c_j^1 = \frac{cov(y, X^j)}{\sqrt{\sum_{j=1}^{p} cov(y, X^j)^2}}$$

On a $C^1 = \frac{X^T y}{||X^T y||^2}$ (puisque X et y sont centrées on a $cov(y, X^j) = E(yX^j) = \sum_{l=1}^{n} y_l X_l^j$), et $T^1 = XC^1$

On fait ensuite la régression linéaire de X par T^1 et de y par T^1. On obtient alors W^1 de dimension $p \times 1$, $R1_x$ de dimension $n \times p$, $V^1 \in \mathbb{R}$ et $R1_y$ tels que :

$$X = T^1 W^{1T} + R1_x$$
$$y = T^1 V^1 + R1_y$$

On a, en s'appuyant sur les formules des coefficients directeurs lors de la régression d'une variable A par une unique variable B : $W^1 = \frac{X^T T^1}{||T^1||^2}$ et $V^1 = \frac{y^T T^1}{||T^1||^2}$

Etape 2 :

L'étape 2 consiste simplement à répéter la procédure de l'étape 1 en considérant la matrice $R1_x$ à la place de la matrice X et la matrice $R1_y$ à la place de la matrice y. On cherche alors $C^2 = (c_1^2, c_2^2, ..., c_p^2) \in \mathbb{R}^p$ tel que : $cov(\sum_{j=1}^{p} c_j^2 R1_x^j, R1_y)$ soit maximale et $||C^2|| = 1$.

Une fois qu'on a calculé les nouvelles variables $T^1, T^2, ..., T^q$, on a finalement :

$$\begin{aligned} Y &= \sum_{j=1}^{q} V^j T^j + Rq_Y \\ &= \sum_{j=1}^{q} V^j \sum_{i=1}^{p} C_i^j X^j + Rq_Y \end{aligned}$$

On peut donc bien réaliser une prédiction pour un nouvel individu à partir des valeurs des "anciennes" variables $X^j \ \forall j \in [\![1,p]\!]$.

On montre que les nouvelles variables $T^1, T^2, ..., T^q$ construites sont bien orthogonales entre elles (*cf.* [Tenenhaus et al., 1995]).

Le pseudo-code de l'algorithme de la régression PLS simple, où on a choisi le nombre q ($q <= p$) de nouvelles variables à calculer, est disponible en annexe A.

2.2 Algorithmes pour la régression clusterwise

On considère un problème avec n individus décrits par p variables quantitatives où on cherche à prédire la valeur d'une autre variable quantitative. On considère que l'on a $n > p$. On note X la matrice de dimension $n \times p$ qui contient la valeur de chacune des p variables explicatives pour chacun des n individus. On note X^j la j^e colonne de la matrice $X \ \forall j \in [\![1,p]\!]$. On note y le vecteur de taille n qui contient la valeur de la variable réponse pour chaque individu et y_i le i^e élément du vecteur y. On suppose que les individus sont classés en K groupes, sans toutefois savoir a priori à quel groupe appartient un individu. On aimerait alors déterminer la répartition optimale des n individus dans ces K groupes de sorte qu'au sein de chaque groupe on ait la meilleure régression possible de y par X.

2.2.1 Algorithmes géométriques

Algorithme de Spaeth :

On présente ici une première méthode permettant d'effectuer une régression clusterwise, qui correspond à la méthode présentée dans [Späth, 1979].

On note y_i la valeur de la variable réponse pour l'individu i ($\forall i \in [\![1, n]\!]$).
Si on note $C_1, C_2, ..., C_G$ une partition de l'ensemble $\{1, ..., n\}$, on cherche $C_1, C_2, ..., C_G$ telle que :

$$\begin{aligned} C_1, C_2, ..., C_G &= \underset{C_1, C_2, ..., C_G}{\operatorname{argmin}} \sum_{g=1}^{G} \left[\sum_{i \in C_g} \left((y_i - \sum_{j=1}^{p} X_i^j \beta_{g,j})^2 \right) \right] \\ &= \underset{C_1, C_2, ..., C_G}{\operatorname{argmin}} \sum_{g=1}^{K} E(C_g) \\ &= \underset{C_1, C_2, ..., C_G}{\operatorname{argmin}} D(C_1, C_2, ..., C_G) \end{aligned}$$

où :
X_i^j est la valeur de la variable j ($\forall j \in [\![1, p]\!]$) pour l'individu i ($\forall i \in [\![1, n]\!]$)
$\beta_{g,j}$ est la valeur du coefficient dans la régression linéaire de y par X uniquement pour les individus du groupe C_g ($\forall g \in [\![1, K]\!]$) pour la variable X^j ($\forall j \in [\![1, p]\!]$).

La méthode que l'on explique ici pour trouver une partition qui atteint un minimum local de $D(C_1, C_2, ..., C_G) = \sum_{g=1}^{G} E(C_g)$ consiste à partir d'une partition initiale aléatoire des individus telle que chaque groupe d'individus a un effectif supérieur au nombre de variables. Au sein de ces groupes on effectue une régression linéaire de Y par X. Ensuite, pour chaque individu i ($\forall i \in [\![1, n]\!]$), si il appartient par exemple au groupe C_k ($k \in [\![1, G]\!]$) on décide de le déplacer vers un autre groupe C_j ($j \in [\![1, G]\!]$ et $k \neq j$) si c'est le déplacement vers ce groupe qui induit la plus grande diminution de la valeur de $D(C_1, C_2, ..., C_G)$ (les coefficients de régression linéaire sont mis à jour quand on considère le déplacement d'un individu d'un groupe à l'autre), et qu'on respecte toujours $card(C_k - \{i\}) \geq p$. On arrête quand on arrive à parcourir tous les individus sans jamais pour aucun trouver un déplacement qui induit une diminution dans la valeur de $D(C_1, C_2, ..., C_G)$.
On voit bien que cette méthode dépend de la partition initiale des individus.

Les coefficients de régression sont mis à jour au sein de chaque groupe lorsque l'on considère l'ajout ou le retrait d'un individu de ce groupe.
On appelle algorithme Spaeth cet algorithme, et son pseudo-code est disponible en annexe B.

Algorithme mbpls :
On voit que la principale difficulté que l'on risque de rencontrer est d'avoir un problème où $n \leq p$ ou encore lorsque le nombre K de groupes est tel qu'il est difficile de déplacer un individu d'un groupe à l'autre sans violer la contrainte de cardinalité de chaque groupe d'individus vis-à-vis du nombre p de variables. Ainsi, lorsque l'on a p plus grand que n ou p trop grand par rapport à n, on pourra par exemple envisager d'effectuer une régression PLS au sein de chaque groupe, à la place d'une régression linéaire comme présenté ici. On parle alors de régression Clusterwise PLS. Un tel algorithme est présenté dans [Bougeard et al., 2017], et est implémenté dans le *package R mbclusterwise*. On appelera mbpls cet algorithme.

Algorithme kreg :
Une autre version, dite *batch* de cet algorithme existe. On appelle algorithme kreg cette version. Cette version peut s'apparenter à un algorithme qui reprend le principe des nuées dynamiques (*cf.* [Diday, 1971]) pour un problème de régression clusterwise. La fonction de coût qui diminue à chaque étape est $J(\beta, A) = \sum_{i=1}^{n} (y_i - X_i^T \beta^{A(Z_i)})^2$, où $\beta^c \in \mathcal{R}^p$, $\beta = (\beta^1, \beta^2, ..., \beta^G)$, et A est une fonction d'affectation qui à un individu fait correspondre un numéro de groupe. Les valeurs de β^c dépendent uniquement des individus qui sont classés dans le groupe c par la fonction A. L'algorithme kreg consiste en la répétition d'une phase d'affectation et d'une phase de minimisation jusqu'à ce que J ne décroisse plus. La phase d'affectation considère l'ensemble des individus et affecte chaque individu au groupe tel que l'erreur quadratique pour ce groupe soit plus petite que pour les autres groupes. La phase de minimisation, effectuée après la phase d'affectation, met à jour la valeur des β^c au sein de chaque groupe en effectuant une régression linéaire de y par X au sein de chaque groupe en ne prenant en compte que les individus du groupe en question. L'algorithme kreg est simplement la version *batch* de l'algorithme présenté dans [Späth, 1979].

Plus précisément, on a :

Phase d'affectation :
$\forall i \in [\![1, n]\!]\ A(i) = \text{argmin}_{r \in [\![1,G]\!]} (y_i - X_i^T \beta^r)^2$

Phase de minimisation :
$\forall c \in [\![1, G]\!]\ \beta^c$ est le vecteur de coefficients linéaires de y par X en ne considérant que les individus du groupe c.

On appelle algorithme kreg cet algorithme, et son pseudo-code est disponible en annexe C.

2.2.2 Algorithme probabiliste

On présente ici une autre méthode permettant de faire de la régression ClusterWise, qui correspond à la méthode présentée dans [DeSarbo and Cron, 1988].
Contrairement à précédemment, cet algorithme ne cherchera pas à chaque étape la nouvelle répartition des individus qui minimisera le plus une fonction de coût , mais sera basée sur l'estimation de paramètre pour maximiser une vraisemblance. L'estimation des paramètres de sorte à maximiser une vraisemblance est réalisée avec un algorithme dont chaque itération est décomposable en deux phases (détaillées plus loin).
On commence par expliciter le modèle que l'on considère, ainsi que certains résultats concernant l'algorithme utilisé.

On se place dans le cadre d'un modèle de mélange gaussien pour la variable réponse y.
On se place donc dans le modèle tel que :

$$\forall i \in [\![1, n]\!], y_i \sim \sum_{g=1}^{G} \lambda_g f_{i,g}(y_i | X_i, \sigma_g, B_g)$$

avec :
X_i un vecteur de taille p (qui correspond aux valeurs des p variables pour l'individu i)
$\sigma_g \in \mathbb{R}$
B_g est un vecteur de $\mathbb{R}^p$
et où $f_{i,g}(y_i | X_i, \sigma_g, B_g)$ correspond à une loi gaussienne de moyenne $X_i B_g$ et d'écart-type σ_g
Pour certaines raisons qu'on ne précise pas ici, on doit avoir $\sum_{g=1}^{G} \lambda_g = 1$ et $\forall g \in [\![1, G]\!]\ \lambda_g \geq 0$.
La log-vraisemblance correspondant à ce modèle est alors :

$$L = \sum_{i=1}^{n} ln(\sum_{g=1}^{G} \lambda_g f_{i,g}(y_i | X_i, \sigma_g, B_g))$$

En utilisant des résultats d'optimisation mathématique qu'on ne détaillera pas ici, on trouve que les expressions de λ_g, σ_g et B_g qui maximisent L sont telles que :

- $\forall g \in [\![1, G]\!]\ \lambda_g = h(p_{i,g})$ où $p_{i,g} = \frac{\lambda_g f_{i,g}(y_i | X_g, \sigma_g, B_g)}{\sum_{g=1}^{G} \lambda_g f_{i,g}(y_i | X_i, \sigma_g, B_g)}$ correspond en fait à la probabilité d'appartenance de l'individu i au groupe C_g calculée avec la règle de Bayes (on ne détaille pas l'expression de h)

- B_g est le vecteur de coefficients de la régression linéaire de y par X pondérée par $p_{i,g}^{\frac{1}{2}}$ (c'est-à-dire que lors de la régression linéaire avec la méthodes des moindres carrés on va chercher B_g tel que $B_g = \text{argmin}_{B_g} \left[\sum_{i=1}^{n} p_{i,g}(y_i - X_i B_g)^2\right]$ où X_i correspond aux valeurs des p variables pour l'individu i)

- $\sigma_g = m(p_{i,g}, B_g, Xg)$ (on ne détaille pas l'expression de m).

Pour l'algorithme, on considère donc des valeurs de départ de B_g, σ_g, λ_g et $Xg\ \forall g \in [\![1, G]\!]$.
On répète ensuite les deux phases suivantes jusqu'à ce que l'augmentation de la log-vraisemblance

soit aussi petite que souhaitée :
Une première phase consiste ensuite à mettre à jour les valeurs de $p_{i,g}$ (avec $p_{i,g} = \frac{\lambda_g f_{i,g}(y_i|X_g,\sigma_g,B_g)}{\sum_{g=1}^{G} \lambda_g f_{i,g}(y_i|X_i,\sigma_g,B_g)}$) puis de λ_g (avec $\lambda_g = h(p_{i,g})$).
Une seconde phase consiste à mettre à jour les valeurs de B_g et σ_g.
Avec les valeurs finales de B_g, σ_g et λ_g $\forall g \in [\![1, G]\!]$ on a naturellement une partition $C_1, ..., C_k$ de $\{1, ..., n\}$ par le biais de $p_{i,g}$.

Remarque :
En réalité, l'algorithme en deux phases que l'on vient de présenter peut être vu comme un algorithme EM (Espérance-Maximisation) où la variable latente serait une variable qui indiquerait pour chaque individu son groupe d'appartenance. Ici, lors de la phase "Espérance" on détermine les valeurs de $p_{i,g}$ et λ_g, et lors de la phase "Maximisation" on détermine les valeurs de B_g et σ_g.

De manière pratique :
On commence par déterminer une partition aléatoire $C_1, ..., C_G$ des individus telle que $\forall g \in [\![1, G]\!], card(C_g) > p$ (en supposant que cela est possible). La proportion initiale d'individus au sein du groupe g est notée λ_g. On note Y_i la valeur de la variable réponse pour l'individu i. Au sein de chaque groupe g on effectue une régression linéaire de y par X. On obtient, pour chaque groupe C_g, un vecteur de coefficients $(\beta_{g,1}, ..., \beta_{g,p})$ noté B_g issu de cette régression linéaire. On note σ_g l'écart-type au sein du groupe C_g.
Ces valeurs de B_g, σ_g et λ_g $\forall g \in [\![1, G]\!]$ constituent les valeurs de départ de l'algorithme décrit précédemment.
On déroule ensuite l'algorithme précédemment décrit.

On note Xg les valeurs des variables de la matrice X uniquement pour les individus du groupe C_g et y^g les valeurs de la variable réponse uniquement pour les individus du groupe C_g, $\forall g \in [\![1, G]\!]$.
On appelle algorithme flexmix cet algorithme, et son pseudo-code est disponible en annexe D.

2.2.3 Prédiction pour un nouvel individu

On propose maintenant d'expliquer une méthode permettant, à partir d'une partition $C_1, ..., C_G$ de $\{1, ..., n\}$ obtenue par exemple après avoir appliqué une des deux méthodes décrites précédemment, de prédire la valeur de y pour un nouvel individu.
On considère un nouvel individu i' décrit par p valeurs des variables $X^1, ..., X^p$, notées $X_i'^1, ..., X_i'^p$.
On suppose qu'au sein de chaque groupe C_g d'individus ($\forall g \in [\![1, G]\!]$), les valeurs du vecteur $X = (X^1, ..., X^p)$ proviennent d'une loi de densité propre à chaque groupe C_g notée $f_g(X)$. Généralement, la loi de densité est estimée empiriquement au sein de chaque groupe C_g si on dispose d'un assez grand nombre d'individus dans chacun de ces groupes. On note λ_g la proportion d'individus dans le groupe C_g.
Pour le nouvel individu, on peut calculer sa probabilité d'appartenance à un groupe C_g en utilisant la règle de Bayes. En effet, on a :

$$\forall i \in [\![1, n]\!], \forall g \in [\![1, K]\!] P(i \in C_g) = P(C_g|(X_i'^1, ..., X_i'^p)) = \frac{\lambda_g f_g(X_i'^1, ..., X_i'^p)}{\sum_{k=1}^{K} \lambda_k f_k(X_i^1, ..., X_i^p)}$$

On peut ensuite décider d'affecter à l'individu la prédiction de la régression linéaire au sein du groupe C_g tel que $g = \text{argmax}_k\, P(i \in C_k)$ ou encore, si on note $p_g(X_i'^1, ..., X_i'^p)$ la prédiction de la régression linéaire au sein du groupe C_g, d'affecter la valeur $\sum_{g=1}^{G} P(i' \in C_g) p_g((X_i'^1, ..., X_i'^p)$

Si on ne peut pas supposer qu'au sein de chaque groupe C_g d'individus ($\forall g \in [\![1, G]\!]$), les valeurs du vecteur $X = (X^1, ..., X^p)$ proviennent d'une loi de densité propre à chaque groupe C_g (par exemple si au sein d'un groupe C_g on a trop peu d'individus et qu'on ne peut conséquemment pas obtenir une estimation empirique de la loi de densité pour ce groupe), une autre méthode pour obtenir la valeur de la prédiction pour un nouvel individu i' décrit par p valeurs des variables $X^1, ..., X^p$, notées $X_i'^1, ..., X_i'^p$ peut être :

— lui affecter la prédiction obtenue au sein du groupe C_g dont le barycentre noté $(G_g^1, ..., C_g^p)$ est le plus proche au sens d'une distance (euclidienne, Manhattan, etc.) de $(X_i'^1, ..., X_i'^p)$.

— lui affecter la prédiction obtenue au sein du groupe C_g tel que le groupe C_g est le groupe d'appartenance prédit avec la méthodes des plus proches voisins, pour une certaine distance (euclidienne, Mahalanobis, etc.)

2.2.4 Illustration sur données simulées

Nous proposons une illustration de la pertinence de la régression clusterwise sur un exemple simple simulé dans les figures 1 et 2. Pour les figures 1 et 2, nous avons simulé deux groupes d'individus selon deux modèles linéaires différents. Ces groupes correspondent aux points rouges et bleus des figures 1 et 2. Pour la figure 1, nous avons ensuite appris un modèle de régression linéaire pour les individus de l'ensemble de ce deux groupes. La droite de régression linéaire relative à ce modèle linéaire correspond à la droite noire de la figure 1. Pour la figure 2, nous avons exécuté l'algorithme flexmix (*cf.* partie 2.2.2) sur les individus de l'ensemble de ce deux groupes avec un nombre de groupes en paramètres égal à 2. Les deux droites de régression linéaire trouvées sont représentées par les droites bleue et rouge sur la figure 2. Comme nous pouvons le constater sur la

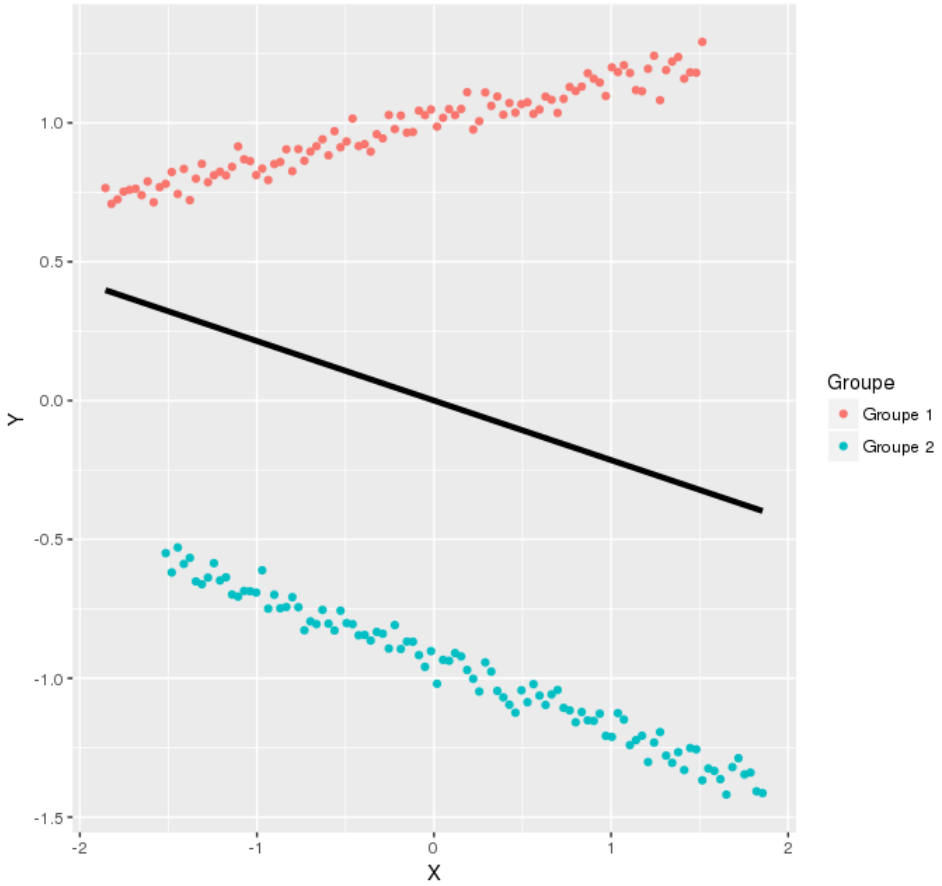

FIGURE 1 – Régression linéaire sur donnés simulées
Régression linéaire classique sur des données simulées présentant une structure claire en deux groupes : la droite noire correspond à la droite de régression linéaire

figure 1, le modèle appris ne correspond pas du tout à la réalité. 2.2.2). Comme nous pouvons le constater sur la figure 2, les deux groupes d'individus sont bien retrouvés par l'algorithme flexmix, et le modèle appris au sein de chaque groupe correspond bien à la réalité, pas comme pour la figure 1. En effet, pour la figure 1, le coefficient de détermination est égal à 0.04, alors que pour la figure 2, les valeurs du coefficient de détermination pour les deux groupes sont égales à 0.94 et 0.97.

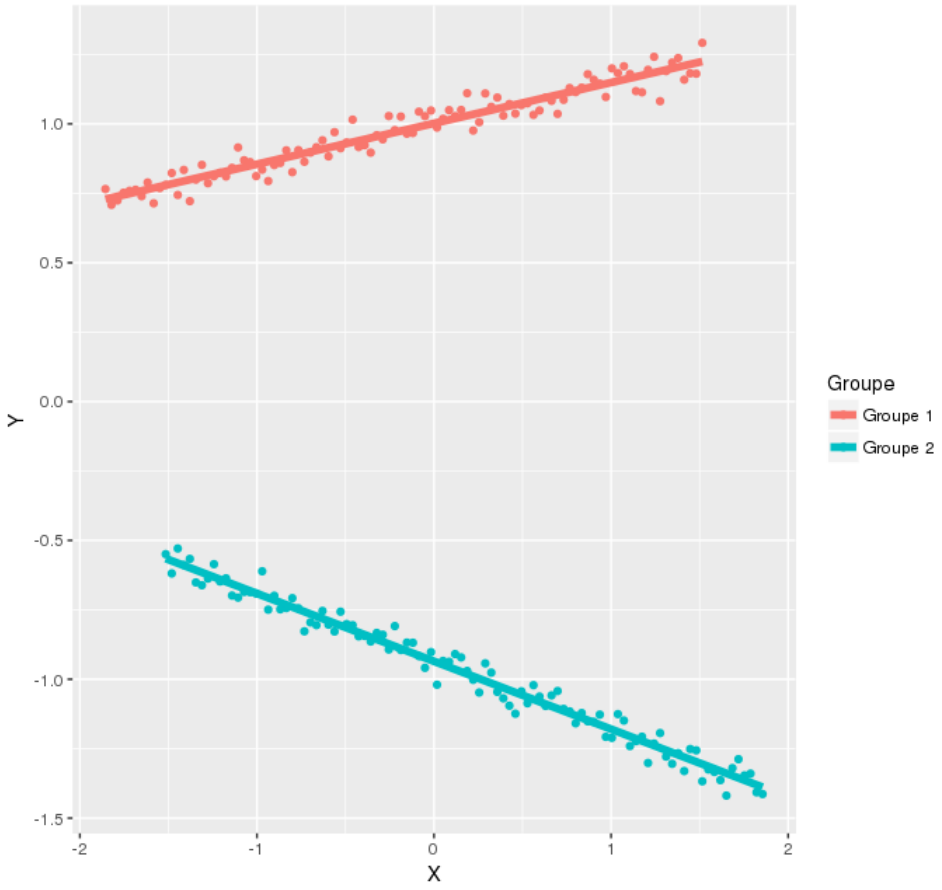

FIGURE 2 – Régression linéaire Clusterwise sur donnés simulées
Résultats pour la régression Clusterwise linéaire avec deux groupes sur des données simulées présentant une structure claire en deux groupes

2.3 Algorithme des cartes de Kohonen

On considère un problème avec n individus décrits par p variables quantitatives où on cherche à regrouper les n individus en G groupes, avec chaque groupe qui correspond à un sommet d'une grille 2D, tels que deux individus voisins dans l'espace des données ($\mathcal{R}^p$) mais n'appartenant pas au même groupe ont leur groupe qui correspondent à des sommets voisins de la grille. On note X la matrice de données qui contient la valeur de chacune des p variables explicatives pour chacun des n individus. On note $z_i \in \mathcal{R}^p$ le vecteur contenant la valeur des p variables pour l'individu i $\forall i \in [\![1, n]\!]$. On note $C_1, ..., C_G$ les G groupes que l'on veut finalement obtenir.

2.3.1 Principe de l'algorithme

Le principe de l'algorithme des cartes de Kohonen (*cf.* [Kohonen, 2001] et [Dreyfus, 2008]) est de chercher à déterminer G groupes d'individus, chacun caractérisé par un point de l'espace des données (i.e un point de $\mathcal{R}^p$), appelé référent, qui est associé à un sommet de la grille, tels que la somme des distances (on considère généralement la distance euclidienne) entre chaque individu et chaque référent soit minimale. De plus, la distance euclidienne entre un individu et un référent est pondérée par une valeur qui dépend de la distance sur la grille 2D entre les sommets relatifs au groupe de l'individu et au référent.
La grille 2D permet d'avoir une représentation dans un espace de dimension 2 des données qui sont dans un espace de départ de dimension p, tout en respectant la topologie de cet espace de départ. Deux points de $\mathcal{R}^p$ proches pour une distance définie sur $\mathcal{R}^p$ seront associés à des sommets proches sur la grille 2D.
On note $w_g \in \mathcal{R}^p$ le référent du groupe g $\forall g \in [\![1, G]\!]$, et $W = (w_1, ..., w_G)$. On appelle neurone un

sommet de la grille 2D.

L'algorithme des cartes de Kohonen cherche $C_1, ..., C_G$ et W de sorte à minimiser une certaine fonction de coût, notée L, telle que :

$$J(A, W) = \sum_{g=1}^{G} \sum_{i=1}^{n} K(\delta(A(z_i), g)) \|Z_i - w_g\|_2$$

avec :

—

$$A \colon \mathcal{R}^{p+1} \to 1, 2, ..., G$$
$$Z \mapsto c$$

qui affecte à chaque vecteur z_i le numéro de son groupe (le neurone sur la grille).

— K une fonction de voisinage (détaillé après)

— $\delta(A(z_i), g)$ le plus court chemin dans la grille entre le neurone c le neurone $A(z_i)$

— les référents qui sont notés $W = (w_1, w_2, ..., w_G)$, avec $\forall g \in [\![1, G]\!]\ w_g \in \mathcal{R}^p$

La valeur de la fonction de voisinage K est d'autant plus grande que $\delta(A(z_i), g)$ est petite. De plus, pour une valeur de $\delta(A(z_i), g)$ fixée, la valeur de K dépend d'une valeur t (non-mentionnée ici) qu'on appellera température, et est d'autant plus grande que la valeur de t est grande. Deux formes très courantes de la fonction de voisinage sont :

$$K_t \colon \mathcal{R} \to \mathcal{R}$$
$$\delta(A(z_i), g) \mapsto e^{-\frac{\delta(A(z_i), g)}{t}}$$

appelée fonction de voisinage gaussienne
ou

$$K_t \colon \mathcal{R} \to \mathcal{R}$$
$$\delta(A(z_i), g) \mapsto 1 \quad si \quad \delta(A(z_i), g) \leq t \qquad et \qquad 0 \quad sinon$$

appelée fonction de voisinage porte
On comprend bien que pour une valeur de t élevée, la fonction de coût J tient compte, pour un individu i, de la distance euclidienne pondérée de cet individu à tous les référents. De même, pour une valeur de t assez faible pour que $K(\delta(A(z_i), g)$ soit nulle pour $A(z_i) \neq g$ et vaille 1 pour $A(z_i) = g$, la fonction de coût peut s'apparenter à $\sum_{g=1}^{G} \sum_{i \in C_g} K(\delta(A(z_i), g)) \|z_i - w_g\|_2$. On montre qu'une valeur élevée de la température favorise la "mise en ordre" de la grille 2D (le fait que les maillons de la grille ne se touchent pas) et qu'une valeur très petite de la température favorise le déploiement de la grille sur la projection de l'espace des données ($\mathcal{R}^p$) dans un espace 2D.
Dans ce qui suit, on présente l'algorithme pour une valeur donnée de la température t.

2.3.2 Algorithme

L'algorithme des cartes de Kohonen consiste en la répétition de deux phases jusqu'à ce que J soit aussi petite que souhaitée, en partant d'une répartition aléatoire initiale des individus dans les G groupes. La première phase, dite "d'affectation", consiste en réalité en la minimisation de J selon A. On a alors : $\forall i \in [\![1, n]\!]\ A(z_i) = \text{argmin}_{r \in [\![1,G]\!]} \sum_{g=1}^{G} K(\delta(r, c)) \|z_i - w_g\|_2$. La seconde phase, dite de "minimisation", consiste en réalité en la minimisation de L selon W. On a alors : $\forall g \in [\![1, G]\!]\ w_g = \frac{\sum_{r=1}^{G} K(\delta(r,g)) z_r}{\sum_{r=1}^{G} K(\delta(r,g)) n_r}$ avec n_r le nombre d'individus dans le groupe C_r.
On montre que cet algorithme converge en un temps fini.
Dans la pratique, on exécute cet algorithme plusieurs fois en considérant des valeurs décroissantes de la température t.

Le pseudo-code de cet algorithme est disponible en annexe E.

3 Algorithmes proposés

Dans cette partie nous présentons le principe des algorithmes que nous avons développé pendant le stage

3.1 Présentation du problème

On considère un problème avec n individus décrits par p variables quantitatives, appelées variables explicatives, où on cherche à prédire la valeur d'une autre variable quantitative, appelée variable réponse. On suppose que les n individus sont répartis en G groupes, sans toutefois savoir a priori à quel groupe chaque individu appartient. On aimerait alors déterminer la répartition optimale des n individus dans ces G groupes de sorte qu'au sein de chaque groupe on ait la meilleure régression linéaire possible de la variable réponse par les variables explicatives. On souhaite également faire correspondre chaque groupe à un sommet d'une grille 2D, telle que deux individus proches pour une certaine distance dans l'espace des données ($\mathcal{R}^p$) appartiennent à des groupes relatifs à des sommets proches pour la grille 2D. La grille 2D est appelée carte topologique et ses sommets sont appelés neurones.
On introduit ici quelques notations que l'on utilisera par la suite. Pour ne pas s'encombrer de certaines notations qui seraient un peu pénibles, on considère que parmi les p variables explicatives une variable explicative prend la valeur 1 pour tous les individus. Cela permet notamment de ne pas à avoir à rentrer dans certaines précisions concernant l'*intercept* de la régression linéaire.
On note X la matrice de dimension $n \times p$ qui contient la valeur de chacune des p variables explicatives pour chacun des n individus. On note X_i le vecteur de dimension $p \times 1$ qui contient les p valeurs des variables explicatives pour le i$^{\text{e}}$ individu ($\forall i \in [\![1, n]\!]$). On note $X_{i,j}$ la valeur à la ligne i et à la colonne j de la matrice X. On note y le vecteur de taille n qui contient la valeur de la variable réponse pour chaque individu, et y_i le i$^{\text{e}}$ élément de Y (la valeur de la variable réponse pour le i$^{\text{e}}$ individu). On note z_i le vecteur de dimension $(p+1)\times 1$ avec $z_i = \begin{pmatrix} y_i \\ X_i \end{pmatrix}$.

3.2 Algorithme SOM-Clusterwise

Nous présentons un algorithme que nous avons mis au point pour effectuer de la régression clusterwise. Nous détaillons notamment son principe et la preuve de sa convergence en un temps fini. Nous abordons certaines simplifications dans son implémentation qui ont dû être réalisées afin d'avoir un temps d'exécution réaliste. Nous discutons de plusieurs critères d'arrêt de l'algorithme afin de limiter son sur-apprentissage et améliorer sa robustesse à certains cas.

3.2.1 Principe de l'algorithme SOM-clusterwise

Le principe de l'algorithme que l'on présente ici est de déterminer G groupes d'individus, chacun caractérisé par un vecteur de $\mathcal{R}^p$ appelé référent (défini dans la suite). On veut d'une part, au sein de chaque groupe, pour tout individu du groupe, minimiser le carré du résidu dans le modèle linéaire où la valeur à prédire est la variable réponse et les coefficients de régression sont les éléments du référent de ce groupe. On considère d'autre part, à l'instar de la méthode des cartes de Kohonen, une grille 2D dont chaque neurone correspond à un référent, qui préserve la topologie de l'espace d'entrée (ici $\mathcal{R}^{p+1}$) et permet d'en faire une représentation simplifiée.

Nous définissons une fonction de coût, notée J et cherchons alors à déterminer G groupes ainsi que G référents qui minimisent sa valeur. La fonction de coût J est telle que :

$$
\begin{aligned}
J(A,\beta) &= \sum_{i=1}^{n}\sum_{g=1}^{G} K(\delta(A(z_i),g))(y_i - \hat{y_i^c})^2 \\
&= \sum_{i=1}^{n}\sum_{g=1}^{G} K(\delta(A(z_i),g))(y_i - X_i^T\beta^c)^2 \\
&= \sum_{i=1}^{n}\sum_{g=1}^{G} K(\delta(A(z_i),g))(y_i - \sum_{j=1}^{p} X_{i,j}\beta_j^c)^2 \\
&= \sum_{i=1}^{n} d(A(z_i), y_i)
\end{aligned}
$$

avec :

—

$$
\begin{aligned}
A\colon \mathcal{R}^{p+1} &\to 1,2,...,G \\
z &\mapsto c
\end{aligned}
$$

qui affecte à chaque vecteur z_i le numéro de son groupe (le neurone sur la grille).

— K une fonction de voisinage comme pour les cartes de Kohonen

— δ le plus court chemin dans la grille entre le neurone g le neurone $A(Z_i)$

— les référents qui sont notés $\beta = (\beta^1, \beta^2, ..., \beta^G)$, $\forall g \in [\![1, G]\!]$ $\beta^g \in \mathcal{R}^p$ et $\forall g \in [\![1, G]\!]$ $\beta^g = (\beta_1^g, \beta_2^g, ..., \beta_p^g)$

La distance euclidienne entre y_i et $\hat{y}_i^c$ est pondérée par une fonction de la distance sur la grille 2D entre chaque neurone et le neurone qui a été affecté à z_i.

3.2.2 Algorithme

La méthode que l'on présente ici consiste en la répétition de deux phases jusqu'à ce que J ne décroisse plus, en partant d'une répartition aléatoire initiale des individus dans les G groupes. La première phase, dite phase d'affectation, consiste à mettre à jour la fonction A en affectant à chaque Z_i ($i \in [\![1, n]\!]$) le neurone r qui minimise $d(r, Y_i)$. La deuxième phase, dite phase de minimisation, met à jour les valeurs des G référents de sorte à minimiser la fonction de coût. La première phase consiste en réalité en la minimisation de la fonction J selon A pour β fixé, et la seconde phase en la minimisation de la fonction J selon β pour A fixée.

Plus précisément, pour les deux phases, on a :

Phase d'affectation :
$\forall i \in [\![1, n]\!]$ $A(z_i) = \text{argmin}_{r \in [\![1,G]\!]} \sum_{g=1}^{G} K(\delta(r, g))(y_i - X_i^T\beta^g)^2$

Phase de minimisation :

$\forall g \in [\![1, G]\!]$, on cherche β^g tel que $\sum_{i=1}^{n} K(\delta(A(z_i), g))(y_i - X_i^T\beta^g)^2$ soit minimal (ce qui correspond à chercher β tel que $J(A, \beta)$ soit minimal pour A fixée).

On montre que la fonction

$$
\begin{aligned}
f\colon \mathcal{R}^p &\to \mathcal{R} \\
\beta^g = (\beta_1^g, \beta_2^g, ..., \beta_p^g) &\mapsto \sum_{i=1}^{n} K(\delta(A(z_i), c))(y_i - X_i^T\beta^g)^2
\end{aligned}
$$

est convexe si et seulement si $X^T M_g X$ est définie positive, avec $M_g = diag(M_{i,g})$ et $M_{i,g} = K(\delta(A(z_i), g)$ $\forall i \in [\![1, n]\!]$.

En effet, on a : $\frac{\partial f}{\partial \beta^g_{j_1} \beta^g_{j_2}} = \sum_{i=1}^{n} 2K(\delta(A(z_i), g) X_{i,j_1} X_{i,j_2}$, qui est égal à la valeur à la ligne j_1 et à la colonne j_2 de la matrice hessienne de f, qui correspond aussi à la valeur à la ligne j_1 et à la colonne j_2 de la matrice $X^T M_g X$. On sait de plus qu'une fonction à plusieurs variables est convexe si et seulement si sa matrice Hessienne est définie positive. Ici, on peut considérer que $X^T M_g X$ est définie positive si l'on suppose que $X^T M_g X$ est inversible (une matrice définie positive est une matrice positive et inversible et l'on sait que $X^T M_g X$ es positive). On note que si des colonnes de X sont linéairement dépendantes entre elles, alors la matrice $X^T M_g X$ n'est pas inversible. Plus généralement, la matrice $X^T M_g X$ n'est pas inversible si et seulement si il existe (j_1, j_2) dans $[\![1, p]\!] \times [\![1, p]\!]$ tels que $X_{M_g, j_1} = \mu X_{M_g, j_2}$ avec $\mu \in \mathcal{R}$ et X_{M_g, j_1} (resp. X_{M_g, j_2}) qui correspond à la $j_1{}^{\text{e}}$ (resp. $j_2{}^{\text{e}}$) colonne de X avec uniquement les lignes i tels que le i^{e} élément diagonal de M_g est non-nul.

Puisque f est une fonction convexe, si on suppose que $X^T M_g X$ est inversible, le minimum global est atteint en l'unique point critique, c'est-à-dire l'unique $\beta^{g,*}$ tel que $\frac{\partial f}{\partial \beta^{g,*}} = (\frac{\partial f}{\partial \beta_1^{g,*}}, \frac{\partial f}{\partial \beta_2^{g,*}}, ..., \frac{\partial f}{\partial \beta_p^{g,*}})^T = (0, 0, ..., 0)^T = 0_p$
On a :

$$\frac{\partial f}{\partial \beta^g}(\beta^g) = 0_p$$

$$\Leftrightarrow \quad \sum_{i=1}^{n} K(\delta(A(z_i), g)) X_i (y_i - X_i^T \beta^g) = 0_p$$

$$\Leftrightarrow \quad \sum_{i=1}^{n} K(\delta(A(z_i), g)) X_i y_i = \sum_{i=1}^{n} K(\delta(A(Z_i), g)) X_i (X_i^T \beta^g)$$

$$\Leftrightarrow \quad \beta^g = \left(\sum_{i=1}^{n} K(\delta(A(z_i), g)) X_i X_i^T \right)^{-1} \left(\sum_{i=1}^{n} K(\delta(A(z_i), g)) X_i y_i \right)$$

$$\Leftrightarrow \quad \beta^g = \left(X^T M_g X \right)^{-1} X^T M_g Y$$

Durant la phase de minimisation, on effectue donc :
$\forall g \in [\![1, G]\!]\ \beta^g = \left(X^T M_g X \right)^{-1} X^T M_g Y$

Convergence :

La fonction de coût J diminue à chaque étape. En effet, si on se place au début de l'étape t, on a :

— Lors de la phase d'affectation :
$\forall i \in [\![1, n]\!] \sum_{g=1}^{G} K(\delta(A^t(z_i), g))(y_i - X_i^T \beta^{g,t-1})^2 \leq \sum_{g=1}^{G} K(\delta(A^{t-1}(z_i), g))(y_i - X_i^T \beta^{c,t-1})^2$,
donc $J(\beta^{t-1}, A^t) \leq J(\beta^{t-1}, A^{t-1})$, où A^t représente la fonction A déterminée lors de la phase d'affectation de l'étape t et β^{t-1} le vecteur β déterminé précédemment à l'étape $t-1$.
— Lors de la phase de minimisation : on a déjà justifié plus haut que $J(\beta^t, A^t) \leq J(\beta^{t-1}, A^t)$
Finalement, on a bien : $J(\beta^t, A^t) \leq J(\beta^{t-1}, A^{t-1})$

On a vu que la fonction de coût J décroît bien à chaque étape. Puisque la valeur de J est minorée par 0, notre algorithme fait bien converger la valeur de J. On montre aussi que la valeur de J n'évolue plus au bout d'un nombre fini d'étapes. Cependant, la manière dont on fait décroître J (on ne regarde qu'un paramètre de J à la fois : A puis β), il est tout à fait possible de rester bloqué à un minimum local.

Le pseudo-code de cet algorithme est disponible en annexe F.

Remarque :
La fonction de voisinage K dépend d'une valeur t appelée température. Nous ne détaillons pas les différentes fonctions de voisinage possibles. Nous précisons uniquement que pour $\delta(r, g)$ fixée, la valeur de $K(\delta(r, g))$ est d'autant plus petite que la valeur de t est petite. Ainsi, plus la valeur de

t diminue, plus on comprend que la valeur de chaque référent β^g est apprise principalement avec les individus du groupe g. En effet, comme on l'a vu, la phase de minimisation consiste à affecter à β^g la valeur des coefficients dans la régression linéaire pondérée de y par X par les valeurs de $K(\delta(A(z_i), g))$ $(i \in [\![1, n]\!])$.
Pour une valeur de température t assez petite telle que $K(\delta(r, g)) = 1$ si $r = g$ et $K(\delta(r, g)) = 0$ si $r \neq g$, on voit que l'algorithme SOM-Clusterwise correspond à l'algorithme kreg.

Nous avons ici montré la décroissance de la fonction de coût J à une température t fixée. Dans la pratique, on fait décroître la valeur de la température t au fur et à mesure de l'algorithme, notamment avant chaque passage dans la boucle **tant que** de l'algorithme F. On peut montrer que les grandes valeurs de t favorisent la "mise en ordre" (les maillages de la carte 2D ne se croisent pas) de la carte 2D tandis que les petites valeurs de T favorisent l'étalement de la carte dans l'espace des données" (*cf.* [Dreyfus, 2008]).

3.2.3 Détails d'implémentation et illustration

L'implémentation a été réalisé en reprenant du code R du package R *SOMbrero*. Nous avons naturellement modifié les parties de code relatives aux phases d'affectation et de minimisation, mais aussi les parties de code relatives aux calculs des valeurs de fonction de voisinage. Plus précisément, dans le package R *SOMbrero*, le calcul de la fonction de voisinage était réalisé autant de fois qu'il y avait de paires de neurones sur la carte, ce qui peut se révéler coûteux en temps pour des phases d'entraînement nécessitant beaucoup d'itérations, si on change la valeur de la fonction de voisinage à chaque itération, et pour des grilles comportant un nombre élevé de neurones. Pour une grille carrée de taille $k \times k$, les valeurs des distances sur la grille vont de 0 à $2(k-1)$. Il n'est donc en réalité nécessaire que d'effectuer $2k-1$ calculs de la fonctions de voisinage. Cette modification que nous avons effectuée permet donc de passer d'une complexité en $\mathcal{O}(k^2)$ à une complexité en $\mathcal{O}(k)$ pour le calcul de la fonction de voisinage.
Pour des problèmes liés à la vitesse de calcul, la phase d'affectation telle que nous l'avons implémentée n'est plus :
$\forall i \in [\![1, n]\!]\ A(z_i) = \text{argmin}_{r \in [\![1,G]\!]} \sum_{g=1}^{G} K(\delta(r, g))(y_i - X_i^T \beta^g)^2$
mais :
$\forall i \in [\![1, n]\!]\ A(z_i) = \text{argmin}_{r \in [\![1,G]\!]} (y_i - X_i^T \beta^r)^2$
De plus, la décroissance de la température ne s'effectue pas de manière optimale, c'est-à-dire avant chaque passage dans la boucle **tant que** de l'algorithme F, mais à chaque entrée dans cette boucle. Pour chaque valeur de la température, on ne considère donc qu'une fois l'ensemble des individus.
Une fois l'implémentation terminée, nous effectuons des tests sur des données simulées, avec une fonction de voisinage gaussienne.
Nous constatons que la température initiale et la manière dont on fait décroître la température influent naturellement sur le temps d'exécution et sur la classification finalement obtenue des individus.
On constate que l'on atteint la classification finale des individus ainsi que des valeurs très approchées des valeurs finales pour les référents bien avant que la fonction de coût ait atteinte un minimum local. Pour une température basse (i.e pour des valeurs de la fonction de voisinage petites), on favorise le fait que les référents se rapprochent des coefficients de régression de leur groupe uniquement. En effet, pour une température basse (i.e pour des valeurs de la fonction de voisinage petites), durant la phase de minimisation, lorsque l'on met à jour le référent β^g, les valeurs de la diagonale de M_g correspondant à des individus affectés à un autre neurone qu'au neurone g sont quasiment nulles. La valeur de β^g calculée correspond alors simplement à la valeur des coefficients de régression linéaire de y par X uniquement dans le groupe g. En effet, β^g tel qu'on le calcule correspond aux coefficients de régression linéaire pondérée de y par X avec des pondérations quasiment nulles pour les individus ne faisant pas partie du groupe g.
Nous réalisons des tests informels (i.e des tests sans méthodologie précise destinés à des premières constatations simples) sur des données simulées assez simples. Nous exécutons notre algorithme avec une fonction de voisinage gaussienne, avec une certaine valeur de température initiale et nous faisons décroître cette température linéairement à chaque passage dans la boucle **tant que** de l'algorithme F, jusqu'à une certaine valeur de température choisie à l'avance. Nous prêtons attention aux performances de prédiction en généralisation et à la classification finale obtenue. La prédiction de la valeur obtenue pour un nouvel individu se fait avec la méthode des plus proches voisins, telle que décrite dans la partie 2.2.3.Plus précisément, la valeur de k choisie pour les données simulées

est celle qui minimise l'erreur de classement, c'est-à-dire celle qui permet, avec la classification réelle des individus avec lesquels on a exécute l'algorithme (l'ensemble d'apprentissage), d'arriver à retrouver le mieux la classification réelle des individus pour lesquels on va effectuer la prédiction (l'ensemble de test). Nous constatons tout d'abord que nous obtenons de meilleures performances en apprentissage (mesurées avec la *Mean Squared Error* et en classification (indice rand et indice de Calinsky-Harabasz) que si l'on exécutait d'abord un algorithme des k-moyennes en prenant en compte les valeurs des variables explicatives et de la variable réponse, puis qu'on apprenait un modèle linéaire au sein de chaque groupe déterminé. Les figures 3 et 4 illustrent ce propos. Pour ces figures, nous avons simulé trois groupes d'individus au sein de chacun desquels on a un modèle linéaire particulier. Les résultats de la figure 3 correspondent à l'exécution de l'algorithme des 3-moyenne en prenant en compte la variable explicative X et la variable réponse y, puis à l'apprentissage d'un modèle linéaire au sein de chacun de ces groupes. Les couleurs représentent les groupes trouvés avec l'algorithme des 4-moyennes, et les droites grises aux droites de régression linéaires au sein de chacun de ces groupes. On constate que les modèles de régression linéaires sont bien moins pertinents sur la figure 3 que sur la figure 4. On remarque notamment que lors de l'attribution d'un numéro de groupe pour un nouvel individu, on fera bien plus d'erreur avec le résultat présenté en figure 3 qu'avec celui présenté en 4.

Nous avons bien sûr exécuté notre algorithme avec plusieurs initialisations des groupes différentes et plusieurs valeurs de température initiales. Sur les figures 3 et 4 nous présentons le résultat obtenu majoritairement (parmi toutes les initialisations différentes réalisées) en classification avec respectivement l'algorithme de 4-moyennes et notre algorithme. Sur les figures 3 et 4, les droites grises correspondent aux modèles appris au sein de chaque groupe.

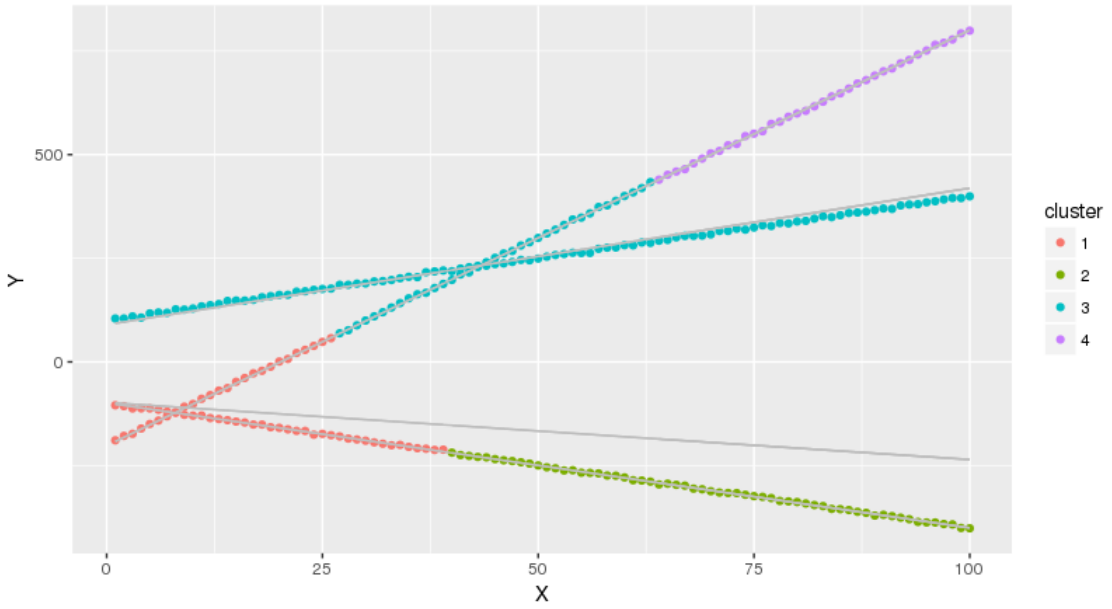

FIGURE 3 – Algorithme des 4-moyennes sur donnés simulées
Résultats pour l'algorithme des 4-moyennes suivi d'une régression linéaire au sein de chacun des groupes sur des données simulées présentant une structure claire en trois groupes ; les couleurs sont relatives à la classification trouvée et les droites de régression pour le modèle linéaire relatif à chque groupe sont les droites grises

Nous constatons aussi que pour tous les exemples simulés simples considérés, nous arrivons à trouver une valeur initiale de la température qui nous permette de retrouver la classification réelle des individus.

Nous proposons maintenant une illustration de la notion d'ordre relative aux cartes topologiques de Kohonen de l'algorithme SOM-Clusterwise, dans la figure 5. Pour la figure 5, nous avons considéré la fonction sinusoïde à laquelle nous avons rajouté un bruit gaussien, pour un total de 500 points. Pour chaque point on a donc la valeur de la variable explicative et la valeur de la variable

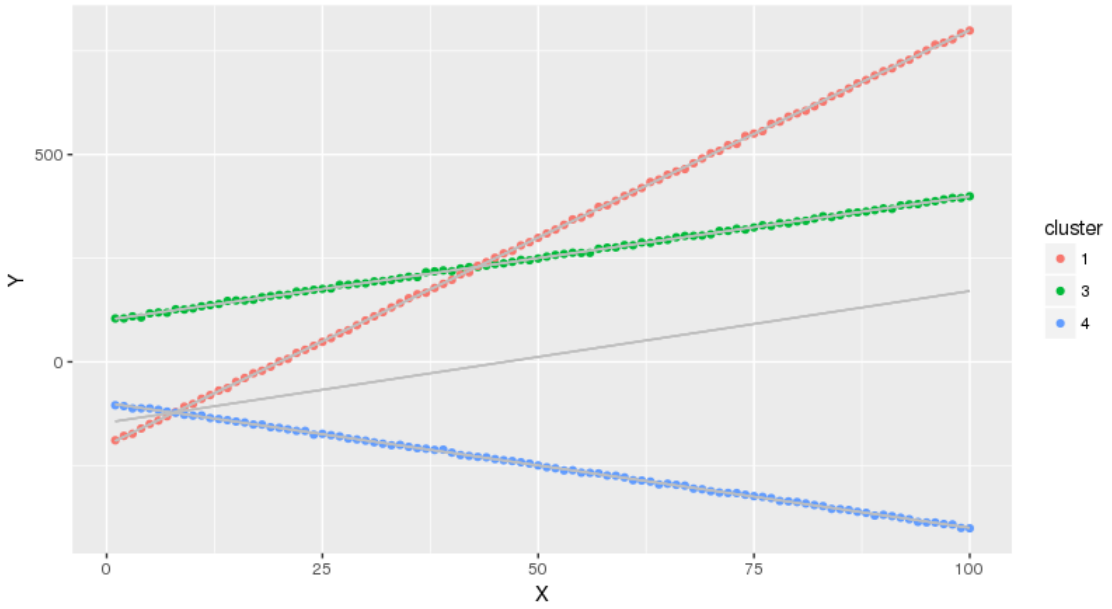

FIGURE 4 – Régression SOM-Clusterwise sur donnés simulées
Résultats pour la régression SOM-Clusterwise linéaire avec quatre groupes sur des données simulées présentant une structure claire en trois groupes ; les couleurs sont relatives à la classification trouvée et les droites de régression pour le modèle linéaire relatif à chque groupe sont les droites grises

réponse qui correspond au sinus de la valeur de la variable explicative. Nous avons ensuite exécuté l'algorithme SOM-Clusterwise avec un nombre de groupes passé en paramètres égal à 100. Dans la figure 5 est présenté le résultat de cette exécution, notamment la valeur prédite de la variable réponse pour chaque point.

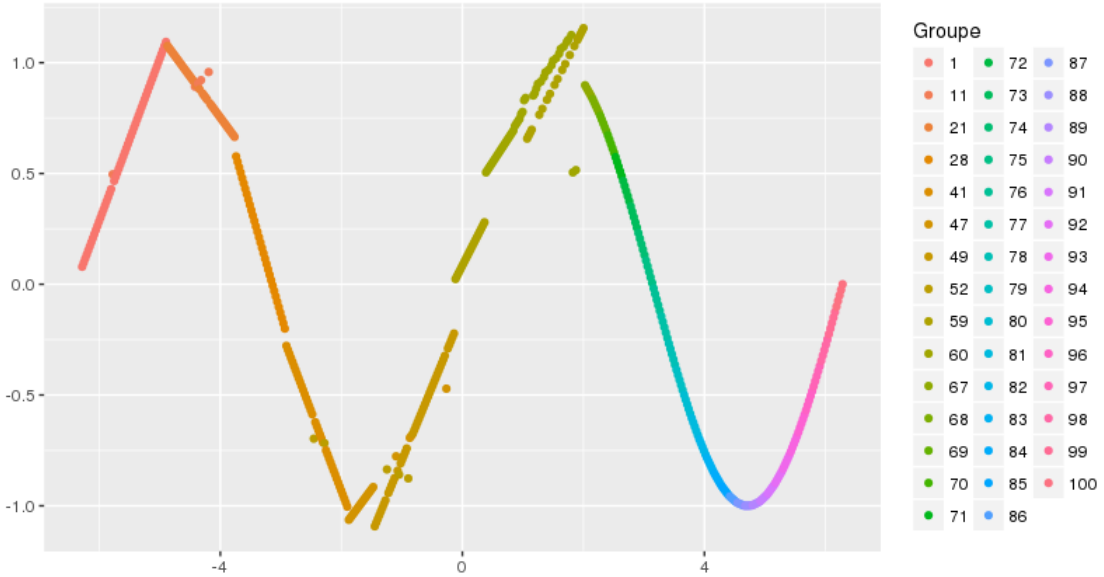

FIGURE 5 – Illustration de la notion d'ordre pour l'algorithme SOM-Clusterwise
Résultats pour l'algorithme SOM-Clusterwise sur la simulation d'une sinusoïde bruitée de 500 points : chaque point du graphique présente en ordonnée la valeur prédite trouvée

Nous constatons sur la figure 5 que l'on retrouve bien l'aspect de la sinusoide de départ. Nous avons refait la même expérience avec les algorithmes flexmix et mbpls. Afin de ne pas surcharger le rapport, nous ne présentons pas le résultat de ces deux algorithmes. Nous précisons uniquement que pour l'algorithme flexmix le résultat de cet algorithme ne permet pas de retrouver l'aspect de sinusoïde de départ. Pour l'algorithme mbpls, un problème d'implémentation ne permet pas de donner un nombre de groupes en paramètres supérieur à 25, quelque soit le jeu de données étudié. De plus, toujours pour un problème d'implémentation, on ne peut pas considérer une seule variable explicative.

3.2.4 Robustesse de l'algorithme face à $n < p$ dans un groupe

Comme nous l'avons vu, lors de la phase de minimisation, nous réalisons G régressions linéaires pondérées. Lorsque la valeur t de la température est assez petite pour qu'un certain nombre de lignes de la matrice M_g ($g \in [\![1, G]\!]$) soient nulles, cette régression pondérée peut ne pas être possible car il se peut que la matrice $X^T M_g X$ soit non-inversible tout simplement car la dimension de l'espace engendré par ses colonnes est inférieur à p. Un tel cas mettrait en faillite notre algorithme. Arrêter l'algorithme quand la valeur de la température n'est pas trop petite est donc pertinent dans le sens où ça augmentera la robustesse de l'algorithme dans certains cas où le nombre de variables explicatives p est grand par rapport au nombre d'individus n et au nombre de groupes G.
De plus, on fait l'hypothèse qu'arrêter l'algorithme avant que la plus petite valeur de température possible soit atteinte permet de limiter le sur-apprentissage. En effet quand la température est trop petite, les valeurs d'un référent pour un groupe sont apprises uniquement avec les individus de ce groupe. Nous pensons qu'il est préférable que les valeurs du référent d'un groupe g ($g \in [\![1, G]\!]$) soient apprises majoritairement avec les individus du groupe en question mais que les individus des groupes dont le neurone est proche sur la carte du groupe g jouent aussi un rôle (moindre). Nous devons donc définir un critère qui arrêtera notre algorithme avant que la température devienne trop petite. Les critères que l'on a envisagés sont :

1. s'arrêter quand les individus commencent à ne plus changer de groupe. On peut justifier ce choix en disant que quand l'on continue, i.e quand la température baisse, on ne fait que de plus en plus "coller" les valeurs de chaque référent à son groupe uniquement, et cela va un peu à l'encontre du fait qu'on aimerait que les valeurs de ce référent tiennent aussi compte des individus pour les autres neurones proches sur la carte. De plus généralement, la classification finale est atteinte un certain temps avant le minimum pour la fonction de coût, ce qui nous assure une valeur de la température pas trop petite et on peut alors espérer ne pas avoir de problème avec le calcul de l'inverse des matrices. [**?**]
2. définir un ensemble de validation et s'arrêter quand l'erreur de prédiction sur l'ensemble de validation se met à augmenter significativement (*cf* [Prechelt, 1998]).
3. s'arrêter quand l'erreur sur l'ensemble d'apprentissage ne diminue plus significativement d'itération en itération. On rappelle que l'erreur sur l'ensemble d'apprentissage correspond simplement à la *Mean Squared Error* entre la valeur de la variable réponse pour chaque individu de l'ensemble d'apprentissage et la prédiction du modèle linéaire relatif au groupe auquel il a été affecté. Si on note Err_t la valeur absolue de la différence de l'erreur sur l'ensemble d'apprentissage entre l'itération t et l'itération $t-1$, on peut observer trois phases dans son évolution. La première phase, qui correspond aux valeurs de température assez grandes, et donc au moment où la carte se met simplement "en ordre", se traduit par des petites diminutions de l'erreur sur l'ensemble d'apprentissage à chaque itération, et donc des petites valeurs des Err_t. Une deuxième phase, qui correspond aux premières valeurs de température assez petites, et qui est l'équivalent pour l'algorithme des cartes de Kohonen classique aux premiers instants où la carte commence à se dilater et se répandre sur l'ensemble des données projeté dans un espace 2D, se traduit naturellement par de fortes diminutions de l'erreur sur l'ensemble d'apprentissage et donc des valeurs de Err_t qui augmentent significativement. Cette deuxième étape correspond dans notre cas aux premiers instants où les référents commencent à mieux représenter le modèle linéaire de leur groupe car les pondérations pour les individus des autres groupes commencent à être plus faibles. Une troisième phase, qui correspond aux dernières valeurs de la température (et donc aux

plus petites) et à la période de l'apprentissage où les valeurs des référents ne sont quasiment plus apprises qu'en fonction des individus appartenant au groupe relatif à chaque référent. L'erreur sur l'ensemble d'apprentissage entre deux itérations recommence à diminuer, et cela se traduit donc par des valeurs de Err_t qui diminuent aussi. L'observation de ces trois phases nous motive à nous arrêter à la fin de la deuxième phase, c'est-à-dire au moment où on pense ne pas commencer le sur-apprentissage.

Sur la figure 6 est illustré l'évolution de Err_t suivant les itérations. La première phase est colorée en rouge, la seconde phase en colorée en vert et la dernière phase est colorée en bleu. On s'arrêterait ici juste avant que la dernière phase commence.

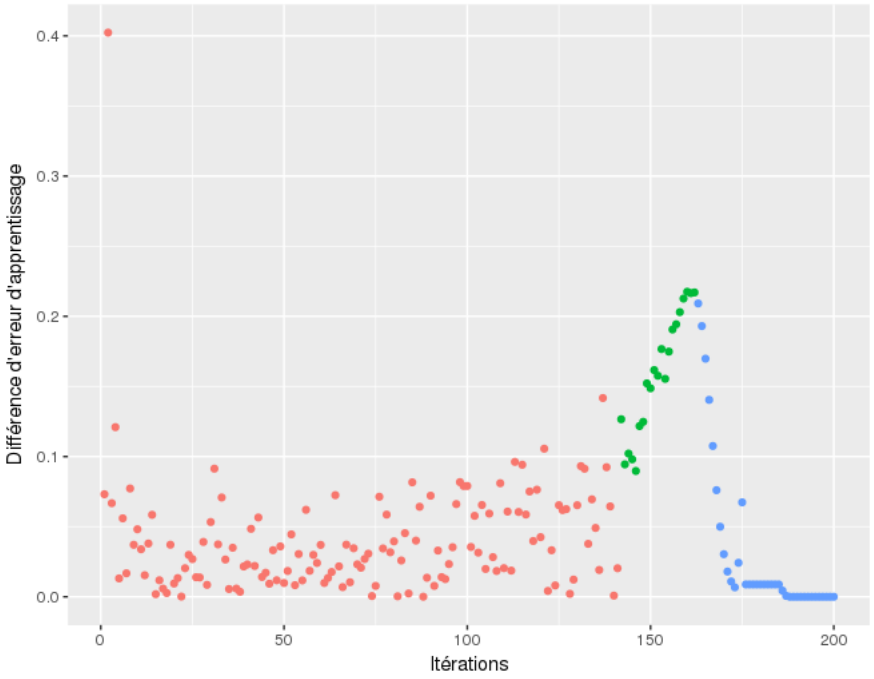

FIGURE 6 – *Early-stopping* : méthode 3
Les couleurs sont relatives aux différents phases de l'apprentissage

3.3 Algorithme SOM-PLS-Clusterwise

Cet algorithme peut être vu comme une version de l'algorithme mbpls avec des cartes de Kohonen.
On considère toujours une grille 2D dont chaque neurone correspond à un référent, qui préserve la topologie de l'espace d'entrée (ici $\mathcal{R}^{p+1}$) et permet d'en faire une représentation simplifiée.

3.3.1 Principe de l'algorithme SOM-PLS-Clusterwise

Nous définissons une fonction de coût, notée J et cherchons alors à déterminer G groupes et G modèles PLS qui minimisent sa valeur. La fonction de coût J est telle que :

$$J(A, \beta, mpls^1, ..., mpls^G) = \sum_{i=1}^{n} \sum_{g=1}^{G} K(\delta(A(z_i), g))(y_i - \hat{y_i^g})^2$$

avec :

—

$$A : \mathcal{R}^{p+1} \to 1, 2, ..., G$$
$$z \mapsto g$$

qui affecte à chaque vecteur z_i le numéro de son groupe (le neurone sur la grille).
— K une fonction de voisinage comme pour les cartes de Kohonen
— $\delta(A(z_i), g)$ le plus court chemin dans la grille entre le neurone g le neurone $A(z_i)$
— $mpls^g$ ($g \in [\![1, G]\!]$) un modèle PLS de y par X avec seulement certaines lignes de sélectionnées (détails donnés ci-après)
— $\hat{y_i^g}$ ($g \in [\![1, G]\!]$, $i \in [\![1, n]\!]$) la prédiction de la variable réponse pour l'individu i issue de $mpls^g$

La distance euclidienne entre y_i et $\hat{y}_i$ est pondérée par une fonction de la distance sur la grille 2D entre chaque neurone et le neurone qui a été affecté à z_i.
Cet algorithme reprend le principe de l'algorithme précédent, mais le modèle de régression PLS du groupe g ($g \in [\![1, G]\!]$) est appris uniquement avec les matrices y^c et X^c, qui correspondent aux matrices y et X desquelles on ne garde que les lignes correspondant aux individus du groupe g.

3.3.2 Algorithme

L'algorithme que l'on présente ici consiste à considérer chacun des n individus un par un jusqu'à ce que J ne décroisse plus.
Pour un individu z_i donné :
La première phase, dite phase d'affectation, consiste à mettre à jour la valeur de $A(z_i)$ ($i \in [\![1, n]\!]$). Pour cela, pour l'individu z_i, si il appartient au groupe $c_1 \in [\![1, G]\!]$, on commence par calculer les $G - 1$ valeurs de la fonction J en supposant que l'on ait déplacé cet individu dans tous les autres groupes à l'exception du groupe c_1. Chaque déplacement envisagé vers un groupe $c_2 \in [\![1, G]\!]$ avec $c_1 \neq c_2$ nécessite de mettre à jour les modèles PLS associés aux groupes c_1 et c_2. Le détail de ces mises à jour est détaillé ci-après. On affecte ensuite à $A(z_i)$ le numéro du groupe tel que le déplacement envisagé dans ce groupe induise la plus grande diminution de la fonction de coût J. Il est ainsi possible que la valeur de $A(z_i)$ ne change pas si aucun déplacement envisagé n'induit de diminution dans la fonction de coût J. La deuxième phase, dite phase de minimisation, met à jour les modèles PLS des groupes c_1 et c_2. Pour un groupe $c \in [\![c_1, c_2]\!]$, le modèle PLS du groupe c correspond à la régression PLS de y_t^c par X_t^c, avec y_t^c et X_t^c les matrices y et X où on ne garde que les lignes des individus relatifs au groupe c.

On comprend bien qu'on diminue la fonction de coût J après le passage de chaque individu car le déplacement de cet individu dans un autre groupe (accompagné de la mise à jour des modèles PLS) ne s'effectue que si cela induit une diminution de la fonction de coût J. Plus formellement : On note $J(A, \beta, mpls^1, ..., mpls^G)$ la valeur de la fonction de coût quand l'individu i ($i \in [\![1, n]\!]$) est dans le groupe g ($c \in [\![1, G]\!]$). On se place à une itération i de l'algorithme, c'est à dire qu'on considère l'individu i. On suppose que l'individu i est actuellement dans le groupe c_1. On connait la valeur de $J_{i,c_1}(A, \beta, mpls^1, ..., mpls^G)$. On calcule les valeurs de $J_{i,c}(A, \beta, mpls^1, ..., mpls^G)$ $\forall g \in [\![1, G]\!]$ et on déplace l'individu vers le groupe c_2 uniquement si $J_{i,c_2}(A, \beta, mpls^1, ..., mpls^G) = \text{argmin}_{g \in [\![1,G]\!]} J_{i,g}(A, \beta, mpls^1, ..., mpls^G)$. A chaque itération, la fonction de coût J diminue donc.

Le pseudo-code de cet algorithme est disponible en annexe G.

Remarque :
Comme on l'avait déjà précisé pour l'algorithme SOM-clusterwise, la valeur de la fonction de voisinage dépend d'une valeur t appelée la température. En pratique, on fait décroître cette valeur de la température au fur et à mesure de l'algorithme. Pour une valeur de température assez petite pour que $K(\delta(A(Z_i), g)) = 1$ uniquement si $A(z_i) = g$, cet algorithme est le même que l'algorithme mbpls du *package* R mbclusterwise.
Le temps d'exécution de cet algorithme est égal au temps d'exécution de l'algorithme mbpls multiplié par le nombre de valeurs de la température considéré.

Pour une valeur de la température telle que $K(\delta(A(z_i), g)) \neq 0$ pas uniquement si $A(z_i) = g$, la fonction de coût est différente de celle de l'algorithme mbpls, les minima locaux le sont donc aussi (donc les répartitions des individus dans les groupes et les modèles PLS associées aussi). La phase de minimisation est la même que pour l'algorithme mbpls. La phase d'affectation est différente. Pour l'algorithme mbpls, les valeurs de la fonction de coût calculées lors de la phase d'affectation tiennent uniquement compte pour chaque individu de la valeur d'une seule prédiction de la variable

réponse : celle relative au groupe auquel il a a été affecté. Pour notre algorithme, les valeurs de la fonction de coût calculées tiennent compte pour chaque individu de la prédiction de la variable réponse relative au groupe auquel il a a été affecté, mais aussi des prédictions relatives aux autres groupes. Alors qu'avec l'algorithme mbpls un individu peut être déplacé d'un groupe c_1 à un autre groupe c_2 sans tenir compte des valeurs des prédictions pour les groupes autres que c_1 et c_2, avec notre algorithme, si les valeurs des prédictions sont trop mauvaises pour les groupes voisins (sur la grille) de c_2, cela peut ne pas se produire.

4 Expérimentations

Dans cette partie, nous présentons les résultats des comparaisons des algorithmes que nous avons présentés jusqu'ici, dans un premier temps sur des données simulées, puis dans un second temps sur le jeu de données réelles issues du domaine de la pollution de l'air, fourni par Monsieur François Kaly, présenté dans [Kali, 2016]. La définition des critères utilisés est présentée dans l'annexe H. Dans les différents tableaux de cette partie, "SOM-Clus" désigne l'algorithme SOM-Clusterwise, "SOM-Clust-d" désigne l'algorithme SOM-Clusterwise-d, "SOM-Clust-c" désigne l'algorithme SOM-Clusterwise-c, "kreg" désigne l'algorithme kreg, "SOMPLS" désigne l'algorithme SOM-PLS-Clusterwise, "mbpls" désigne l'algorithme mbpls et "Spaeth" désigne l'algorithme Spaeth.

4.1 Simulations

Dans un premier temps, nous réalisons différentes simulations pour comparer l'algorithme kreg avec notre algorithme SOM-clusterwise avec et sans critère d'*early-stopping*.
Dans un second temps, nous comparons l'algorithme SOM-PLS-Clusterwise avec l'algorithme SOM-Clusterwise.
Dans un troisième temps, nous comparons l'algorithme SOM-PLS-Clusterwise avec l'algorithme mbpls.
Dans toute cette partie, les simulations sont réalisées de la manière suivante :
Pour un nombre n d'individus, un nombre p de variables explicatives, et un nombre G de groupes réels :
Pour les variables explicatives : pour chaque groupe, les variables sont simulées selon une loi normale de même moyenne au sein de chaque groupe, mais dont les valeurs peuvent changer entre les différents groupes selon que l'on veuille avoir des groupes réels plus ou moins séparés. L'écart-type de chaque variable explicative est pris égal à 1 pour tous les groupes. Pour les variables réponse : les coefficients de régression sont les mêmes au sein de chaque groupe, mais différent naturellement entre les groupes (nous décidons que la valeur des coefficients de régressions entre les groupes sont répartis uniformément entre -1 et 1). La valeur de la variable réponse pour chaque groupe est ensuite calculé en fonction des valeurs des variables explicatives simulées et coefficients de régression calculées pour chaque groupe ; on ajoute à la variable réponse pour chaque groupe un bruit simulé selon une loi normale de moyenne nulle et dont la valeur l'écart-type est égale à 0.1.

Pour toutes les simulations, on a séparé de manière aléatoire et stratifiée les données simulées en un ensemble d'apprentissage et un ensemble de test représentant respectivement 80% et 20% des données. La prédiction de la valeur de la variable réponse pour un nouvel individu correspond à la prédiction issue du modèle du groupe auquel il a été affecté, avec la méthode des plus k-proches voisins avec la distance euclidienne. La valeur de k choisie est celle qui minimise l'erreur de classement.

Dans les différents tableaux de résultats, err_{train} correspond à l'erreur (*Mean Squared Error*) sur l'ensemble d'apprentissage, err_{test} correspond à l'erreur (*Mean Squared Error*) sur l'ensemble de test, rand correspond à la valeur de l'indice rand, CH correspond à la valeur de l'indice de Calinsky-Harabasz.

Dans les différents tableaux de simulation, l'attribut "Séparation" est un indicateur du fait que les différences de moyenne des variables explicatives entre les groupes d'individus sont plus ou moins grandes. Par exemple, pour le cas "bien séparés", si on a $G = 3$: dans le premier groupe, la moyenne de chaque variable explicative est égale à 0, dans le second groupe, la moyenne de chaque variable explicative est égale à 4, dans le troisième groupe la moyenne de chaque variable explicative est à égale à 8. De même, pour le cas "moyennement séparés", si on a $G = 3$: dans le premier groupe, la moyenne de chaque variable explicative est égale à 0, dans le second groupe, la moyenne de chaque variable explicative est égale à 2, dans le troisième groupe la moyenne de chaque variable explicative est à égale à 4. L'attribut "Répartition" est un indicateur du fait que les individus sont répartis équitablement dans les G groupes ou non. Pour le cas "Egale", chaque groupe comporte $\frac{100}{G}$ % des individus. Pour le cas "Dispr", le pourcentage de répartition des individus au sein de chaque groupe est précisé. L'attribut "G_{algo}" représente le nombre de groupes passé en paramètres aux deux algorithmes.

4.1.1 Comparaisons entre SOM-Clusterwise et kreg

Nous réalisons différentes simulations pour comparer l'algorithme kreg avec notre algorithme SOM-clusterwise avec et sans critère d'*early-stopping*. Nous voulons d'une part montrer que notre algorithme avec un critère d'*early-stopping* est robuste aux cas où un groupe a de grandes chances de se retrouver avec trop peu d'individus, en comparaison de notre algorithme sans critère d'*early-stopping* et de l'algorihtme kreg. D'autre part, nous voulons montrer que notre algorithme avec critère d'*early-stopping* présente de meilleures performances avec certains critères que pour notre algorithme sans *early-stopping* et l'algorithme kreg. Les critères d'*early-stopping* qu'on a retenus sont les critères 1 et 3 définis dans la partie 3.2.4, et on considère donc les algorithmes SOM-Clusterwise-d et SOM-Clusterwise-c.

Les différentes simulations effectuées selon cette méthode sont détaillées dans le tableau 1. Les cas 5,6 et 7 sont ici destinés à mettre en défaut notre algorithme sans *early-stopping* et l'algorithme des k-moyennes (peu d'individus ou beaucoup de variables explicatives), les cas 1,2,3 et 4 testent les limites de mise en défaut de notre algorithme sans *early-stopping* et l'algorithme des k-moyennes (carte plus ou moins grande), tandis que les cas 8 et 9 font en sorte qu'ils ne soient pas en défaut, pour pouvoir les comparer selon les critères qu'on a définis.

Cas numéro	n	p	G	Séparation	G_{algo}
1	540	2	4	bien séparés	9
2	540	2	4	moyennement séparés	9
3	540	2	4	bien séparés	16
4	540	2	4	moyennement séparés	16
5	180	2	4	bien séparés	16
6	540	20	4	bien séparés	16
7	540	20	4	moyennement séparés	16
8	900	2	9	bien séparés	9
9	900	2	9	moyennement séparés	9

TABLE 1 – Cas simulés

Dans le tableau 2 nous présentons les résultats obtenus avec les différents algorithmes. Pour chaque cas, nous avons réalisé 5 simulations différentes. Pour chaque simulation, les différents algorithmes ont été initialisés avec les mêmes groupes initiaux, les mêmes valeurs des référents et la même température initiale. Les valeurs présentées ici sont moyennées. Pour un cas simulé, si un algorithme rencontrait un problème (i.e une régression linéaire qui n'est pas possible au sein d'un groupe), on recommençait l'exécution de tous les algorithmes en changeant les groupes initiaux et les référents. Si malgré avoir recommencé plusieurs fois, un algorithme rencontrait toujours un problème, on le signifie par un X. Nous avons observé que pour un cas simulé, un algorithme rencontrant un problème le rencontrait pour les 5 simulations.

Les résultats présentés dans le tableau 2 sont assez positifs. Premièrement nous constatons de l'utilité du *early-stopping* dans notre algorithme. Cela empêche, dans certains cas (cas numéro 5, 6 et 7 : peu d'individus par rapport au nombre de groupes G_{algo} ou p grand par rapport au nombre d'individus n) à notre algorithme d'avoir des problèmes d'inversion de matrice liés à des valeurs de température trop petites. Deuxièmement, nous constatons que notre algorithme avec *early-stopping* est bien plus efficace que le simple algorithme des k-moyennes, quand ce dernier n'échoue pas (cas numéro 8 et 9 : n grand par rapport à p ; cas numéro 2 : n relativement grand par rapport à p). En termes d'erreur sur l'ensemble de test, notre algorithme avec *early-stopping* donne de meilleurs résultats que notre algorithme sans *early-stopping*, ce qui était un des buts recherché de l'*early-stopping* (en plus de le rendre plus robuste à certains cas où le nombre d'individus dans un groupe serait trop petit). Généralement, notre algorithme avec l'*early-stopping* qui dépend de l'erreur sur l'ensemble d'apprentissage (SOM-clust-d dans le tableau)donne de meilleurs résultats en généralisation que notre algorithme avec l'*early-stopping* qui dépend de la classification (SOM-clust-c dans le tableau). Concernant l'indice CH, on constate pour le cas 1,2,3 et 4 que quand notre algorithme sans *early-stopping* donne de meilleurs résulats que notre algorihtme avec *early-stopping*. Cependant, dans les cas 8 et 9, c'est l'inverse qui se produit. Dans les cas où seuls nos algorithmes avec

Cas numéro	algorithme	rand	CH	err_{train}	err_{test}
1	SOM-clus	0.9	184	0.04	0.12
	kreg	X	X	X	X
	SOM-clust-d	0.9	37	0.07	0.13
	SOM-clust-c	0.9	184	0.07	0 .12
2	SOM-clus	0.88	137	0.08	4.25
	kreg	0.85	67	0.1	5.03
	SOM-clust-d	0.84	58	0.09	**3.31**
	SOM-clust-c	0.84	137	0.09	3.52
3	SOM-clus	0.81	55	0.03	0.34
	kreg	X	X	X	X
	SOM-clust-d	0.8	22	0.05	0.28
	SOM-clust-c	0.8	55	0.05	0.26
4	SOM-clus	0.81	17	0.03	3.89
	kreg	X	X	X	X
	SOM-clust-d	0.8	22	0.05	**3.36**
	SOM-clust-c	0.8	17	0.05	4.00
5	SOM-clus	X	X	X	X
	kreg	X	X	X	X
	SOM-clust-d	0.8	5.23	0.01	0.25
	SOM-clust-c	0.8	5.32	0.01	0.26
6	SOM-clus	X	X	X	X
	kreg	X	X	X	X
	SOM-clust-d	0.8	25	1.9	1750
	SOM-clust-c	0.8	16	1.9	2000
7	SOM-clus	X	X	X	X
	kreg	X	X	X	X
	SOM-clust-d	0.75	6.58	1.9	1686
	SOM-clust-c	0.75	7.37	1.9	1887
8	SOM-clus	0.89	22	0.63	1.11
	kreg	0.2	8	0.67	1.79
	SOM-clust-d	0.89	31	0.64	**1.06**
	SOM-clust-c	0.85	22	0.54	1.09
9	SOM-clus	0.89	23	0.3	3.99
	kreg	0.2	17	0.35	4.29
	SOM-clust-d	0.89	27	0.35	**3.79**
	SOM-clust-c	0.85	23	0.33	5.36

TABLE 2 – Comparaisons des RMSE en généralisation (SOM-Clusterwise, kreg)
Pour chaque cas, 5 simulations différentes, même classification et référents initiaux pour chaque algorithme, X si échec sur les 5 simulations

early-stopping n'échouent pas (cas numéro 5,6 et 7), l'algorithme SOM-clust-d donne de meilleurs résultats que l'algorithme SOM-clust-c. Comme nous l'avons déjà expliqué, pour la plus petite valeur de température possible, notre algorithme est équivalent à l'algorithme kreg. Pour les cas 1 et 3 où notre algorithme n'échoue pas alors que l'algorithme kreg échoue, cela s'explique simplement par le fait que la classification des individus et les prototypes quand on atteint la température minimale pour notre algorithme est différente de la classification des individus et des prototypes pour l'algorithme kreg. Dans les cas 2, 8 et 9 où notre algorithme donne de meilleurs résultats en généralisation que l'algorithme kreg, la même explication est valable. Généralement, l'algorithme des kreg ne détermine que 4 ou 5 groupes, et la grande majorité des individus sont répartis sur deux groupes. Pour $G = 4$ et $G_{algo} = 9$ ou $G_{algo} = 16$, notre algorithme détermine généralement 9 ou 10 groupes, avec la majorité des individus dans 4 groupes. Pour $G = 9$, notre algorithme détermine toujours 9 ou 10 groupes mais les individus sont mieux répartis au sein de ces groupes. Cela est encourageant dans la capacité de notre algorithme à retrouver le bon nombre de groupes réels.
Ces simulations nous permettent de légitimement supposer que notre algorithme est pertinent par

rapport au simple algorithme kreg, et que le critère d'*early-stopping* est aussi pertinent à considérer.

Remarque :
Nous avons aussi implémenté l'algorithme Spaeth, tel que présenté dans [Späth, 1979], c'est-à-dire en version stochastique et non en version *batch*. Cette version ne peut pas "échouer" car dans son implémentation on ne peut pas déplacer un individu d'un groupe à un autre si cela induit des problèmes d'inversion de matrices. Afin de ne pas surcharger le rapport, et puisque notre but était de montrer la supériorité de l'algorithme SOM-Clusterwise avec *early-stopping* à l'algorithme kreg (une version *batch* de l'algorithme de [Späth, 1979]), afin de montrer la pertinence de l'introduction des cartes topologiques, nous ne présentons pas les résultats obtenus. Nous constatons que pour les simulations numéro 1 à 9, les résultats en termes de classification et de performance en généralisation restent moins bons que pour l'algorithme SOM-Clusterwise avec ou sans (quand cela est possible) critère d'*early-stopping*.

Nous avons développé une première version encourageante d'un algorithme permettant d'effectuer de la régression clusterwise. Nous constatons cependant que les performances obtenues ici en généralisation sont surpassées par l'algorithme mbpls. Nous constatons néanmoins que le temps d'exécution de notre algorithme SOM-Clusterwise est bien moindre que pour l'algorithme mbpls. Pour des jeux de données de bien plus grande taille, notre algorithme SOM-Clusterwise peut être alors un bon compromis entre temps d'exécution et performances.

4.1.2 Comparaisons entre SOM-Clusterwise et SOM-PLS-Clusterwise

Comme nous l'avons déjà précisé, pour une valeur de température assez petite pour que $K(\delta(A(z_i), g)) = 1$ uniquement si $A(z_i) = g$, l'algorithme SOM-PLS-Clusterwise est le même que l'algorithme mbpls du *package R* mbclusterwise. Nous décidons alors d'arrêter la température à une valeur telle que $K(\delta(A(z_i), g)) \neq 0$ pas uniquement si $A(z_i) = g$. Comme nous l'avons déjà expliqué, la fonction de coût est alors différente de celle de l'algorithme mbpls, notamment les minima locaux sont différents et ainsi les différents classifications des individus dans les groupes et les modèles PLS au sein de chacun des groupes associés à ces minima locaux sont aussi différents. Afin que le temps d'exécution ne soit pas trop grand, pour une valeur de la température donnée, nous ne considérons pas l'ensemble des individus autant de fois que nécessaire jusqu'à ce que J ne décroisse plus, mais uniquement une seule fois.
Nous reprenons les simulations effectuées lorsque nous comparions l'algorithme SOM-Clusterwise à ses versions avec *early-stopping* et à l'algorithme kreg. Pour chaque simulation, nous exécutons notre algorithme SOM-PLS-Clusterwise avec 2 initialisations différentes des groupes d'individus, et considérons les résultats de l'exécution ayant aboutie à la plus petite valeur de la fonction de coût. Nous rappelons les simulations qui avaient été effectuées et présentons les résultats obtenus avec notre algorithme SOM-PLS-Clusterwise pour ces simulations, dans les deux tableaux suivants :

Cas numéro	n	p	G	Séparation	G_{algo}
1	540	2	4	bien séparés	9
2	540	2	4	moyennement séparés	9
3	540	2	4	bien séparés	16
4	540	2	4	moyennement séparés	16
5	180	2	4	bien séparés	16
6	540	20	4	bien séparés	16
7	540	20	4	moyennement séparés	16
8	900	2	9	bien séparés	9
9	900	2	9	moyennement séparés	9

TABLE 3 – Cas simulés

Nous constatons que les résultats obtenus, notamment l'erreur en généralisation, sont bien meilleurs que pour l'algorithme SOM-Clusterwise (*cf.* tableau 2). Notamment, pour les simulations 6,7, où nous avions eu des performances bien moindres que pour les autres simulations avec l'algorithme SOM-Clusterwise, nous obtenons maintenant des performances similaires aux autres

Cas numéro	rand	CH	err_{train}	err_{test}
1	0.89	167	0.006	0.02
2	0.87	112	0.006	2.96
3	0.72	32	0.01	0.25
4	0.83	54	0.006	3.31
5	0.78	13	0.03	0.05
6	0.88	70	0.12	1.3
7	0.89	20	0.09	2.1
8	0.89	51	0.04	1.98
9	0.89	23	0.06	3.15

TABLE 4 – RMSE sur l'ensemble de test (SOM-PLS-Clusterwise)
Pour chaque cas, 2 simulations différentes, même classification initiale pour chaque algorithme

simulations.

4.1.3 Comparaisons entre mbpls et SOM-PLS-Clusterwise

Nous comparons maintenant l'algorithme SOM-PLS-Clusterwise à l'algorithme mbpls, en faisant en sorte qu'aucun groupe ne puisse ne plus contenir d'individu, car l'algorithme mbpls ne permet pas à un groupe de ne pas contenir d'individu. De même, le nombre de groupes passé en paramètres à notre algorithme et à l'algorithme mbpls est le même. Pour une simulation, les deux algorithmes sont exécutés avec plusieurs initialisations des groupes différents. On a en effet vu que chacun des algorithmes présentés précédemment atteignent un minimum local de la fonction de coût J et que le minimum local atteint dépend de l'initialisation des groupes choisie. Les résultats présentés pour chaque simulation sont ceux relatifs à l'initialisation des groupes qui a conduit à la plus petite valeur de la fonction de coût, pour notre algorithme et pour l'algorithme mbpls. Ici, l'algorithme mbpls a été exécuté avec 20 initialisations différentes et notre algorithme avec 2 initialisations seulement, en raison de son temps d'exécution. Le nombre de composantes PLS est le même pour les deux algorithmes, choisi arbitrairement égal à 1.

Les simulations du tableau 5 permettent de comparer notre algorithme pour un nombre de variables explicatives relativement élevé par rapport au nombre d'individus, et un nombre de groupes réels ($G = 4$) inférieur au nombre de groupes passé en paramètres aux deux algorithmes ($G_{algo} = 9$).
Les simulations du tableau 6 correspondent à certaines simulations du tableau 5 lorsque le nombre de groupes réels ($G = 9$) et le nombre de groupes passé en paramètres aux algorithmes est le même ($G_{algo} = 9$).
Les simulations du tableau 7 correspondent aussi à certaines simulations du tableau 5 lorsque le nombre de groupes réels ($G = 4$) est très inférieur au nombre de groupes passé en paramètres aux algorithmes est le même ($G_{algo} = 16$).
Les simulations du tableau 8 correspondent aux simulations du tableau 6 avec un nombre réduit de variables explicatives.
Dans aucun des tableaux 6, 7 ou 8, l'attribut "Répartition" est différent de "Egale". Nous avons en effet attesté du fait que les constatations tirées des résultats pour des simulations où l'attribut "Répartition" n'était pas "Egale" étaient les mêmes que pour le simulations où l'attribut "Répartition" était "Egale".

Dans les tableaux 9, 10, 11 et 12, nous présentons les résultats obtenus avec notre algorithme SOMPLS-clusterwise et l'algorithme mbpls.

Les résultats pour le premier groupe et le troisième groupe de simulations (cf tableaux 9 et 11) semblent indiquer que notre algorithme SOM-PLS-Clusterwise dépasse en capacité de prédiction en généralisation l'algorithme mbpls, lorsque le nombre de groupes réels est inférieur au nombre de groupes passé en paramètres de l'algorithme. Nous ne pouvons cependant pas légitimement affirmer que les résultats sont meilleurs pour $G_{algo} = 9$ ou $G_{algo} = 16$, pour aucun des deux algorithmes. Les résultats du deuxième groupe (*cf.* tableau 10 semblent montrer que pour $G = G_{algo}$,

Cas numéro	n	p	G	Séparation	Répartition	G_algo
1	540	10	4	bien séparés	Egale	9
2	540	10	4	moyennement séparés	Egale	9
3	540	10	4	bien séparés	(35,35,15,15)	9
4	540	10	4	moyennement séparés	(35,35,15,15)	9
5	180	10	4	bien séparés	Egale	9
6	180	10	4	moyennement séparés	Egale	9
7	180	10	4	bien séparés	(35,35,15,15)	9
8	180	10	4	moyennement séparés	(35,35,15,15)	9

TABLE 5 – Cas simulés : $\mathbf{G_{algo}} = 9 > \boldsymbol{G} = 4$, $p = 10$

Cas numéro	n	p	G	Séparation	Répartition	G_algo
9	540	10	9	bien séparés	Egale	9
10	540	10	9	moyennement séparés	Egale	9
11	180	10	9	bien séparés	Egale	9
12	180	10	9	moyennement séparés	Egale	9

TABLE 6 – Cas simulés : $\mathbf{G_{algo}} = 9 = \boldsymbol{G}$, $p = 10$

l'algorithme mbpls surpasse notre algorithme SOM-PLS-CLusterwise. Concernant les résultats du quatrième groupe de simulations ,notamment pour les simulation numéro 19 et 20, où l'algorithme mbpls est meilleur que notre algorithme de très peu, nous pensons que cela est dû au nombre trop petit d'individu par rapport au nombre de groupes et de variables explicatives.

Les résultats pour le premier groupe et le troisième groupe de simulations (cf tableaux 9 et 11) semblent indiquer que notre algorithme SOM-PLS-Clusterwise dépasse en capacité de prédiction en généralisation l'algorithme mbpls, lorsque le nombre de groupes réels est inférieur au nombre de groupes passé en paramètres de l'algorithme. Nous ne pouvons cependant pas légitimement affirmer que les résultats sont meilleurs pour $G_{algo} = 9$ ou $G_{algo} = 16$, pour aucun des deux algorithmes. Les résultats du deuxième groupe (*cf.* tableau 10 semblent montrer que pour $G = G_{algo}$, l'algorithme mbpls surpasse notre algorithme SOM-PLS-CLusterwise. Concernant les résultats du quatrième groupe de simulations ,notamment pour les simulation numéro 19 et 20, où l'algorithme mbpls est meilleur que notre algorithme de très peu, nous pensons que cela est dû au nombre trop petit d'individu par rapport au nombre de groupes et de variables explicatives.

Remarque :
Pour chacune des simulations, pour chacun des deux algorithmes (SOM-PLS-Clusterwise et mbpls), nous avons effectué un certain nombre d'exécutions relatives à des initialisations des groupes d'individus différentes, et nous avons gardé les résultats (indices rand et Calinsky-Harabasz, erreurs en apprentissage et en test) des exécutions qui avaient abouties à la plus petite valeur de la fonction de coût. Nous avions alors dans certains cas pu conclure à la supériorité de notre algorithme SOM-PLS-Clusterwise par rapport à l'algorithme mbpls, notamment en regardant l'erreur en généralisation.
Si l'on considère pour les deux algorithmes les résultats de toutes les exécutions, nous constatons que les meilleurs résultats (notamment pour l'erreur en généralisation) ne sont pas forcément obtenus pour les exécutions qui aboutissent à la plus petite valeur de la fonction de coût. De plus, nous constatons que les meilleurs résultats pour l'algorithme mbpls surpassent des fois les meilleurs résultats pour notre algorithme SOM-PLS-Clusterwise, notamment pour l'erreur en généralisation. Cependant, pour comparer les deux algorithmes nous ne pouvions pas logiquement choisir de comparer les résultats des exécutions qui aboutissaient à la plus petite erreur en généralisation : nous avions besoin d'un critère interne à l'algorithme.

Remarque :
Afin d'améliorer la performance en prédiction de notre algorithme, nous avons construit lors de son apprentissage un ensemble de validation. A chaque fois qu'un individu était déplacé d'un groupe

Cas numéro	n	p	G	Séparation	Répartition	G_algo
13	540	10	4	bien séparés	Egale	16
14	540	10	4	moyennement séparés	Egale	16
15	180	10	4	bien séparés	Egale	16
16	180	10	4	moyennement séparés	Egale	16

TABLE 7 – Cas simulés : $\mathbf{G_{algo}} = 16 > \boldsymbol{G} = 4$, $p = 10$

Cas numéro	n	p	G	Séparation	Répartition	G_algo
17	540	2	9	bien séparés	Egale	9
18	540	2	9	moyennement séparés	Egale	9
19	180	2	9	bien séparés	Egale	9
20	180	2	9	moyennement séparés	Egale	9

TABLE 8 – Cas simulés : $\mathbf{G_{algo}} = 9 = \boldsymbol{G}$, $p = 2$

Cas numéro	algorithme	rand	CH	err_{train}	err_{test}
1	SOMPLS	0.88	138	0.05	0.5
	mbpls	0.89	231	4.06	2.59
2	SOMPLS	0.83	58	0.11	11.3
	mbpls	0.9	136	5.6	15.86
3	SOMPLS	0.75	60	0.1	2.07
	mbpls	0.84	230	6.69	5.59
4	SOMPLS	0.87	205	0.06	0.35
	mbpls	0.79	131	5.26	4.43
5	SOMPLS	0.82	14	0.08	1.99
	mbpls	0.78	9	8.51	10.96
6	SOMPLS	0.89	16	0.04	1.07
	mbpls	0.82	10.93	0.87	2.44
7	SOMPLS	0.73	28.8	0.08	1.98
	mbpls	0.71	14	3.5	9.67
8	SOMPLS	0.7	15	0.04	3.56
	mbpls	0.75	23	3.31	4.97

TABLE 9 – Comparaison des RMSE en généralisation (SOM-PLS-Clusterwise, mbpls) : $\mathbf{G_{algo}} = 9 > \boldsymbol{G} = 4$
$p = 10$ *; Pour chaque cas, 2 simulations différentes, même classification initiale pour chaque algorithme*

Cas numéro	algorithme	rand	CH	err_{train}	err_{test}
9	SOMPLS	0.94	40	0.62	2.49
	mbpls	1	5849	0.71	1.72
10	SOMPLS	0.92	76	0.91	2.48
	mbpls	0.97	170	0.04	0.64
11	SOMPLS	0.92	33	1.6	1.56
	mbpls	1	1897	0.65	1.49
12	SOMPLS	0.91	12	11.8	3.67
	mbpls	0.86	5	35	36

TABLE 10 – Comparaison des RMSE en généralisation (SOM-PLS-Clusterwise, mbpls) : $\mathbf{G_{algo}} = 9 = \boldsymbol{G}$
$p = 10$ *; Pour chaque cas, 2 simulations différentes, même classification initiale pour chaque algorithme*

Cas numéro	algorithme	rand	CH	err_{train}	err_{test}
13	SOMPLS	0.82	49	0.04	1.31
	mbpls	0.84	34	3.33	5.05
14	SOMPLS	0.81	28	0.03	3.61
	mbpls	0.83	41	1.89	5.15
15	SOMPLS	0.81	6	0.01	3.88
	mbpls	0.78	8	2.35	7.34
16	SOMPLS	0.79	6	0.01	1.24
	mbpls	0.79	9	0.04	1.24

TABLE 11 – Comparaison des RMSE en généralisation (SOM-PLS-Clusterwise, mbpls) : $\mathbf{G_{algo}} = 16 > \boldsymbol{G} = 4$
$p = 10$ *; Pour chaque cas, 2 simulations différentes, même classification initiale pour chaque algorithme*

Cas numéro	algorithme	rand	CH	err_{train}	err_{test}
17	SOMPLS	0.89	159	0.50	0.59
	mbpls	0.97	3431	0.2	0.2
18	SOMPLS	0.88	78	0.16	3.19
	mbpls	0.97	304	0.02	2.93
19	SOMPLS	0.92	24	0.06	0.16
	mbpls	0.93	1129	0.08	0.13
20	SOMPLS	0.85	5	0.17	5.25
	mbpls	0.86	8.7	1.17	5.08

TABLE 12 – Comparaison des RMSE en généralisation (SOM-PLS-Clusterwise, mbpls) : $\mathbf{G_{algo}} = 9 = \boldsymbol{G}$ $p = 2$
$p = 10$ *; Pour chaque cas, 2 simulations différentes, même classification initiale pour chaque algorithme*

à un autre, nous avons calculé l'erreur en prédiction sur cet ensemble de validation. Nous avons tout d'abord constaté que l'erreur en prédiction sur l'ensemble de validation ne suivait pas de schéma particulier. Il n'est donc malheureusement pas possible d'arrêter l'algorithme en se basant sur un critère dépendant de l'erreur sur l'ensemble de validation. Nous choisissons donc de garder les modèles PLS et la classification des individus relatifs à la plus petite erreur sur l'ensemble de validation. On constate premièrement que l'erreur sur l'ensemble de test diminue. Deuxièmement, nous constatons deux choses :

1. soit la plus petite erreur sur l'ensemble de validation est obtenue pour une valeur de la température assez élevée
2. soit la plus petite erreur sur l'ensemble de validation est obtenue pour la valeur de la température la plus basse envisagée

Pour les simulations où on avait trouvé que l'algorithme mbpls était meilleur que notre algorithme SOM-PLS-Clusterwise (en partant de 20 initialisations différentes des individus dans les groupes et en gardant les résultats de celle qui aboutissait à la plus petite valeur de la fonction de coût), le fait de considérer un ensemble de validation ne permet pas à notre algorithme de faire mieux que l'algorithme mbpls.
Comme nous l'avons déjà précisé, pour chaque valeur de température, on ne considère qu'une seule fois l'ensemble des individus. On ne considère pas autant de fois que nécessaire l'ensemble des individus pour qu'à une température donnée la fonction de coût ne décroisse plus. Toujours afin d'améliorer la performance en prédiction de notre algorithme, nous essayons de constater des effets de considérer plus d'une unique fois l'ensemble des individus pour une valeur donnée de la température. Nous avons constaté que cela n'améliorait pas forcément l'erreur en généralisation.

4.1.4 Conclusion

Dans un premier temps nous avons d'abord montré la robustesse des algorithmes SOM-Clusterwise-c et SOM-Clusterwise-d, qui correspondent à l'algorithme SOM-Clusterwise avec critère d'*early-stopping*, à certains cas où un groupe présente moins d'individus que de variables, par rapport aux algorithmes SOM-Clusterwise et kreg. Nous avons aussi montré les meilleures performances en généralisation des algorithmes SOM-Clusterwise-c et SOM-Clusterwise-d par rapport aux algorithmes SOM-Clusterwise et kreg.
Dans un second temps nous avons montré les meilleures performances en généralisation de l'algorithme SOM-PLS-Clusterwise par rapport à l'algorithme SOM-Clusterwise.
Dans un troisième temps, les simulations laissent supposer que l'algorithme SOM-PLS-Clusterwise est plus performant en généralisation que l'algorithme mbpls si $G_{algo} > G$. Ainsi, supposons qu'on mène une étude sur un jeu de données réel dans le cadre de la régression clusterwise où le nombre de groupes réel est inconnu mais qu'on suppose quand même qu'il est compris entre deux valeurs G_{min} et G_{max}. Si la taille du jeu de données réel est trop important et qu'on ne peut envisager qu'une exécution d'un algorithme, on exécutera donc avec profit l'algorithme SOM-PLS-Clusterwise avec un nombre de groupe passé en paramètre G_{algo} tel que G_{algo} est légèrement supérieur à G_{max}.

4.2 Données réelles

Dans cette partie nous proposons principalement de comparer nos algorithmes SOM-Clusterwise (*cf.* Partie 3.2)et SOM-PLS-Clusterwise (*cf* Partie 3.2) à l'algorithme mbpls et à certains résultats présentés dans [Fall, 2017], sur le jeu de données réelles issues de données de capteurs oour l'étude de la pollution de l'air fourni par Monsieur François Kali. Nous nous appuierons aussi sur certains résultats présentés dans un mémoire ([Fall, 2017]) pour le choix du nombre de groupes. Dans un premier temps, nous comparons l'algorithme SOM-Clusterwise à l'algorithme Spaeth.

4.2.1 Présentation du jeu de données

Le jeu de données réelles sur lequel nous avons travaillé correspond à des relevés de différentes grandeurs physiques effectués dans une station de Banizoumbou, ville du Niger. Nous disposons de 2960 relevés, effectués du 1er janvier 2006 au 31 décembre 2010, durant les mois de janvier à mai et d'octobre à décembre, quasiment chaque jour, généralement à 6,9, 12 et 15 heures. Chaque relevé nous donne la valeur de 953 variables, qui peuvent être décomposées en deux groupes. Le premier groupe concerne les variables "locales", au nombre de 17 et le second les variables de "grande échelle", au nombre de 936.

Les variables "locales" sont :

- le jour du relevé (entre 0 et 31)
- le mois du relevé (entre 1 et 5 et entre 10 et 12)
- l'année du relevé (entre 2006 et 2010)
- l'heure du relevé (6,9,12 ou 15)
- la vitesse du vent (en $m.s^{-1}$)
- la direction du vent (de 0 à 360 degrés)
- la température (en degrés Celsius)
- l'humidité (en pourcentage)
- la pollution en particules PM10 (concentration massique en $\mu g.m^{-3}$)
- la pression (en Bar)
- quatre mesures de AOT (*Aerosol Optical Thickness* : Epaisseur Optique en Aérosols) (sans unité). L'AOT est un indicateur de l'"opacité" de l'air liée à la présence de particules fines. Il est d'autant plus élevé qu'il y a de particules fines dans l'air. Les quatre mesures correspondent à quatre longueurs d'onde différentes.

— trois mesures du coefficient d'Angström (sans unité). Il est un indicateur de la dépendance à la longueur d'onde de certaines propriétés optiques (sa valeur varie en fonction de la longueur d"onde). Notamment, plus les particules sont petites, plus il est élevé.

Les variables de "grande échelle" correspondent à 7 variables (température, vent zonal, vent méridien, géopotentiel, humidité spécifique, vitesse verticale et hauteur de la couche limite). Pour chacune des 6 premières variables de grande échelle on a 153 valeurs qui correspondent à des mesures effectuées à 9 niveaux de pression différents. Pour la dernière variable de grande échelle on a effectué les 17 mesures à seulement un seul niveau de pression.

Nous décidons de ne pas considérer les variables relatives au jour, au mois, à l'année et à l'heure des relevés. On ne considère alors plus que 13 variables locales.
Notre objectif est de prédire la pollution en particules PM10 en fonction des autres variables, tout en arrivant à classer les individus en un certain nombre de groupes.
La concentration en pollution en particules PM10, considérées comme dangereuses pour la santé, est soumise à des standards dans le monde entier. En 2006, le taux limite annuel légal dans l'Union Européenne était de 50 $\mu g.m^{-3}$. La région où se trouve Banizoumbou est une des régions les plus polluées en particules PM10 du monde. Ainsi, les moyennes annuelles observées sont ici pour les années 2006, 2007, 2008, 2009 et 2010 respectivement de 193, 232, 182, 157 et 240 $\mu g.m^{-3}$ (arrondies à l'unité). L'appareil utilisé pour mesurer la concentration en PM10 n'est pas connu. On peut cependant considérer, en se basant sur les caractéristiques d'appareils destinées au même usage à la vente, que la précision est de l'ordre de 3%.

Dans le cadre de ce rapport nous ne nous intéressons qu'aux variables "locales" autre que la pollution en particules PM10, au nombre de 12.

4.2.2 Notations

On note n (ici $n = 2960$)le nombre de relevés effectués et p (ici $p = 12$)le nombre de variables locales considérées comme explicatives.
On appellera variable réponse la variable relative à la concentration en PM10.
On note y le vecteur de dimension $n \times 1$ qui contient les valeurs de la concentration en PM10 pour chaque relevé. On note X la matrice de dimension $ntimesp$ qui contient les valeurs des p variables explicatives pour chacun des n relevés. On note y_i ($\forall i \in [\![1, n]\!]$) la valeur de la variable réponse pour le relevé i, X_i ($\forall i \in [\![1, n]\!]$) le vecteur des valeurs des p variables pour le relevé i, X^j ($\forall j \in [\![1, p]\!]$) la colonne j de la matrice X.

4.2.3 Méthodes linéaires classiques

Nous avons tout d'abord appris des modèles de régression multiples classiques, en considérant qu'il n'y avait qu'un seul groupe de relevés. Les coefficients de régression obtenus sont donc les mêmes pour tous les relevés. Les différents résultats obtenus nous permettront de pouvoir émettre des hypothèses quant à la pertinence des modèles que l'on apprendra dans les parties suivantes, où une classification des individus en un certain nombre de groupes sera effectuée. On rappelle que l'écart-type de la variable réponse y est égal à 288 $\mu g.m^{-3}$.
On réalise une *10-cross validation* afin d'estimer notamment la *Root Mean Square Error* et la *Mean Average Percentage Error*. Les différents résultats pour la régression linéaire sont présentés dans le tableau 13. On constate notamment que la *Root Mean Squared Error* est plus grande que l'écart-type de la variable réponse y (égal à 288.85), ce qui participe à montrer que ce modèle est très peu satisfaisant.
Nous avions cependant constaté avant d'effectuer la régression linéaire que certaines variables explicatives sont très linéairement corrélées entre elles, d'une part en calculant le coefficient de corrélation linéaire classique, et d'autre part en calculant le coefficient *Variance Inflation Factor* (la valeur du *Variance Inflation Factor* pour une variable correspond à $\frac{1}{1-R^2}$ où R^2 est le coefficient de détermination dans la régression linéaire de cette variable par toutes les autres variables). Étant donné que le fait d'avoir des variables linéairement corrélées entre elles peut poser des problèmes dans le calcul des coefficients de régression, nous avons décidé d'apprendre un modèle linéaire en ne

conservant que certaines variables de sorte à ne plus avoir de coefficient de corrélation linéaire significatif entre ces variables, ni de valeur du *Variance Inflation Factor* trop élevée. Nous considérons ainsi plus que 7 variables explicatives, sachant que les variables que l'on a enlevées correspondent à trois variables relatives à la mesure de l'AOT et deux variables relatives au coefficient d'Angström. Nous aurions aussi pu réaliser une régression de type Lasso. Nous trouvons avec la *10-cross validation* des valeurs de *Root Mean Squared Error* un peu moins élevées que quand l'on considérait toutes les variables explicatives. Les différents résultats pour la régression linéaire sont présentés dans le tableau 13.

Régression linéaire	12 variables	7 variables
Coefficient de détermination R^2 (10-cross validation)	0.66	0.62
RMSE (10-cross validation, en $\mu g.m^{-3}$)	440	343

TABLE 13 – Données réelles : Performances en généralisation pour la régression linéaire classique (10-cross validation)
Résultats pour la régression linéaire classique avec toutes les variables explicatives et seulement les variables explicatives non-corrélées entre elles

Parallèlement à cela, nous avons aussi appris un modèle de régression PLS1 (*cf.* Partie 2.1). La valeur de RMSE trouvée par *10-cross validation* est minimale lorsque l'on considère 8 composantes PLS et est de l'ordre de l'écart-type de la variable réponse. Le modèle appris est donc encore peu satisfaisant.

4.2.4 Résultats existants

Nous disposons de premiers résultats déjà obtenus sur ce jeu de données réelles dans [Fall, 2017]. Une des parties intéressantes de ce mémoire consiste notamment dans le travail réalisé pour déterminer un certain nombre de groupes de relevés dans ce jeu de données. Une fois ces groupes fixés, l'auteur réalise une régression linéaire au sein de chacun de ces groupes.
Pour déterminer le nombre de groupes d'individus, l'auteur utilise l'algorithme des cartes de Kohonen (*cf.* Partie 2.3)en ne considérant que les variables explicatives pour chaque relevé. En visualisant la valeur de la fonction de coût de l'algorithme des cartes de Kohonen pour différents nombre de groupes de relevés et en utilisant le critère du coude, l'auteur détermine alors qu'il y a six groupes d'individus. L'auteur sépare ensuite le jeu de données en un ensemble d'apprentissage et un ensemble de test. Il apprend un modèle linéaire dans chaque groupe d'individus de l'ensemble d'apprentissage. Il attribue ensuite à chaque individu de l'ensemble de test un numéro de groupe qui correspond au numéro du groupe du référent qui lui est le plus proche au sens de la distance euclidienne. Il effectue ensuite la prédiction de la variable réponse pour chaque individu de l'ensemble de test en utilisant le modèle linéaire appris dans le groupe qui lui a été attribué. Les résultats obtenus sont présentés dans les tableaux 14 et 15. Pour l'ensemble du jeu de données, nous ne pouvons donner qu'une valeur de la RMSE, et pas de coefficient de détermination, car cela reviendrait à apprendre un modèle linéaire sur l'ensemble du jeu de données, ce que nous avons déjà fait. De même, nous ne pouvons pas donner de valeur du coefficient de détermination pour l'ensemble de test.

Ces premiers résultats sont meilleurs que ceux obtenus avec une régression linéaire ou PLS simple.

4.2.5 Comparaisons

Détails des procédures de comparaison :

Pour les comparaisons effectuées dans cette partie, nous avons séparé de manière aléatoire le jeu de données en un ensemble d'apprentissage et un ensemble de test représentant respectivement 80% et 20% des données. Nous exécutons les algorithmes kreg, Spaeth, SOM-CLusterwise, SOM-PLS-Clusterwise et mbpls sur les ensembles d'apprentissage et calculons la valeur de la *RMSE* sur

Groupe numéro	*RMSE*	Coefficient de détermination
1	58.61	0.65
2	97.11	0.81
3	71.99	0.63
4	108.75	0.78
5	162.14	0.76
6	142.67	0.85
1,2,3,4,5 et 6	131.55	X

TABLE 14 – Données réelles : performances en apprentissage, cartes de Kohonen puis régression linéaire

Groupe numéro	*RMSE*	Coefficient de détermination
1	59.67	X
2	168.29	X
3	104.51	X
4	185.26	X
5	161.82	X
6	196.68	X
1,2,3,4,5 et 6	148.62	X

TABLE 15 – Données réelles : performances en généralisation, cartes de Kohonen puis régression linéaire

l'ensemble de test. Pour un relevé de l'ensemble de test, on prédit d'abord son groupe d'appartenance avec la méthode des 5 plus proches voisins, puis on lui associe comme valeur prédite de la variable réponse la prédiction issue du groupe qu'on lui a attribué.
Les algorithmes kreg, Spaeth et SOM-Clusterwise sont exécutés avec trois initialisations des groupes de relevés différents. Pour les algorithmes kreg et Spaeth, nous gardons les résultats de l'exécution ayant donné lieu à la plus petite valeur de la fonction de coût. Pour l'algorithme SOM-Clusterwise, nous construisons un ensemble de validation et à chaque étape de l'algorithme nous conservons la valeur de l'erreur de prédiction sur cet ensemble de validation, ainsi que la répartition des individus dans les groupes et les référents associés. A la fin de chacune des trois exécutions, nous gardons la classification et les référents relatifs à la plus petite erreur sur l'ensemble de validation. Pour les algorithmes mbpls et SOM-PLS-Clusterwise, les nombres de composantes PLS considérés sont 1 et 2. Nous avons fait en sorte qu'aucun groupe ne puisse se vider pour l'algorithme SOM-PLS-Clusterwise, puisque l'algorithme mbpls est implémenté de telle façon qu'aucun groupe ne puisse se vider. Pour chaque nombre de groupes et chaque nombre de composantes PLS considéré, nous exécutons notre algorithme SOM-PLS-Clusteriwse avec une unique initialisation des groupes de relevés différents, et l'algorithme mbpls avec vingt initialisations de groupes différentes. Nous n'exécutons notre algorithme qu'avec une seule initialisation des groupes de relevés car celui-ci est très coûteux en temps de calcul. De plus, pour chaque valeur de la température, on ne fait pas autant d'itérations que nécessaire jusqu'à ce que la fonction de coût ne décroisse plus : pour chaque valeur de la température on décide de considérer au plus 10 fois chaque relevé (*cf.* Partie **??** pour plus de détails). Pour l'algorithme mbpls, nous gardons les résultats de l'exécution ayant donné lieu à la plus petite valeur de la fonction de coût. Pour l'algorithme SOM-PLS-Clusterwise, nous construisons un ensemble de validation et conservons finalement pour chacune des trois exécutions la classification et les modèles PLS relatifs à la plus petite erreur sur l'ensemble de validation.

Résultats des comparaisons :

Dans un premier temps, nous allons comparer en termes de performances en prédiction notre algorithme SOM-Clusterwise (*cf.* Partie 3.2) aux algorithmes kreg et Spaeth (*cf.* 2.2.1) sur le jeu de données réelles, avec un nombre de groupes égal à 4 ou 6. Nous avions en effet déjà montré la supériorité de notre algorithme SOM-Clusterwise sur l'algorithme kreg sur données simulées. Nous voulons maintenant les comparer sur un jeu de données réelles. Pour l'algorithme SOM-Clusterwise,

nous étudierons aussi la répartition des relevés dans les différents groupes.
Les résultats concernant l'ensemble d'apprentissage sont présentés dans le tableau 16, et les résultats concernant l'ensemble de test sont présentés dans le tableau 17.

RMSE(ensemble d'apprentissage)		Algorithme		
		kreg	Spaeth	SOM-clus
Nombre de groupes	4	56.91	54.27	44.67
	6	33.16	33.67	31.92

Table 16 – Données réelles : Comparaison de la RMSE sur l'ensemble d'apprentissage de l'algorithme SOM-Clusterwise aux algorithmes Spaeth et kreg
Trois initialisations des groupes et référents, les résultats concernent l'exécution qui a aboutie à la plus petite valeur de la fonction de coût

RMSE(ensemble de test)		Algorithme		
		kreg	Spaeth	SOM-clus
Nombre de groupes	4	177.1	174.9	149.42
	6	183.39	172.81	168.44

Table 17 – Données réelles : Comparaison de la RMSE sur l'ensemble de test de l'algorithme SOM-Clusterwise aux algorithmes Spaeth et kreg
Trois initialisations des groupes et référents, les résultats concernent l'exécution qui a aboutie à la plus petite valeur de la fonction de coût

Nous constatons avec les tableaux 16 et 17 tout d'abord que les performances de notre algorithme SOM-Clusterwise sont supérieures en apprentissage et en généralisation à celles des versions *batch* et stochastique de l'algorithme présenté dans [Späth, 1979], c'est-à-dire les algorithmes kreg et Spaeth. Ensuite, si nous comparons les performances de notre algorithme SOM-Clusterwise à celles de [Fall, 2017] présentées dans 4.2.4, nous constatons que pour le même nombre de groupes, à savoir 6, les résultats de 4.2.4 sont légèrement supérieures aux nôtres en termes de généralisation, mais bien inférieures aux nôtres en termes d'apprentissage. Pour un nombre de groupes égal à 4, nous avons des performances en généralisation similaires à celles de la partie 4.2.4, mais aussi des performances en apprentissage bien meilleures.
Au sein de chaque groupe de relevés de l'ensemble d'apprentissage, on apprend un modèle linéaire afin de calculer le coefficient de détermination au sein de ce groupe. Les valeurs des coefficients de détermination sont présentés dans les tableaux 18 et 19. Pour l'ens

Groupe numéro	Ensemble d'apprentissage
1	0.96
2	0.83
3	0.98
4	0.98

Table 18 – Données réelles : coefficients de détermination après SOM-Clusterwise, $G_{algo} = 4$

Dans le tableaux 18 et 19, nous constatons que les valeurs des coefficients de détermination au sein de chaque groupe sont meilleurs que ceux présentés dans 4.2.4.

Ces premières expériences sur un jeu de données réel, nous ont tout d'abord montré la pertinence des cartes topologiques de Kohonen telles qu'on les a introduites dans notre algorithme SOM-Clusterwise par rapport à d'autres algorithmes sans carte de Kohonen tels que les algorithmes Spaeth et kreg. Ensuite, nous avons constaté de la pertinence de notre algorithme en termes de performances en apprentissage et en généralisation par rapport à d'autres méthodes où la classification est réalisée en amont de l'apprentissage de modèles, notamment celle appliquée pour obtenir

Groupe numéro	Ensemble d'apprentissage
1	0.72
2	0.67
3	0.73
4	0.71
5	0.68
6	0.74

TABLE 19 – Données réelles : coefficients de détermination après SOM-Clusterwise, $G_{algo} = 6$

des résultats existants (*cf.* partie 4.2.4).

Nous allons maintenant nous intéresser à interpréter les résultats obtenus avec notre algorithme SOM-Clusterwise, pour $G_{algo} = 4$ ou $G_{algo} = 6$. Nous commençons par regarder, pour chaque groupe de relevés, quels sont les mois prépondérants parmi ces groupes. Nous rappelons que nous n'avons pas utilisé les mois dans notre étude jusqu'à présent.
Sur la figure 7 nous représentons, pour chaque groupe, l'effectif du nombre de relevés pour chaque mois où des relevés ont été faits, pour $G_{algo} = 4$. Sur la figure 8 nous représentons, pour chaque groupe, l'effectif du nombre de relevés pour chaque mois où des relevés ont été faits, pour $G_{algo} = 6$.

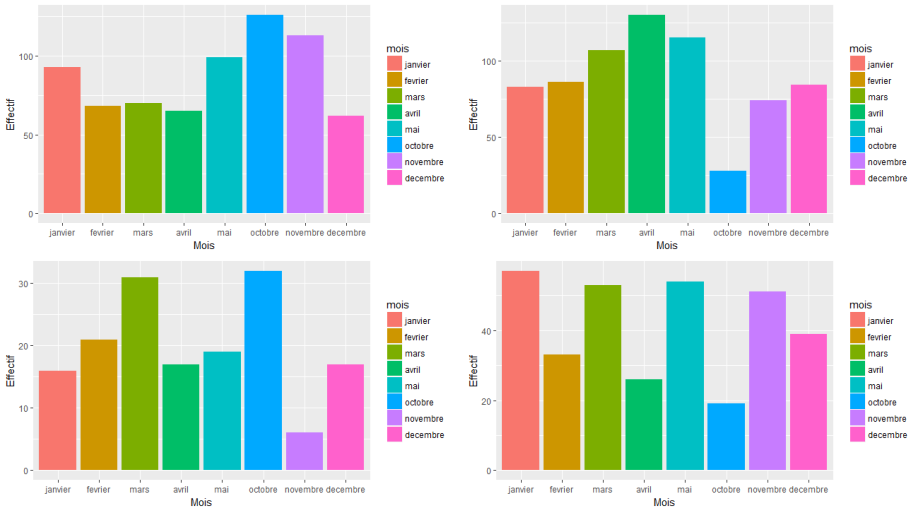

FIGURE 7 – SOM-Clusterwise : répartition des relevés suivant les mois, $G_{algo} = 4$

Pour $G_{algo} = 4$, nous constatons que la grande différence au niveau de la répartition des mois dans les groupes d'individus se situe au niveau des mois d'octobre et novembre. Deux groupes ne captent quasiment pas de relevé du mois d'octobre, et beaucoup de relevés du mois de novembre, tandis que deux autres groupes ne captent quasiment pas de relevé du mois d'octobre, mais beaucoup de relevés du mois de novembre. Il est à noter que le mois d'octobre est le mois d'intersection entre la saison des pluies et la saison sèche. Pour $G_{algo} = 6$, nous n'arrivons pas à identifier de phénomène similaire.

Nous nous intéressons maintenant aux différences d'ordre de grandeur qu'il peut y avoir pour les variables explicatives entre les différents groupes de relevés déterminés. Pour cela, pour $G_{algo} = 4$ et $G_{algo} = 6$, pour chaque variable explicative, nous traçons les boites à moustache de la variable pour chaque groupe. Cela nous permet d'avoir une première idée quant à l'existence ou non d'une différence d'ordre de grandeur. Ensuite, nous utilisons le test de Student à 95% pour comparer deux à deux les moyennes des valeurs de la variable explicative entre les groupes déterminés. Nous savons que l'on peut utiliser ce test pour comparer les moyennes entre deux groupes si on suppose que la distribution au sein de chacun de ces groupes est celle d'une loi normale. Ici, on ne peut pas légitimement faire cette supposition. Cependant, au sein de chaque groupe, étant donné le nombre

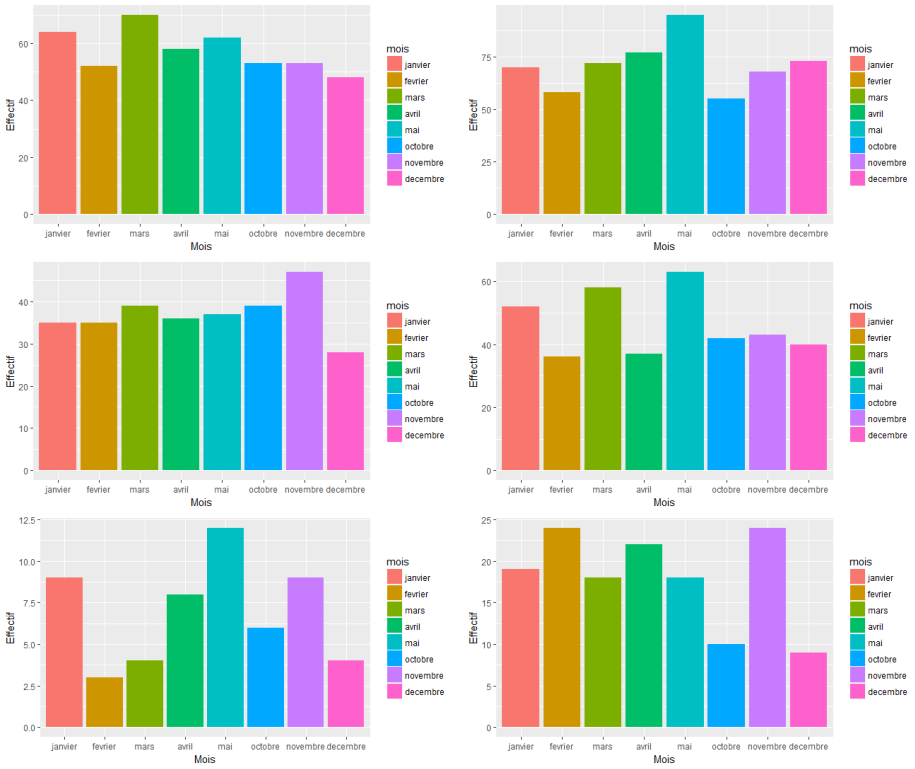

FIGURE 8 – SOM-Clusterwise : répartition des relevés suivant les mois, $G_{algo} = 6$

élevé de relevés, nous pouvons utiliser le théorème central limite, qui rend alors le test de Student applicable. Nous ne rentrons volontairement pas dans les détails, mais nous précisons juste que les relevés au sein de chaque groupe peuvent être considérés indépendants entre eux car ils ont perdu leur structure temporelle lors de la séparation aléatoire en un ensemble d'apprentissage et un ensemble de test.
Finalement, on constate que les variables pour lesquelles il existe une différence significative de moyenne entre les groupes sont :

— pour $G_{algo} = 4$: la mesure de l'AOT, la mesure du coefficient d'Angström, la pression et la direction du vent
— pour $G_{algo} = 6$: la direction du vent, la vitesse du vent, la température, l'humidité, la pression, la mesure de l'AOT et la mesure de l'Angström

On constate que pour $G_{algo} = 6$, toutes les variables explicatives présentent des différences significatives de moyenne entre les 6 groupes trouvés.

Sur la figure 9 sont représentées les boîtes à moustache d'une des quatre mesures de l'AOT pour $G_{algo} = 4$. Les boîtes à moustache pour les trois autres mesures de l'AOT sont très similaires.

Toujours dans un but d'interprétation, nous regardons, pour $G_{algo} = 4$ ou $G_{algo} = 6$, pour chaque variable explicative, pour chaque groupe de relevés déterminé, si la moyenne de cette variable au sein de ce groupe est significativement différente de la moyenne de cette variable pour l'ensemble des relevés. Nous réalisons cela avec un test de Student à 95%.
Dans les tabelaux 20 et 21, nous présentons, pour $G_{algo} = 4$ et $G_{algo} = 6$ respectivement, pour chaque groupe de relevés déterminé, pour quelles variables on peut légitimement affirmer que la moyenne au sein de ce groupe est significativement différente de la moyenne sur l'ensemble des relevés.

Dans un second temps, nous comparons les performances sur ce jeu de données réelles de notre algorithme SOM-PLS-Clusterwise (*cf.* Partie 3.3) et de l'algorithme mbpls présenté dans [Bougeard et al., 2017]. Les nombres de groupes considérés sont 4,6 et 9. Les résultats obtenus pour l'algorithme SOM-PLS-Clusterwise pour l'ensemble d'apprentissage et l'ensemble de test sont présentés dans

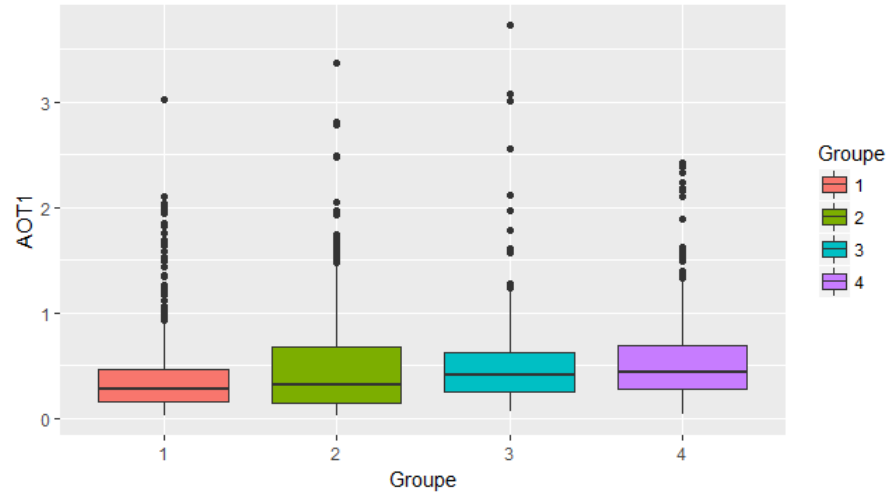

FIGURE 9 – SOM-Clusterwise : Boîtes à moustaches des mesures d'AOT selon les groupes, $G_{algo} = 4$

Groupe 1	AOT
Groupe 2	Pression
Groupe 3	AOT
Groupe 4	Angström, Direction du vent

TABLE 20 – SOM-Clusterwise, $G_{algo} = 4$, variables prépondérantes dans chaque groupe

le tableau 22 et 23. Les résultats obtenus pour l'algorithme mbpls pour l'ensemble d'apprentissage et l'ensemble de test sont présentés dans le tableau 24 et 25.

Nous analysons maintenant les résultats des tableaux 16, 23, 24 et 25. Pour l'algorithme mbpls, pour un nombre de composantes PLS donné, plus G_{algo} augmente, plus l'erreur en apprentissage diminue mais plus l'erreur en généralisation augmente. Pour l'algorithme SOM-PLS-Clusterwise, l'erreur en apprentissage et en généralisation est minimale pour $G_{algo} = 9$. Si on compare les erreurs en généralisation des deux algorithmes, on constate que ceux de l'algorithme mbpls sont meilleurs que ceux de l'algorithme SOM-PLS-Clusterwise. Cependant, nous n'avons pas exécuté l'algorithme SOM-PLS-Clusterwise comme il devrait l'être en théorie, c'est-à-dire que nous n'avons pas été jusqu'à la stabilisation de valeur de la fonction de coût pour une valeur de température donnée. Enfin, les résultats des deux algorithmes en généralisation sont les plus proches pour $G_{algo} = 9$. Cela peut s'interpréter en relation avec ce qu'on avait constaté pour les données simulées : on peut supposer que le nombre de groupes réel se situe entre 6 et 9.

Nous ne pouvons pas légitimement comparer ces résultats avec ceux des tableaux 16 et 17, car nous n'avons pas ici calculé les erreurs en prédiction et en généralisation pour tous les nombres de composantes PLS possibles. Nous pouvons cependant souligner la différence de temps d'exécution des deux algorithmes qui est très importante.

Remarque :
Nous avons aussi exécuté l'algorithme SOM-PLS-Clusterwise avec une carte de taille 10×10, en faisant en sorte que les groupes puissent se vider. Pour chaque valeur de la température, nous avons considéré au maximum trois fois chaque individu. Finalement, nous trouvons que 85 groupes sont-non vides. La RMSE sur l'ensemble d'apprentissage est assez faible (égale à 31 $\mu g.m^{-3}$) mais la RMSE sur l'ensemble de test est assez élevée (égale à 300 $\mu g.m^{-3}$).
Nous notons que l'algorithme mbpls échoue (renvoie un message d'erreur) lorsqu'on lui passe en paramètre un nombre de groupe supérieur à 20, sur ce jeu de données réel.

Groupe 1	Humidité
Groupe 2	Humidité, Pression, Direction du vent
Groupe 3	AOT
Groupe 4	Pression, Direction du vent
Groupe 5	
Groupe 6	Température

TABLE 21 – SOM-Clusterwise, $G_{algo} = 6$, variables prépondérantes dans chaque groupe

RMSE(ensemble d'apprentissage)		Nombre de groupes		
		4	6	9
Nombre de composantes	1	168.14	230.45	132.79
	2	246.44	254.22	231.26

TABLE 22 – Données réelles : Résultats en apprentissage de l'algorithme SOM-PLS-Clusterwise
Une initialisation des groupes d'individus

5 Conclusion

Durant ce stage, nous avons réussi à exploiter pertinemment les cartes topologiques de Kohonen pour effectuer de la régression clusterwise . Dans un premier temps, nous avons développé l'algorithme SOM-Clusterwise (*cf.* partie 3.2), qui est une version *batch* de l'algorithme présenté dans [Späth, 1979] à laquelle on rajoute les cartes topologiques de Kohonen. Les avantages de cet algorithme sont sa robustesse générale lorsque l'on introduit des critères d'*early-stopping*, son temps d'exécution réduit, et ses performances qui se révèlent être meilleures qu'une version *batch* ou stochastique de l'algorithme présenté dans [Späth, 1979] sans carte topologique. Cependant, l'algorithme présenté dans [Bougeard et al., 2017], qui reprend le concept de l'algorithme présenté dans [Späth, 1979], en remplaçant la régression linéaire par la régression PLS, obtenait de meilleures performances que l'algorithme SOM-Clusterwise. Les problèmes de robustesse étaient ainsi résolus avec la régression PLS. Nous avons alors développé un nouvel algorithme, appelé SOM-PLS-Clusterwise (*cf.* partie 3.3), qui reprend le principe de l'algorithme présenté dans [Bougeard et al., 2017] en y introduisant les cartes topologiques de Kohonen. Cet algorithme est assez coûteux en temps mais nous avons réussi à identifier des cas, dont nous avons su expliquer la particularité, où il se révèle bien meilleur en termes de performances que l'algorithme présenté dans [Bougeard et al., 2017].
Nous avons par la suite travaillé sur un jeu de données réelles. Nous avons dans un premier temps exécuté l'algorithme SOM-Clusterwise avec deux nombres de groupes différents : 4 et 6. Pour un nombre de groupes égal à 4, nous avons eu de meilleures performances (en prédiction et classification) qu'une autre méthode où le nombre de groupes, déterminé après avoir utilisé les cartes de Kohonen, précédait une régression linéaire dans chacun de ces groupes. De plus, pour les deux nombres de groupes considérés, l'algorithme SOM-Clusterwise surpasse l'algorithme présenté dans [Späth, 1979]. Dans un second temps, nous avons comparé l'algorithme SOM-PLS-Clusterwise à l'algorithme présenté dans [Bougeard et al., 2017]. Nous notons que les performances de l'algorithme présenté dans [Bougeard et al., 2017] sont très légèrement meilleures à celles de l'algorithme SOM-PLS-Clusterwise. Ces deux algorithmes sont bien plus coûteux en termes de temps d'exécution que l'algorithme SOM-Clusterwise.

RMSE(ensemble de test)		Nombre de groupes		
		4	6	9
Nombre de composantes	1	241.91	260.12	198.77
	2	175.53	180.24	175.56

TABLE 23 – Données réelles : Résultats en généralisation de l'algorithme SOM-PLS-Clusterwise
Une initialisation des groupes d'individus

RMSE(ensemble d'apprentissage)		Nombre de groupes		
		4	6	9
Nombre de composantes	1	112.70	89.81	73.24
	2	103.44	87.07	74.83

TABLE 24 – Données réelles : Résultats en apprentissage de l'algorithme mbpls
20 initialisations des groupes d'individus ; on garde le résultat relatif à la plus petite valeur de la fonction de coût

6 Perspectives

Comme nous l'avons expliqué, pour les algorithmes SOM-Clusterwise et SOM-PLS-Clusterwise, afin que l'exécution ne nécessite pas trop de temps, nous avons dû opérer certaines simplifications par rapport aux algorithmes théoriques. Pour l'algorithme SOM-Clusterwise, la phase d'affectation a été simplifiée et pour une valeur donnée de la température, nous ne considérons qu'une seule fois l'ensemble des individus. Pour l'algorithme SOM-PLS-Clusterwise, pour une valeur donnée de la température, nous ne considérons aussi qu'une seule fois l'ensemble des individus. En fait, pour les deux algorithmes, pour une valeur donnée de la température, nous n'avons pas considéré autant de fois que nécessaire tous les individus pour que la fonction de coût ne décroisse plus. Nous pensons que cela pourrait améliorer les performances de nos algorithmes en généralisation. En effet, quand nous atteignons une valeur donnée de la température, la valeur de la fonction de coût est plus haute avec notre manière d'exécution actuelle que si pour toutes les valeurs de température précédentes on avait considéré autant de fois les relevés de sorte qu'à chaque valeur de température la fonction de coût ne décroisse plus. Nous pensons alors que nous débutons le traitement pour cette valeur de température donnée avec des groupes et des référents (ou modèles PLS pour l'algorithme SOM-PLS-Clusterwise) qui ne sont pas optimaux. En résumé, si on suppose que pour une certaine valeur de la température, nous pouvons au maximum diminuer la fonction de coût d'une quantité l, avec notre manière d'exécution actuelle, pour une valeur de température donnée, on la diminue d'une quantité inférieure à l. Le résultat final n'est donc pas optimal.

Remarque :
Étant donné le sur-apprentissage qui peut avoir lieu pour des petites valeurs de la température, il est aussi judicieux de conserver notre manière d'exécution actuelle pour les plus petites valeurs de la température uniquement.

Finalement, une solution à ce problème serait d'implémenter notre algorithme avec Spark par exemple pour exploiter la parallélisation sur plusieurs *clusters* de calcul.

RMSE(ensemble de test)		Nombre de groupes		
		4	6	9
Nombre de composantes	1	181.08	203.73	197.67
	2	153.98	165.83	168.21

TABLE 25 – Données réelles : Résultats en généralisation de l'algorithme mbpls
20 initialisations des groupes d'individus ; on garde le résultat relatif à la plus petite valeur de la fonction de coût

7 Nomenclature

On appelle algorithme **Spaeth** l'algorithme présenté dans la partie 2.2.1, dont le pseudo-code est disponible en annexe B.
Il s'agit d'un algorithme stochastique dont la fonction de coût est la somme des carrés des résidus pour la régression linéaire dans chaque groupe d'individus.

On appelle algorithme **kreg** l'algorithme présenté dans la partie 2.2.1, qui est une version *batch* de l'algorithme de Spaeth, dont le pseudo-code est disponible en annexe C.
Il s'agit d'un algorithme *batch* dont la fonction de coût est la somme des carrés des résidus pour la régression linéaire dans chaque groupe d'individus.

On appelle algorithme **mbpls** l'algorithme cw.multiblock du *package R mbclusterwise* présenté dans la partie 2.2.1, qui correspond à l'algorithme de Spaeth avec la régression PLS à la place de la régression linéaire.
Il s'agit d'un algorithme stochastique dont la fonction de coût est la somme des carrés des résidus pour la régression PLS dans chaque groupe d'individus.

On appelle algorithme **flexmix** l'algorithme présenté dans la partie 2.2.2, dont le pseudo-code est disponible en annexe D.
Il s'agit d'un algorithme de type Espérance-Maximisation.

On appelle algorithme **SOM-Clusterwise** l'algorithme présenté dans la partie 3.2, dont le pseudo-code est disponible en annexe F.
Il s'agit d'un algorithme *batch* dont la fonction de coût est similaire à celle de l'algorithme des cartes de Kohonen, adaptée à la régression linéaire.

On appelle algorithme **SOM-Clusterwise-c** l'algorithme SOM-Clusterwise avec le critère d'*early-stopping* 1. défini en partie 3.2.4, c'est-à-dire qu'on arrête l'algorithme quand les individus ne changent plus de groupe.

On appelle algorithme **SOM-Clusterwise-d** l'algorithme SOM-Clusterwise avec le critère d'*early-stopping* 3. défini en partie 3.2.4, c'est-à-dire qu'on arrête l'algorithme quand l'erreur sur l'ensemble d'apprentissage commence à décroître trop lentement.

On appelle algorithme **SOM-PLS-Clusterwise** l'algorithme présenté dans la partie 3.3, dont le pseudo-code est disponible en annexe G.
Il s'agit d'un algorithme stochastique dont la fonction de coût est similaire à celle de l'algorithme des cartes de Kohonen, adaptée à la régression PLS.

Références

Hervé Abdi. Partial least squares regression and projection on latent structure regression (pls regression). *Wiley Interdisciplinary Reviews : Computational Statistics*, 2(1) :97–106, 2010.

Stéphanie Bougeard, Hervé Abdi, Gilbert Saporta, and Ndèye Niang. Clusterwise analysis for multiblock component methods. *Advances in Data Analysis and Classification*, 2017.

Wayne S DeSarbo and William L Cron. A maximum likelihood methodology for clusterwise linear regression. *Journal of classification*, 5(2) :249–282, 1988.

Edwin Diday. Une nouvelle méthode en classification automatique et reconnaissance des formes : la méthode des nuées dynamiques. *Revue de statistique appliquée*, 19(2) :19–33, 1971.

Gérard Dreyfus. *Apprentissage statistique*, pages 351–428. Eyrolles, 2008.

Maguette Fall. Estimation de la concentration en pm10 en fonction de l'épaisseur optique des aérosols (aot) et des variables météorologiques : Etude comparative de deux méthodes (la carte topologique de kohonen et la r*regression typologique*), 2017.

Francois Kali. *Etude statistique de la variabilité des teneurs atmosphériques en aérosols désertiques en Afrique de l'Ouest*. PhD thesis, Université Cheikh Anta Diop de Dakar, 2016.

Teuvo Kohonen. Self-organizing maps, springer series in information sciences. *Berlin : Springer*, 3 :30, 2001.

Lutz Prechelt. *Early Stopping - But When ?* Springer Berlin Heidelberg, 1998.

Helmuth Späth. Algorithm 39 clusterwise linear regression. *Computing*, 22(4) :367–373, 1979.

M Tenenhaus, J.-P Gauchi, and C Ménardo. Régression pls et applications. *Revue de statistique appliquée*, 43(1) :7–63, 1995.

Table des figures

Liste des tableaux

A Pseudo-code : algorithme de régression PLS

Détails en partie 2.1.

Données : X une matrice de dimension $n \times p$ centrée et y un vecteur de dimension $n \times 1$ centrée
Résultat : T une matrice de dimension $n \times q$ dont les colonnes sont $(T^1, T^2, ..., T^q)$
$X_1 \leftarrow X$
$y_1 \leftarrow y$
pour *i allant de 1 à q* **faire**
  $C^i \leftarrow \frac{X_i^T y_i}{||X_i^T y_i||^2}$
  $T^i \leftarrow X_i C^{iT}$
  $W^i \leftarrow \frac{X_i^T T^i}{||T^i||^2}$
  $V^i \leftarrow \frac{y_i^T T^i}{||T^i||^2}$
  $X_{i+1} = X_i - T^i W^{iT}$
  $y_{i+1} = y_i - T^i V^{iT}$
fin
retourner $T^1, T^2, ..., T^q$

B Pseudo-code : algorithme Spaeth

Détails en partie 2.2.1.
Données : X une matrice de dimension $n \times p$, y un vecteur de dimension $n \times 1$ avec $n > p$ et G entier naturel
Résultat : Une partition $C_1, C_2, ..., C_G$ de l'ensemble $\{1, ..., n\}$
$C_1, C_2, ..., C_G \leftarrow$ partition aléatoire de $\{1, ..., n\}$ avec $\forall g \in [\![1, G]\!]$, $card(C_g) \geq p$
`/* Si on ne peut pas trouver une telle partition, on arrête l'algorithme */`
(1) **pour** *i allant de 1 à n* **faire**
  $j \leftarrow$ numéro du groupe d'appartenance de l'individu i (i.e $i \in C_j$)
  si *il existe k tel que* $k \neq j$ *et* $E(C_k + \{i\}) + E(C_j - \{i\}) \leq E(C_k) + E(C_j)$ *et* $k = \text{argmin}_h [E(C_h + \{i\}) + E(C_j - \{i\})]$ *et* $card(C_j - \{i\}) \geq p$ **alors**
    Déplacer l'individu du groupe j au groupe k
    Mettre à jour les modèles au sein des groupes j et k
fin
si *Aucun individu n'a été déplacé* **alors**
  retourner $C_1, C_2, ..., C_G$
sinon
  Revenir en (1)
fin

C Pseudo-code : algorithme kreg

Détails en partie 2.2.1.

Données : X une matrice de dimension $n \times p$, y un vecteur de dimension $n \times 1$, G entier naturel

Résultat : Une partition $C_1, C_2, ..., C_G$ de l'ensemble $\{1, ..., n\}$, $\beta^1, \beta^2, ..., \beta^G$ G vecteurs de $\mathcal{R}^p$

$C_1, C_2, ..., C_G \leftarrow$ partition aléatoire de $\{1, ..., n\}$

On initialise A à partir de cette partition aléatoire

$\forall g \in [\![1, G]\!]$ $\beta^c \leftarrow$ vecteur de coefficients linéaires de y par X en ne considérant que les individus du groupe g

Calculer $J(A, \beta)$

tant que $J(A, \beta)$ *décroît* **faire**

- $\forall i \in [\![1, n]\!]$ $A(z_i) = \text{argmin}_{g \in [\![1,G]\!]}(y_i - X_i^T \beta^g)^2$
- Mettre à jour $C_1, C_2, ..., C_G$
- $\forall g \in [\![1, G]\!]$ $\beta_g \leftarrow$ vecteur de coefficients linéaires de y par X en ne considérant que les individus du groupe g
- Calculer $J(A, \beta)$

fin

Retourner $C_1, C_2, ..., C_G$ et $\beta^1, \beta^2, ..., \beta^G$

D Pseudo code : algorithme flexmix

Détails en partie 2.2.2.

Données : X une matrice de dimension $n \times p$, y un vecteur de dimension $n \times 1$ avec $n > p$, G entier naturel et s réel

Résultat : Une partition $C_1, C_2, ..., C_G$ de l'ensemble $\{1, ..., n\}$, une matrice B de dimension $p \times G$, $(\lambda_1, ..., \lambda_g) \in \mathbb{R}^p$, une matrice P de dimension $n \times G$

INITIALISATION

$C_1, C_2, ..., C_G \leftarrow$ partition aléatoire de $\{1, ..., n\}$ avec $\forall g \in [\![1, G]\!]$, $card(C_g) \geq p$

`/* Si on ne peut pas trouver une telle partition, on arrête l'algorithme */`

pour *g allant de 1 à G* **faire**

- $\lambda_g \leftarrow$ proportion d'individus dans le groupe C_g
- $\sigma_g \leftarrow$ écart-type au sein du groupe C_g
- $y^g \leftarrow$ vecteur Y pour les individus du groupe C_g
- $Xg \leftarrow$ matrice X pour les individus du groupe C_g
- $B_g \leftarrow$ vecteur de $\mathbb{R}^p$ des coefficients dans la régression linéaire de y^g par Xg

fin

$L \leftarrow \sum_{i=1}^{n} ln(\sum_{g=1}^{G} \lambda_g f_{i,g}(y_i | X_i, \sigma_g, B_g))$

ITERATIONS

tant que $L < s$ **faire**

- **Première phase**
- **pour** *i allant de 1 à n* **faire**
  - **pour** *g allant de 1 à G* **faire**
    - $p_{i,g} \leftarrow \frac{\lambda_g f_{i,g}(y_i | X_g, \sigma_g, B_g)}{\sum_{g=1}^{K} \lambda_g f_{i,g}(y_i | X_g, \sigma_g, B_g)}$
  - **fin**
- **fin**
- $\lambda_g \leftarrow h(p_{i,g})$ $\forall g \in [\![1, G]\!]$
- **Deuxième phase**
- $B_g \leftarrow$ vecteur de $\mathbb{R}^p$ des coefficients dans la régression linéaire de y par X pondérés par $p_{i,g}^{\frac{1}{2}}$ $\forall g \in [\![1, G]\!]$
- $\sigma_g \leftarrow m(p_{i,g}, B_g, Xg)$ $\forall g \in [\![1, G]\!]$
- $L \leftarrow \sum_{i=1}^{n} ln(\sum_{g=1}^{K} \lambda_g f_{i,g}(y_i | X_i, \sigma_g, B_g))$

fin

retourner $C_1, ..., C_G$, $B = (B_1 ... B_g)$, $\lambda_1, ..., \lambda_G$, $P = (p_{i,g})_{i \in \{1,...,n\}, g \in \{1,...,G\}}$

E Pseudo code : algorithme des cartes de Kohonen

Détails en partie 2.3.
Données : X une matrice de dimension $n \times p$, G entier naturel, $s \in \mathcal{R}$
Résultat : Une partition $C_1, C_2, ..., C_G$ de l'ensemble $\{1, ..., n\}$
$C_1, C_2, ..., C_G \leftarrow$ partition aléatoire de $\{1, ..., n\}$
On initialise A et K à partir de cette partition aléatoire
$w_g \leftarrow \frac{\sum_{r=1}^{G} K(\delta(r,g))z_r}{\sum_{r=1}^{G} K(\delta(r,g))n_r}$ $\forall g \in [\![1, G]\!]$
Calculer $j(A, W)$
tant que $J(W, \chi) > s$ **faire**
  $\forall i \in [\![1, n]\!]$ $A(z_i) \leftarrow \text{argmin}_{r \in [\![1,G]\!]} \sum_{g=1}^{G} K(\delta(r, g))\|z_i - w_g\|_2$
  Mettre à jour $C_1, C_2, ..., C_G$
  $\forall g \in [\![1, G]\!]$ $w_g \leftarrow \frac{\sum_{r=1}^{G} K(\delta(r,g))z_r}{\sum_{r=1}^{G} K(\delta(r,g))n_r}$ $\forall g \in [\![1, G]\!]$
  Calculer $J(A, W)$
fin
Retourner $C_1, C_2, ..., C_G$

F Pseudo-code : algorithme SOM-Clusterwise

Détails en partie 3.2.
Données : X une matrice de dimension $n \times p$, Y un vecteur de dimension $n \times 1$, G entier naturel
Résultat : Une partition $C_1, C_2, ..., C_G$ de l'ensemble $\{1, ..., n\}$, $\beta^1, \beta^2, ..., \beta^G$ G vecteurs de $\mathcal{R}^p$
$C_1, C_2, ..., C_G \leftarrow$ partition aléatoire de $\{1, ..., n\}$
On initialise A et les M_c à partir de cette partition aléatoire
$\forall g \in [\![1, G]\!]$ $\beta^g \leftarrow \left(X^T M_g X\right)^{-1} X^T M_g X$
Calculer $J(A, \beta)$
tant que $J(A, \beta)$ *décroît* **faire**
  $\forall i \in [\![1, n]\!]$ $A(z_i) = \text{argmin}_{r \in [\![1,G]\!]} \sum_{g=1}^{G} K(\delta(r, g))(y_i - X_i^T \beta^g)^2$
  Mettre à jour $C_1, C_2, ..., C_G$
  $\forall g \in [\![1, G]\!]$ $\beta_g \leftarrow \left(X^T M_g X\right)^{-1} X^T M_g X$
  Calculer $J(A, \beta)$
fin
Retourner $C_1, C_2, ..., C_G$ et $\beta^1, \beta^2, ..., \beta^G$

G Pseudo-code : algorithme SOM-PLS-Clusterwise

Détails en partie 3.3.1.

Données : X une matrice de dimension $n \times p$, Y un vecteur de dimension $n \times 1$ et G entier naturel

Résultat : Une partition $C_1, C_2, ..., C_G$ de l'ensemble $\{1, ..., n\}$, $mpls^1, mpls^2, ..., mpls^G$ G modèles PLS

$C_1, C_2, ..., C_G \leftarrow$ partition aléatoire de $\{1, ..., n\}$
On initialise A à partir de cette partition aléatoire
$\forall g \in [\![1, G]\!]$ $mpls^g \leftarrow$ modèle de régression PLS de Y^g par X^g
Calculer $J(A, \beta, mpls^1, ..., mpls^G)$
tant que $J(A, mpls^1, ..., mpls^G)$ *décroît* **faire**
  pour i *allant de 1 à* n **faire**
    $j \leftarrow$ numéro du groupe d'appartenance de l'individu i (i.e $i \in C_j$)
    $k \leftarrow$ le numéro du groupe qui induit la plus grande diminution de J quand on déplace l'individu du groupe C_j au groupe C_k
    Déplacer l'individu du groupe C_j au groupe C_k
    Mettre à jour $mpls^j$ et $mpls^k$
    Calculer $J(A, \beta, mpls^1, ..., mpls^G)$
  fin
fin
Retourner $C_1, C_2, ..., C_G$, $mpls^1, mpls^2, ..., mpls^G$

H Critères de comparaison

On rappelle ici les différents critères de comparaison utilisés lors de la comparaison des algorithmes.

1. l'indice de Calinsky-Harabasz, qui est utilisé comme une mesure de la qualité d'une classification de n individus en G groupes, noté CH, est tel que CH $= \frac{(G-1)V_{inter}}{(n-G)V_{intra}}$ avec V_{inter} la variance inter-groupes et V_{intra} la variance intra-groupe.
2. l'indice rand, qui est utilisé pour comparer deux classifications de n individus en G groupes, noté rand, est tel que rand$= \frac{a+d}{2}$ où a est le nombre de paires d'individus classés ensemble dans le même groupes (au sens où les deux individus de la paire ne sont pas dans deux groupes différents) pour les deux classifications et d est le nombre de paires d'individus classés dans deux groupes différents (au sens où les deux individus de la paire sont dans deux groupes différents) pour les deux classifications.
3. la *Root Mean Squared Error* entre deux vecteurs y et $\hat{y}$ de taille n notée RMSE, est telle que RMSE$= \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ avec y_i (respectivement $\hat{y}_i$) le i^e élément du vecteur y (respectivement $\hat{y}$).