L'algorithme GLRM

Tutoriel Tanagra

1 Introduction

La réduction de dimensionnalité est une tâche essentielle de l'analyse exploratoire. L'objectif est de concentrer la représentation des données sur un faible nombre de facteurs, exprimés à l'aide des variables initiales, qui traduisent les « formes » (*pattern*) pertinentes qui les régissent. En résumant l'essentiel de l'information sur un jeu de caractères réduits : nous pouvons représenter graphiquement les observations quand leur nombre est faible (souvent dans le plan) pour mieux apprécier les proximités ; les algorithmes de machine learning subséquents (supervisé ou non) sont plus efficaces lorsqu'ils sont appliqués sur un espace de représentation plus consistant.

La littérature francophone est pléthorique à ce sujet (Rakotomalala, 2020). La très grande majorité des auteurs se concentrent majoritairement sur l'ACP (analyse en composantes principales) lorsque les descripteurs sont quantitatifs, sur l'ACM (analyse des correspondances multiples) lorsqu'ils sont qualitatifs. Lorsque les descripteurs sont mixtes, l'AFDM (analyse factorielle des données mixtes) est la solution la plus naturelle. Elle est peu visible dans les supports. Elle s'inscrit pourtant dans l'obédience de l'analyse des données à la française que nous connaissons bien. Elle constitue une véritable généralisation de l'ACP normée et de l'ACM.

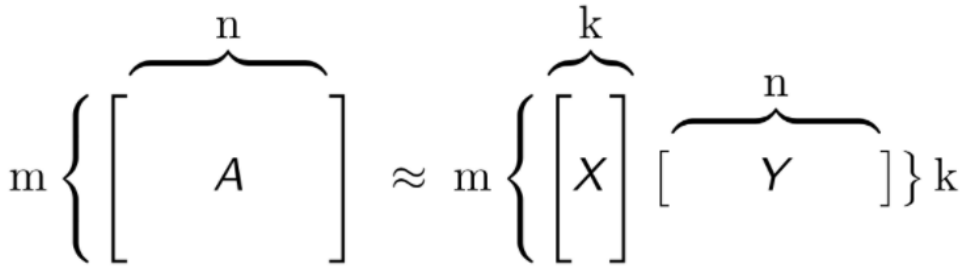
Mais d'autres approches sont possibles, plutôt visibles dans la littérature anglo-saxonne. Mon attention a été attirée récemment par l'algorithme GLRM (*generalized low rank model*) (Boehmke et Greenwell, 2020 ; chapitre 18). Plusieurs qualités sont mises en avant pour justifier son utilisation : à l'instar de l'AFDM, elle sait traiter nativement les descripteurs mixtes ; en s'affranchissant de la définition des facteurs sous la forme de combinaisons linéaires des variables initiales, elle dépasse les limitations de l'ACP notamment et est capable de capturer des « formes » non-linéaires. Une solution performante qui nous

sortirait donc du sempiternel cadre de la décomposition en valeurs singulières d'une variante de la matrice des données initiales. L'idée est d'autant plus séduisante que l'appréhension des problèmes à très forte dimensionalité est devenu un standard dans le contexte du traitement des données non-structurées, en particulier le « *text mining* ».

Dans ce tutoriel, nous décrivons succinctement la méthode, puis nous étudions son comportement en l'opposant à l'ACP lorsque les descripteurs sont tous quantitatifs, à l'AFDM lorsqu'ils sont mixtes. Nous utilisons les jeux de données décrits dans l'ouvrage consacré à l'analyse factorielle qui a été mise en ligne récemment (Rakotomalala, 2020). Nous exploitons l'implémentation disponible dans la librairie H2O que nous avons explorée déjà à plusieurs reprises (Tutoriels Tanagra – H20).

2 La méthode GLRM

Dans cette section, nous adoptons les notations de la documentation de H20. L'idée de l'algorithme GLRM est de résumer une matrice de données A composée de m lignes (m observations) et n colonnes (n variables) en deux matrices X (m, k) et Y (n, k) où « k » serait la dimension de la représentation factorielle.



« n » est le nombre de variables si elles sont toutes quantitatives. Sinon, il correspond à l'addition du nombre quantitatives et d'indicatrices 0/1 issues du codage disjonctif complet des descripteurs qualitatifs. « X » représente la description des observations dans le nouvel espace de représentation. « Y » enfin correspond aux « archétypes » permettant de comprendre la nature des « facteurs » mis en avant par la méthode. On peut faire le parallèle avec les vecteurs propres de l'ACP par exemple.

Le produit « X.Y » représente une approximation des données initiales A. L'algorithme chercher à minimiser une fonction de perte qui quantifie l'écart entre A et les données reconstituées « X.Y », principalement :

$$quadratic\ loss = minimize\{\sum_{i=1}^{m}\sum_{j=1}^{n}\left(A_{i,j} - X_i Y_j\right)^2\}$$

D'autres types de fonctions de perte sont à privilégier selon la nature des données et du problème à traiter (présence de valeurs atypiques par ex.), mais le principe est là.

Une heuristique est utilisée pour obtenir la solution. L'idée de la minimisation alternée (*alternating minimization*) est d'ajuster alternativement via une descente du gradient les valeurs des matrices X et Y de manière à réduire itérativement les valeurs calculées de la fonction de perte. Cela peut être un inconvénient, le nombre d'itérations peut être très élevé avant de parvenir à la convergence ; c'est surtout, selon ses auteurs, un avantage car : nous pouvons utiliser différents types de fonctions adaptées aux données et au problème à traiter, la liaison entre les « archétypes » et les variables initiales ne sont pas régies par un type de relation prédéfinie (elle peut être non-linéaire notamment), il est possible d'introduire des mécanismes de régularisation (de type « ridge » ou « lasso ») pour prévenir les phénomènes de surapprentissage (Boehmke et Greenwell, 2020 ; section 18.3.3).

Comme de coutume dans les algorithmes de réduction de dimensionnalité, le choix du nombre de dimensions « k » reste un problème ouvert.

Enfin, en ce qui me concerne, j'ai surtout été intéressée par cette approche en raison de sa capacité à traiter les descripteurs actifs mixtes (*heterogenous features*), quantitatifs ou qualitatifs. En ce sens, elle se place comme une alternative de l'AFDM (analyse factorielle des données mixtes), très utile et pourtant peu décrite dans la littérature francophone.

3 Equivalence avec l'ACP - Descripteurs quantitatifs

Les résultats sont « similaires » à ceux de l'analyse en composantes principales lorsque nous traitons une base composée exclusivement de descripteurs quantitatifs. Nous

disposons à la sortie d'un jeu de variables synthétiques réduites (facteurs, composantes), expression des variables initiales. Mais GLRM diffère par le fait que ses facteurs ne sont pas explicitement formés à partir de combinaisons linéaires des variables originelles, ils ne sont pas nécessairement orthogonaux entre eux. Nous allons vérifier cela empiriquement dans cette section en appliquant l'algorithme sur un jeu de données exemple et en comparant les propriétés des facteurs avec ceux de l'ACP.

3.1 Données "autos"

Nous reprenons les données « autos » utilisées pour illustrer les mécanismes de l'ACP dans notre ouvrage de référence (Rakotomalala, 2020 ; chapitre 1).

3.1.1 Chargement, inspection

La base est composée de (m = 18) observations décrites par (n = 6) variables.

```
#changer de répertoire
setwd("... votre dossier de travail ...")

#librairie
library(xlsx)

#autos ACP
autos <- read.xlsx("Data_Methodes_Factorielles.xlsx",sheetIndex = 1)

#labels des lignes
rownames(autos) <- autos$Modele
autos$Modele <- NULL

#nombre de lignes du tableau de données
m <- nrow(autos)

#nombre de descripteurs
n <- ncol(autos)

#affichage
print(autos)
##                  CYL PUISS LONG LARG POIDS VMAX
## Alfasud TI      1350    79  393  161   870  165
## Audi 100        1588    85  468  177  1110  160
## Simca 1300      1294    68  424  168  1050  152
## Citroen GS Club 1222    59  412  161   930  151
## Fiat 132        1585    98  439  164  1105  165
## Lancia Beta     1297    82  429  169  1080  160
## Peugeot 504     1796    79  449  169  1160  154
## Renault 16 TL   1565    55  424  163  1010  140
## Renault 30      2664   128  452  173  1320  180
## Toyota Corolla  1166    55  399  157   815  140
```

```
## Alfetta 1.66    1570   109  428  162  1060  175
## Princess 1800   1798    82  445  172  1160  158
## Datsun 200L     1998   115  469  169  1370  160
## Taunus 2000     1993    98  438  170  1080  167
## Rancho          1442    80  431  166  1129  144
## Mazda 9295      1769    83  440  165  1095  165
## Opel Rekord     1979   100  459  173  1120  173
## Lada 1300       1294    68  404  161   955  140
```

3.1.2 Centrage et réduction

Nous centrons et réduisons les données en amont pour traiter les deux approches sur un pied d'égalité. Nous utilisons la variance population pour disposer des résultats conformes à ceux produits par les outils francophones.

```
#centrer et réduire
Z <- as.data.frame(lapply(autos,function(x){(x-mean(x))/sqrt((m-1)/m*var(x))}))
rownames(Z) <- rownames(autos)
print(Z)
##                        CYL       PUISS       LONG       LARG        POIDS
## Alfasud TI      -0.7750989 -0.28335818 -1.8850808 -1.0973453 -1.569006764
## Audi 100        -0.1201633  0.01963869  1.6058095  2.0010414  0.234161425
## Simca 1300      -0.9292014 -0.83885242 -0.4421794  0.2581989 -0.216630623
## Citroen GS Club -1.1273332 -1.29334771 -1.0007219 -1.0973453 -1.118214717
## Fiat 132        -0.1284188  0.67613189  0.2559986 -0.5163978  0.196595421
## Lancia Beta     -0.9209459 -0.13185975 -0.2094534  0.4518481  0.008765401
## Peugeot 504      0.4522175 -0.28335818  0.7214507  0.4518481  0.609821464
## Renault 16 TL   -0.1834554 -1.49534562 -0.4421794 -0.7100469 -0.517158654
## Renault 30       2.8408062  2.19111619  0.8610863  1.2264447  1.811933590
## Toyota Corolla  -1.2814357 -1.49534562 -1.6058095 -1.8719420 -1.982232807
## Alfetta 1.66    -0.1696962  1.23162613 -0.2559986 -0.9036961 -0.141498615
## Princess 1800    0.4577211 -0.13185975  0.5352698  1.0327956  0.609821464
## Datsun 200L      1.0080872  1.53462299  1.6523548  0.4518481  2.187593629
## Taunus 2000      0.9943280  0.67613189  0.2094534  0.6454972  0.008765401
## Rancho          -0.5219305 -0.23285870 -0.1163630 -0.1290994  0.376912239
## Mazda 9295       0.3779180 -0.08136027  0.3025438 -0.3227486  0.121463413
## Opel Rekord      0.9558024  0.77713084  1.1869027  1.2264447  0.309293432
## Lada 1300       -0.9292014 -0.83885242 -1.3730835 -1.0973453 -0.930384697
##                       VMAX
## Alfasud TI       0.56976043
## Audi 100         0.14597168
## Simca 1300      -0.53209032
## Citroen GS Club -0.61684807
## Fiat 132         0.56976043
## Lancia Beta      0.14597168
## Peugeot 504     -0.36257482
## Renault 16 TL   -1.54918332
## Renault 30       1.84112668
## Toyota Corolla  -1.54918332
## Alfetta 1.66     1.41733793
## Princess 1800   -0.02354382
## Datsun 200L      0.14597168
## Taunus 2000      0.73927593
```

```
## Rancho          -1.21015232
## Mazda 9295       0.56976043
## Opel Rekord      1.24782243
## Lada 1300       -1.54918332
```

3.2 ACP sur les données "autos"

Les données étant déjà standardisées, nous lançons une ACP non-normée à l'aide de la librairie "FactoMineR". Nous fixons à « k = 2 » la taille du nouvel espace de représentation.

```
#k = 2, taille du nouvel espace de représentation
k <- 2

#acp
library(FactoMineR)
acp <- PCA(Z,ncp=k,scale.unit=FALSE,graph=FALSE)
summary(acp)
##
## Call:
## PCA(X = Z, scale.unit = FALSE, ncp = k, graph = FALSE)
##
##
## Eigenvalues
##                        Dim.1   Dim.2   Dim.3   Dim.4   Dim.5   Dim.6
## Variance               4.421   0.856   0.373   0.214   0.093   0.043
## % of var.             73.681  14.268   6.218   3.565   1.547   0.722
## Cumulative % of var.  73.681  87.949  94.166  97.732  99.278 100.000
##
## Individuals (the 10 first)
##                     Dist    Dim.1    ctr   cos2    Dim.2    ctr   cos2
## Alfasud TI       |  2.868 | -2.139  5.749  0.556 |  1.786 20.693  0.388 |
## Audi 100         |  2.583 |  1.561  3.064  0.365 | -1.527 15.133  0.349 |
## Simca 1300       |  1.469 | -1.119  1.575  0.580 | -0.675  2.953  0.211 |
## Citroen GS Club  |  2.604 | -2.574  8.324  0.977 |  0.113  0.083  0.002 |
## Fiat 132         |  1.081 |  0.428  0.230  0.157 |  0.696  3.140  0.414 |
## Lancia Beta      |  1.065 | -0.304  0.116  0.082 | -0.196  0.250  0.034 |
## Peugeot 504      |  1.230 |  0.684  0.588  0.309 | -0.933  5.650  0.575 |
## Renault 16 TL    |  2.374 | -1.948  4.771  0.674 | -0.980  6.238  0.171 |
## Renault 30       |  4.668 |  4.410 24.437  0.892 |  1.064  7.342  0.052 |
## Toyota Corolla   |  4.036 | -3.986 19.964  0.975 |  0.236  0.362  0.003 |
##
## Variables
##                    Dim.1    ctr   cos2    Dim.2    ctr   cos2
## CYL              |  0.893 18.057  0.798 |  0.115  1.542  0.013 |
## PUISS            |  0.887 17.791  0.787 |  0.385 17.287  0.148 |
## LONG             |  0.886 17.763  0.785 | -0.381 16.959  0.145 |
## LARG             |  0.814 14.971  0.662 | -0.413 19.899  0.170 |
## POIDS            |  0.905 18.534  0.819 | -0.225  5.889  0.050 |
## VMAX             |  0.755 12.884  0.570 |  0.574 38.423  0.329 |
```

Nous notons essentiellement que les deux premiers facteurs regroupent 87.949% de l'information disponible. Toutes les variables, mis à part VMAX, sont fortement corrélés avec la première composante.

3.3 Inspection des résultats de l'ACP

Inspectons maintenant les résultats de l'ACP en mettant en avant les propriétés des facteurs.

3.3.1 Coordonnées des individus

Lorsque les effectifs sont réduits, inspecter les coordonnées – qui sont en relation directe avec les contributions – des individus est toujours intéressant, surtout lorsque les observations ne sont pas anonymes. Lorsque le nombre de dimensions est supérieur à 2, les possibilités de tri permettant de mettre en évidence les observations situées aux extrémités des axes me paraît même plus intéressant que d'avoir à jongler avec plusieurs plans factoriels simultanément.

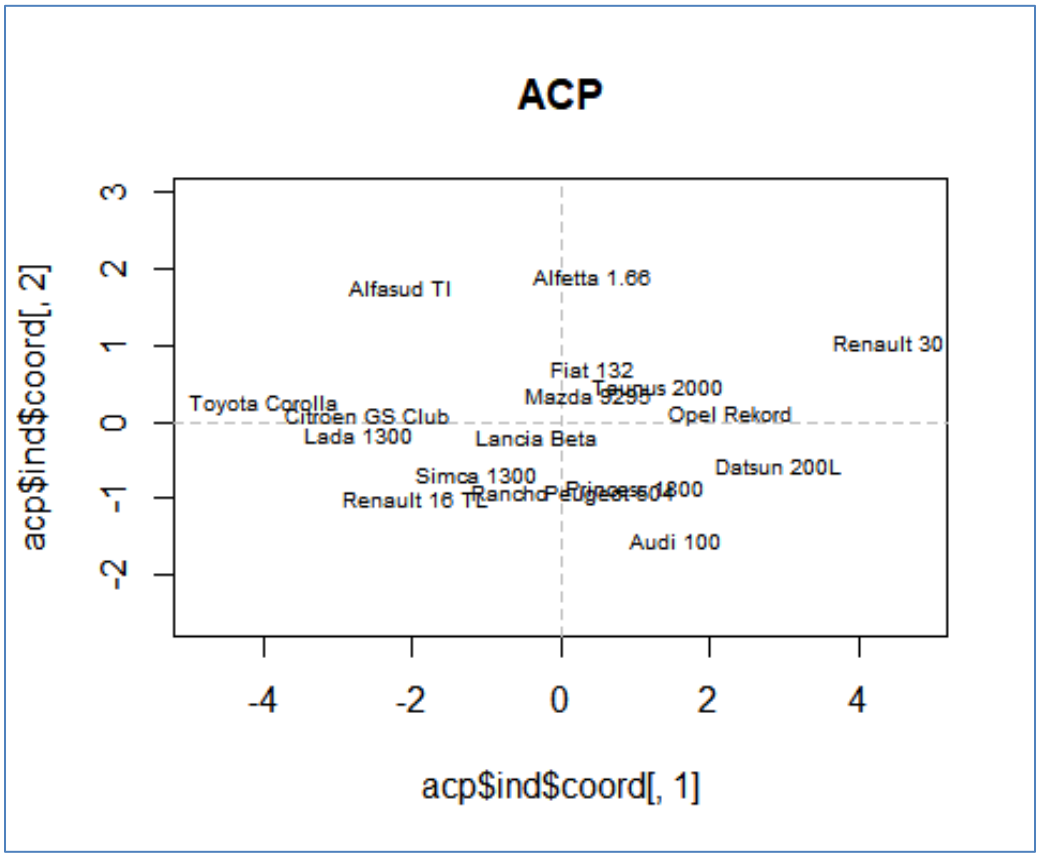
```
#coordonnées factorielles des individus
print(acp$ind$coord)
##                      Dim.1      Dim.2
## Alfasud TI      -2.1389236  1.7856810
## Audi 100         1.5614586 -1.5270403
## Simca 1300      -1.1193853 -0.6745052
## Citroen GS Club -2.5737417  0.1128842
## Fiat 132         0.4278552  0.6955673
## Lancia Beta     -0.3042376 -0.1961488
## Peugeot 504      0.6839285 -0.9330568
## Renault 16 TL   -1.9484926 -0.9804480
## Renault 30       4.4097346  1.0636332
## Toyota Corolla  -3.9857824  0.2362404
## Alfetta 1.66     0.4376582  1.9124482
## Princess 1800    1.0181754 -0.8417121
## Datsun 200L      2.9410803 -0.5591746
## Taunus 2000      1.3148804  0.4865225
## Rancho          -0.6911114 -0.8977214
## Mazda 9295       0.3857089  0.3561846
## Opel Rekord      2.2897682  0.1043446
## Lada 1300       -2.7085736 -0.1436988
```

3.3.2 Représentation des individus dans le plan

Comme nous n'avons que « k = 2 » facteurs, la représentation graphique reste un outil privilégié. On utilise "eqscplot" de la librairie « MASS » pour que les unités en abscisse et

ordonnées soient identiques et ainsi mieux apprécier les dispersions. On voit bien ici que l'essentiel de l'information repose sur le 1er facteur.

```
#représentation des individus
library(MASS)
eqscplot(acp$ind$coord[,1],acp$ind$coord[,2],type="n",xlim=c(-5,+5),main="ACP")
abline(h=0,col="gray",lty="dashed")
abline(v=0,col="gray",lty="dashed")
text(acp$ind$coord[,1],acp$ind$coord[,2],rownames(autos),cex=0.65)
```



3.3.3 Propriétés des facteurs

Les variances associées aux composantes correspondent à ce qu'on appelle communément les « valeurs propres ». Nous pouvons les calculer a posteriori à partir des coordonnées des individus dans le repère factoriel. Nous réutiliserons ce procédé plus loin pour apprécier les propriétés des « facteurs » mis en évidence par l'algorithme « GLRM ».

```
#variance calculée sur les facteurs
print(apply(acp$ind$coord,2,function(x){(m-1)/m*var(x)}))
##     Dim.1     Dim.2
## 4.4208581 0.8560623
```

Et, conformément à la théorie, les composantes de l'ACP sont orthogonales, leurs corrélations croisées sont nulles (aux erreurs de troncature près).

```
#orthogonalité
print(cor(acp$ind$coord))
##               Dim.1         Dim.2
## Dim.1  1.000000e+00 -5.265045e-17
## Dim.2 -5.265045e-17  1.000000e+00
```

3.3.4 Vecteurs propres

Les vecteurs propres correspondrait à la notion d'archétype pour l'ACP. Nous pouvons les déduire à partir des corrélations des variables aux axes, fournies par la très grande majorité des outils.

```
#vec. propres - dérivés des corrélations
vecp <- sapply(1:2,function(j){acp$var$cor[,j]/sqrt(acp$eig[,1][j])})
print(vecp)
##            [,1]       [,2]
## CYL   0.4249360  0.1241911
## PUISS 0.4217944  0.4157739
## LONG  0.4214599 -0.4118177
## LARG  0.3869222 -0.4460870
## POIDS 0.4305120 -0.2426758
## VMAX  0.3589443  0.6198626
```

Nous aurions pu aussi les calculer directement à partir de la matrice de corrélation (les signes diffèrent pour la première composante, mais ce n'est pas un problème, ce sont les positions relatives qui importent).

```
#calcul des val. et vec. propres à partir de la matrice des corrélations
print(eigen(cov(Z)))
## eigen() decomposition
## $values
## [1] 4.68090853 0.90641889 0.39501114 0.22650574 0.09826011 0.04583676
##
## $vectors
##            [,1]       [,2]        [,3]       [,4]       [,5]        [,6]
## [1,] -0.4249360  0.1241911 -0.35361252  0.8077865 -0.1515800 -0.05889517
## [2,] -0.4217944  0.4157739 -0.18492049 -0.3577920  0.2937346 -0.63303302
## [3,] -0.4214599 -0.4118177  0.06763394 -0.2797523 -0.7305690 -0.19029153
## [4,] -0.3869222 -0.4460870  0.60486812  0.2115694  0.4781901 -0.10956624
## [5,] -0.4305120 -0.2426758 -0.48439601 -0.3017114  0.3045584  0.58081220
## [6,] -0.3589443  0.6198626  0.48547226 -0.0735743 -0.1886551  0.45852167
```

Ces vecteurs propres sont normés comme nous pouvons le constater ci-dessous.

```
#vec.propres normées
print(apply(vecp,2,function(x){sum(x^2)}))
## [1] 1 1
```

Et ils sont orthogonaux (les deux premiers ici, mais ils le sont deux à deux si nous prenons tous les facteurs), leur produit scalaire est nul.

```
#vec.P orthogonaux (test pour les 2 premiers)
print(sum(vecp[,1]*vecp[,2]))
## [1] 1.526557e-16
```

3.3.5 Reconstitution des données

Mis en lumière dans la description de GLRM, moins dans les ouvrages consacrés à l'ACP, les méthodes de réduction de dimensionnalité s'apparentent à une compression avec perte que l'on peut quantifier avec la qualité de représentation associée aux facteurs. Il est possible de réaliser une rétro-ingénierie (*reverse engineering*) en revenant sur la représentation initiale. L'écart entre les données originelles et celles reproduites est le critère central de GLRM. Essayons de voir ce qu'il en est lorsque nous le calculons pour l'ACP.

Voici les données reconstituées dans l'espace des variables centrées et réduites, nous l'obtenons via le produit matriciel entre les coordonnées des individus et les vecteurs propres (Rakotomalala, 2020 ; section 1.4).

```
#reconstituer les données
ZHatAcp <- acp$ind$coord %*% t(vecp)
print(ZHatAcp)
##                       CYL       PUISS        LONG         LARG       POIDS
## Alfasud TI      -0.6871400 -0.15974647 -1.63684570 -1.624166203 -1.35417384
## Audi 100         0.4738752  0.02371103  1.28695448  1.285355868  1.04280237
## Simca 1300      -0.5594347 -0.75259209 -0.19400286 -0.132227072 -0.31822269
## Citroen GS Club -1.0796564 -1.03865558 -1.13121670 -1.046194062 -1.13542090
## Fiat 132         0.2681943  0.46966566 -0.10612313 -0.144736844  0.01539943
## Lancia Beta     -0.1536415 -0.20987928 -0.04744643 -0.030216888 -0.08337739
## Peugeot 504      0.1747485 -0.09946343  0.67249777  0.680851639  0.52086972
## Renault 16 TL   -0.9497476 -1.22950797 -0.41744566 -0.316549981 -0.60091837
## Renault 30       2.0059488  2.30223229  1.42050343  1.231751448  1.64032553
## Toyota Corolla  -1.6643636 -1.58295814 -1.77713557 -1.647571640 -1.77325693
## Alfetta 1.66     0.4234857  0.97974779 -0.60312467 -0.683778577 -0.27568782
## Princess 1800    0.3281263  0.07949877  0.77575209  0.769431524  0.64259987
## Datsun 200L      1.1803265  1.00804104  1.46982552  1.387409899  1.40186847
## Taunus 2000      0.6191618  0.75689253  0.35381082  0.291725119  0.44800453
## Rancho          -0.4051671 -0.66475603  0.07842183  0.133055484 -0.07967646
## Mazda 9295       0.2081365  0.31078209  0.01587771 -0.009649969  0.07961491
## Opel Rekord      0.9859637  1.00919522  0.92207459  0.839415468  0.96045074
## Lada 1300       -1.1688166 -1.20220742 -1.08237754 -0.983905212 -1.13120118
##                      VMAX
## Alfasud TI       0.3391225
## Audi 100        -0.3860785
## Simca 1300      -0.8198975
## Citroen GS Club -0.8538572
```

```
## Fiat 132         0.5847323
## Lancia Beta     -0.2307897
## Peugeot 504     -0.3328748
## Renault 16 TL   -1.3071433
## Renault 30       2.2421554
## Toyota Corolla  -1.2842372
## Alfetta 1.66     1.3425500
## Princess 1800   -0.1562776
## Datsun 200L      0.7090725
## Taunus 2000      0.7735459
## Rancho          -0.8045344
## Mazda 9295       0.3592335
## Opel Rekord      0.8865785
## Lada 1300       -1.0613005
```

Nous calculons la fonction de perte quadratique pour exprimer la qualité de la reconstitution.

```
#quadratic loss
QLAcp <- sum((as.matrix(Z)-ZHatAcp)^2)
print(QLAcp)
## [1] 13.01543
```

La valeur (**13.01543**) obtenue pour l'ACP constituera un repère clé pour évaluer les performances de l'algorithme GLRM qui, rappelons-le, cherche à minimiser explicitement ce critère durant son processus d'apprentissage.

3.4 Algorithme GLRM sur descripteurs quantitatifs

Nous utilisons l'implémentation de la librairie "H2O" que nous avons déjà exploré dans de précédents tutoriels. Elle est pour spécificité de proposer de nombreux algorithmes de machine learning et de deep learning, avec d'excellentes capacités de traitement grâce à la parallélisation et des techniques de compression de mémoire.

3.4.1 Analyse avec la méthode glrm

Il faut démarrer le "moteur" H20 dans un premier temps.

```
#h2o
library(h2o)
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
```

```
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## ----------------------------------------------------------------------
##
## Attaching package: 'h2o'
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
## The following objects are masked from 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
h2o.init()
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 22 minutes
##     H2O cluster timezone:       Europe/Paris
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.30.0.1
##     H2O cluster version age:    6 months and 24 days !!!
##     H2O cluster name:           H2O_started_from_R_Zatovo_eyf120
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.48 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4
##     R Version:                  R version 3.6.3 (2020-02-29)
## Warning in h2o.clusterInfo():
## Your H2O cluster version is too old (6 months and 24 days)!
## Please download and install the latest version from http://h2o.ai/download/
```

La version installée sur ma machine est assez ancienne (6 mois, c'est beaucoup de nos jours) mais, pour ce que nous souhaitons en faire, cela suffit amplement.

Il faut typer le dataset au format "H20" reconnu par la librairie.

```
#transtyper les données
DAutos <- as.h2o(Z)
##   |                                                                      |
|   0%  |
|======================================================================| 100%
print(DAutos)
```

```
##          CYL       PUISS       LONG       LARG        POIDS       VMAX
## 1 -0.7750989 -0.28335818 -1.8850808 -1.0973453 -1.569006764  0.5697604
## 2 -0.1201633  0.01963869  1.6058095  2.0010414  0.234161425  0.1459717
## 3 -0.9292014 -0.83885242 -0.4421794  0.2581989 -0.216630623 -0.5320903
## 4 -1.1273332 -1.29334771 -1.0007219 -1.0973453 -1.118214717 -0.6168481
## 5 -0.1284188  0.67613189  0.2559986 -0.5163978  0.196595421  0.5697604
## 6 -0.9209459 -0.13185975 -0.2094534  0.4518481  0.008765401  0.1459717
##
## [18 rows x 6 columns]
```

Nous pouvons lancer l'algorithme GLRM en spécifiant « k = 2 » facteurs. Les données étant déjà centrées et réduites, aucune transformation n'est demandée (`transform = NONE`).

```
#analyse
glrm <- h2o.glrm(DAutos,k=2,transform="NONE",seed=1)
##   |                                                                      |
|   0%  |
|======                                                                |   9%  |
|======================================================================| 100%
print(glrm)
## Model Details:
## ==============
##
## H2ODimReductionModel: glrm
## Model ID:  GLRM_model_R_1603893550087_5
## Model Summary:
##   number_of_iterations final_step_size final_objective_value
## 1                 1000         0.00825              19.04164
##
##
## H2ODimReductionMetrics: glrm
## ** Reported on training data. **
##
## Sum of Squared Error (Numeric):  19.04165
## Misclassification Error (Categorical):  0
## Number of Numeric Entries:  108
## Number of Categorical Entries:  0
```

Nous retenons avant tout la valeur finale de la fonction objectif (sum of squared error) = **19.04165**, moins bonne que celle de l'ACP alors que GLRM cherche à optimiser directement ce critère. J'ai eu du mal à le croire dans un premier temps. En scrutant la documentation, les paramètres par défaut indiquent bien une optimisation sans entraves particulières : « loss = Quadratic », « regularization_x = None », « regularization_y = None ».

Plus loin, comme pour l'ACP, nous essaierons de calculer ce même critère à partir des données reconstituées. Nous retrouverons effectivement la même valeur. Ce déficit de performances par rapport à l'ACP reste un mystère pour moi à ce jour.

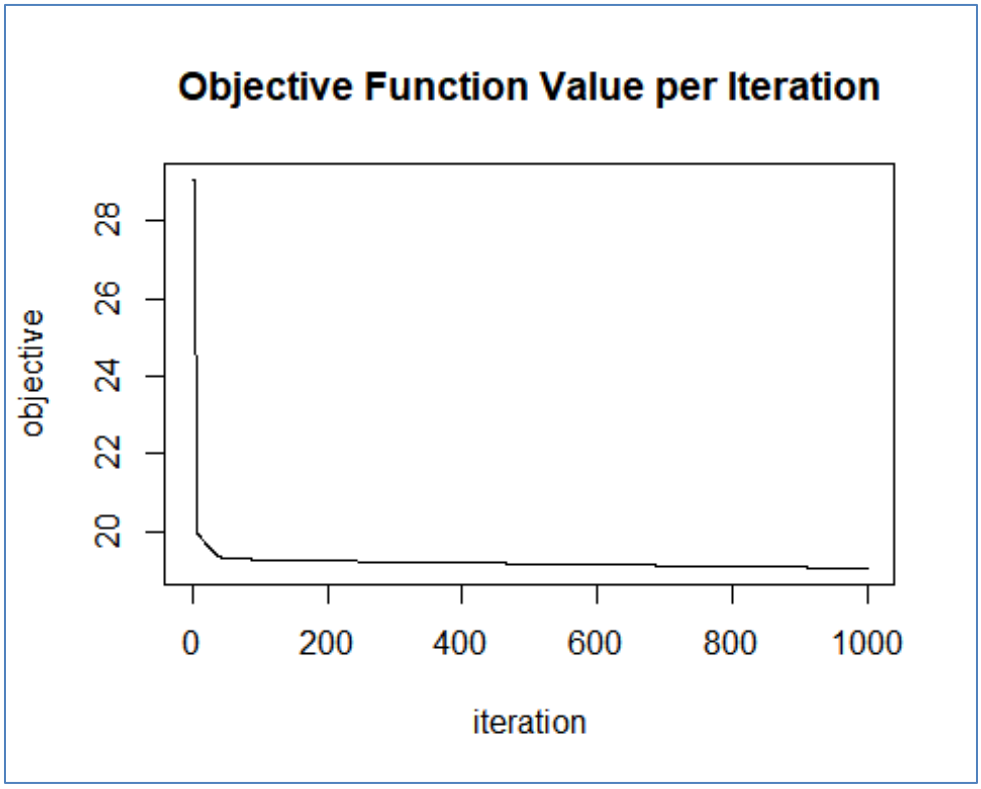
3.4.2 Inspection des résultats

Nous disposons de l'historique des résultats, principalement la décroissance du critère durant les 1000 itérations.

```
#historique du processus d'optimisation
summary(glrm)
## Model Details:
## ==============
##
## H2ODimReductionModel: glrm
## Model Key:  GLRM_model_R_1603893550087_5
## Model Summary:
##   number_of_iterations final_step_size final_objective_value
## 1                 1000         0.00825              19.04164
##
## H2ODimReductionMetrics: glrm
## ** Reported on training data. **
##
## Sum of Squared Error (Numeric):  19.04165
## Misclassification Error (Categorical):  0
## Number of Numeric Entries:  108
## Number of Categorical Entries:  0
##
##
##
## Scoring History:
##             timestamp   duration iterations step_size objective
## 1 2020-10-28 16:21:36  0.018 sec          0   0.66667  29.06913
## 2 2020-10-28 16:21:36  0.021 sec          1   0.44444  29.06913
## 3 2020-10-28 16:21:36  0.021 sec          2   0.22222  29.06913
## 4 2020-10-28 16:21:36  0.022 sec          3   0.07407  29.06913
## 5 2020-10-28 16:21:36  0.022 sec          4   0.01852  29.06913
##
## ---
##                timestamp   duration iterations step_size objective
## 995  2020-10-28 16:21:36  0.514 sec        994   0.00646  19.04324
## 996  2020-10-28 16:21:36  0.514 sec        995   0.00679  19.04273
## 997  2020-10-28 16:21:36  0.515 sec        996   0.00712  19.04242
## 998  2020-10-28 16:21:36  0.516 sec        997   0.00748  19.04215
## 999  2020-10-28 16:21:36  0.517 sec        998   0.00786  19.04190
## 1000 2020-10-28 16:21:36  0.517 sec        999   0.00825  19.04164
```

La même information sous la forme graphique montre que le gain est minime à partir de – à peu près – la 50ème itération.

```
#décroissance sous une forme graphique
plot(glrm)
```



Nous pouvons obtenir la liste des propriétés de l'objet "glrm" avec la commande `str()`. Les sorties étant trop verbeuses, nous n'affichons que la liste réduite ici avec `slotNames()`.

```
#liste des propriétés de l'objet
slotNames(glrm)
## [1] "model_id"      "algorithm"     "parameters"    "allparameters"
## [5] "have_pojo"     "have_mojo"     "model"
```

En particulier, les champs de la propriété "model" nous seront utiles par la suite.

```
#champs de @model
print(attributes(glrm@model))
## $names
##  [1] "names"
##  [2] "original_names"
##  [3] "column_types"
##  [4] "domains"
##  [5] "cross_validation_models"
##  [6] "cross_validation_predictions"
##  [7] "cross_validation_holdout_predictions_frame_id"
##  [8] "cross_validation_fold_assignment_frame_id"
##  [9] "model_summary"
## [10] "scoring_history"
## [11] "training_metrics"
## [12] "validation_metrics"
## [13] "cross_validation_metrics"
## [14] "cross_validation_metrics_summary"
```

```
## [15] "status"
## [16] "start_time"
## [17] "end_time"
## [18] "run_time"
## [19] "help"
## [20] "iterations"
## [21] "updates"
## [22] "objective"
## [23] "avg_change_obj"
## [24] "step_size"
## [25] "archetypes"
## [26] "singular_vals"
## [27] "eigenvectors"
## [28] "representation_name"
## [29] "importance"
```

Nous disposons au premier chef de l'importance des facteurs : le premier restituerait 73.68% de l' information disponible, le second 8.72%. Mais la documentation accessible en ligne n'est pas très diserte sur le mode de calcul de ces indicateurs. Il faudrait se plonger dans l'article originel. En tous les cas, ces valeurs ne correspondent pas aux calculs a posteriori que nous effectuerons plus bas à partir des coordonnées factorielles des individus (section 3.4.4).

```
#importance des facteurs
print(glrm@model$importance)
## Importance of components:
##                              pc1      pc2
## Standard deviation     2.163516 0.744395
## Proportion of Variance 0.736793 0.087223
## Cumulative Proportion  0.736793 0.824016
```

Pour l'heure, nous passons au carré les écarts-type pour obtenir les variances. Nous verrons s'il est possible de retrouver ces valeurs plus loin.

```
#variance
print(glrm@model$importance[1,]^2)
##                          pc1       pc2
## Standard deviation 4.680802 0.5541234
```

3.4.3 Coordonnées des individus dans le "plan factoriel"

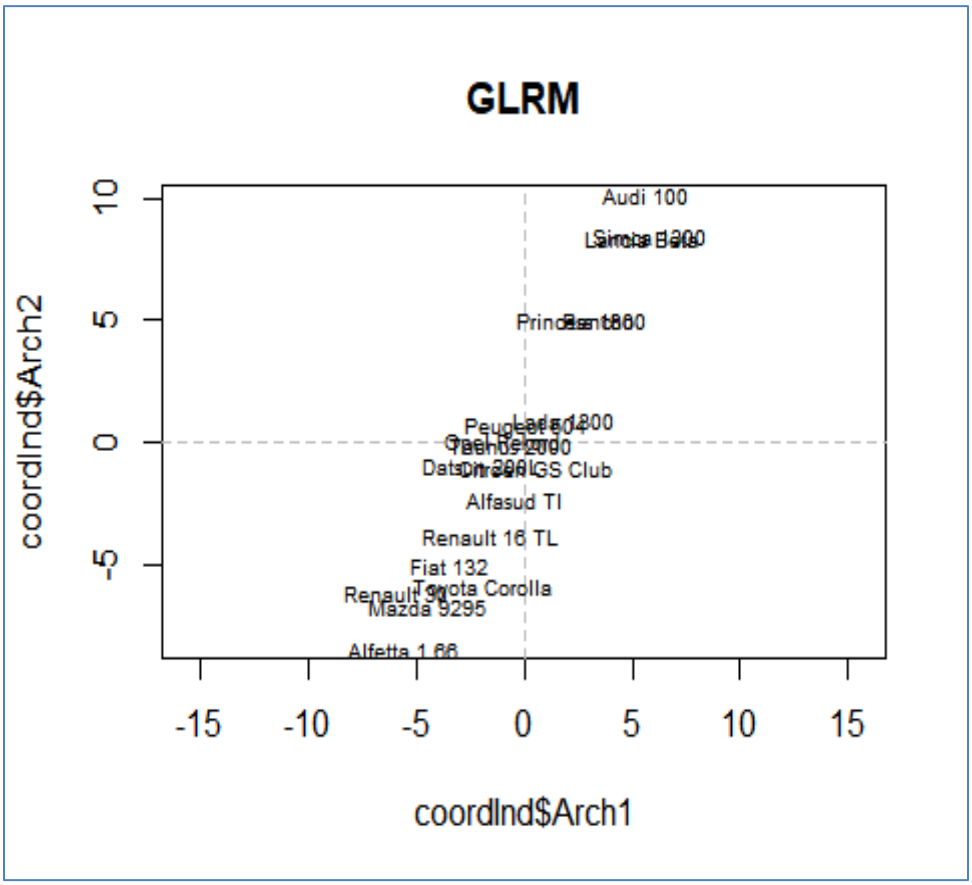
Les coordonnées des individus dans le plan factoriel…

```
#coordonnées "factorielles"
coordInd <- h2o.getFrame(glrm@model$representation_name)
coordInd <- as.data.frame(coordInd)
print(coordInd)
##          Arch1       Arch2
## 1  -0.44630767 -2.25792496
```

```
## 2   5.72470489 10.12024690
## 3   5.89157464  8.43976201
## 4   0.54376492 -0.99811948
## 5  -3.38738181 -5.01048988
## 6   5.47641076  8.36993766
## 7   0.09125017  0.64551856
## 8  -1.50811284 -3.75318246
## 9  -5.88649918 -6.07409665
## 10 -1.94594194 -5.91783248
## 11 -5.53911316 -8.39052918
## 12  2.74643337  5.04762904
## 13 -1.94793919 -0.93730788
## 14 -0.65240792 -0.08393689
## 15  3.50754153  5.00963436
## 16 -4.41958055 -6.65400772
## 17 -1.03542013  0.02445145
## 18  1.85918098  0.96671468
```

… nous permet de réaliser la représentation graphique.

```
#plot
eqscplot(coordInd$Arch1,coordInd$Arch2,type="n",main="GLRM")
abline(h=0,col="gray",lty="dashed")
abline(v=0,col="gray",lty="dashed")
text(coordInd$Arch1,coordInd$Arch2,rownames(autos),cex=0.65)
```



Certains voisinages interrogent. On est forcément perplexe quand on « voit » la proximité entre la « Renault 30 » et la « Toyota Corolla », deux objets a priori diamétralement opposés dans la réalité (grande voiture de luxe vs. petite voiture urbaine) et dans le repère de l'ACP (section 3.3.2). De plus, à l'évidence, les deux « composantes » sont fortement corrélées. Nous constatons également que, contrairement à ce qui est annoncé par la propriété « importance des facteurs », il semble que la dispersion soit plus prononcée sur le 2nd axe. Nous vérifierons par le calcul ces deux intuitions visuelles dans la section suivante.

3.4.4 Propriétés de la représentation - Coordonnées des individus

Les facteurs de GLRM ne sont pas centrés.

```
#coordonnées centrées ? ==> NON
print(colMeans(coordInd))
##       Arch1       Arch2
## -0.05154684 -0.08075183
```

Les variances calculées à partir des coordonnées des observations ne correspondent pas à l'importance des facteurs affichés par l'outil (section 3.4.2). Elles sont plus en phase avec les dispersions empiriquement constatées dans le graphique.

```
#variance calculée ==> différent de Importance
print(sapply(coordInd,function(x){((m-1)/m)*var(x)}))
##    Arch1    Arch2
## 12.71803 29.34877
```

Le repère n'est pas orthogonal. Les deux facteurs sont fortement corrélés, corroborant encore une fois l'analyse visuelle.

```
#repère orthogonal ? ==> NON
print(cor(coordInd))
##           Arch1     Arch2
## Arch1 1.0000000 0.9627944
## Arch2 0.9627944 1.0000000
```

3.4.5 Archétypes

Essayons de comprendre la nature des « facteurs » mis en évidence par GLRM en analysant la matrice des archétypes Y qui définissent le changement de base.

```
#archétype
coordVar <- as.matrix(glrm@model$archetypes)
print(coordVar)
```

```
##              cyl      puiss       long      larg      poids       vmax
## Arch1 -0.9323929 -0.9228577 -0.9177868 -0.835133 -0.9383570 -0.7855958
## Arch2  0.5430816  0.5424560  0.6205984  0.627411  0.6132031  0.4499682
```

Les vecteurs ne sont ni normés…

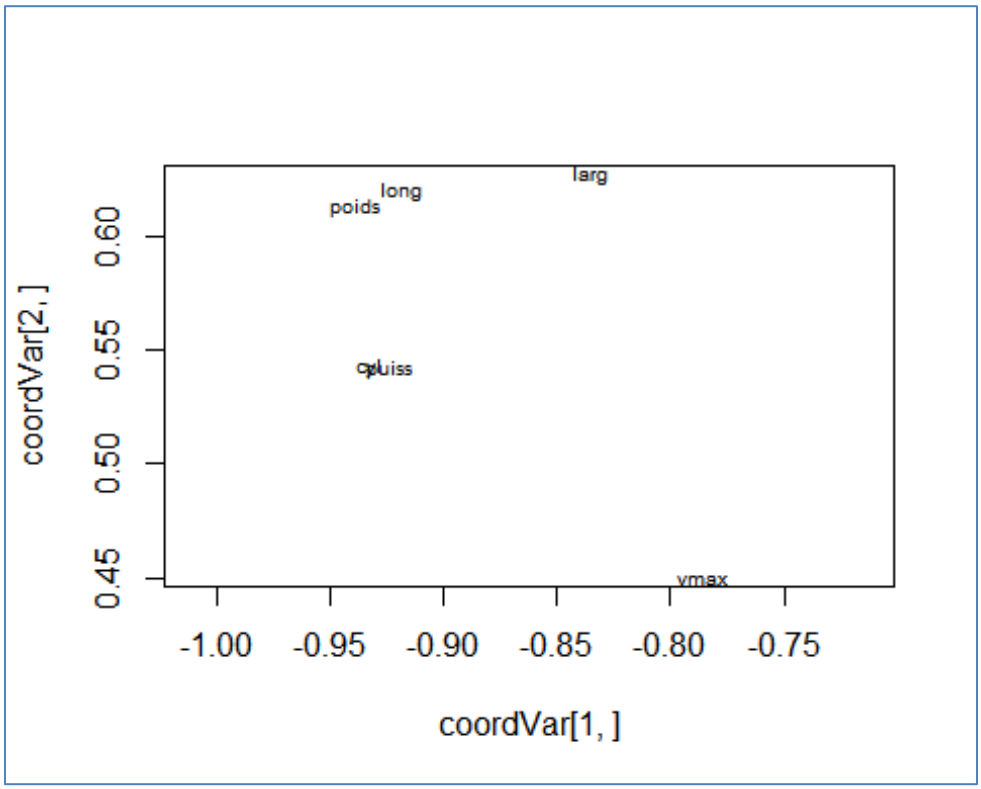
```
#archétype normée ? => NON
print(apply(coordVar,1,function(x){sum(x^2)}))
##    Arch1    Arch2
## 4.758477 1.946472
```

… ni orthogonales (on pouvait s'y attendre puisque les facteurs sont corrélés).

```
#archétype orthogonaux => NON
print(sum(coordVar[1,]*coordVar[2,]))
## [1] -3.02942
```

Nous affichons ci-dessous les coordonnées des variables dans le nouvel espace de représentation. Ce n'est pas un cercle des corrélations, mais les positions relatives peuvent être source de réflexion.

```
#coordonnées des variables
eqscplot(coordVar[1,],coordVar[2,],type="n")
text(coordVar[1,],coordVar[2,],colnames(coordVar),cex=0.65)
```



Si on calcule explicitement les corrélations à partir des coordonnées des individus. On ne voit pas vraiment le lien qu'il peut y avoir entre les vecteurs de Y et ces dernières. Mais ce n'est pas choquant en soi puisque la méthode GLRM est censée mettre en évidence des liaisons non-linéaires.

```
#corrélation des facteurs avec les var. initiales
correl_glrm <- sapply(coordInd,function(x){cor(x,autos)})
rownames(correl_glrm) <- colnames(autos)
print(correl_glrm)
##             Arch1       Arch2
## CYL   -0.49199483 -0.25881093
## PUISS -0.46133278 -0.22954573
## LONG  -0.03637426  0.21048890
## LARG   0.29338557  0.53064506
## POIDS -0.14811393  0.09998797
## VMAX  -0.45420640 -0.25927468
```

3.4.6 Reconstitution des données

Nous obtenons la matrice des données reconstituées en effectuant le produit matriciel entre X et Y.

```
#reconstitution des données - calcul
print(round(as.matrix(coordInd) %*% coordVar,4))
##           cyl   puiss    long    larg   poids    vmax
##  [1,] -0.8101 -0.8129 -0.9916 -1.0439 -0.9658 -0.6654
##  [2,]  0.1584  0.2067  1.0266  1.5687  0.8339  0.0565
##  [3,] -0.9098 -0.8589 -0.1695  0.3750 -0.3531 -0.8308
##  [4,] -1.0491 -1.0433 -1.1185 -1.0803 -1.1223 -0.8763
##  [5,]  0.4373  0.4081 -0.0006 -0.3147  0.1061  0.4066
##  [6,] -0.5606 -0.5136  0.1682  0.6779 -0.0064 -0.5360
##  [7,]  0.2655  0.2660  0.3169  0.3288  0.3102  0.2188
##  [8,] -0.6321 -0.6442 -0.9451 -1.0953 -0.8863 -0.5040
##  [9,]  2.1898  2.1375  1.6330  1.1051  1.7990  1.8913
## [10,] -1.3995 -1.4143 -1.8866 -2.0878 -1.8028 -1.1341
## [11,]  0.6079  0.5603 -0.1234 -0.6384  0.0526  0.5760
## [12,]  0.1805  0.2035  0.6119  0.8733  0.5181  0.1137
## [13,]  1.3072  1.2892  1.2061  1.0387  1.2531  1.1085
## [14,]  0.5627  0.5565  0.5467  0.4922  0.5607  0.4748
## [15,] -0.5498 -0.5195 -0.1102  0.2138 -0.2194 -0.5013
## [16,]  0.5071  0.4691 -0.0732 -0.4839  0.0669  0.4779
## [17,]  0.9787  0.9688  0.9655  0.8801  0.9866  0.8244
## [18,] -1.2085 -1.1914 -1.1064 -0.9461 -1.1518 -1.0256
```

Ou en utilisant la fonction `predict()` qui a été surchargée.

```
#coordoonées reconstituées - outil de la librairie
reconst <- as.matrix(predict(glrm,newdata=DAutos))
##   |                                                                      |
|   0%  |
|======================================================================| 100%
```

```
print(round(reconst,4))
##       reconstr_CYL reconstr_PUISS reconstr_LONG reconstr_LARG reconstr_POIDS
##  [1,]      -0.8101        -0.8129       -0.9916       -1.0439        -0.9658
##  [2,]       0.1584         0.2067        1.0266        1.5687         0.8339
##  [3,]      -0.9098        -0.8589       -0.1695        0.3750        -0.3531
##  [4,]      -1.0491        -1.0433       -1.1185       -1.0803        -1.1223
##  [5,]       0.4373         0.4081       -0.0006       -0.3147         0.1061
##  [6,]      -0.5606        -0.5136        0.1682        0.6779        -0.0064
##  [7,]       0.2655         0.2660        0.3169        0.3288         0.3102
##  [8,]      -0.6321        -0.6442       -0.9451       -1.0953        -0.8863
##  [9,]       2.1898         2.1375        1.6330        1.1051         1.7990
## [10,]      -1.3995        -1.4143       -1.8866       -2.0878        -1.8028
## [11,]       0.6079         0.5603       -0.1234       -0.6384         0.0526
## [12,]       0.1805         0.2035        0.6119        0.8733         0.5181
## [13,]       1.3072         1.2892        1.2061        1.0387         1.2531
## [14,]       0.5627         0.5565        0.5467        0.4922         0.5607
## [15,]      -0.5498        -0.5195       -0.1102        0.2138        -0.2194
## [16,]       0.5071         0.4691       -0.0732       -0.4839         0.0669
## [17,]       0.9787         0.9688        0.9655        0.8801         0.9866
## [18,]      -1.2085        -1.1914       -1.1064       -0.9461        -1.1518
##       reconstr_VMAX
##  [1,]       -0.6654
##  [2,]        0.0565
##  [3,]       -0.8308
##  [4,]       -0.8763
##  [5,]        0.4066
##  [6,]       -0.5360
##  [7,]        0.2188
##  [8,]       -0.5040
##  [9,]        1.8913
## [10,]       -1.1341
## [11,]        0.5760
## [12,]        0.1137
## [13,]        1.1085
## [14,]        0.4748
## [15,]       -0.5013
## [16,]        0.4779
## [17,]        0.8244
## [18,]       -1.0256
```

Nous pouvons dès lors calculer la fonction de perte quadratique en opposant données originelles et reconstituées. Nous retrouvons la valeur rapportée par l'outil. Et, encore une fois, elle est moins intéressante par rapport à celle de l'ACP.

```
#quadratic loss
QLglrm <- sum((reconst-as.matrix(Z))^2)
print(QLglrm)
## [1] 19.04164
```

3.5 Conclusion - Parallèle avec l'ACP

Je suis très mitigé à l'issue de ce premier comparatif. Etudier une nouvelle approche est toujours enthousiasmant. Mais j'avoue que la nature des résultats m'a laissé dubitatif. Je connais parfaitement ce fichier « autos ». Je ne retrouve pas les proximités entre les véhicules que je connais bien par ailleurs. Mais surtout, le fait que la qualité de restitution soit moindre par rapport à une « banale » ACP à taille d'espace de représentation égale est incompréhensible. GLRM est censée se focaliser sur cet aspect de la réduction de dimension.

4 Equivalence avec l'AFDM - Descripteurs mixtes

La capacité à traiter des bases composées de descripteurs mixtes était la première motivation de mon intérêt pour l'algorithme GLRM. En cela, il se positionne comme une alternative à l'AFDM, l'analyse factorielle des données mixtes (Rakotomalala, 2020 ; chapitre 6). Dans cette section, nous étudions les résultats fournis par les deux approches.

4.1 Données "Tennis"

Notre jeu de données recense les caractéristiques et performances sportives de fameux tennismen de ces dernières décennies (Rakotomalala, 2020 ; section 6.4.1). De nouveau, nous chargeons le fichier et nous en affichons le contenu.

```
#chargement des données "tennis"
tennis <- read.xlsx("Data_Methodes_Factorielles.xlsx",sheetIndex = 12)

#labels des lignes
rownames(tennis) <- tennis$Joueur
tennis$Joueur <- NULL
print(tennis)
##            Taille Lateralite MainsRevers Titres Finales TitresGC RolandGarros
## Agassi        180   droitier        deux     60      30        8    vainqueur
## Becker        191   droitier         une     49      28        6         demi
## Borg          180   droitier        deux     64      25       11    vainqueur
## Connors       178    gaucher        deux    109      52        8         demi
## Courier       185   droitier        deux     23      13        4    vainqueur
## Djokovic      188   droitier        deux     79      34       17    vainqueur
## Edberg        187   droitier         une     41      36        6       finale
## Federer       185   droitier         une    103      54       20    vainqueur
## Kafelnikov    190   droitier        deux     26      20        2    vainqueur
## Kuerten       190   droitier         une     20       9        3    vainqueur
## Lendl         187   droitier         une     94      50        8    vainqueur
## McEnroe       180    gaucher         une     77      31        7       finale
## Murray        191   droitier        deux     46      22        3       finale
```

```
## Nadal         185    gaucher        deux     85      37       19    vainqueur
## Nastase       180   droitier         une     58      38        2    vainqueur
## Rafter        185   droitier         une     11      14        2         demi
## Safin         193   droitier        deux     15      12        2         demi
## Sampras       185   droitier         une     64      24       14         demi
## Vilas         180    gaucher         une     62      40        4    vainqueur
## Wilander      182   droitier        deux     33      27        7    vainqueur
##            BestClassDouble
## Agassi                 123
## Becker                   6
## Borg                   890
## Connors                370
## Courier                 20
## Djokovic               114
## Edberg                   1
## Federer                 24
## Kafelnikov               4
## Kuerten                 38
## Lendl                   20
## McEnroe                  1
## Murray                  51
## Nadal                   26
## Nastase                 59
## Rafter                   6
## Safin                   71
## Sampras                 27
## Vilas                  175
## Wilander                 3
```

Voici la liste des variables et leurs types : certaines sont quantitatives (numériques), d'autres qualitatives (factor).

```
#liste des variables
str(tennis)
## 'data.frame':    20 obs. of  8 variables:
##  $ Taille         : num  180 191 180 178 185 188 187 185 190 190 ...
##  $ Lateralite     : Factor w/ 2 levels "droitier","gaucher": 1 1 1 2 1 1 1 1 ...
##  $ MainsRevers    : Factor w/ 2 levels "deux","une": 1 2 1 1 1 1 2 2 1 2 ...
##  $ Titres         : num  60 49 64 109 23 79 41 103 26 20 ...
##  $ Finales        : num  30 28 25 52 13 34 36 54 20 9 ...
##  $ TitresGC       : num  8 6 11 8 4 17 6 20 2 3 ...
##  $ RolandGarros   : Factor w/ 3 levels "demi","finale",..: 3 1 3 1 3 3 2 3 3 3 ...
##  $ BestClassDouble: num  123 6 890 370 20 114 1 24 4 38 ...
```

4.2 Analyse factorielle des données mixtes (AFDM)

L'AFDM s'applique aux bases composées de variables actives de nature différente. Elle constitue une véritable généralisation dans le sens où elle est équivalente à l'ACP normée lorsque les descripteurs sont tous quantitatifs, à l'ACM lorsqu'ils sont tous qualitatifs. Elle

est implémentée dans la librairie « FactoMineR » sous R. Elle est également disponible dans d'autres packages (ade4, pcamixdata).

```
#avec FactoMineR
library(FactoMineR)
afdm <- FAMD(tennis,ncp=k,graph=FALSE)
summary(afdm)
##
## Call:
## FAMD(base = tennis, ncp = k, graph = FALSE)
##
##
## Eigenvalues
##                        Dim.1  Dim.2
## Variance               3.101  1.526
## % of var.             34.454 16.958
## Cumulative % of var. 34.454 51.412
##
## Individuals (the 10 first)
##                      Dist    Dim.1    ctr   cos2    Dim.2    ctr   cos2
## Agassi           |  1.817 |  0.526  0.446  0.084 | -1.207  4.770  0.441 |
## Becker           |  2.540 | -1.320  2.809  0.270 |  1.089  3.887  0.184 |
## Borg             |  4.409 |  1.429  3.292  0.105 | -3.375 37.316  0.586 |
## Connors          |  4.345 |  3.220 16.714  0.549 | -0.426  0.595  0.010 |
## Courier          |  2.359 | -1.634  4.308  0.480 | -1.204  4.746  0.260 |
## Djokovic         |  2.432 |  0.926  1.383  0.145 | -0.895  2.623  0.135 |
## Edberg           |  2.820 | -0.705  0.802  0.063 |  1.923 12.120  0.465 |
## Federer          |  3.638 |  2.523 10.267  0.481 |  0.921  2.779  0.064 |
## Kafelnikov       |  2.471 | -1.908  5.871  0.596 | -0.863  2.440  0.122 |
## Kuerten          |  2.860 | -2.321  8.686  0.659 |  0.047  0.007  0.000 |
##
## Continuous variables
##                     Dim.1    ctr   cos2    Dim.2    ctr   cos2
## Taille           | -0.685 15.150  0.470 |  0.153  1.533  0.023 |
## Titres           |  0.934 28.119  0.872 |  0.143  1.341  0.020 |
## Finales          |  0.860 23.840  0.739 |  0.283  5.233  0.080 |
## TitresGC         |  0.671 14.520  0.450 | -0.066  0.288  0.004 |
## BestClassDouble |  0.374  4.501  0.140 | -0.665 28.935  0.442 |
##
## Categories
##                     Dim.1    ctr   cos2 v.test    Dim.2    ctr   cos2 v.test
## droitier         | -0.527  2.310  0.657 -2.608 | -0.125  0.540  0.037 -0.885 |
## gaucher          |  2.107  9.238  0.657  2.608 |  0.501  2.159  0.037  0.885 |
## deux             | -0.022  0.002  0.000 -0.054 | -0.940 18.971  0.734 -3.317 |
## une              |  0.022  0.002  0.000  0.054 |  0.940 18.971  0.734  3.317 |
## demi             | -0.690  1.239  0.144 -0.986 |  0.269  0.778  0.022  0.549 |
## finale           | -0.354  0.196  0.020 -0.368 |  1.525 14.977  0.370  2.260 |
## vainqueur        |  0.376  0.883  0.175  1.140 | -0.493  6.273  0.300 -2.132 |
```

L'importance des facteurs (51% de la variance restituée sur les 2 premiers) est relativisée ici puisque nous avons de l'information redondante lors du codage disjonctif complet des

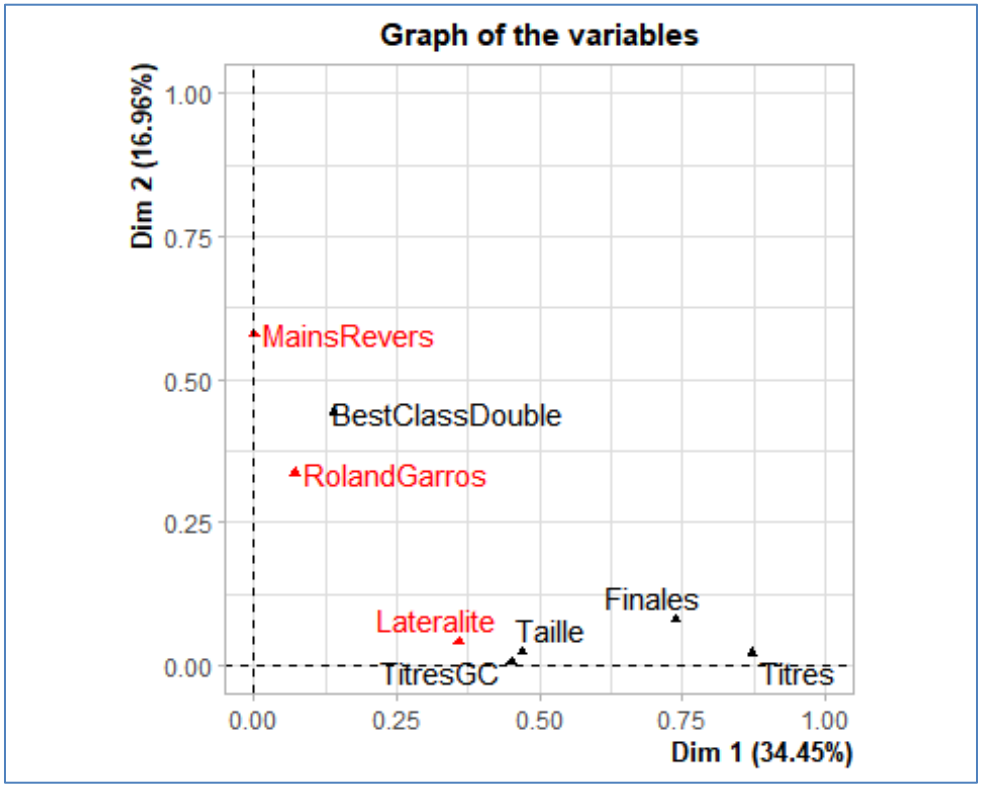
variables qualitatives. Il faudrait introduire des corrections similaires à celles de Benzécri/Greenacre en vigueur sous l'ACM (Rakotomalala, 2020 ; section 5.3).

4.3 Inspection des résultats de l'AFDM

4.3.1 Influence des variables

La carte des variables traduit le carré du rapport de corrélation (variables qualitatives) ou de la corrélation (quantitatives) avec les axes. Elle montre que le 1er facteur est principalement déterminé par le nombre de titres, le second par le type de revers utilisé.

```
#influence des variables
plot(afdm,choix="var")
```



4.3.2 Position des individus

Les coordonnées des individus oppose notamment Connors (multi-titré, carrière à rallonge) à Safin (carrière écourtée par les blessures, peu de titres, mais talent inoubliable) sur le premier facteur. Sur le second, nous observons l'emblématique affrontement entre McEnroe (revers à une main, mais quelle main, joueur de double exceptionnel) et Borg

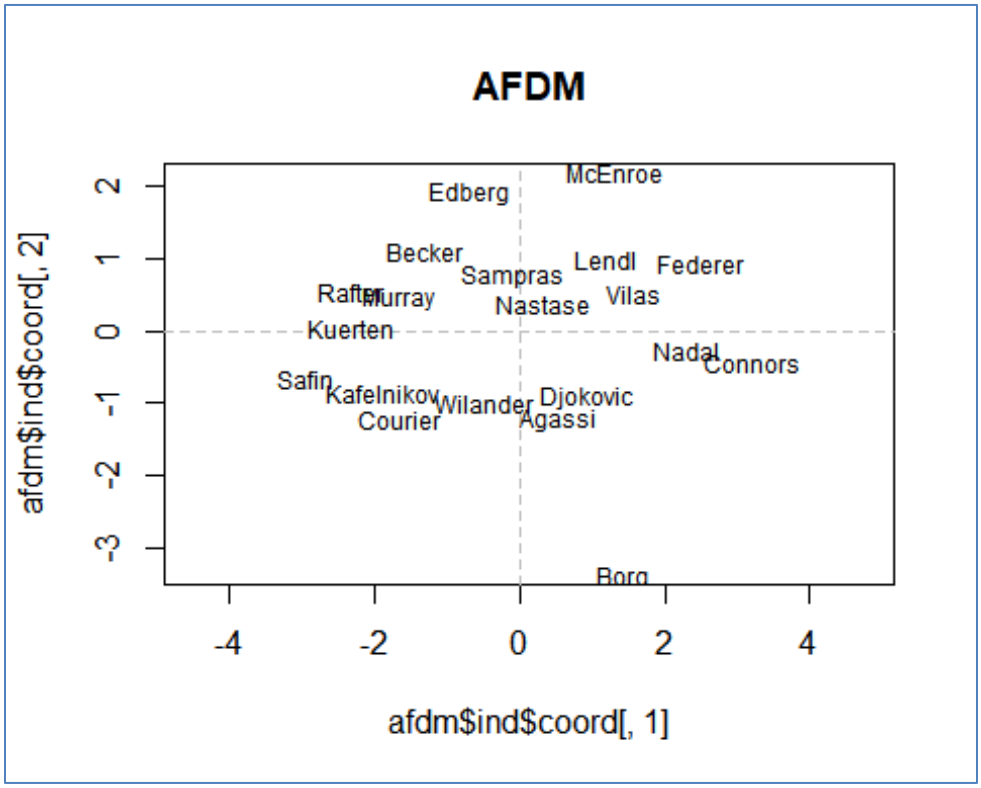
(revers à demain, qui n'a jamais cherché à réaliser quoique ce soit en double même s'il se débrouille plutôt bien dans la discipline comme il a pu le montrer en Coupe Davis).

```
#coordonnées des individus
print(afdm$ind$coord)
##                  Dim.1       Dim.2
## Agassi      0.52585827 -1.20665530
## Becker     -1.31987169  1.08927590
## Borg        1.42878437 -3.37499593
## Connors     3.21960536 -0.42612101
## Courier    -1.63446693 -1.20365108
## Djokovic    0.92607495 -0.89483701
## Edberg     -0.70540858  1.92341872
## Federer     2.52332109  0.92102805
## Kafelnikov -1.90814972 -0.86306701
## Kuerten    -2.32090678  0.04668077
## Lendl       1.20397486  0.99544850
## McEnroe     1.30948565  2.18357157
## Murray     -1.66695490  0.46825330
## Nadal       2.31012654 -0.27290963
## Nastase     0.33097816  0.39028168
## Rafter     -2.29997364  0.55429599
## Safin      -2.95604496 -0.64680237
## Sampras    -0.09479422  0.77596028
## Vilas       1.59067870  0.52121480
## Wilander   -0.46231653 -0.98039021
```

Puisque nous nous en tenons à (k = 2) dimensions, il est naturel de représenter graphiquement les individus dans le plan factoriel.

```
#individus dans le 1er plan
eqscplot(afdm$ind$coord[,1],afdm$ind$coord[,2],type="n",main="AFDM")
abline(h=0,col="gray",lty="dashed")
abline(v=0,col="gray",lty="dashed")
text(afdm$ind$coord[,1],afdm$ind$coord[,2],rownames(tennis),cex=0.75)
```

Un fan de tennis pourrait deviser longuement sur ce graphique.



4.3.3 Propriétés des facteurs

Les facteurs de l'AFDM sont centrés….

```
#centrés ? ==> OUI
print(colMeans(afdm$ind$coord))
##        Dim.1        Dim.2
## 5.134781e-17 7.216450e-17
```

… et ils sont orthogonaux.

```
#orthogonaux ==> OUI
print(cor(afdm$ind$coord))
##              Dim.1        Dim.2
## Dim.1 1.000000e+00 4.872748e-17
## Dim.2 4.872748e-17 1.000000e+00
```

4.4 GLRM sur données mixtes

Voyons comment se comporte l'algorithme GLRM sur les données « tennis ».

4.4.1 Préparation des données

Il n'est pas nécessaire de préparer spécifiquement le dataset (pas de codage disjonctif préalable des qualitatives en particulier). Il s'agit simplement de le typer au format « H2O ».

```
#transtypage au format H2O
DTennis <- as.h2o(tennis)
##   |                                                                      |
|   0%  |
|=================================================================| 100%
print(DTennis)
##   Taille Lateralite MainsRevers Titres Finales TitresGC RolandGarros
## 1    180   droitier        deux     60      30        8    vainqueur
## 2    191   droitier         une     49      28        6         demi
## 3    180   droitier        deux     64      25       11    vainqueur
## 4    178    gaucher        deux    109      52        8         demi
## 5    185   droitier        deux     23      13        4    vainqueur
## 6    188   droitier        deux     79      34       17    vainqueur
##   BestClassDouble
## 1             123
## 2               6
## 3             890
## 4             370
## 5              20
## 6             114
##
## [20 rows x 8 columns]
```

4.4.2 Lancement des calculs

Nous faisons de nouveau appel à la fonction `glrm()` en demandant à ce que les variables soient standardisées. L'idée est de ramener les variables quantitatives sur la même échelle. On ne sait pas en revanche comment cette opération pourrait affecter les qualitatives. La documentation ne dit rien à ce sujet.

```
#demander la projection dans le plan
glrm.tennis <- h2o.glrm(DTennis,k=k,transform = "STANDARDIZE",seed=1)
##   |                                                                      |
|   0%  |
|===========                                                      |  16%  |
|=================================================================| 100%
print(glrm.tennis)
## Model Details:
## ==============
##
## H2ODimReductionModel: glrm
## Model ID:  GLRM_model_R_1603893550087_7
## Model Summary:
##   number_of_iterations final_step_size final_objective_value
## 1                  160         0.00008              85.74740
##
##
## H2ODimReductionMetrics: glrm
## ** Reported on training data. **
##
## Sum of Squared Error (Numeric):  41.98717
## Misclassification Error (Categorical):  10
```

```
## Number of Numeric Entries:  100
## Number of Categorical Entries:  60
```

La qualité de représentation est globalement quantifiée à 85.74. Mais elle est en réalité décomposée – en simplifiant vraiment énormément – selon le type de variables : la somme des carrés des erreurs pour les variables quantitatives (41.98) et le nombre d'erreur pour les qualitatives (10).

4.5 Inspection des résultats

4.5.1 Importance des facteurs

L'importance des facteurs indique une plus forte dispersion sur le 1er facteur. Etrangement, la somme des proportions cumulée est égale à 100% sur les 2 premiers facteurs. On sait que ce n'est pas vrai. Pour l'AFDM par exemple, le nombre maximal de facteurs est égal à l'addition du nombre de variables quantitatives, du nombre total de modalités, auxquels nous retranchons le nombre de variables qualitatives (AFDM, page 15).

```
#importance des facteurs
print(glrm.tennis@model$importance)
## Importance of components:
##                             pc1      pc2
## Standard deviation     2.420102 1.516904
## Proportion of Variance 0.717942 0.282058
## Cumulative Proportion  0.717942 1.000000
```

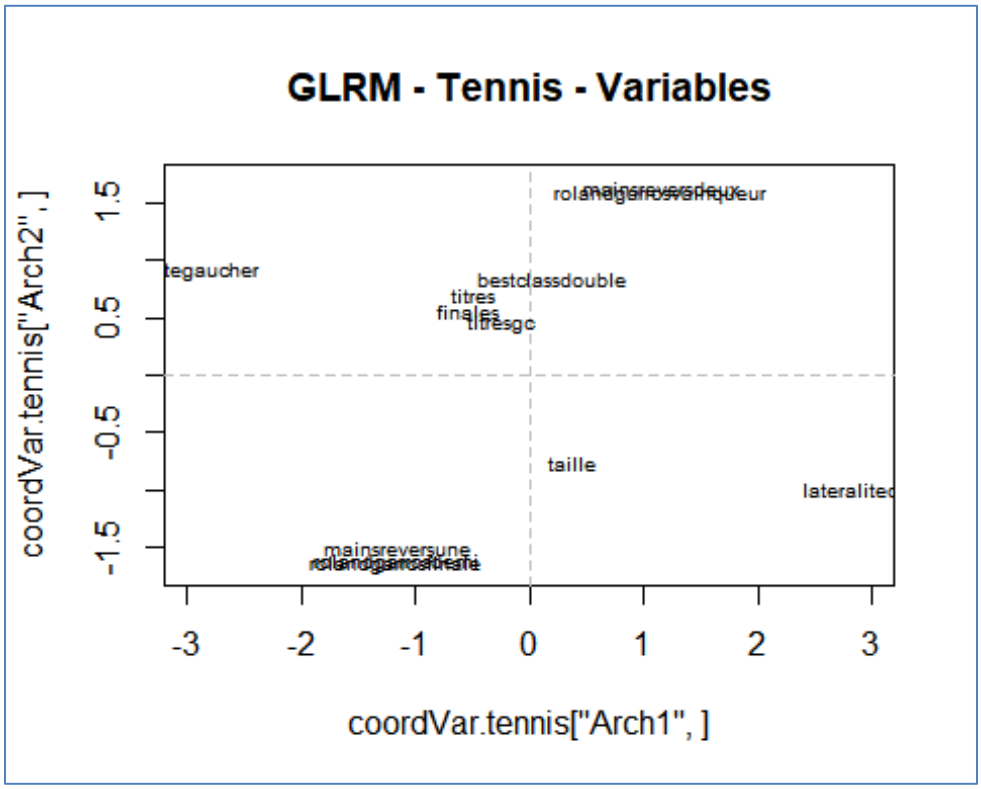
4.5.2 Archétypes - Influence des variables

Les archétypes (matrice Y) montrent que les variables qualitatives ont bien été binarisées en interne via un codage disjonctif complet.

```
#archétype
coordVar.tennis <- as.matrix(glrm.tennis@model$archetypes)
print(coordVar.tennis)
##       rolandgarrosdemi rolandgarrosfinale rolandgarrosvainqueur mainsreversdeux
## Arch1        -1.157286          -1.157229               1.15347        1.149424
## Arch2        -1.573070          -1.572856               1.57786        1.569250
##       mainsreversune lateralitedroitier lateralitegaucher     taille     titres
## Arch1      -1.149435          3.0688692        -3.0685962  0.3701727 -0.4920181
## Arch2      -1.569249         -0.9526472         0.9525322 -0.7413257  0.7496287
##          finales   titresgc bestclassdouble
## Arch1 -0.5222672 -0.2496858       0.2018273
## Arch2  0.6062768  0.5217433       0.8822225
```

Nous pouvons positionner les points-modalités et variables dans le plan. Certes, les mélanger dans le même repère de représentation interroge toujours.

```
#
eqscplot(coordVar.tennis['Arch1',],coordVar.tennis['Arch2',],type="n",main="GLRM - Tennis - Variables")
abline(h=0,col="gray",lty="dashed")
abline(v=0,col="gray",lty="dashed")
text(coordVar.tennis['Arch1',],jitter(coordVar.tennis['Arch2',],100),colnames(coordVar.tennis),cex=0.65)
```



Sur le premier facteur, nous retrouvons l'opposition gauchers vs. droitiers. Le nombre de titres n'a pas d'impact ici en revanche. Sur le second, c'est le type de revers et le fait d'avoir gagné Roland-Garros qui semble primer.

4.5.3 Position des individus

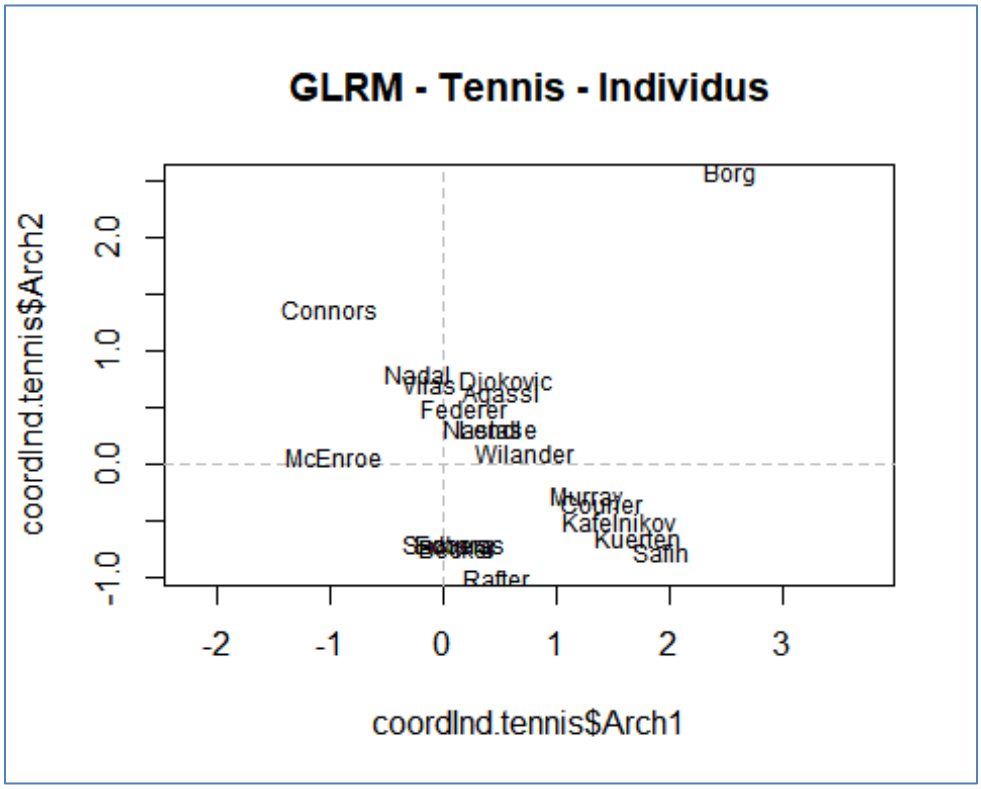
De nouveau, nous extrayons les coordonnées factorielles des individus.

```
#coordonnées factorielles
coordInd.tennis <- as.data.frame(h2o.getFrame(glrm.tennis@model$representation_name))
rownames(coordInd.tennis) <- rownames(tennis)
print(coordInd.tennis[order(coordInd.tennis$Arch1),])
##                 Arch1       Arch2
## Connors    -1.0054274  1.37370320
## McEnroe    -0.9630684  0.06818138
## Nadal      -0.2232376  0.80082233
## Vilas      -0.1052832  0.71073536
## Edberg      0.1043353 -0.71367972
## Sampras     0.1046521 -0.71265498
## Becker      0.1233785 -0.72761855
```

```
## Federer     0.1933290  0.49355825
## Nastase     0.4259489  0.32234794
## Lendl       0.4259580  0.32239096
## Rafter      0.4857189 -0.99301351
## Agassi      0.5214868  0.63015650
## Djokovic    0.5495121  0.72033980
## Wilander    0.7273380  0.10450041
## Murray      1.2618651 -0.29263925
## Courier     1.4040212 -0.33475528
## Kafelnikov  1.5451974 -0.49453406
## Kuerten     1.7293130 -0.63658129
## Safin       1.9181135 -0.77403488
## Borg        2.5238123  2.56071489
```

Nous formons la représentation graphique dans le plan.

```
#position des individus dans le plan
eqscplot(coordInd.tennis$Arch1,coordInd.tennis$Arch2,type="n",main="GLRM - Tennis - Individus")
abline(h=0,col="gray",lty="dashed")
abline(v=0,col="gray",lty="dashed")
text(coordInd.tennis$Arch1,coordInd.tennis$Arch2,rownames(tennis),cex=0.75)
```



En conformité avec l'archétype des points variables et modalités, le groupe des gauchers se démarque (relativement aux autres, pas par rapport à l'origine de l'espace de représentation puisque les facteurs ne sont pas centrés) à l'ouest (Connors, McEnroe, Nadal,

Vilas). Sur le 2nd, Borg se démarque vraiment au nord parce qu'il a gagné Roland-Garros et qu'il a un revers à deux mains. Etrangement, Nadal, qui présente les mêmes caractéristiques, est tout à fait anonyme en revanche.

4.5.4 Propriétés des facteurs

De nouveau ici, nous constatons que les facteurs ne sont ni centrés…

```
#facteurs centrés ? ==> NON
print(colMeans(coordInd.tennis))
##     Arch1     Arch2
## 0.5873482 0.1213970
```

… ni orthogonaux.

```
#orthogonaux ? ==> NON
print(cor(coordInd.tennis))
##             Arch1       Arch2
## Arch1  1.00000000 -0.02644519
## Arch2 -0.02644519  1.00000000
```

5 Conclusion

Très honnêtement, j'en attendais nettement plus de la méthode GLRM. Sur nos fichiers exemples, elle ne se distingue pas du tout. Les corrélations entre les facteurs rendent les interprétations difficiles. Les méthodes factorielles usuelles, basées sur la préservation des distances entre individus, ont encore de beaux jours devant elles.

Après, pour relativiser ce constat, il faut reconnaître que nous nous sommes placés dans un cadre pédagogique très particulier dans ce tutoriel, avec des petits fichiers comportant un nombre réduit de variables. On n'est pas vraiment dans le « big data » pour le coup. Il faudrait analyser le comportement de l'algorithme sur un fichier plus représentatif de la fouille de données massives avec des bases composées de plusieurs milliers - voire de dizaines de milliers - de variables. Peut-être que les qualités de l'approche s'y démarqueraient davantage. Autre point que nous n'avons pas abordé dans ce document, GLRM sait gérer nativement le problème des données manquantes. Un atout fort à son actif dans l'appréhension des problèmes réels.

6 Références

(Boehmke et Greenwell, 2020) Boehmke B., Greenwell B., « Hands-On Machine Learning with R », CRC Press, 2020 ; https://bradleyboehmke.github.io/HOML/

(Rakotomalala, 2020) Rakotomalala R., « Pratique des Méthodes Factorielles avec Python », juillet 2020 ; http://tutoriels-data-mining.blogspot.com/2020/07/pratiques-des-methodes-factorielles.html