Cours / TP d'analyse factorielle discriminante

Contents

1 Rappel sur l'ACP

L'ACP cherche les **composantes principales** C_1, C_2, ..., C_d expliquant aux mieux la **variance du nuage de points** défini par les variables X_1, X_2, ..., X_d.

La **première composante principale** C_1 est définie par :

$$C_1 = a_1X_1 + a_2X_2 + \cdots + a_pX_p$$

On cherche les **coefficients** a_1, ..., a_d, minimisant la variance de C_1.

La variance de C_1 s'écrit :

$$V(C_1) = V(a_1X_1 + a_2X_2 + \cdots + a_pX_p)$$

Il s'agit de la **variance d'une combinaison linéaire** de variables.

En notant $a = (a_1, \dots, a_d)^T$ et $X = (X_1, \dots, X_d)^T$. On a :

$$V(C) = V(a^TX) = a^T.V(X).a$$

où $V = V(X)$ est la matrice de variance covariance du vecteur aléatoire X. L'entrée ij de la matrice est

$$V_{ij} = \text{cov}(X_i, X_j)$$

Exemple sur les données `iris` :

```r
data(iris)
names(iris) = c("X1","X2","X3","X4","Y")
```

La matrice de variance covariance est :

```r
cov(iris[,1:4])
```

```
##            X1         X2         X3         X4
## X1  0.6856935 -0.0424340  1.2743154  0.5162707
## X2 -0.0424340  0.1899794 -0.3296564 -0.1216394
## X3  1.2743154 -0.3296564  3.1162779  1.2956094
## X4  0.5162707 -0.1216394  1.2956094  0.5810063
```

en fait plus précisément la fonction `cov` calcule la variance débiaisée (en divisant par $n-1$), là où la présentation dans le cours est en divisant par n.

On peut écrire une fonction `cov.p` pour régler ce problème :

```r
cov.p <- function(x){
  cov(x) * (nrow(x) - 1)/nrow(x)
}
```

de même on peut écrire une fonction `var.p`

```r
var.p <- function(x){
  var(x) * (length(x) - 1)/length(x)
}
```

Et donc :

```r
V = cov.p(iris[-5])
V
```

```
##             X1          X2         X3         X4
## X1  0.68112222 -0.04215111  1.2658200  0.5128289
## X2 -0.04215111  0.18871289 -0.3274587 -0.1208284
## X3  1.26582000 -0.32745867  3.0955027  1.2869720
## X4  0.51282889 -0.12082844  1.2869720  0.5771329
```

Ainsi $\text{cov}(X_1, X_2) = -0,042$.

Par exemple on peut s'intéresser à la variance de $3X_1 + 2X_2 + X_3 + X_4$

```r
a = matrix(c(3, 2, 1, 1))
a
```

```
##      [,1]
## [1,]    3
## [2,]    2
## [3,]    1
## [4,]    1
```

```r
composante = as.matrix(iris[-5]) %*% a
var.p(composante)
```

```
##          [,1]
## [1,] 21.50446
```

Dont la variance est égale à $21,5$. Qu'on peut retrouver comme suit :

```r
t(a) %*% V %*% a
```

```
##          [,1]
## [1,] 21.50446
```

C_1 se trouve en résolvant le problème de maximisation :

$$a = \arg\max_a V(a^TX) = a^T.V.a$$

sous la contrainte $\|a\| = 1$.

On montre que la solution de ce problème consiste à résoudre :

$$Va = \lambda_1 a$$

avec λ_1 la plus grande valeur propre de la matrice V. Ainsi on trouve a comme vecteur propre associé à la plus grande valeur propre de V.

Le principe est le même pour les valeurs propres suivantes.

Dans R la diagonalisation se fait simplement à l'aide de la fonction `eigen`

```r
res= eigen(V)
res
```

```
## eigen() decomposition
## $values
## [1] 4.20005343 0.24105294 0.07768810 0.02367619
##
## $vectors
##             [,1]        [,2]        [,3]       [,4]
## [1,]  0.36138659 -0.65658877 -0.58202985  0.3154872
## [2,] -0.08452251 -0.73016143  0.59791083 -0.3197231
## [3,]  0.85667061  0.17337266  0.07623608 -0.4798390
## [4,]  0.35828920  0.07548102  0.54583143  0.7536574
```

Le champ `values` nous donnes les valeurs propres et le champ `vectors` les vecteurs propres en colonne.

Ainsi la combinaison linéaire de plus forte variance est :

$$C_1 = 0.3614 \times X_1 - 0.0845 \times X_2 + 0.8567 \times X_3 + 0.3583 \times X_4$$

L'ensemble des composante principales peut simplement s'obtenir à partir du produit matriciel entre le tableau de donnée et la matrice des vecteurs propres :

```r
C = as.matrix(iris[-5]) %*% res$vectors
head(C)
```

```
##          [,1]      [,2]       [,3]        [,4]
## [1,] 2.818240 -5.646350 -0.6597675 -0.03108928
## [2,] 2.788223 -5.149951 -0.8423170  0.06567484
## [3,] 2.613375 -5.182003 -0.6139525 -0.01338332
## [4,] 2.757022 -5.008654 -0.6002933 -0.10892753
## [5,] 2.773649 -5.653707 -0.5417735 -0.09461031
## [6,] 3.221505 -6.068283 -0.4631751 -0.05755257
```

La variance de chacun des composantes principales s'obtient comme suit :

```r
apply(C, 2, var.p)
```

```
## [1] 4.20005343 0.24105294 0.07768810 0.02367619
```

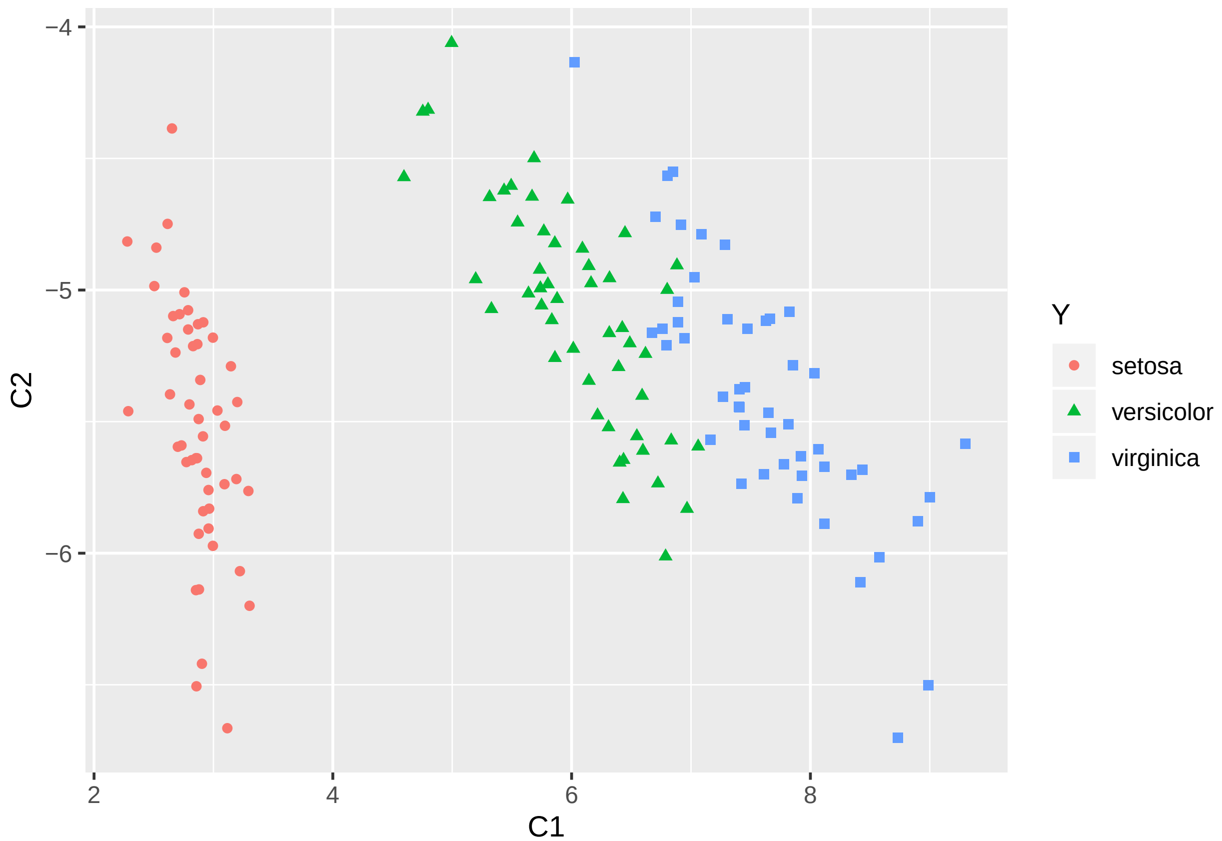
et on retrouve justement les valeurs propres de V ...

```r
res$values
```

```
## [1] 4.20005343 0.24105294 0.07768810 0.02367619
```

Le nuage de points est le suivant :

```r
library(ggplot2)
colnames(C) = paste0("C",1:4)
ACP = cbind.data.frame(C,Y = iris$Y)
ggplot(ACP, aes(x = C1, y = C2, color = Y, shape = Y)) + geom_point()
```



Globalement les classes sont bien séparées, cependant on doit pouvoir faire mieux. Aussi le $R^2_{C_1/Y}$ est égal à :

```r
summary(lm(C1 ~ Y, data = ACP))$r.squared
```

```
## [1] 0.9297843
```

Ce qui est moins bien que $R^2_{X_3/Y} = 94,1\%$. On doit pouvoir trouver des combinaisons linéaires qui font mieux ...

Remarque : cela sort du cadre du cours mais idéalement on utiliserait plutôt le package `FactoMineR` pour faire l'ACP.

2 L'analyse factorielle discriminante (AFD)

Ici on cherche aussi des combinaisons linéaires des variables $X_1, \ldots, X_d$ comme en ACP mais on cherche les composantes pour lesquelles les **classes sont les mieux séparées**.

2.1 Décomposition de la variance

Les effectifs des classes sont obtenus comme suit :

```r
ni = table(iris$Y)
ni
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

La matrice G des moyennes dans chacune des classes s'obtient par :

```r
G = t(simplify2array(by(iris[,1:4],iris$Y, colMeans)))
G
```

```
##               X1    X2    X3    X4
## setosa     5.006 3.428 1.462 0.246
## versicolor 5.936 2.770 4.260 1.326
## virginica  6.588 2.974 5.552 2.026
```

On en déduit la matrice de variance inter-classe B qui est la matrice de covariance de la matrice G en **pondérant** chaque ligne par l'effectif de la classe. Ce qui peut être fait facilement à partir de la fonction `cov.wt`

```r
B = cov.wt(G, wt = as.vector(ni), method = "ML")$cov
```

`wt` sont les poids à donner à chacune des ligne. `method = "ML"` précise qu'il utilise l'estimateur du maximum de vraisemblance (Maximum Likelihood) qui correspond à diviser par l'effectif n (en non pas $n-1$ qui correspondrait à l'option "unbiaised").

La variance intra-classes W se calcule comme moyenne pondérée des variances intra-classe groupe par groupe :

```r
Wi = by(iris[-5], iris$Y, cov.p)
Wi
```

```
## iris$Y: setosa
##          X1       X2       X3       X4
## X1 0.121764 0.097232 0.016028 0.010124
## X2 0.097232 0.140816 0.011464 0.009112
## X3 0.016028 0.011464 0.029556 0.005948
## X4 0.010124 0.009112 0.005948 0.010884
## ------------------------------------------------------------
## iris$Y: versicolor
##          X1      X2      X3       X4
## X1 0.261104 0.08348 0.17924 0.054664
## X2 0.083480 0.09650 0.08100 0.040380
## X3 0.179240 0.08100 0.21640 0.071640
## X4 0.054664 0.04038 0.07164 0.038324
## ------------------------------------------------------------
## iris$Y: virginica
##          X1       X2       X3       X4
## X1 0.396256 0.091888 0.297224 0.048112
## X2 0.091888 0.101924 0.069952 0.046676
## X3 0.297224 0.069952 0.298496 0.047848
## X4 0.048112 0.046676 0.047848 0.073924
```

donnes les matrices de variance covariance dans chacun des groupes.

Et on en déduit $W = \frac{n_1 W_1 + n_2 W_2 + n_3 W3}{n}$:

```r
W = Reduce('+',Map('*',Wi,ni))/sum(ni)
W
```

```
##            X1         X2         X3         X4
## X1 0.25970800 0.09086667 0.16416400 0.03763333
## X2 0.09086667 0.11308000 0.05413867 0.03205600
## X3 0.16416400 0.05413867 0.18148400 0.04181200
## X4 0.03763333 0.03205600 0.04181200 0.04104400
```

Informatiquement la fonction `Map` permet de multiplier chaque élément de la liste `Wi` par chaque élément du vecteur `ni`. La fonction réduce permet d'en faire la somme, et enfin on divise par l'effectif total.

On vérifie que $V = B + W$:

```r
norm(V - (B+W))
```

```
## [1] 2.275957e-15
```

Ainsi l'écart entre la matrice `V` et la matrice `B + W` est très faible 10^{-15}.

2.2 But de l'AFD

Le but de l'AFD est de trouver les composantes discrimantes $D_1, \ldots$ la première composante discriminante est recherchée de sorte à maximiser la part de la variance de de la composante expliquée par Y ($R^2_{D_1/Y}$).

$$\arg\max_a R^2_{D_1/Y} = \arg\max_a \frac{V_Y[E(D_1|Y)]}{V(D_1)} = \arg\max_a \frac{V_Y[E(a^TX|Y)]}{V(a^TX)} = \arg\max_a \frac{a^T.B.a}{a^T.V.a}$$

On montre qu'on trouve a en cherchant le **vecteur propre** associé à la plus grande valeur propre de V^1B. Par ailleurs comme $V = B + W$ on montre que ce problème est équivalent à chercher le vecteur propre associé à la plus grande valeurs propre de $W^{-1}B$.

Attention : même si les matrices $V^{-1}B$ et $W^{-1}B$ ont les mêmes vecteurs propres, ils n'ont cependant pas les mêmes valeurs propres. Soit v_1 le vecteur : propres associé à la plus grande valeurs propres on a :

$V^{-1}Bv_1 = \lambda_1 v_1$ et $W^{-1}Bv_1 = \mu_1 v_1$, quel est le lien entre λ_1 et μ_1 ???

Les nombre de valeurs propres non nulles est égal à $\min(K-1, p)$.

Ici on à $K = 3$ classes et on est en dimension $d = 4$ donc on a au plus 2 valeurs propres non nulles, donc deux composantes discriminantes.

On peut les trouver comme suit :

```
res2 = eigen(solve(V) %*% B)
res2
```

```
## eigen() decomposition
## $values
## [1] 9.698722e-01 2.220266e-01 6.994243e-15 3.019403e-16
##
## $vectors
##            [,1]         [,2]       [,3]       [,4]
## [1,] -0.2087418  0.006531964  0.2546279  0.8846854
## [2,] -0.3862037  0.586610553 -0.4061398 -0.2797345
## [3,]  0.5540117 -0.252561540 -0.4669638 -0.2546818
## [4,]  0.7073504  0.769453092  0.7430747 -0.2724289
```

Ici la première valeur propre est égale à $96,9\%$ ce qui réprésente la part de variance expliquée par la première composante discriminante : $R^2_{D_1/Y} = 96,9\%$

Et on calcule la composante comme suit :

```
D = as.matrix(iris[-5]) %*% res2$vectors
colnames(D) = paste0("D",1:4)
AFD = cbind.data.frame(D,Y = iris$Y)
ggplot(AFD, aes(x = D1, y = D2, color = Y, shape = Y)) + geom_point()
```

On peut aussi calculer $R^2_{D_1/Y}$ comme suit :

```
summary(lm(D1 ~ Y, data = AFD))$r.squared
```

```
## [1] 0.9698722
```

Attention : selon le logiciel la matrice diagonalisée peut être $V^{-1}B$ ou $W^{-1}B$, attention à l'interprétation des valeurs propres ...

Remarque : Dans ce qui a été présenté en ici on a fait des AFD et ACP non centrée (le variables non pas été recentrées), le nuage de point obtenu est même qu'habituellement à une translation près.

En pratique, les coefficients permettant de la combinaison linéaire sont simplement obtenu comme suit :

```
K = nlevels(iris$Y)
library(MASS)
coeff = lda(Y ~ ., data = iris, prior = rep(1/K, K))$scaling
coeff
```

```
##           LD1         LD2
## X1  0.8293776  0.02410215
## X2  1.5344731  2.16452123
## X3 -2.2012117 -0.93192121
## X4 -2.8104603  2.83918785
```

Ces coefficient sont déterminés de telle sorte que les variances intra-classe de chaque composante discriminante soit égale à 1, $a^TWa = 1$ (mais il ici le variance sans biais, d'où la fonction `cov` usuelle):

```r
proj_v2 = cbind.data.frame(as.matrix(iris[-5]) %*% coeff, iris$Y)
names(proj_v2) = c("D1","D2", "Y")
head(proj_v2)
```

```
##         D1       D2      Y
## 1 5.956693 6.961893 setosa
## 2 5.023581 5.874812 setosa
## 3 5.384722 6.396088 setosa
## 4 4.708094 5.990841 setosa
## 5 6.027203 7.175935 setosa
## 6 5.596840 8.123194 setosa
```

```r
Wiproj = by(proj_v2[,1:2], proj_v2$Y, cov)
Wproj = Reduce('+',Map('*',Wiproj,ni))/sum(ni)
Wproj
```

```
##               D1            D2
## D1  1.000000e+00 -6.276461e-16
## D2 -6.276461e-16  1.000000e+00
```

```r
t(coeff) %*% W %*% coeff # idem (au facteur (n-1)/n près)
```

```
##               LD1           LD2
## LD1  9.800000e-01 -6.106227e-16
## LD2 -4.996004e-16  9.800000e-01
```

3 Règle de décision : score linéaire

La partie précédente est essentiellement descriptive, elle peut cependant être utilisée pour faire de la prédiction à l'aide de la règle pour une nouvelle donnée x :

$$\hat{y} = \arg \max_{i \in \{1,\ldots,K\}} (x - \bar{X}_i)^T W^{-1} (x - \bar{X}_i)$$

cela consiste à classer dans le classe la plus proche au sens de la **métrique de Mahalanobis**.

On montre que cette règle est équivalente à maximiser :

$$\arg \max_i x' 2W^{-1}\bar{X}_i - \bar{X}_i' W^{-1} \bar{X}_i = \arg \max_i s_i(x)$$

avec $s_i(x) = \alpha_{0i} + \alpha_{i1}x_1 + \cdots + \alpha_{ip}x_p$

avec $\alpha_{i0} = -\bar{X}_i' W^{-1} \bar{X}_i$ et

$$\begin{pmatrix} \alpha_{i1} \\ \vdots \\ \alpha_{ip} \end{pmatrix} = 2W^{-1}\bar{X}_i$$

Construire le tableau des coefficients :

	Setosa	Versicolor	Virginica
α_0	α_{10}	α_{20}	α_{30}
$X_1 : \alpha_1$	α_{11}	α_{21}	α_{31}
$X_2 : \alpha_2$	α_{12}	α_{22}	α_{32}
$X_3 : \alpha_3$	α_{13}	α_{23}	α_{33}
$X_4 : \alpha_4$	α_{14}	α_{24}	α_{34}

Exemple à la main :

```r
i = 1 # Classe 1 (setosa)
dim(G[i,])
```

```
## NULL
```

```r
dim(G[i,,drop = FALSE])
```

```
## [1] 1 4
```

```r
Xi <- matrix(G[i,],4,1)
# Xi <- t(G[i,,drop = FALSE])
- t(Xi) %*% solve(W) %*% Xi  # Premier alpha_{i0}
```

```
##            [,1]
## [1,] -173.8977
```

```r
2 * solve(W) %*% Xi # Les 4 autres !
```

```
##          [,1]
## X1  48.04932
## X2  48.13851
## X3 -33.53192
## X4 -35.50696
```

```r
# Ou directement dans un tableau :
c(- G[1,,drop=F] %*% solve(W) %*% t(G[1,,drop=F]),
    2 * solve(W) %*% t(G[1,,drop=F]))
```

```
## [1] -173.89767   48.04932   48.13851  -33.53192  -35.50696
```

Automatisable avec un sapply :

```r
A = sapply(levels(iris$Y), function(k) {
  c(- G[k,,drop=F] %*% solve(W) %*% t(G[k,,drop=F]),
    2 * solve(W) %*% t(G[k,,drop=F]))
})
A
```

```
##           setosa versicolor   virginica
## [1,] -173.89767 -146.43672 -210.754506
## [2,]   48.04932   32.03716   25.399692
## [3,]   48.13851   14.43369    7.520979
## [4,]  -33.53192   10.63561   26.054173
## [5,]  -35.50696   13.13108   43.018598
```

On déduite les scores dans chacune des classes pour chacun des individus :

```r
scores = cbind(1,as.matrix(iris[-5])) %*% A
# On rajoute une colonne de 1 dans les données pour prendre en compte l'intercept dans le calcul du sco
head(scores)
```

```
##        setosa versicolor  virginica
## [1,] 185.5926   84.98680  -9.813089
## [2,] 151.9135   71.36252 -18.653517
## [3,] 155.2845   66.77827 -24.834677
## [4,] 138.9593   64.25830 -22.915909
## [5,] 185.6015   83.22645 -11.600960
## [6,] 202.1018  106.18833  17.235182
```

Enfin on en déduit la classe prédite par :

```r
Yp = apply(scores, 1, function(x) names(which.max(x)))
```

Qu'on peut comparer avec la vraie classe :

```r
table(iris$Y, Yp)
```

```
##             Yp
##              setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         48         2
##   virginica       0          1        49
```

4 Exercice d'application sur les données glass

L'objectif est de prédire le type de verre, variable `Type` en fonction des autres variables explicatives.

```r
glass = read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data", header =
names(glass) <- c("RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type")
glass = glass[-1] # suppression de la première variable qui sert d'identifiant
summary(glass)
```

```
##        Na              Mg              Al              Si       
##  Min.   :1.511   Min.   :10.73   Min.   :0.000   Min.   :0.290  
##  1st Qu.:1.517   1st Qu.:12.91   1st Qu.:2.115   1st Qu.:1.190  
##  Median :1.518   Median :13.30   Median :3.480   Median :1.360  
##  Mean   :1.518   Mean   :13.41   Mean   :2.685   Mean   :1.445  
##  3rd Qu.:1.519   3rd Qu.:13.82   3rd Qu.:3.600   3rd Qu.:1.630  
##  Max.   :1.534   Max.   :17.38   Max.   :4.490   Max.   :3.500  
##        K               Ca                Ba               Fe       
##  Min.   :69.81   Min.   :0.0000   Min.   : 5.430   Min.   :0.000  
##  1st Qu.:72.28   1st Qu.:0.1225   1st Qu.: 8.240   1st Qu.:0.000  
##  Median :72.79   Median :0.5550   Median : 8.600   Median :0.000  
##  Mean   :72.65   Mean   :0.4971   Mean   : 8.957   Mean   :0.175  
##  3rd Qu.:73.09   3rd Qu.:0.6100   3rd Qu.: 9.172   3rd Qu.:0.000  
```

```
##  Max.   :75.41   Max.   :6.2100   Max.   :16.190   Max.   :3.150
##       Type               NA
##  Min.   :0.00000   Min.   :1.00
##  1st Qu.:0.00000   1st Qu.:1.00
##  Median :0.00000   Median :2.00
##  Mean   :0.05701   Mean   :2.78
##  3rd Qu.:0.10000   3rd Qu.:3.00
##  Max.   :0.51000   Max.   :7.00
```

1. Faire une ACP sur les données `glass`
2. Faire une AFD sur les données `glass`
3. Comparer aux résultats de l'ACP
4. Utiliser un score linéaire pour calculer la prédiction et comparer à la valeur réelle.