TD4 - Détection de nouveauté par One-class SVM et Kernel PCA

Romain Dudoit, Franck Doronzo, Marie Vachet

01/01/2022

Contents

1 Présentation de l'étude de cas

Nous nous intéressons à des données sur le cancer du sein, recueillies à l'hôpital de l'université de Wisconsin par le Dr. Wolberg. Cette étude de cas à pour objectif la détection des tumeurs malignes à partir de différentes variables explicatives numériques. Les données médicales fournies sont déséquilibrées avec 2/3 de cas bénins. Cette particularité nous pousse à utiliser des méthodes de détection de nouveauté : le One Class SVM et le kernel PCA. L'objectif de la détection de nouveauté est d'identifier l'appartenance d'un objet 'a une classe spécifique parmi plusieurs autres classes et dont les observations sont relativement rares. Dans notre cas, la classe que l'on souhaite modéliser est celle des tumeurs bégnines (puisqu'il s'agit de la classe majoritaire). Une observation maligne sera celle qui ne rentre pas dans la classe "classique" et elle est donc nouvelle. Cette étude est tirée de l'article suivant : Hoffmann, H. (2007). Kernel PCA for novelty detection. Pattern Recognition, 40(3), 863-874

2 Les méthodes utilisées

One class SVM : Le but de cette méthode, basée sur les Support Vector Machine, est de séparer les points de la classe majoritaire de l'origine de l'espace : on cherche l'hyperplan le plus éloigné de l'origine qui sépare les données de l'origine. Pour cela, la technique cherche à maximiser la distance (la marge) entre cette frontière linéaire et l'origine.

Kernel PCA : Cette fois-ci nous utilisons une méthode non supervisée de détection d'anomalies: l'Analyse en Composantes Principales (ACP) à noyau. Afin de classer les données de test, nous nous basons sur un score appelé "reconstruction error" (erreur de reconstruction) : il s'agit de la distance entre le point à tester, représenté dans un espace de redescription, et sa projection sur l'espace généré par l'ACP à noyau sur un échantillon d'appentissage.

3 Réponses aux questions

3.1 Lecture et description des données

Question 1:

Question 2:

```
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
```

La commande na.strings permet de préciser quelles valeurs concidérer comme na.

Question 3:

```
print(class(D))
```

```
## [1] "data.frame"
```

```
print(str(D))
```

```
## 'data.frame':    699 obs. of  11 variables:
##  $ code_number               : int  1000025 1002945 1015425 1016277 1017023 1017122 1018099 1018561
##  $ clump_thickness           : int  5 5 3 6 4 8 1 2 2 4 ...
##  $ cell_size_uni             : int  1 4 1 8 1 10 1 1 1 2 ...
##  $ cell_shape_uni            : int  1 4 1 8 1 10 1 2 1 1 ...
##  $ marginal_adhesion         : int  1 5 1 1 3 8 1 1 1 1 ...
##  $ single_epithetial_cell_size: int  2 7 2 3 2 7 2 2 2 2 ...
##  $ bare_nuclei               : int  1 10 2 4 1 10 10 1 1 1 ...
##  $ bland_chromatin           : int  3 3 3 3 3 9 3 3 1 2 ...
##  $ normal_nucleoli           : int  1 2 1 7 1 7 1 1 1 1 ...
##  $ mitoses                   : int  1 1 1 1 1 1 1 1 5 1 ...
##  $ class                     : int  2 2 2 2 2 4 2 2 2 2 ...
## NULL
```

La commande 'class' nous indique que D est un objet de type data.frame. La commande 'str' nous donne une synthèse des données avec le type et un extrait de chaque colonnes. De plus, on notera que nous étudierons 699 cas et 11 variables. La dernière variable est la variable cible à prédire.

```
print(head(D))
```

```
##   code_number clump_thickness cell_size_uni cell_shape_uni marginal_adhesion
## 1     1000025               5             1              1                 1
## 2     1002945               5             4              4                 5
```

```
## 3     1015425               3             1              1                 1
## 4     1016277               6             8              8                 1
## 5     1017023               4             1              1                 3
## 6     1017122               8            10             10                 8
##   single_epithetial_cell_size bare_nuclei bland_chromatin normal_nucleoli
## 1                           2           1               3               1
## 2                           7          10               3               2
## 3                           2           2               3               1
## 4                           3           4               3               7
## 5                           2           1               3               1
## 6                           7          10               9               7
##   mitoses class
## 1       1     2
## 2       1     2
## 3       1     2
## 4       1     2
## 5       1     2
## 6       1     4
```

La commande 'head' affiche les premières lignes de D.

```
summary(D)
```

```
##   code_number       clump_thickness  cell_size_uni    cell_shape_uni
##  Min.   :   61634   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.:  870688   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Median : 1171710   Median : 4.000   Median : 1.000   Median : 1.000
##  Mean   : 1071704   Mean   : 4.418   Mean   : 3.134   Mean   : 3.207
##  3rd Qu.: 1238298   3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000
##  Max.   :13454352   Max.   :10.000   Max.   :10.000   Max.   :10.000
##
##  marginal_adhesion single_epithetial_cell_size  bare_nuclei
##  Min.   : 1.000    Min.   : 1.000              Min.   : 1.000
##  1st Qu.: 1.000    1st Qu.: 2.000              1st Qu.: 1.000
##  Median : 1.000    Median : 2.000              Median : 1.000
##  Mean   : 2.807    Mean   : 3.216              Mean   : 3.545
##  3rd Qu.: 4.000    3rd Qu.: 4.000              3rd Qu.: 6.000
##  Max.   :10.000    Max.   :10.000              Max.   :10.000
##                                                NA's   :16
##  bland_chromatin  normal_nucleoli     mitoses           class
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   :2.00
##  1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:2.00
##  Median : 3.000   Median : 1.000   Median : 1.000   Median :2.00
##  Mean   : 3.438   Mean   : 2.867   Mean   : 1.589   Mean   :2.69
##  3rd Qu.: 5.000   3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:4.00
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :4.00
##
```

La commande 'summary' nous affiche quelques indicateurs statistiques comme la mediane et la moyenne des colonnes.

3.2 Séparation des donneés en "train" et "test"

Question 4:

La variable D comporte des données manquantes. Identifiez les observations comportant au moins une donnée manquante à l'aide de la commande complete.cases. Vous devez identifier 16 cas.

```
na_row = complete.cases(D)[complete.cases(D)==FALSE]
length(na_row)
```

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

La variable na_row contient un vecteur de booleéns des observations contenant des valeurs manquantes. On a bien 16 observations non completes.

Question 5:

```
D = D[complete.cases(D),]
nrow(D)
```

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

On obtient donc 699-16 = 683 lignes

Question 6:

```
X = as.matrix(D[, 2:10])
y = D$class
```

On stocke dans X les données des colonnes 2 à 10 et dans y la variable cible (class)

Question 7:

```
length(y[y==2])
```

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

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

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

```
y<-factor(y)
levels(y) <- c(0,1)
length(y[y==0])
```

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

```
length(y[y==1])
```

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

En transformant y en factor, il devient facile de changer ces niveaux en 0 et 1.

Quesion 8:

```
benin = which(y == 0, arr.ind = TRUE)
length(benin)
```

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

```
malin = which(y == 1, arr.ind = TRUE)
length(malin)
```

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

Question 9:

```
train_set = benin[1:200]
test_set = -benin[1:200]
```

Le - devant benin[1:200] pour définir l'échantillon de test permet de ne pas sélectionner les indices des 200 premières valeurs. Les données d'entrainement sont donc toutes bénines alors que les données de test sont mixtes. L'échantillon de test est composé de 483 individus.

3.3 One-class SVM

Question 10:

Question 11:

```
oc_svm_fit = svm(as.matrix(X[train_set,]), type = "one-classification", gamma = 1/2)
oc_svm_fit
```

```
##
## Call:
## svm.default(x = as.matrix(X[train_set, ]), type = "one-classification",
##     gamma = 1/2)
##
##
## Parameters:
##    SVM-Type:  one-classification
##  SVM-Kernel:  radial
##       gamma:  0.5
##          nu:  0.5
##
## Number of Support Vectors:  106
```

Notre modèle a 106 support vectors, c'est à dire 106 observations servant à la définition de l'hyperplan.

Question 12:

```
oc_svm_pred_test = predict(oc_svm_fit, newdata = X[test_set,], decision.values = TRUE)
str(oc_svm_pred_test)
```

```
##  Named logi [1:483] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  - attr(*, "names")= chr [1:483] "6" "13" "15" "16" ...
##  - attr(*, "decision.values")= num [1:483, 1] -9.07 -9.05 -9.07 -9.06 -9.07 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:483] "6" "13" "15" "16" ...
##   .. ..$ : chr "/"
```

On applique le modèle entrainé sur les données de test. La commande str permet de voir les attributs de l'objet oc_svm_pred_test:

Question 13:

```
head(attr(oc_svm_pred_test, "decision.values"))
```

```
##            /
## 6  -9.073103
## 13 -9.047651
## 15 -9.073103
## 16 -9.064651
## 19 -9.073103
## 21 -9.073103
```

On obtient les scores d'appartenance à la classe majoritaire de chaque observation.

```
oc_svm_score_test = -as.numeric(attr(oc_svm_pred_test ,"decision.values"))
head(oc_svm_score_test)
```

```
## [1] 9.073103 9.047651 9.073103 9.064651 9.073103 9.073103
```

Le signe et le type des prédictions ont été modifiés.

3.4 Courbe ROC

Question 14:

Question 15:

```
pred_oc_svm = prediction(oc_svm_score_test, y[test_set])
```

La commande 'prediction' transforme le modele en un objet de la classe utilisée par les fonctions du package ROCR.

```
oc_svm_roc = performance(pred_oc_svm, measure = "tpr", x.measure = "fpr")
str(oc_svm_roc)
```

```
## Formal class 'performance' [package "ROCR"] with 6 slots
##   ..@ x.name      : chr "False positive rate"
##   ..@ y.name      : chr "True positive rate"
##   ..@ alpha.name  : chr "Cutoff"
##   ..@ x.values    :List of 1
##   .. ..$ : num [1:273] 0 0.0041 0.0041 0.0041 0.0041 ...
```

```
##   ..@ y.values    :List of 1
##   .. ..$ : num [1:273] 0 0.347 0.377 0.385 0.393 ...
##   ..@ alpha.values:List of 1
##   .. ..$ : num [1:273] Inf 9.07 9.07 9.07 9.07 ...
```

```
head(oc_svm_roc@alpha.values[[1]])
```

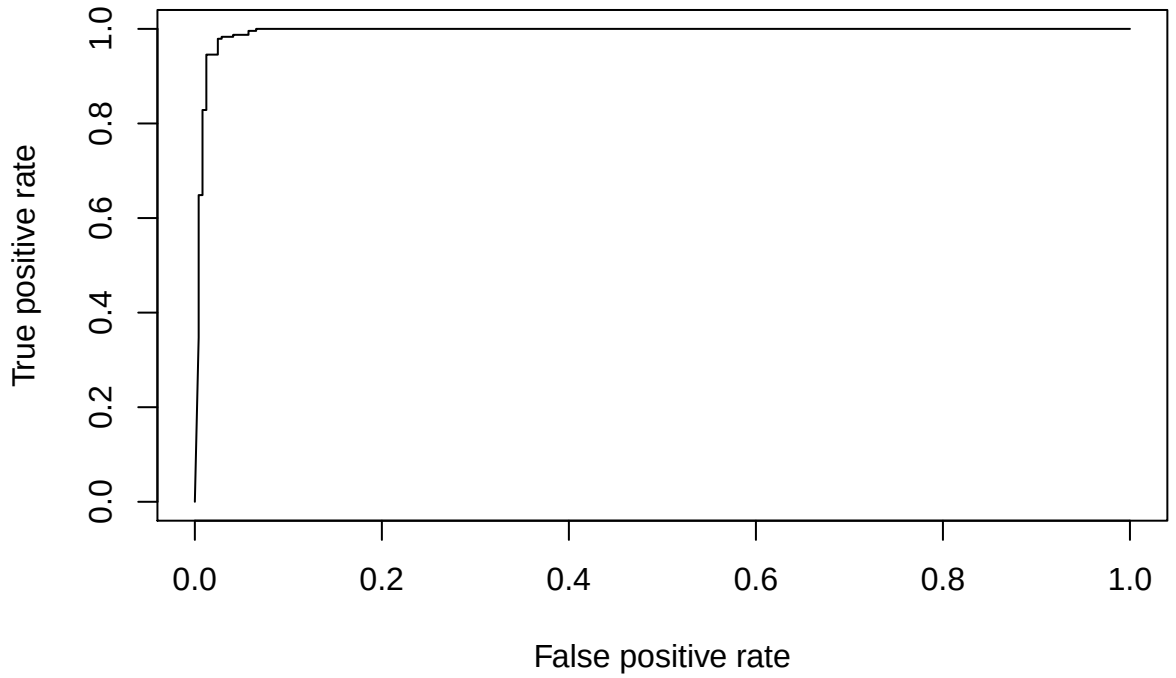
```
## [1]      Inf 9.073103 9.073103 9.073103 9.073103 9.073103
```

```
tail(oc_svm_roc@alpha.values[[1]])
```

```
## [1] -1.971688 -2.092267 -2.099773 -2.192363 -3.199670 -3.501672
```

On obtient un obet de classe performance composé des mesures testées (taux de vrais positifs et le taux de faux positifs) pour diffé©rentes valeurs de alpha (de Inf à -3.501672)

```
plot(oc_svm_roc)
```



Le graphique ci-dessus représente la courbe ROC du modèle que nous venons de créer.

Question 16:

Il semblerait que le classifieur créé soit assez performant puisque, au premier coup d'oeil, il est au dessus du classifieur aléatoire (une droite diagonale au repère) et, plus précisément, son meilleur 'point' est assez proche du point idéal (0,1).

Pour aller plus loin, nous calculons son aire sous la courbe (AUC) :

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

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

L'aire sour la courbe (AUC) est de 0.993, ce qui est tres proche de 1 (AUC du meilleur modele possible). Notre modele est donc très bon.

3.5 Kernel PCA

Question 17:

Nous procédons à la création d'un noyau gaussien: c'est un objet de type 'kernel'. A la suite de quoi nous appliquons ce noyau sur les données d'apprentissage afin de créer la matrice K (Ktrain). Ces lignes de commandes implémente cette partie du sujet : " Projeter implicitement les observations d'entraˆınement dans F en calculant la matrice 'a noyaux K de terme général K(xi, xj) = <Φ(xi), Φ(xj)>, i, j = 1, . . . , n.

Question 18:

```
k2=apply(Ktrain,1,sum)
k3=apply(Ktrain,2,sum)
k4=sum(Ktrain)
n=nrow(Ktrain)
KtrainCent=matrix(0,ncol=n,nrow=n)
```

Dans la formule 1, k2 représente la somme en ligne, k3 la somme en colonnes et k4 la somme globale. KtrainCent est initialisée avec des 0.

```
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
      }
}
```

Le code dans cette boucle transforme la matrice noyau K en un produit scalaire des vecteurs centres dans F, en utilisant la formule 1.

Question 19:

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

```
## List of 2
##  $ values : num [1:200] 31.58 16.79 11.11 5.51 4.06 ...
##  $ vectors: num [1:200, 1:200] -0.1231 -0.0427 -0.0274 -0.0427 -0.0809 ...
##  - attr(*, "class")= chr "eigen"
```

On effectue notre ACP sur la matrice KtrainCent.

Question 20:

```
s = 80
A=eigen_KtrainCent$vectors[,1:s]%*%diag(1/sqrt(
  eigen_KtrainCent$values[1:s]))
dim(A)
```

```
## [1] 200  80
```

Nous gardons les 80 permiers axes principaux. Ces lignes de code implémentent les coefficients alpha, définis par cette formule : $m = 1 /(\sqrt{\hat{}}m) * v\hat{}m$

Question 21:

```
K=kernelMatrix(kernel,X)
dim(K)
```

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

La matrice K ainsi définie correspond à la projection dans l'espace F de toutes les données.

Question 22:

```
n=683
p1=as.numeric(diag(K[test_set,test_set]))
p2 =apply(K[test_set,train_set],1,sum)
p3=sum(Ktrain)
```

On calcule le carre de la distance euclidienne entre l'origine et le vecteur dans F, dont la formule est la 4. La variable p1 correspond à k(z,z) La variable p2 correspond à la somme en ligne de K(z, xi) La variable p3 correspond à la double somme de k(xi, xj)

Question 23:

```
ps=p1-(2/n *p2)+(1/n^2 *p3)
length(ps)
```

```
## [1] 483
```

Question 24:

```
f1 = K[test_set,train_set]

f2 = apply(K[train_set,train_set], 1, sum)

f3 = apply(K[test_set,train_set], 1, sum)

f4 = sum(K[train_set,train_set])
```

Nous cherchons à obtenir f: les vecteurs comportant les coordonnées des vecteur $\Phi\tilde{}(z) - \Phi\tilde{}(0)$ sur chaque axe pour tout z appartenant à l'ensemble de test. On retrouve f1 : K(z, xi), f2 : somme en ligne de K(xi, xr), f3 : somme en ligne de K(z, xr), f4 : double somme K(xr, xs).

f4 a déjà été calculé précédemment: il s'agit de k4 et de p3.

Question 25:

```
intermediaire = f1 - (1/n * f2) - (1/n * f3) + (1/n^2 * f4)
dim(intermediaire)
```

```
## [1] 483 200
```

Nous obtenons ainsi la somme entre les grandes parenthèses de la formule 5.

```
f = intermediaire %*% A
dim(f)
```

```
## [1] 483  80
```

La matrice f est de bonnes dimensions (le nombre de lignes vaut le nombre de données tests et le nombre de colonne, le nombre d'axes principaux retenus).

Question 26:

Le score défini en (3) est composé de la différence entre 2 éléments : — le carré de la distance euclidienne entre l'origine et le vecteur dans F, — le carré de la distance euclidienne entre l'origine et le vecteur dans le sous-espace réduit de F obtenu par l'ACP à noyaux. Nous avons déjà calculé le premier terme, calculons le second à l'aide de la formule n°6.

```
eq6 = apply(f^2,1,sum)
str(eq6)
```

```
##  Named num [1:483] 0.692 0.629 0.406 0.702 0.465 ...
##  - attr(*, "names")= chr [1:483] "6" "13" "15" "16" ...
```

A présent, nous pouvons calculer le score de "reconstruction error":

```
kpca_score_test = ps - eq6
head(kpca_score_test)
```

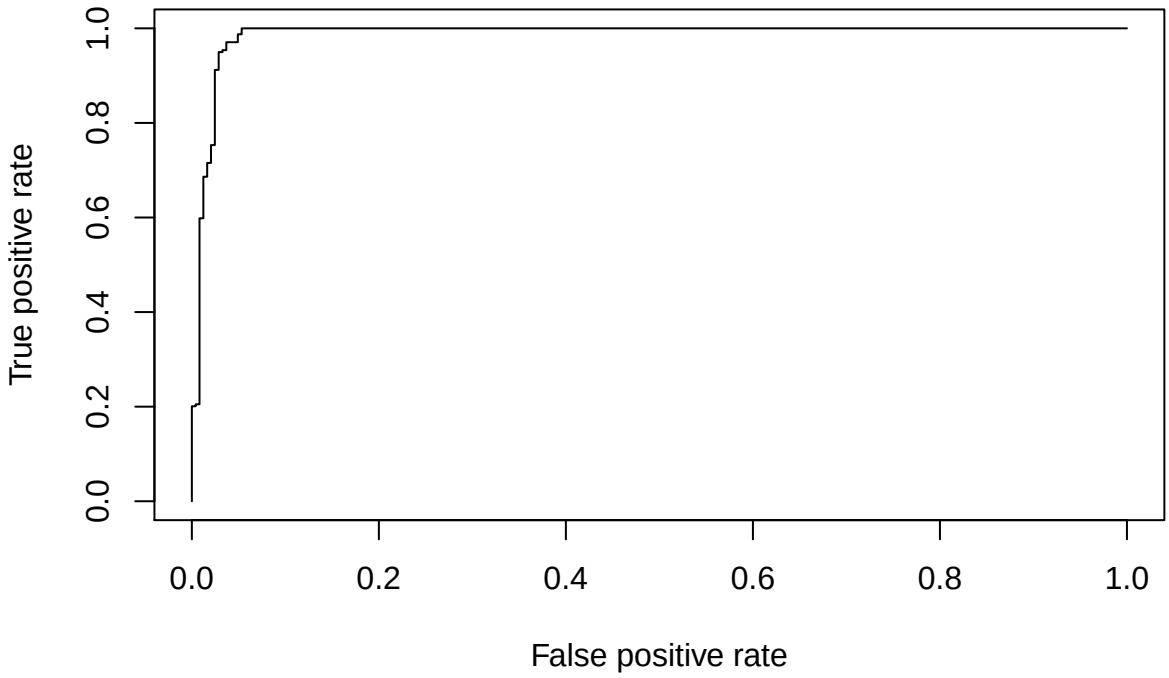
```
##         6        13        15        16        19        21
## 0.3370645 0.3870940 0.6237064 0.3262567 0.5640810 0.5533948
```

Plus ce score est grand, plus z est un point nouveau.

Question 27:

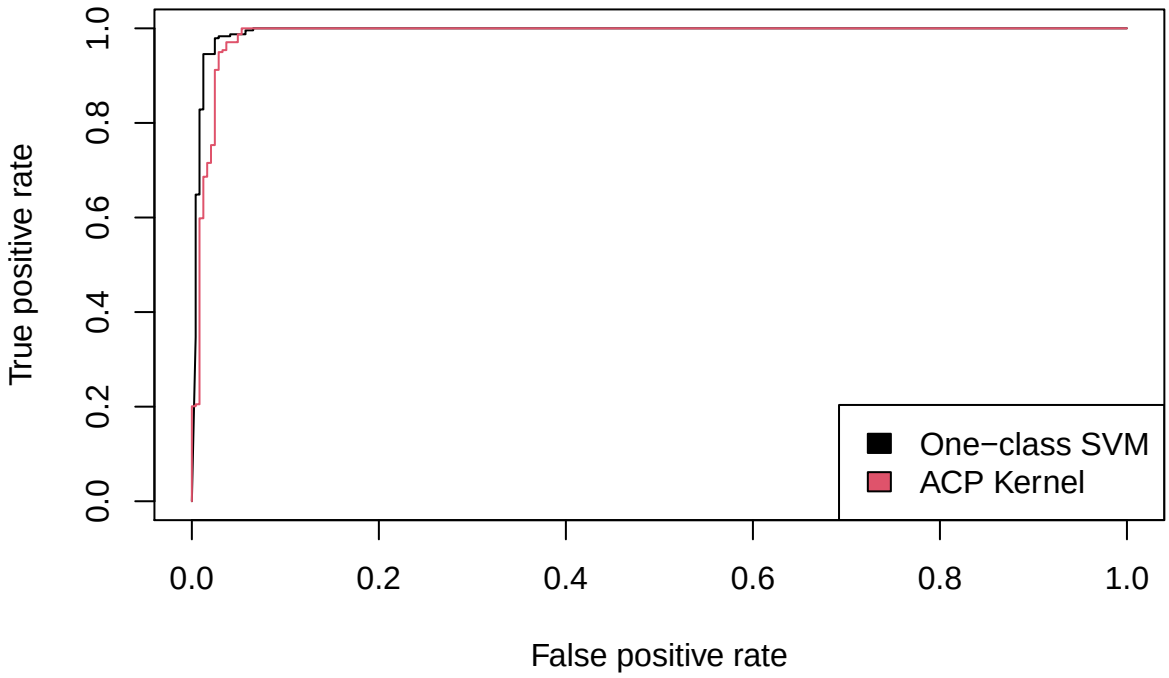
Afin d'évaluer notre modèle, nous allons à nouveau construire la courbe ROC.

```
pred_oc_kpca = prediction(kpca_score_test,y[test_set])
oc_kpca_roc = performance(pred_oc_kpca, measure = "tpr", x.measure= "fpr")
plot(oc_kpca_roc)
```



L'allure de cette courbe semble indiquer que notre modèle "kernel PCA" est plutôt bon. Comparons les deux modèles que nous avons utilisés.

```
plot(oc_svm_roc)
plot(oc_kpca_roc, add=TRUE, col=2)
legend(x = "bottomright",
       legend = c("One-class SVM", "ACP Kernel"),
       fill = 1:2)
```



On observe que les deux courbes sont très proches mais le OneClass SVM semble plus performant. Pour vérifier cette intuition, calculons l'aire sous la courbe.

```
auc_ROCR <- performance(pred_oc_kpca, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]
print(auc_ROCR)
```

```
## [1] 0.9871905
```

Nous obtenons une AUC de 0.987 ce qui est très performant mais un peu moins que pour la première méthode où nous obtenions 0.993.

4 Discussion et comparaison des modèles

Si nous devions choisir un des deux modèles construits, nous prendrions le premier. En effet, ses performances pour détecter les individus 'nouveaux' c'est à dire les tumeurs malignes, sont légèrement meilleures que l'ACP à noyau. L'important à retenir étant quand même que la méthode non supervisée est presque aussi efficace que la méthode supervisée dans ce cas.