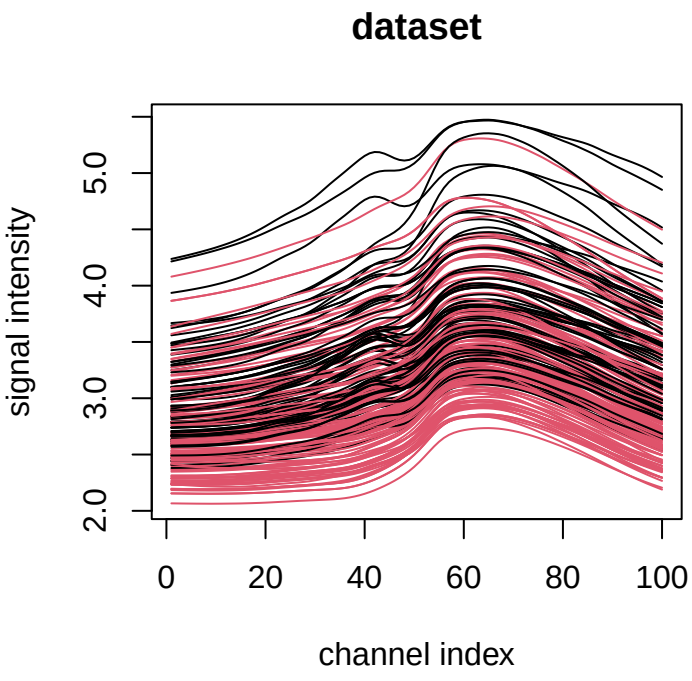
TP no 3 Apprentissage non-supervisé: Multi-Dimensional Scaling et SVM à une classe - solution

UE Apprentissage Statistique

1 Exercice 1

1.1 Questions 1 et 2

```
############################
#### STARTING EXERCICE 1 ####
############################
# question 1 - load dataset
load("datasets/spectra.Rdata")
# question 2 - plot dataset
plot(X[1,], type = "l", col = y[1], ylim = range(X), main = "dataset", xlab = "channel index", ylab = "s
for(i in 1:nrow(X)){
    lines(X[i,], type = "l", col = y[i])
}
```

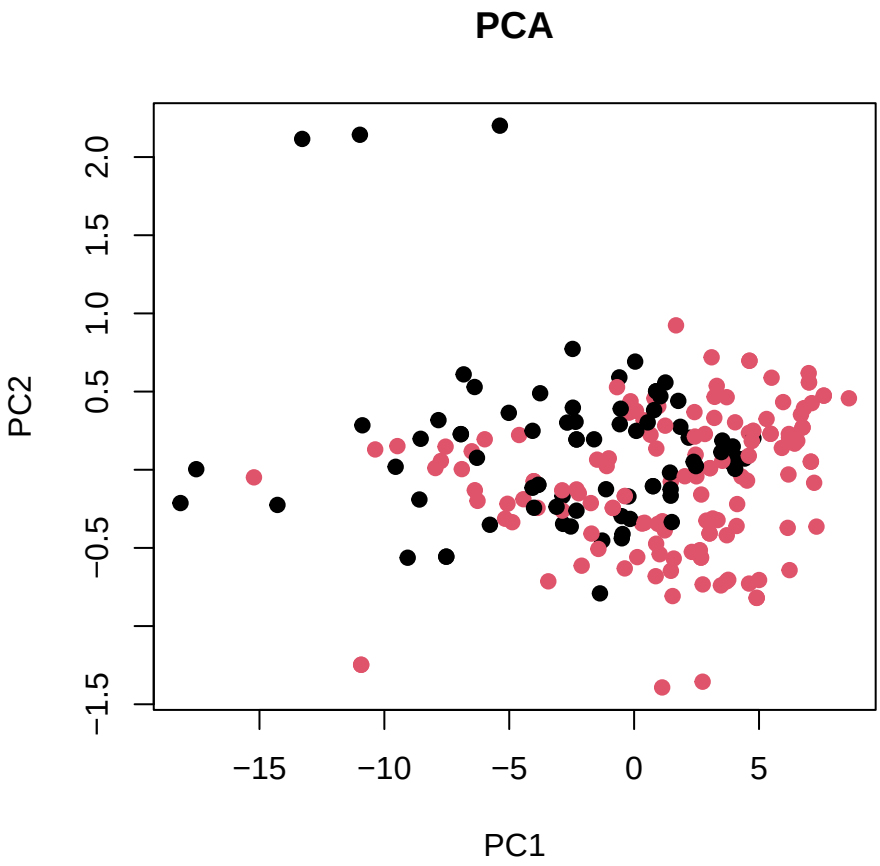


1.2 Questions 3

```
# question 3 - compute distance matrix
d  = dist(X)
```

1.3 Question 4

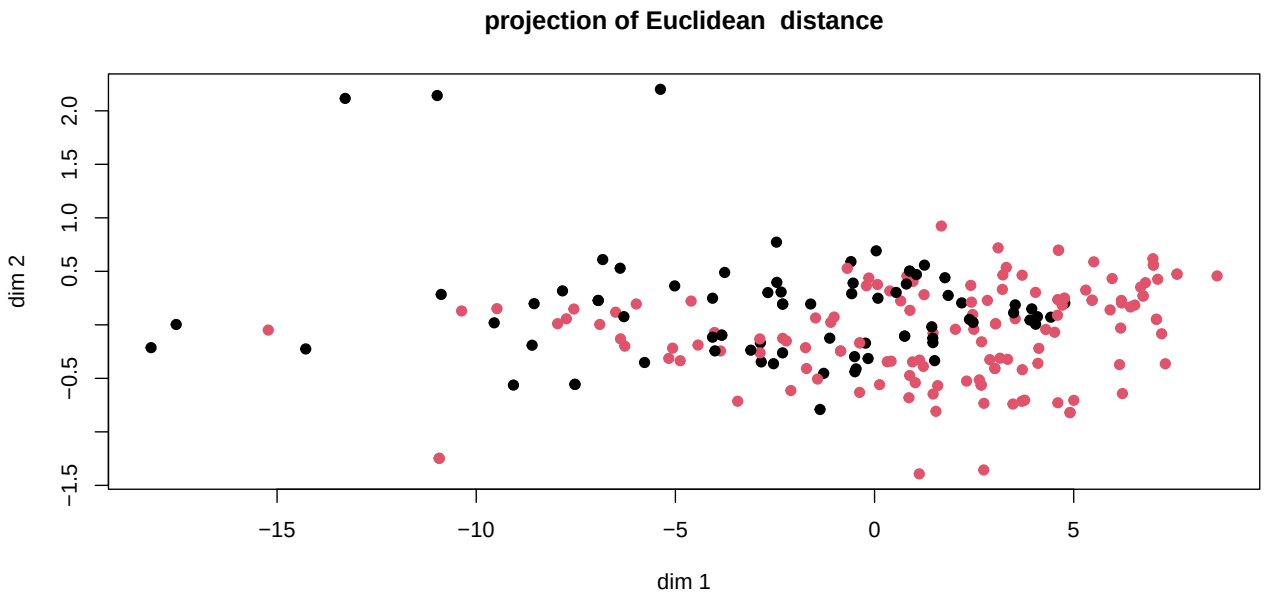
```
# question 5 - pca
pca = prcomp(X)
plot(-pca$x[,1], -pca$x[,2], col = y,  pch = 19, xlab = "PC1", ylab = "PC2", main  = "PCA")
```



On note que l'ACP ne permet pas de bien discriminer les deux catégories de courbes.

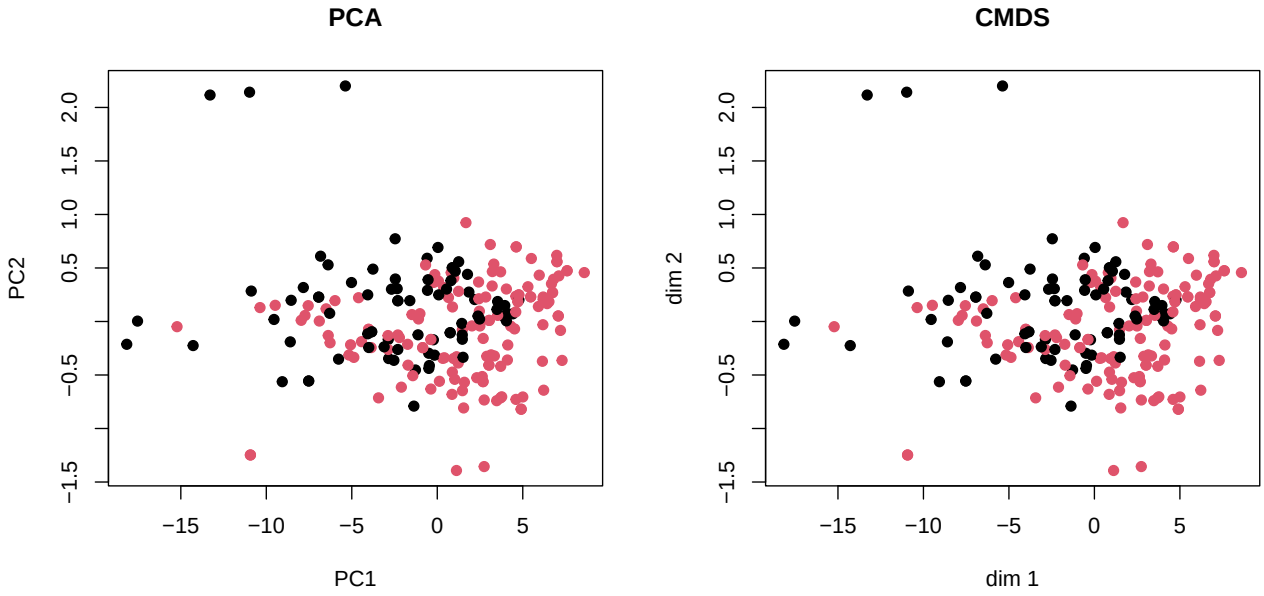
1.4 Question 5

```
# question 5 - represent it in 2d using cmdscale
ds = cmdscale(d, 2)
plot(ds[,1], ds[,2], col = y, xlab = "dim 1", ylab = "dim 2", main = "projection of Euclidean  distance'
```



On note que la distance Euclidienne ne permet pas de bien discriminer les deux catégories de courbes. Le code suivant compare les résultats obtenus par `cmdscale` à partir de la matrice de distance Euclidienne à l'ACP. Les résultats sont bien identiques. On note néanmoins que l'ACP étant définie à un signe près, il a fallu ici utiliser un facteur négatifs sur les composantes principales pour que les nuages de points soient effectivement superposables.

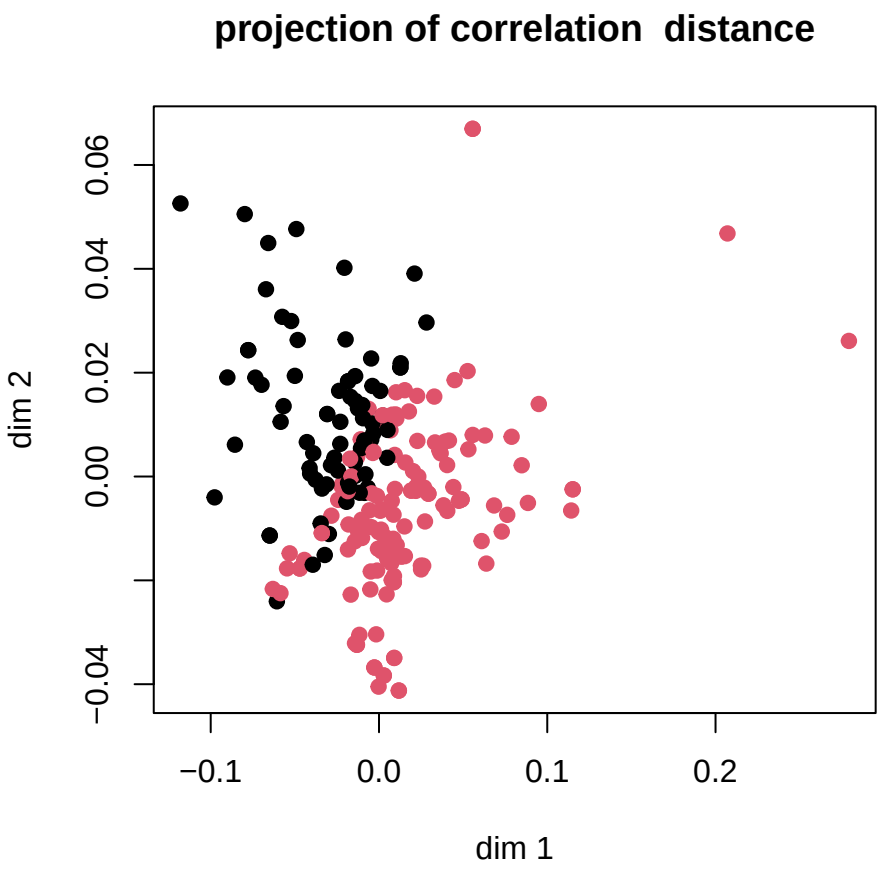
```
# compare to PCA
pca = prcomp(X)
par(mfrow = c(1,2))
plot(-pca$x[,1], -pca$x[,2], col = y,  pch = 19, xlab = "PC1", ylab = "PC2", main  = "PCA")
plot(ds[,1], ds[,2], col = y, xlab = "dim 1", ylab = "dim 2", main = "CMDS", pch = 19)
```



1.5 Question 6

```
# compute a distance object from correlation
  # 1) compute correlation matrix
```

```r
C = cor(t(X))
  # 2) build a distance object as 1 - C
dC = as.dist(1-C)
# represent it in 2D
dsC = cmdscale(dC, 2)
plot(dsC[,1], dsC[,2], col = y, xlab = "dim 1", ylab = "dim 2", main = "projection of correlation  dista
```



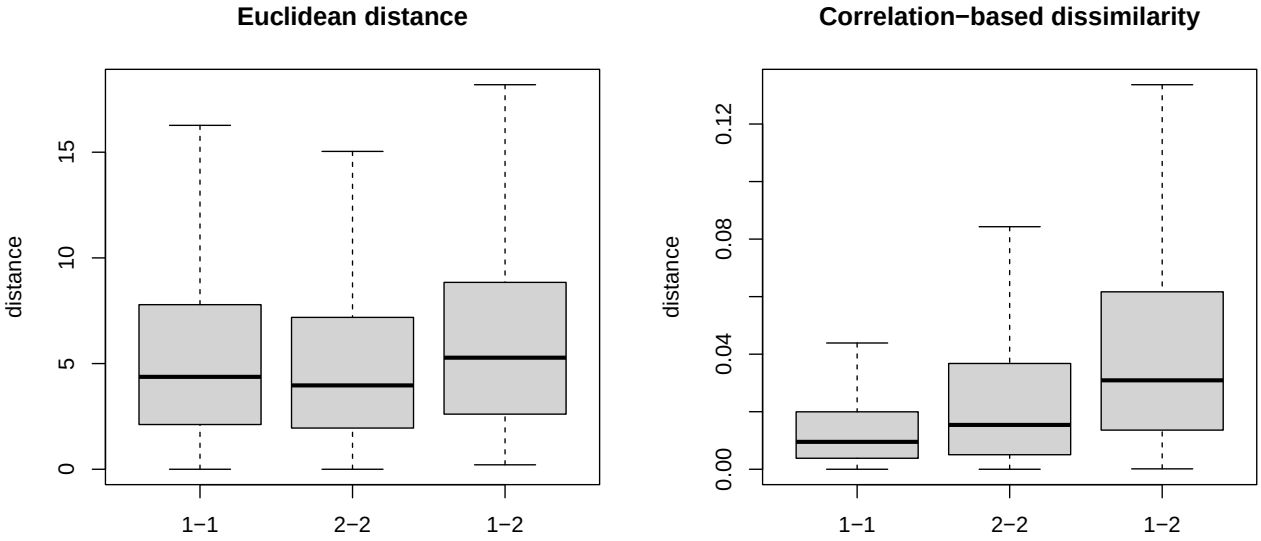
On constate qu'en utilisant la corrélation pour quantifier la similarité entre courbes on discrimine beaucoup mieux les deux catégories de courbe. La corrélation semble en effet mieux adaptée que la distance Euclidienne pour comparer ces courbes car elle est moins sensible aux effets "d'offset", i.e., de décalage global des courbes.

1.6 Question 7

Le code ci-dessous compare les distributions des distances/dissimilarité intra- et inter-classes, pour la distance Euclidienne (gauche) et pour la dissimiarité basée sur la corrélation. On note en effet qu'en se basant sur la corrélation, la distance inter-classe est en général plus importante que la distance intra-classe, ce qui n'est pas le cas avec la distance Euclidienne.

```r
# show distribution of numerical values
  # extract matrices
dX  = as.matrix(d)
dCX  = as.matrix(dC)
  # get indices of both classes
ind1 = which(y == 1)
ind2 = which(y == 2)
  # show distribution
par(mfrow = c(1,2))
```

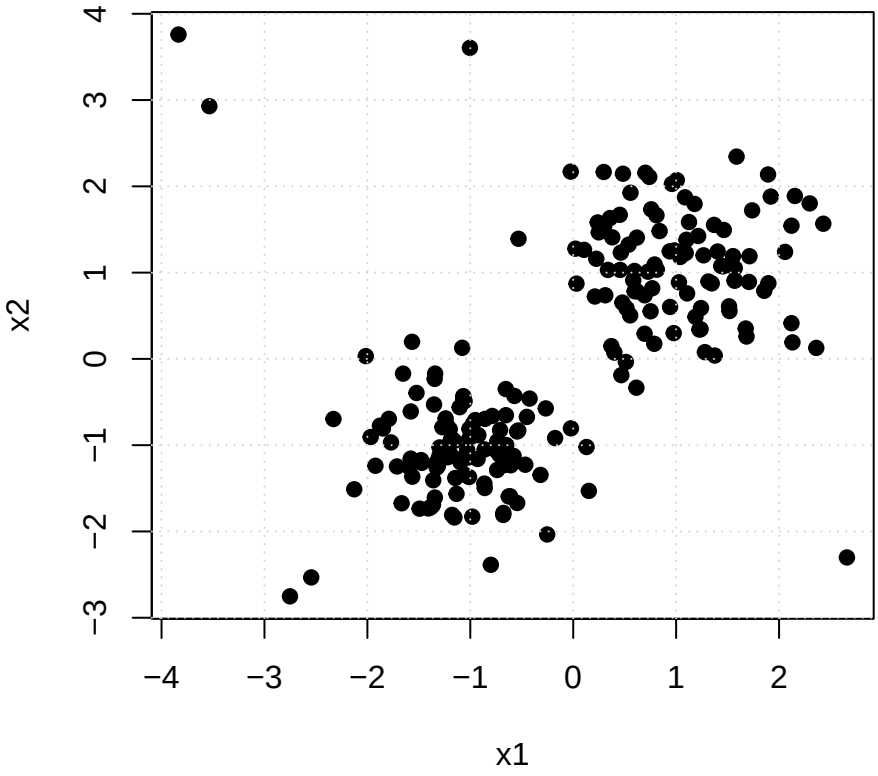
```r
boxplot(list("1-1"=dX[ind1,ind1], "2-2"=dX[ind2,ind2], "1-2"=dX[ind1,ind2]), outline  = F, main = "Eucli
boxplot(list("1-1"=dCX[ind1,ind1], "2-2"=dCX[ind2,ind2], "1-2"=dCX[ind1,ind2]), outline = F, main = "Cor
```



2 Exercice 2

2.1 Question 1 - 2

```r
##############################
#### STARTING EXERCICE 4 ####
##############################
library(e1071)
source("utils.R")
load("datasets/exo-4.Rdata")
plot(X[,1], X[,2], xlab = "x1", ylab = "x2", pch = 19)
grid()
```

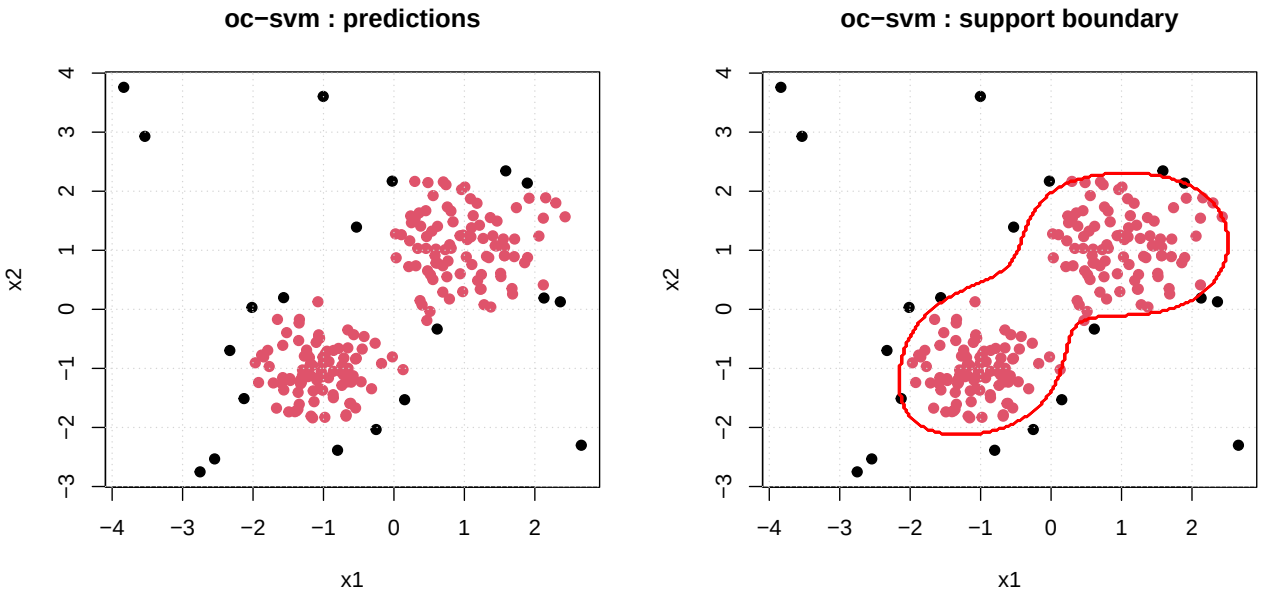


2.2 Question 3

Le code ci-dessous permet d'estimer le support de la distribution définie par notre nuage de points, et de représenter sa frontière par la fonction "contour" de R.

```
#--------------------------#
# illustrate one-class SVM #
#--------------------------#
par(mfrow = c(1,2))
# fit model
svm.model = svm(X, type='one-classification', nu=0.10, kernel="radial")
# make predictions
y.pred = predict(svm.model, X)
# show outliers
plot(X[,1], X[,2], col = factor(y.pred), xlab = "x1", ylab = "x2", pch = 19)
title('oc-svm : predictions')
grid()

# show decision boundary
source("utils.R")
svm_pred = function(x){
  return( predict(svm.model, x))
}
plot_with_contours(X, svm_pred, y = factor(y.pred))
title('oc-svm : support boundary')
```

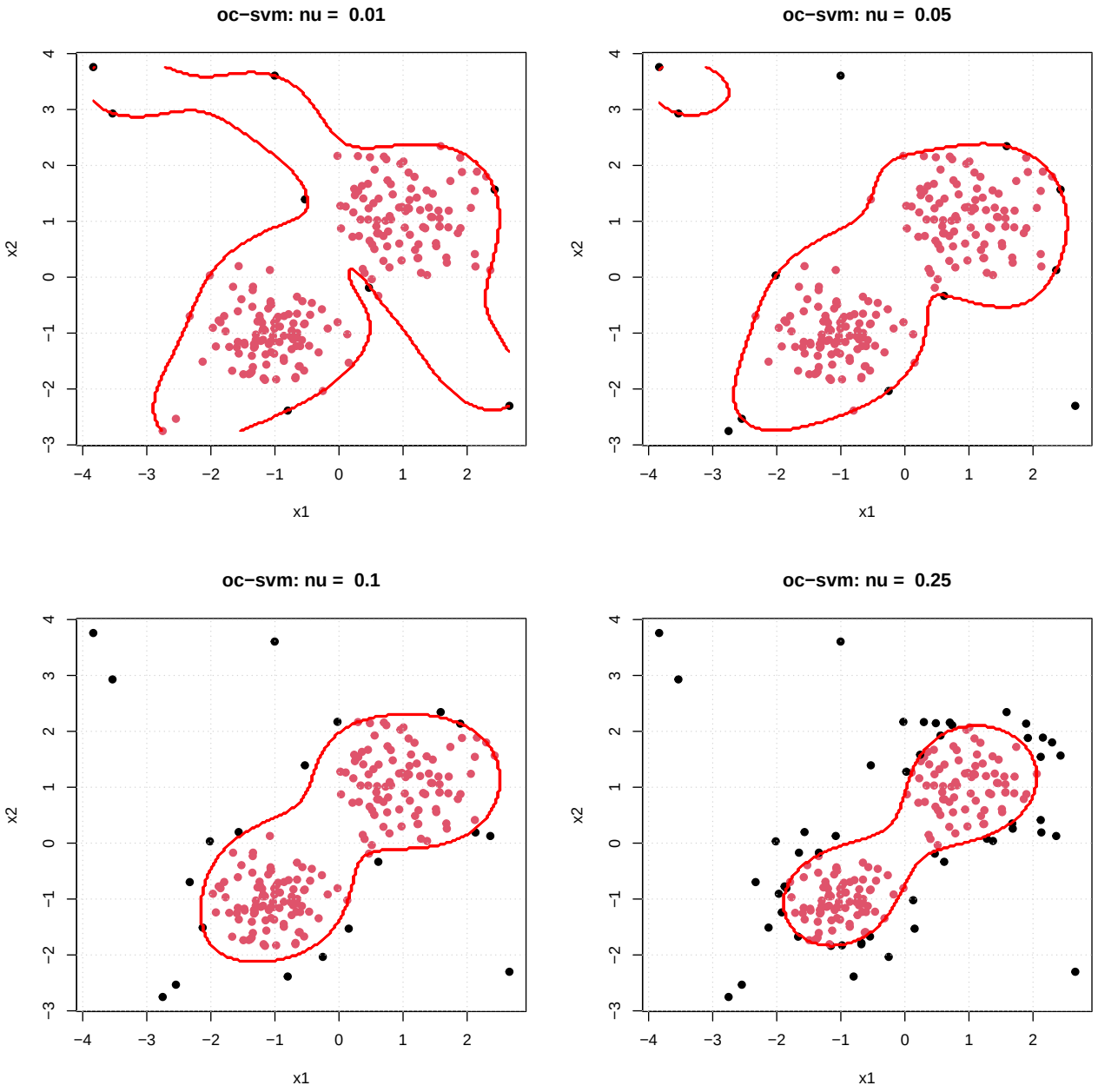


2.3 Question 4

Le code suivant permet d'évaluer l'impact du paramètre ν sur le support que l'on obtient. Il correspond directement à la proportion de points que l'on veut exclure du support.

```
#------------------------------#
# fit for several values of nu #
#------------------------------#
# define candidate nu values
nu.list = c(0.01, 0.05, 0.1, 0.25)
# plot
par(mfrow = c(2,2))
for(nu in nu.list){
  # fit model
  svm.model = svm(X, type='one-classification', nu=nu, kernel="radial")
  # get predictions
  y_pred = predict(svm.model, X)
  cat("\t- nu =", nu, ", fraction of points in support =", mean(y_pred), "\n")
  # define utility function
  svm_pred = function(x){
    return( predict(svm.model, x))
  }
  # plot
  plot_with_contours(X, svm_pred, y = factor(y_pred))
  # add title
  title( paste("oc-svm: nu = ", nu) )
}
```

```
##  - nu = 0.01 , fraction of points in support = 0.9619048
##  - nu = 0.05 , fraction of points in support = 0.9428571
##  - nu = 0.1 , fraction of points in support = 0.9047619
##  - nu = 0.25 , fraction of points in support = 0.752381
```



2.4 Question 5

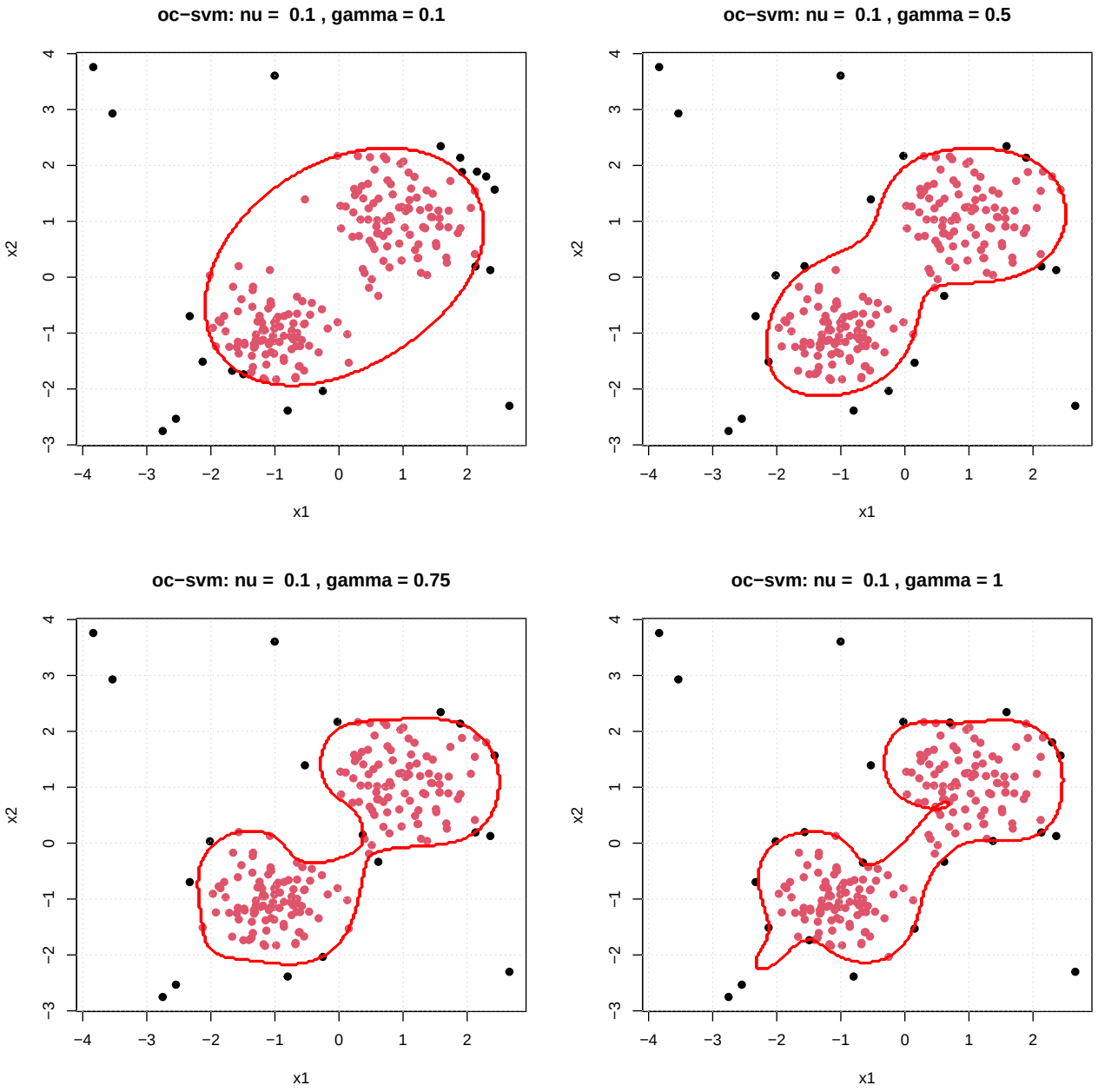
On fixe la valeur de ν à 0.1 et on évalue l'impact du paramètre γ mis en jeu de la fonction "noyau" de la SVM (se référer à la documentation de la foction `svm` pour davantage de détails).

Ce paramètre joue le rôle de l'inverse du paramètre de variance d'une loi normale. Plus il est grand, plus la fonction noyau est concentrée autour de chacun des points du jeu de données et plus le support est irrégulier.

```
#--------------------------------#
# fit for several values of gamma #
#--------------------------------#
# pick value for nu
nu = 0.1
# define candidate gamma values
gamma.list = c(0.1, 0.5, 0.75, 1.0)
```

```r
# run process
par(mfrow = c(2,2))
for(gamma in gamma.list){
  # fit model
  svm.model = svm(X, type='one-classification', nu=nu, kernel="radial", gamma = gamma)
  # get predictions
  y_pred = predict(svm.model, X)
  cat("\t- nu =", nu, ", fraction of points in support =", mean(y_pred), "\n")
  # define utility function
  svm_pred = function(x){
    return( predict(svm.model, x))
  }
  # plot
  plot_with_contours(X, svm_pred, y = factor(y_pred))
  # add title
  title( paste("oc-svm: nu = ", nu, ", gamma =", gamma) )
}
```

```
##  - nu = 0.1 , fraction of points in support = 0.9047619
##  - nu = 0.1 , fraction of points in support = 0.9047619
##  - nu = 0.1 , fraction of points in support = 0.9095238
##  - nu = 0.1 , fraction of points in support = 0.8857143
```



2.5 Question 6

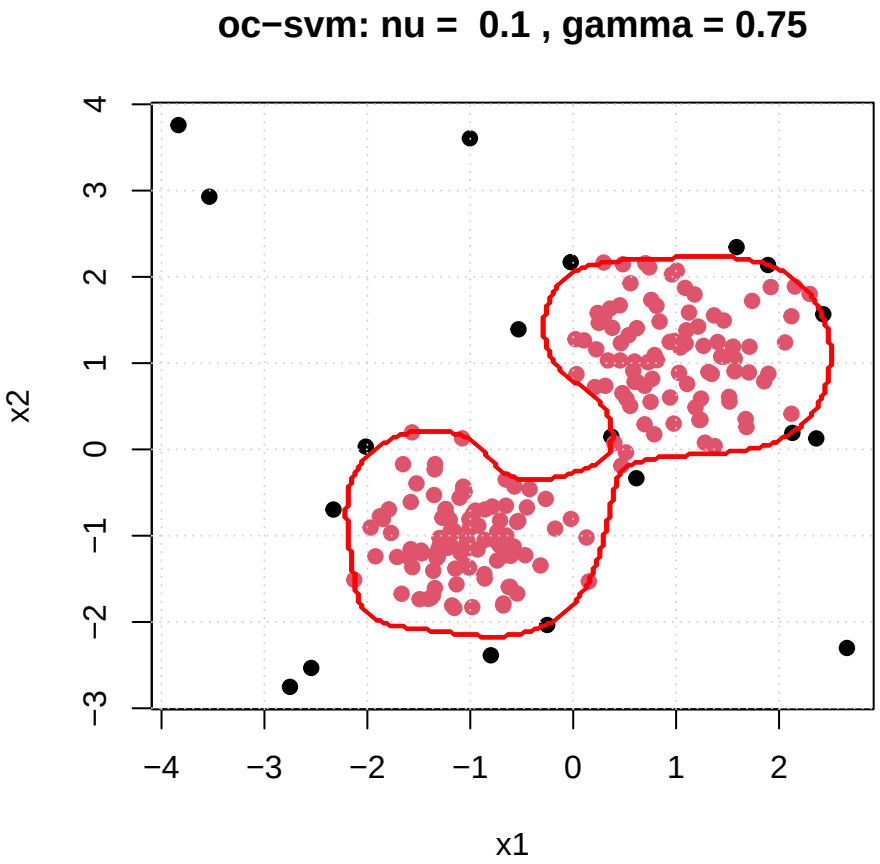
Pour comparer les résultats de la SVM à une classe à une approche basée sur une gaussienne multivariée on commence par choisir notre configuration finale, ici $\nu = 0.1$ et $\gamma = 0.75$, et on estime le support.

```
#---------------------------#
# build final configuration #
#---------------------------#
par(mfrow = c(1,1))
nu = 0.1
gamma = 0.75
# fit model
svm.model = svm(X, type='one-classification', nu=nu, kernel="radial", gamma = gamma)
# get predictions
y_pred = predict(svm.model, X)
```

```r
cat("\t- nu =", nu, ", fraction of points in support =", mean(y_pred), "\n")
```

```
##  - nu = 0.1 , fraction of points in support = 0.9095238
```

```r
# define utility function
svm_pred = function(x){
  return( predict(svm.model, x))
}
# plot
plot_with_contours(X, svm_pred, y = factor(y_pred))
# add title
title( paste("oc-svm: nu = ", nu, ", gamma =", gamma) )
```



Pour mettre en oeuvre une approche paramétrique basée sur une gaussienne multivariée, on commence par estimer ses paramètres via la fonction `mvn` du package `Mclust`. On peut ensuite utiliser la fonction `plot_with_contours` utilisée précédemment pour représenter ses lignes de niveau (il suffit pour cela d'implémenter une fonction calculant la valeur de la densité pour un point $x \in \mathbb{R}^2$). Notons qu'ici la fonction est continue (par opposition à la fonction de décision de la SVM à une classe qui valait 0 ou 1), ce qui se traduira par plusieurs lignes de niveaux.

Comme on pouvait s'en douter, on constate que les "outliers" identifiés par la SVM à une classe ne tombent pas systématiquement dans des zones où la densité de la gaussienne multivariée est faible. C'est notamment le cas des outliers se situant entre les deux "modes" de la distribution (définis par les deux sous-populations), qui ne sont par construction pas capturés par une simple gaussienne multivariée. Il faudrait ici mettre en oeuvre une estimation par un modèle de mélanges de deux gausiennes pour avoir une solution plus appropriée.

```r
#-------------------------------------#
# compare with a multivariate gaussian #
```

```r
#-------------------------------------#
par(mfrow = c(1,1))
# fit parameters with mclust
library(mclust)
```

```
## Package 'mclust' version 6.0.1
## Type 'citation("mclust")' for citing this R package in publications.
```

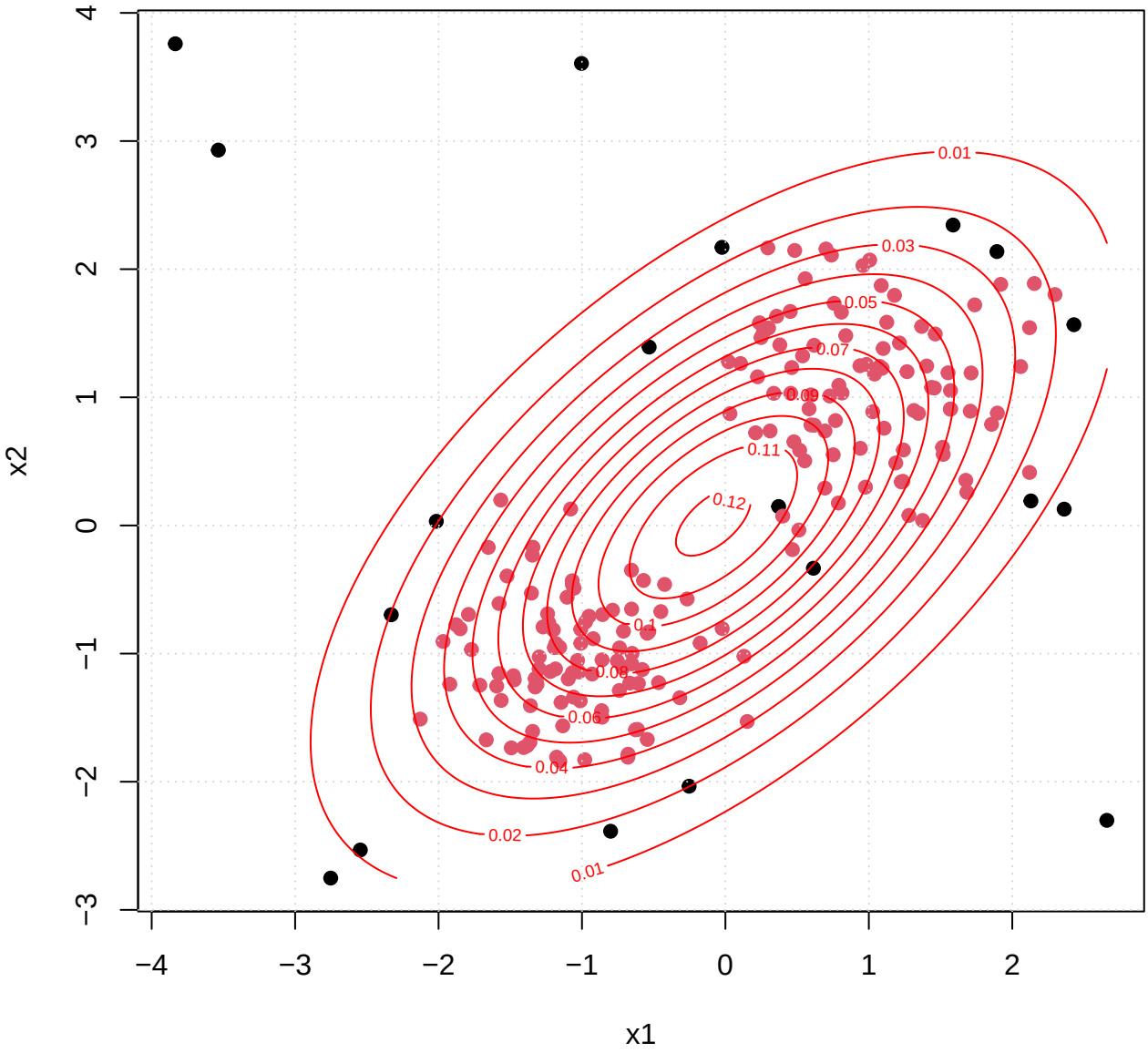
```r
fit.mvn = mvn("XXX", X)
mu = fit.mvn$parameters$mean
S = fit.mvn$parameters$variance$Sigma
cat("t- estimated mean =", fit.mvn$parameters$mean, "\n")
```

```
## t- estimated mean = -0.08413834 0.03147252
```

```r
cat("t- estimated Sigma =", fit.mvn$parameters$variance$Sigma, "\n")
```

```
## t- estimated Sigma = 1.572481 0.9634524 0.9634524 1.661439
```

```r
# define utility function
mv = function(x){
  p = length(mu)
  xt = t(x)
  return( 1/( sqrt((2*pi)^p*det(S)) ) *  exp( -0.5 * t(xt-mu) %*% solve(S) %*% (xt-mu)) )
}
# plot contours
plot_with_contours(X, mv, show.labels = TRUE, y = factor(y_pred))
```


x1
x2
−4
−3
−2
−1
0
1
2
−3
−2
−1
0
1
2
3
4
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
0.11
0.12