Détection de nouveauté par One-class SVM et Kernel PCA

Introduction

Le but de cette étude est de prédire si une tumeur est maligne ou bégnine à partir de différentes variables numériques. L'objectif est mettre en place deux méthodes: le kernel PCA et le one-class SVM.

Le one-class SVM

La méthode du one-class SVM est une technique d'apprentissage supervisé dont le but est de prédire l'appartenance d'un objet à une classe spécifique parmi plusieurs autres classes dont les observations sont rares.
Le one-class SVM repose sur l'identification de la plus petite hypersphère constituée de tous les points de données. Cette méthode est appelée "description des données vectorielles de support" (SVDD). La formulation étant restrictive et sensible aux valeurs aberrantes, une formulation flexible permettant la présence de valeurs aberrantes est formulée grâce à l'ajout de variables "slacks".

Le kernel PCA

La méthode du kernel PCA est une méthode non supersivée fondée sur l'analyse en composantes principales (ACP). Nous nous intéressons à la méthode "Kernel PCA for novelty detection" de Hoffmann.
Cette méthode consiste à projeter les données d'entrainement dans un espace F en calculant la matrice à noyau puis à transformer cette matrice. On représente ensuite les données d'entraînement dans un espace de dimension réduit grâce à une ACP puis on projette les données de test dans cet espace réduit.

Réponses aux questions

Question 1

Téléchargement des données

Question 2

Nous utilisons la fonction read.table afin de charger les données en ajoutant l'option na.strings = "?" pour signaler que les données manquantes sont indiquées par des points d'interrogation. La commande colnames permet quant à elle de définir le nom des colonnes.

```
D = read.table("breast-cancer-wisconsin.data", sep = ",", na.strings = "?")
colnames(D) = c("code_number", "clump_thickness", "cell_size_uni", "cell_shape_uni", "marginal_adhesion
```

Question 3

```
class(D) #data.frame
str(D) #nbr observations et variables + type de chaque var
head(D) #les premieres lignes de D
summary(D) #resumé de chaque variable
```

La fonction "class" permet d'obtenir la classe de l'objet D qui est un data.frame.
La fonction "str" affiche la structure interne de l'objet D. D est composé de 699 lignes et 11 variables. Toutes

les variables sont de type “int”.
La fonction “head” retourne les premières lignes du data.frame D.
La commande “summary” retourne un résumé de chaque variables de D.

Question 4

Identifions les données manquantes à l’aide de la commande “complete.cases”.

```
na_row = complete.cases(D)[complete.cases(D)==FALSE]
length(na_row) #il y a bien 16 lignes non completes
```

```
## [1] 16
```

Question 5

Modifions D afin de ne garder que les données complètes

```
D = D[complete.cases(D),]
nrow(D) #il y a bien 699-16 = 683 lignes
```

```
## [1] 683
```

Question 6

Nous définissons les variables X et y qui désignent respectivement les variables explicatives et la variable cible.

```
X = D[, 2:10] #donnees explicatives
y = D$class #variable cible
```

Question 7

Recodons la variable y en [0,1].

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
#Distribution de y
length(y[y==2]) #444
```

```
## [1] 444
```

```
length(y[y==4]) #239
```

```
## [1] 239
```

```
y = recode(y, "2" = 0, "4" = 1)
#Vérification de la distribution de y après recodage :
length(y[y==0]) #444 -> benin
```

```
## [1] 444
```

```r
length(y[y==1]) #239 -> maligne
```

```
## [1] 239
```

```r
#Le recodage s'est bien passé
```

Question 8

Découpage des variables benin et malin

```r
benin = which(y == 0, arr.ind = TRUE)
#length(benin)
malin = which(y == 1, arr.ind = TRUE)
#length(malin)
```

Question 9

Echantillonage: séparation en train/test.
Nous gardons uniquement les 200 premières observations bégnines dans l'ensemble d'entraînement.

```r
Xtrain_set = X[benin[1:200],]
Xtest_set = X[-benin[1:200],]
ytrain_set = y[benin[1:200]]
ytest_set = y[-benin[1:200]]
```

Question 10

Chargement de la librairie e1071

```r
library(e1071)
```

Question 11

Nous estimons le modele avec un noyau gaussien et de .type "one-classification"

```r
oc_svm_fit = svm(ytrain_set ~ ., data=Xtrain_set, type = "one-classification", gamma = 1/2)
```

Question 12

Les scores des observations de test sont obtenus à l'aide de la commande suivante:

```r
oc_svm_pred_test = predict(oc_svm_fit, newdata = Xtest_set, decision.values = TRUE)
#oc_svm_pred_test
```

Question 13

La premère commande correspond aux scores des observations.
La seconde correspond à la même quantité mais avec le signe inversé

Question 14

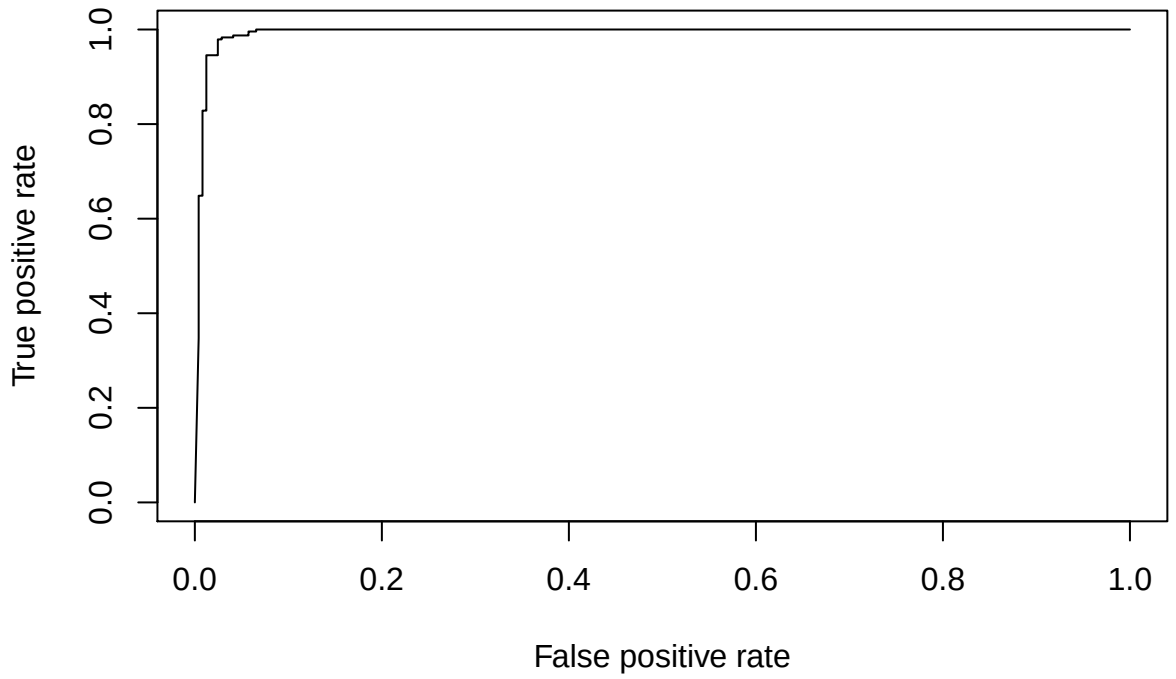
Chargement de la librairie ROCR

```r
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 4.0.3
```

Question 15

```
pred_oc_svm = prediction(oc_svm_score_test, ytest_set)
oc_svm_roc = performance(pred_oc_svm, measure = "tpr", x.measure = "fpr")
plot(oc_svm_roc)
```



On a en abscisse le taux de vrai positive qui est la sensibilité et en ordonnée le taux de faux positive qui vaut 1-la spécificité.
La courbe ROC de notre modèle est au dessus du classifieur aléatoire. Nous avons donc une bonne classification.

Question 16

Nous allons calculer l'aire sous la courbe ROC afin de commenter les performances de notre modèle

```
oc_svm_auc <- performance(pred_oc_svm, "auc")
oc_svm_auc@y.values[[1]]
```

```
## [1] 0.9932694
```

L'aire sous la courbe vaut 0.9932694, ce qui est très proche de la meilleure valeur qui vaut 1. Nous avons donc un très bon modèle de prédiction.

Question 17

```
library(kernlab) #chargement de la librairie
kernel = rbfdot(sigma = 1/8) #kernel generating function
#echatillon d'entrainement :
```

```
Ktrain = kernelMatrix(kernel, as.matrix(Xtrain_set))
```

Ktrain est la projection des données d'entraînement dans l'espace F.

Question 18

```
k2 = apply(Ktrain, 1, sum)
k3 = apply(Ktrain, 2, sum)
k4 = sum (Ktrain)
n = ncol(Ktrain) #n = nbr de ligne de Xtrain_set = nombre de donnees de l'ensemble d'apprentissage
KtrainCent = matrix (0, ncol = n, nrow = n)
for(i in 1:n){
  for(j in 1:n){
    KtrainCent[i, j] = Ktrain[i, j] - 1/n*k2[i] - 1/n*k3[j] + 1/n^2*k4
  }
}
#KtrainCent contient les coordonnees des objets sur ces axes des composantes principales
```

Les variables k2, k3 et k4 sont respectivement la 1ère somme, la 2ème somme et la double somme de l'équation (1).
Ces variables permettent de calculer KtrainCent qui est la transformation de K.

Question 19

Décomposition spectrale de KtrainCent:

```
eigen_KtrainCent = eigen(KtrainCent)
```

Question 20

Calcul des coefiicients alpha:

```
s = 80 #nombre d'axes principaux gardés
A = eigen_KtrainCent$vectors[, 1:s]%*%diag(1/sqrt(eigen_KtrainCent$values[1:s]))
#On recupère le F pour les 80 premières composantes principales
```

Question 21

```
K = kernelMatrix(kernel, as.matrix(X))
#noyau K_{i,j}
```

X représente l'échantillon total.
K est donc la matrice à noyau sur l'échantillon totale.

Question 22

Nous instancions les variables p1, p2 et p3 qui sont les trois termes composants l'equation donnée en (4).

```
n=dim(K)[1]
p1 = K
p2 = apply(K, 1, sum)
p3=sum(Ktrain)
```

Question 23

A partir des variables précédentes, nous créons le vecteur ps qui pour toute observation des données de test donne la quantité (4).

```
ps = matrix(0, ncol = n, nrow = n)
for(i in 1:n){
  for(j in 1:n){
    ps[i, j] = p1[i,j] - (2/n)*p2[i] + (1/n^2)*p3
  }
}
```

Question 24

Nous instancions les variables f1, f2, f3 et f4 qui sont les termes de l'équation (5).

```
f1 = K
f2 = apply(Ktrain, 1, sum)
f3 = apply(K, 1, sum)
f4 = sum(Ktrain)
```

Question 25

A partir des variables précédentes, nous créons le vecteur fl qui pour toute observation des données de test donne la quantité (5).

```
fl = matrix(0, ncol = s, nrow = nrow(Xtest_set))
for(i in 1:nrow(Xtest_set)){
  for(j in 1:s){
    fl[i, j] = A[i] * (f1[i,j] - 1/n*f2[i] - 1/n*f3[i] + (1/n)^2 * f4)
  }
}
```

Question 26

Enfin, nous créons le vecteur qui pour toute observation des données de test donne le score défini en (3).

```
kpca_score_test =  matrix(0, ncol = s, nrow = nrow(Xtest_set))
for(i in 1:nrow(Xtest_set)){
  for(j in 1:s){
    kpca_score_test[i,j] = ps[i,j] - fl[i,j]^2
  }
}
```

Question 27

Nous avons voulu calculer les prédictions puis afficher la courbe mais nous avons l'erreur suivantes: "Erreur : 'predictions' contains NA."
Cette erreur est surement due à une erreur dans l'évaluation d'une ou plusieurs des quantités précédentes.

```
pred = prediction(kpca_score_test, ytest_set) #Erreur : 'predictions' contains NA.
# roc = performance(pred, measure = "tpr", x.measure = "fpr")
# plot(roc, add=TRUE)
```