Université Lille 1 – Master 1

RDF – TP n°07

Réduction de la dimension par analyse en composantes principales et analyse factorielle discriminante

BARCHID Sami – SLIMANI Anthony
10/03/2019

Introduction

L'objectif de ce TP est d'apprendre la réduction de l'espace de représentation en identifiant un axe de projection optimal. Pour ce faire, nous avons vu deux méthodes dans ce TP : l'analyse en composantes principales (non supervisé) et l'analyse factorielle discriminante (supervisé).

Ce rapport est divisé en 4 parties :

- **Analyse des ensembles de données** : présentation et analyse des données multivariées utilisées pour ce TP.
- **Analyse en composantes principales :** expérimentation et interprétations de l'espace de représentation obtenu par analyse en composantes principales puis sur la classification résultante grâce à l'analyse linéaire discriminante.
- **Analyse factorielle discriminante :** expérimentation et interprétations de l'espace de représentation obtenu par analyse factorielle discriminante puis sur la classification résultante grâce à l'analyse linéaire discriminante.
- **Comparaison avec l'espace d'origine :** comparaison des résultats obtenus avec les classifications par analyse linéaire discriminante dans les parties précédentes avec les résultats obtenus avec la classification par analyse linéaire discriminante dans un l'espace de représentation d'origine.

Analyse des ensembles de données

Durant ce TP, nous avons eu à disposition deux ensembles :

- Un ensemble d'apprentissage présenté en figure 1.
- Un ensemble de test présent en figure 2.

Le but pour la suite du TP sera de réduire l'espace de représentation selon un axe optimale selon une des deux techniques présentées aux point suivants. Ensuite, nous appliquerons une classification par analyse linéaire discriminante sur les données d'apprentissage projetées dans le nouvel espace de représentation trouvé. Et enfin, nous appliquerons la classification identifiée sur les données de test projetées dans le nouvel espace de représentation calculé.

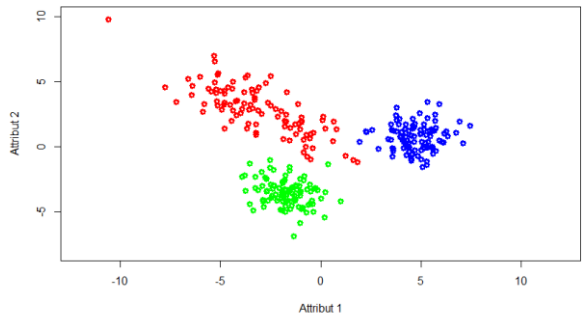

FIGURE 1 – DONNÉES D'APPRENTISSAGE

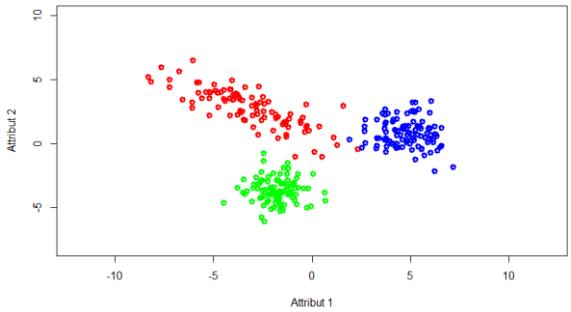

FIGURE 2 – DONNÉES DE TEST

Les données sont multivariées où nous avons D, le nombre d'attributs, qui vaut 2. L'objectif ici est donc de réduire D en un d=1 et par conséquent, passer d'un espace de représentation en deux dimensions à un espace à une seule dimension.

Nous avons les classes suivantes :

- Classe 1 en rouge
- Classe 2 en vert
- Classe 3 en bleu

Nous avons ensuite analysé les relations entre les deux attributs étudiés en calculant la matrice de covariance sur les données d'apprentissage :

$$\begin{pmatrix} \mathbf{13.96} & \mathbf{-0.95} \\ \mathbf{-0.95} & \mathbf{8.86} \end{pmatrix}$$

Cette matrice de covariance indique deux choses :

- Aucun attribut est indépendant (rien n'est nul dans la matrice de covariance)
- $\sum_{1,2} = \sum_{2,1}$ (car une matrice de covariance est symétrique) et ces deux valeurs sont négatives. Ce qui veut dire que, si l'attribut 1 augmente, l'attribut 2 va diminuer et vice-versa.

Analyse en composantes principales

Pour cette partie du TP, nous sommes dans le cadre d'un apprentissage non-supervisé. Nous allons donc réduire l'espace de représentation grâce à l'analyse en composantes principales (ACP).

ACP sur les données d'apprentissage

Nous avons commencé par appliquer l'ACP (analyse en composantes principales) sur les données d'apprentissage pour fixer un axe principal le plus discriminant possible. Le résultat obtenu est montré sur l'image suivante :

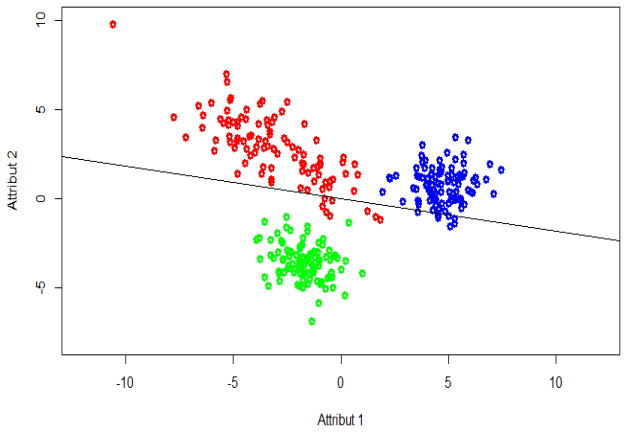

FIGURE 3 – DONNÉES D'APPRENTISSAGE AVEC AXE PRINCIPALE

La droite vue ici est l'axe principale (qui est la droite dont le vecteur directeur est le vecteur propre principale de la matrice de covariance pour les données d'apprentissage). Cette droite sera notre nouvel espace de représentation obtenu par analyse de composantes principales.

En projetant les données d'apprentissage sur cet axe, nous obtenu l'image suivante :

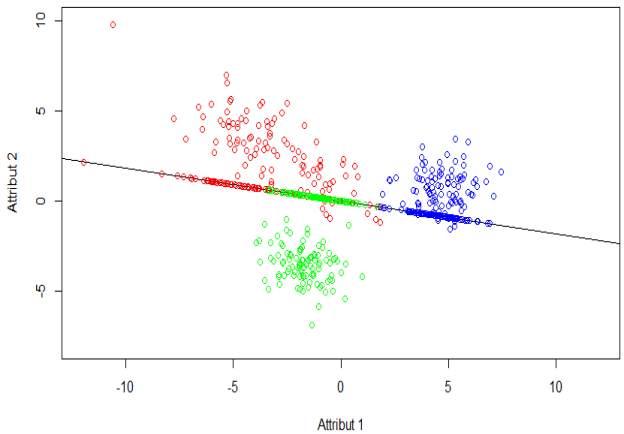

FIGURE 4 – DONNÉES D'APPRENTISSAGE PROJETÉES SUR L'AXE PRINCIPALE

En observant la projection des données d'apprentissage sur l'axe, nous observons que les données peuvent être, en majorité, discriminées convenablement. En effet, nous observons bien des

regroupements des observations projetées pour chaque classe sur le nouvel espace de représentation (l'axe).

Cependant, nous remarquons déjà qu'il y aura une portion d'erreurs puisque, certaines observations d'une classe seront projetées sur une autre classe (par exemple, les points rouges au centre seront projetés dans la classe 2 en vert).

Classification par ALD des données de test projetées

Nous allons appliquer la règle de décision de l'ALD apprise avec les données d'apprentissage sur les données de test projetées sur l'axe principale calculé auparavant.

Tout d'abord, nous projetons les données de tests sur l'espace de représentation obtenu sur les données d'apprentissage, ce qui nous donne l'image suivante :

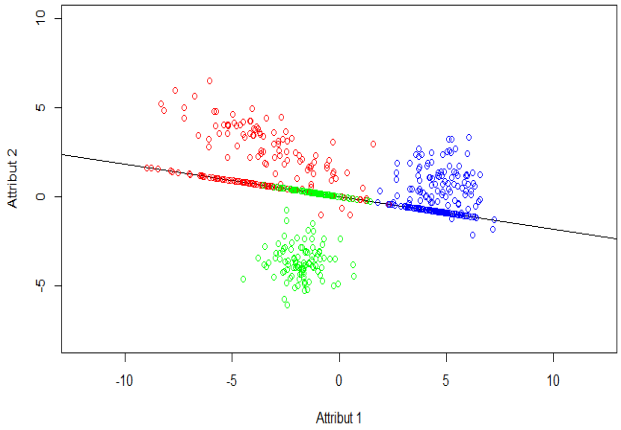

FIGURE 5 – DONNÉES DE TEST PROJETÉES SUR L'AXE PRINCIPALE

Nous remarquons que, comme pour les données d'apprentissage, les données de test pourront être discriminées convenablement.

Nous appliquons, donc, l'analyse linéaire discriminante sur les données de test projetées, ce qui donne la figure suivante :

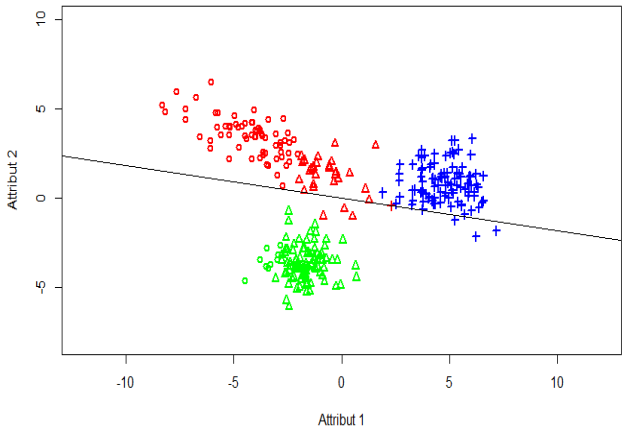

FIGURE 6 – ANALYSE LINÉAIRE DISCRIMINANTE DES DONNÉES DE TEST PROJETÉES SUR L'AXE PRINCIPALE

Dans la figure 6, les classes originales des observations sont données par la couleur tel que (rien ne change par rapport à avant) :

- Classe 1 en rouge
- Classe 2 en vert
- Classe 3 en bleu

Et les classes trouvées par classification suivant l'ALD sont données de la manière suivante :

- Classe 1 donnée par des cercles
- Classe 2 donnée par des triangles
- Classe 3 donnée par des croix (+)

Ainsi, nous observons plusieurs choses :

- Des observations de la classe 1 (en rouge) ont été identifiées comme des éléments de la classe 2 (triangle) et inversement. Ces points sont ceux qui étaient projetés (dans la figure 5) dans les regroupements de classes autres
- Les observations de la classe 3 (en bleu) ont tous été bien classifiés. On remarque aussi qu'une observation de la classe rouge (très éloignée du reste de sa classe) a été identifiée comme appartenant à la classe 3.

Nous pouvons, ensuite, calculer les taux de bonne classification des données de test. Le taux de bonne classification est de $0.866666 \approx 0.87$ donc **87%**.

Pour chaque classe, le taux de bonne classification est :

- Classe 1 : 69%
- Classe 2 : 91%
- Classe 3 : 100%

Nous pouvons donc confirmer que les données ont pu être correctement discriminées avec un taux de bonne classification correcte (mais pas exceptionnel non plus, 13% d'erreur ça fait beaucoup). Cependant, la classe 1 a été celle avec le plus d'erreurs et donne un taux de bonne classification qui, lui, n'est pas satisfaisant.

Analyse factorielle discriminante

Pour cette partie du TP, Nous sommes dans le contexte d'un apprentissage supervisé. Nous allons donc réduire l'espace de représentation grâce à l'analyse factorielle discriminante, dite AFD.

AFD sur les données d'apprentissage

Nous commencions par calculer les moyennes et les matrices de covariances des données d'apprentissage des classes 1, 2 et 3.

$$\boldsymbol{\mu_1} = \begin{pmatrix} \mathbf{-3.09} \\ \mathbf{2.85} \end{pmatrix} \ \boldsymbol{\mu_2} = \begin{pmatrix} \mathbf{-1.70} \\ \mathbf{-3.51} \end{pmatrix} \ \boldsymbol{\mu_3} = \begin{pmatrix} \mathbf{4.73} \\ \mathbf{0.72} \end{pmatrix} \boldsymbol{\mu} = \begin{pmatrix} \mathbf{-0.02} \\ \mathbf{0.02} \end{pmatrix}$$

$$\boldsymbol{S_1} = \begin{pmatrix} \mathbf{5.04} & \mathbf{-3.28} \\ \mathbf{-3.28} & \mathbf{3.58} \end{pmatrix} \ \boldsymbol{S_2} = \begin{pmatrix} \mathbf{0.98} & \mathbf{-0.24} \\ \mathbf{-0.23} & \mathbf{1.03} \end{pmatrix} \ \boldsymbol{S_3} = \begin{pmatrix} \mathbf{1.02} & \mathbf{0.04} \\ \mathbf{0.04} & \mathbf{1.03} \end{pmatrix}$$

Les moyennes correspondent aux coordonnées des barycentres des nuages de points de chacune de classes. Tandis que les matrices de covariances représentent les dépendances de chacune des classes par rapport aux deux attributs. À partir de ces résultats, nous allons calculés la dispersion intra $\boldsymbol{S^W}$ et inter $\boldsymbol{S^B}$ classes.

$$\boldsymbol{S^W} = \begin{pmatrix} \mathbf{7.04} & \mathbf{-3.48} \\ \mathbf{-3.48} & \mathbf{5.65} \end{pmatrix} \ \boldsymbol{S^B} = \begin{pmatrix} \mathbf{34.77} & \mathbf{0.60} \\ \mathbf{0.60} & \mathbf{20.90} \end{pmatrix}$$

Grâce à ces deux résultats, cela va nous permettre de rechercher l'attribut qui maximise la distance entre les moyennes et qui minimise la dispersion intra-classes selon le critère de Fisher qu'on cherche à maximiser. On calcule alors les vecteurs et les valeurs propres de la matrice $\boldsymbol{S_W^{-1}S_B}$. On obtient alors les valeurs et les vecteurs propres suivants :

$$\boldsymbol{v_1 = 9.93} \ \ \boldsymbol{v_2 = 2.65} \ \ \boldsymbol{V_p} = \begin{pmatrix} \mathbf{0.71} & \mathbf{-0.52} \\ \mathbf{0.71} & \mathbf{0.85} \end{pmatrix}$$

Nous avons alors comme vecteur principal : $\vec{v} = (\mathbf{0.71} \quad \mathbf{0.71})$. Ce vecteur définie la droite dont le projecteur des nuages des points est optimal. Comme pour ALD, nous pouvons tracer cette droite sur les données d'apprentissage et projeter l'ensemble des données sur celle-ci. Nous obtenons la figure 7 suivante :

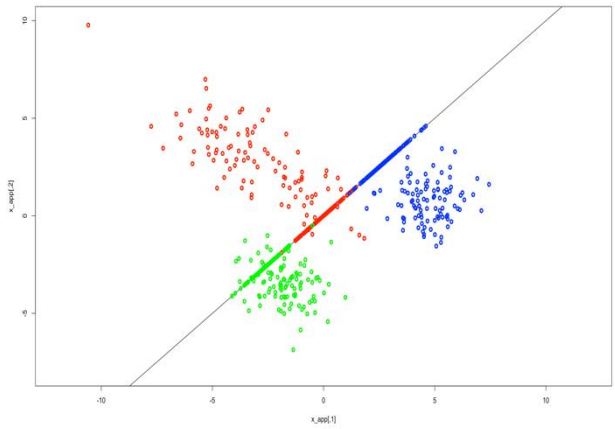

FIGURE 7 - DONNÉES D'APPRENTISSAGE PROJETÉES SUR L'AXE PRINCIPALE

Classification par ALD des données tests projetés par AFD

Nous allons appliquer la règle de décision de l'AFD apprise avec les données d'apprentissage sur les données de test projetées sur l'axe principale calculé auparavant. Nous obtenons la figure suivante :

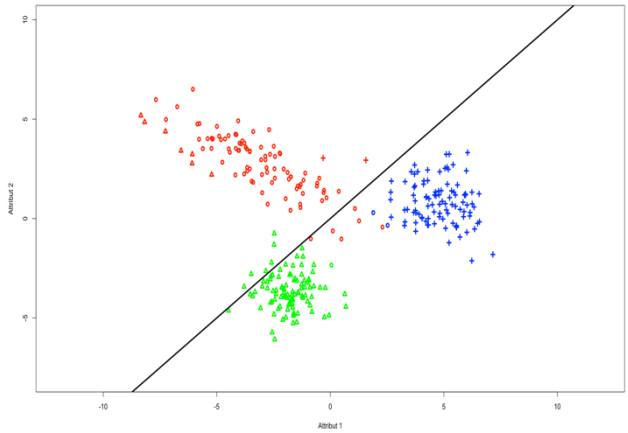

FIGURE 8 – ANALYSE LINÉAIRE DISCRIMINANTE DES DONNÉES DE TEST PROJETÉES SUR L'AXE PRINCIPALE

Nous gardons la même légende que pour l'analyse composante principales.

Ainsi, nous observons plusieurs choses :

- Des observations de la classe 1 ont été identifiées comme des éléments des classes 2 et 3 mais restent globalement correctes
- Des observations de la classe 2 et 3 sont pratiquement bien classés

Nous pouvons, ensuite, calculer les taux de bonne classification des données de test. Le taux de bonne classification est de **96%**.

Pour chaque classe, le taux de bonne classification est :

- Classe 1 : 91%
- Classe 2 : 99%
- Classe 3 : 98%

Comparaison ACP vs AFD

On constate que l'AFD est plus performante, dans notre cas, que l'ACP avec un écart de 9%. Cela peut s'expliquer de la manière suivante : l'ACP maximise la variance des projections sur le sous-espace tandis que l'AFD maximise la différenciation entre les classes dans le sous-espace. On sait que l'AFD maximise le critère de Fisher, c'est-à-dire qu'elle recherche l'attribut qui maximise la distance entre les moyennes et minimise la dispersion intra-classe. L'axe trouvé par AFD est donc optimal pour la réduction de l'espace de représentation. Il faut noter tout de même que nous ne sommes pas dans le même contexte (supervisé vs non-supervisé).

Comparaison avec l'espace d'origine

Dans cette partie du TP, nous allons comparer l'analyse linéaire discriminante (ALD) après réduction de l'espace de représentation par AFD ou ACP et l'ALD obtenue dans l'espace de représentation d'origine.

Nous allons donc classifier les données de test avec un classifieur ALD appris à partir des données d'apprentissage dans l'espace 2D d'origine. La classification trouvée est la suivante :

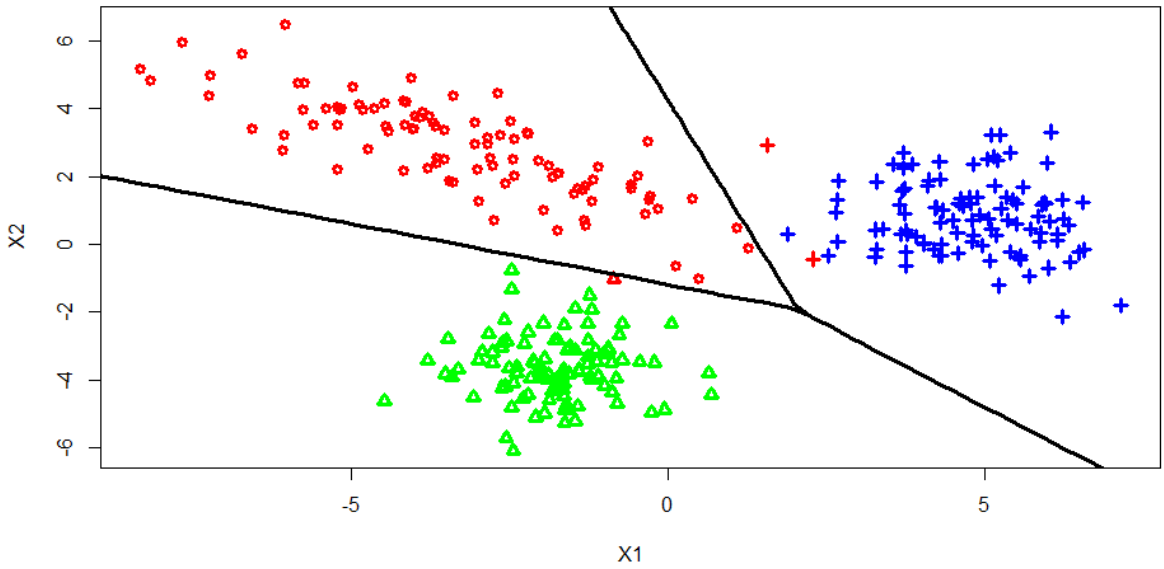

FIGURE 8 – ANALYSE LINÉAIRE DISCRIMINANTE DES DONNÉES DE TEST DANS L'ESPACE 2D D'ORIGINE

Nous obtenons alors **un taux de bonne classification de 99%.** Nous avons alors le tableau de comparaison des taux de bonne classification suivant :

Par ACP	**Par ALD**	**Dans l'espace d'origine**
87%	96%	99%

Nous pouvons directement remarquer que le taux de bonne classification est clairement plus avantageux avec une ALD obtenue à partir de l'espace d'origine. Toutefois, nous devons tenir compte du fait que cela est possible seulement si nous ne tombons pas dans la malédiction de la dimensionnalité (donc, que nous avons bel et bien des dimensions utiles à notre classification et pas d'attributs qui sont hors-sujets).

De plus, une ALD obtenue sur un espace en 2 dimensions demandera plus de calcul qu'une ALD obtenue sur un espace en une seule dimension.

Nous pouvons conclure sur le fait que, suivant l'objectif recherché et le contexte dans lequel on est (supervisé vs non-supervisé), il faut trouver la méthode de classification qui est à la fois satisfaisante dans son taux de classification correcte et à la fois satisfaisante en termes de performance.

Annexes

Annexe 1 : macro R pour la classification par analyse linéaire discriminante de données de test dans un espace réduit de dimension obtenu par l'analyse par composantes principales.

```
library("MASS")
library("lattice")

lwd_default = 3 # pour l'affichage

# Chargement des données d'apprentissage et de test
################################################
load(file = "x_app.data")
load(file = "classe_app.data")
load(file = "x_test.data")
load(file = "classe_test.data")


################################################
# Afficher les données d'apprentissage
################################################
plot(x_app[classe_app==1,], col="red", xlab = "Attribut 1", ylab = "Attribut 2" , xlim=c(-12,12), ylim = c(-8, 10), lwd = lwd_default) # Classe 1 en rouge
points(x_app[classe_app==2,], col="green", lwd = lwd_default) # Classe 2 en bleu
points(x_app[classe_app==3,], col="blue", lwd = lwd_default) # Classe 3 en vert
################################################


################################################
# ACP sur données d'apprentissage
################################################
# Calcul de la covariance des données d'apprentissage
covariance_app = cov(x_app)

# Calcul des valeurs et vecteurs propres de covariance
Vp = eigen(covariance_app)

# Vp$vectors[,1] est le vecteur propre principale
# Tracer la droite correspondant au vecteur propre principale
# dont la valeur propre la plus ´elev´ee
pente <- Vp$vectors[2,1]/Vp$vectors[1,1]
abline(a = 0, b = pente, col = "black")


# Projection des points sur l'axe principale
# Calcul du vecteur d'apprentissage projeté sur l'axe principale
ScalarProduct_app_ACP = x_app %*% (Vp$vectors[,1])/sqrt(sum(Vp$vectors[,1]^Vp$vectors[,1]))

XP_app = x_app
# Projection des points
XP_app[,1]= ScalarProduct_app_ACP * Vp$vectors[1,1]
XP_app[,2]= ScalarProduct_app_ACP * Vp$vectors[2,1]
# Affichage des points projet´es
points(XP_app[classe_app==1,], col="red", lwd = lwd_default)
points(XP_app[classe_app==2,], col="green", lwd = lwd_default)
points(XP_app[classe_app==3,], col="blue", lwd = lwd_default)

################################################
```

```
################################################
# Afficher les données de test
################################################
plot(x_test[classe_test==1,], col="red", xlab = "Attribut 1", ylab = "Attribut 2" , xlim=c(-12,12), ylim = c(-8, 10), lwd=lwd_default) # Classe 1 en rouge
points(x_test[classe_test==2,], col="green", lwd = lwd_default) # Classe 2 en bleu
points(x_test[classe_test==3,], col="blue", lwd = lwd_default) # Classe 3 en vert
################################################



################################################
# ACP sur données de test
################################################

# Vp$vectors[,1] est le vecteur propre principale
# Tracer la droite correspondant au vecteur propre principale
# dont la valeur propre la plus ´elev´ee
pente <- Vp$vectors[2,1]/Vp$vectors[1,1]
abline(a = 0, b = pente, col = "black")


# Projection des points sur l'axe principale
# Calcul du vecteur d'apprentissage projeté sur l'axe principale
ScalarProduct_test_ACP = x_test %*% (Vp$vectors[,1])/sqrt(sum(Vp$vectors[,1]^Vp$vectors[,1]))

XP_test = x_test
# Projection des points
XP_test[,1]= ScalarProduct_test_ACP * Vp$vectors[1,1]
XP_test[,2]= ScalarProduct_test_ACP * Vp$vectors[2,1]
# Affichage des points projet´es
points(XP_test[classe_test==1,], col="red", lwd = lwd_default)
points(XP_test[classe_test==2,], col="green", lwd = lwd_default)
points(XP_test[classe_test==3,], col="blue", lwd = lwd_default)

################################################



################################################
# Appliquer, sur les données de test, la LDA (règle de décision) obtenues grâce aux données d'apprentissage
################################################
# Appliquer la LDA sur les données d'apprentissage
x_app_ACP.lda = lda(ScalarProduct_app_ACP,classe_app)

# Appliquer à partir des données d'apprentissage
assigne_test = predict(x_app_ACP.lda, newdata=ScalarProduct_test_ACP)

# Estimation des taux de bonnes classifications
table_classification_test = table(classe_test, assigne_test$class)

# table of correct class vs. classification
print("Bonne classification par classe : ")
print(diag(prop.table(table_classification_test, 1)))
```

```r
# total percent correct
taux_bonne_classif_test <-sum(diag(prop.table(table_classification_test)))
print("Taux de bonne classification :")
print(taux_bonne_classif_test)

# couleur de la classe 1 LABEL ORIGINAL
couleur<-rep("red",length(x_test)) ;
couleur[classe_test==1]='red'
couleur[classe_test==2]='green'
couleur[classe_test==3]='blue'

# Formes des données (pour le grahique)
shape<-rep(1,length(x_test)) ;
# Forme des données assignées
shape[assigne_test$class==1]=1 ; # classe 1 = cercle
shape[assigne_test$class==2]=2 ; # Classe 2 = triangle
shape[assigne_test$class==3]=3 ; # Classe 3 = croix

# Affichage des projections apprentissage classées
plot(x_test,col=couleur,pch=shape, xlab = "Attribut 1", ylab = "Attribut 2" , xlim=c(-12,12), ylim = c(-8, 10), lwd = lwd_default)
pente <- Vp$vectors[2,1]/Vp$vectors[1,1]
abline(a = 0, b = pente, col = "black")
```

Annexe 2 : macro R pour la classification par analyse linéaire discriminante de données de test dans un espace réduit de dimension obtenu par l'analyse factorielle discriminante.

```
library("MASS")
library("lattice")

lwd_default = 3 # pour l'affichage

# Chargement des données d'apprentissage et de test
#####################################################
load(file = "x_app.data")
load(file = "classe_app.data")
load(file = "x_test.data")
load(file = "classe_test.data")


#####################################################
# Afficher les données d'apprentissage
#####################################################
plot(x_app[classe_app==1,], col="red", xlab = "Attribut 1", ylab = "Attribut 2" , xlim=c(-12,12), ylim = c(-8, 10), lwd = lwd_default) # Classe 1 en rouge
points(x_app[classe_app==2,], col="green", lwd = lwd_default) # Classe 2 en bleu
points(x_app[classe_app==3,], col="blue", lwd = lwd_default) # Classe 3 en vert
#####################################################


#####################################################
# Calculer les dispersions inter et intra classe sur
# données d'apprentissage
#####################################################
# moyennes classe
mean1 = colMeans(x_app[classe_app==1,])
mean2 = colMeans(x_app[classe_app==2,])
mean3 = colMeans(x_app[classe_app==3,])
mean = colMeans(x_app)

# covariances intra-classe
S1 = cov(x_app[classe_app==1,])
S2 = cov(x_app[classe_app==2,])
S3 = cov(x_app[classe_app==3,])

# Calcul de la dispersion intra-classe
Sw=S1+S2+S3
# Calcul de la dispersion inter-classe
Sb=(mean1-mean)%*%t(mean1-mean)+(mean2-mean)%*%t(mean2-mean)+(mean3-mean)%*%t(mean3-mean)
#####################################################


#####################################################
# Effectuer AFD sur les données d'apprentissage
#####################################################
# Résolution de l'équation pour AFD
invSw= solve(Sw)
invSw_by_Sb= invSw %*% Sb
Vp<- eigen(invSw_by_Sb)
# Affichage de la droite correspondant au vecteur propre
# dont la valeur propre la plus ´elev´ee
pente <- Vp$vectors[2,1]/Vp$vectors[1,1]
abline(a = 0, b = pente, col = "black")
#####################################################


#####################################################
# Projection des données d'apprentissage sur l'axe
# principale trouvé avec AFD
#####################################################
# Calcul du vecteur d'apprentissage projeté sur l'axe principale
ScalarProduct_app_AFD = x_app %*% (Vp$vectors[,1])/sqrt(sum(Vp$vectors[,1]*Vp$vectors[,1]))

XP_app = x_app
# Projection des points
XP_app[,1]= ScalarProduct_app_AFD * Vp$vectors[1,1]
XP_app[,2]= ScalarProduct_app_AFD * Vp$vectors[2,1]
# Affichage des points projetés
points(XP_app[classe_app==1,], col="red", lwd = lwd_default)
points(XP_app[classe_app==2,], col="green", lwd = lwd_default)
points(XP_app[classe_app==3,], col="blue", lwd = lwd_default)
#####################################################


#####################################################
# Projection des données de test sur l'axe principale
# obtenu par AFD sur les données d'apprentissage
#####################################################

# Afficher les données de test
plot(x_test[classe_test==1,], col="red", xlab = "Attribut 1", ylab = "Attribut 2" , xlim=c(-12,12), ylim = c(-8, 10), lwd=lwd_default) # Classe 1 en rouge
points(x_test[classe_test==2,], col="green", lwd = lwd_default) # Classe 2 en bleu
points(x_test[classe_test==3,], col="blue", lwd = lwd_default) # Classe 3 en vert

# Vp$vectors[,1] est le vecteur propre principale
# Tracer la droite correspondant au vecteur propre principale
# dont la valeur propre la plus ´elev´ee
pente <- Vp$vectors[2,1]/Vp$vectors[1,1]
abline(a = 0, b = pente, col = "black")

# Projection des points sur l'axe principale
```

```
# Calcul du vecteur d'apprentissage projeté sur l'axe principale
ScalarProduct_test_AFD = x_test %*% (Vp$vectors[,1])/sqrt(sum(Vp$vectors[,1]*Vp$vectors[,1]))

XP_test = x_test
# Projection des points
XP_test[,1]= ScalarProduct_test_AFD * Vp$vectors[1,1]
XP_test[,2]= ScalarProduct_test_AFD * Vp$vectors[2,1]
# Affichage des points projetés
points(XP_test[classe_test==1,], col="red", lwd = lwd_default)
points(XP_test[classe_test==2,], col="green", lwd = lwd_default)
points(XP_test[classe_test==3,], col="blue", lwd = lwd_default)
################################################



################################################
# ALD sur les données de test projetées sur l'axe principale
################################################
# Appliquer la LDA sur les données d'apprentissage
x_app_AFD.lda = lda(ScalarProduct_app_AFD,classe_app)

# Appliquer à partir des données d'apprentissage
assigne_test = predict(x_app_AFD.lda, newdata=ScalarProduct_test_AFD)

# Estimation des taux de bonnes classifications
table_classification_test = table(classe_test, assigne_test$class)

# table of correct class vs. classification
print("Bonne classification par classe : ")
print(diag(prop.table(table_classification_test, 1)))

# total percent correct
taux_bonne_classif_test <-sum(diag(prop.table(table_classification_test)))
print("Taux de bonne classification :")
print(taux_bonne_classif_test)

# couleur de la classe 1 LABEL ORIGINAL
couleur<-rep("red",length(x_test)) ;
couleur[classe_test==1]='red'
couleur[classe_test==2]='green'
couleur[classe_test==3]='blue'

# Formes des données (pour le grahique)
shape<-rep(1,length(x_test)) ;
# Forme des données assignées
shape[assigne_test$class==1]=1 ; # classe 1 = cercle
shape[assigne_test$class==2]=2 ; # Classe 2 = triangle
shape[assigne_test$class==3]=3 ; # Classe 3 = croix

# Affichage des projections apprentissage classées
plot(x_test,col=couleur,pch=shape, xlab = "Attribut 1", ylab = "Attribut 2" , xlim=c(-12,12), ylim = c(-8, 10), lwd = lwd_default)
pente <- Vp$vectors[2,1]/Vp$vectors[1,1]
abline(a = 0, b = pente, col = "black")
################################################
```

Annexe 3 : analyse linéaire discriminante calculée sur les données de test dans l'espace 2D d'origine

```
library("MASS")
library("lattice")
load(file = "x_app.data")
load(file = "classe_app.data")
load(file = "x_test.data")
load(file = "classe_test.data")

#Valeur pour mettre plus en valeur les screens pour le rapport
lwd_default <- 3

########################################
# Q1.1
########################################
#Affichage de l'ensemble d'apprentissage
couleur_app<-rep('red',length(x_app))
couleur_app[classe_app==1]='red'
couleur_app[classe_app==2]='green'
couleur_app[classe_app==3]='blue'

plot(x_app, col = couleur_app, lwd = lwd_default)

#Affichage de l'ensemble de test
couleur_test<-rep('red',length(x_test))
couleur_test[classe_test==1]='red'
couleur_test[classe_test==2]='green'
couleur_test[classe_test==3]='blue'
plot(x_test, col = couleur_test, lwd = lwd_default)

########################################
# Q1.2
########################################
#Affichage de l'ensemble d'apprentissage
plot(x_app, col = couleur_app, lwd = lwd_default)

# Déclaration des vecteurs moyennes
M1 = seq(1,2)
M2 = seq(1,2)
M3 = seq(1,2)

# Vecteur moyenne de la classe 1
M1[1] = mean(x_app[classe_app==1,1])
M1[2] = mean(x_app[classe_app==1,2])

# Vecteur moyenne de la classe 2
M2[1] = mean(x_app[classe_app==2,1])
M2[2] = mean(x_app[classe_app==2,2])

# Vecteur moyenne de la classe 3
M3[1] = mean(x_app[classe_app==3,1])
M3[2] = mean(x_app[classe_app==3,2])

points(M1[1], y = M1[2], pch = 20, col = "red", lwd=15)
points(M2[1], y = M2[2], pch = 20, col = "green", lwd=15)
points(M3[1], y = M3[2], pch = 20, col = "blue", lwd=15)

#Affichage de l'ensemble d'apprentissage
plot(x_app, col = couleur_app, lwd = lwd_default)

# Matrices de co variance
Sigma1 <- matrix(1,2,2)
Sigma2 <- matrix(1,2,2)
Sigma3 <- matrix(1,2,2)

for (i in 1:2) {
  for (j in 1:2) {
    Sigma1[i,j]=cov(as.vector(x_app[classe_app==1,i]), as.vector(x_app[classe_app==1,j]))
    Sigma2[i,j]=cov(as.vector(x_app[classe_app==2,i]), as.vector(x_app[classe_app==2,j]))
    Sigma3[i,j]=cov(as.vector(x_app[classe_app==3,i]), as.vector(x_app[classe_app==3,j]))
  }
}

########################################
# Q1.3
########################################

# Grille d'estimation de la densit´e de probabilit´e en 50 intervalles selon 1er attribut
xp1 <- seq(min(x_app[,1]),max(x_app[,1]),length=50)
# Grille d'estimation de la densit´e de probabilit´e en 50 intervalles selon 2eme attribut
xp2 <- seq(min(x_app[,2]),max(x_app[,2]),length=50)

grille <- expand.grid(x1=xp1,x2=xp2)
x_app_lda<-lda(x_app,classe_app)

# Estimation des densit´es de probabilit´es a posteriori dans Zp
grille = cbind(grille[,1],grille[,2])
Zp <- predict(x_app_lda,grille)

#Affichage des lignes de décision
zp_3<-Zp[["posterior"]][,3]-pmax(Zp[["posterior"]][,1],Zp[["posterior"]][,2])
zp_2<-Zp[["posterior"]][,2]-pmax(Zp[["posterior"]][,1],Zp[["posterior"]][,3])

contour(xp1,xp2,matrix(zp_3,50),add=TRUE,levels=0,drawlabels=FALSE, col="green", lwd = lwd_default)
contour(xp1,xp2,matrix(zp_2,50),add=TRUE,levels=0,drawlabels=FALSE, col="blue", lwd = lwd_default)

########################################
# Q1.4
########################################

assigne_test <- predict(x_app_lda, newdata=x_test)
# Estimation des taux de bonnes classifications
table_classification_test_lda <-table(classe_test, assigne_test[["class"]])
# table of correct class vs. classification
diag(prop.table(table_classification_test_lda, 1))
# total percent correct
taux_bonne_classif_test_lda <-sum(diag(prop.table(table_classification_test_lda)))
```

```
# Création du vecteur contenant le code de la forme des données test assignées aux
shape<-rep(1,length(x_test)) ;
# Forme des données assignées
shape[assigne_test[["class"]]==1]=1 ;
shape[assigne_test[["class"]]==2]=2 ;
shape[assigne_test[["class"]]==3]=3 ;
plot(x_test,col=couleur_test,pch=shape,xlab = "X1", ylab = "X2", lwd = lwd_default)

#Affichage des lignes de décision
contour(xp1,xp2,matrix(zp_3,50),add=TRUE,levels=0,drawlabels=FALSE, col="black", lwd = lwd_default)
contour(xp1,xp2,matrix(zp_2,50),add=TRUE,levels=0,drawlabels=FALSE, col="black", lwd = lwd_default)
```