Projet : TD4

Alexandre Rives, Jacky Madi Corodji et Elisa Frintz

2. Lecture et description des données

```
setwd("~/Documents/M2_SISE/Data_Mining_et_Apprentissage_statistique/Projet_AhPine")

D = read.table("breast-cancer-wisconsin.data", sep = ",", na.strings = "?")

class(D)
```

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

```
str(D)
```

```
## 'data.frame':    699 obs. of  11 variables:
##  $ V1 : int  1000025 1002945 1015425 1016277 1017023 1017122 1018099 1018561 1033078 1033078 ...
##  $ V2 : int  5 5 3 6 4 8 1 2 2 4 ...
##  $ V3 : int  1 4 1 8 1 10 1 1 1 2 ...
##  $ V4 : int  1 4 1 8 1 10 1 2 1 1 ...
##  $ V5 : int  1 5 1 1 3 8 1 1 1 1 ...
##  $ V6 : int  2 7 2 3 2 7 2 2 2 2 ...
##  $ V7 : int  1 10 2 4 1 10 10 1 1 1 ...
##  $ V8 : int  3 3 3 3 3 9 3 3 1 2 ...
##  $ V9 : int  1 2 1 7 1 7 1 1 1 1 ...
##  $ V10: int  1 1 1 1 1 1 1 1 5 1 ...
##  $ V11: int  2 2 2 2 2 4 2 2 2 2 ...
```

```
head(D)
```

```
##        V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11
## 1 1000025  5  1  1  1  2  1  3  1   1   2
## 2 1002945  5  4  4  5  7 10  3  2   1   2
## 3 1015425  3  1  1  1  2  2  3  1   1   2
## 4 1016277  6  8  8  1  3  4  3  7   1   2
## 5 1017023  4  1  1  3  2  1  3  1   1   2
## 6 1017122  8 10 10  8  7 10  9  7   1   4
```

```
summary(D)
```

```
##        V1                 V2               V3               V4
##  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
##
##        V5               V6               V7               V8
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 1.000   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 2.000
##  Median : 1.000   Median : 2.000   Median : 1.000   Median : 3.000
##  Mean   : 2.807   Mean   : 3.216   Mean   : 3.545   Mean   : 3.438
##  3rd Qu.: 4.000   3rd Qu.: 4.000   3rd Qu.: 6.000   3rd Qu.: 5.000
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000
##                                    NA's   :16
##        V9              V10              V11
##  Min.   : 1.000   Min.   : 1.000   Min.   :2.00
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:2.00
##  Median : 1.000   Median : 1.000   Median :2.00
##  Mean   : 2.867   Mean   : 1.589   Mean   :2.69
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:4.00
##  Max.   :10.000   Max.   :10.000   Max.   :4.00
##
```

3. Séparation des données en “train” et “test”

```
## Q4) 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.

obs_miss_val = which(complete.cases(D)==F)

# Affichage
print(obs_miss_val)
```

```
##  [1]  24  41 140 146 159 165 236 250 276 293 295 298 316 322 412 618
```

```
# Nombre d'observations comportant au moins une donnée manquante
print(length(obs_miss_val))
```

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

```
## Q5) Modifiez D de sorte à ce qu'il ne possède que des données complètes.

D = D[-obs_miss_val,]
summary(D)
```

```
##        V1                V2               V3               V4
##  Min.   :   63375   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.:  877617   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Median : 1171795   Median : 4.000   Median : 1.000   Median : 1.000
##  Mean   : 1076720   Mean   : 4.442   Mean   : 3.151   Mean   : 3.215
```

```
##  3rd Qu.: 1238705   3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000
##  Max.   :13454352   Max.   :10.000   Max.   :10.000   Max.   :10.000
##        V5               V6               V7               V8
##  Min.   : 1.00   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 1.00   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 2.000
##  Median : 1.00   Median : 2.000   Median : 1.000   Median : 3.000
##  Mean   : 2.83   Mean   : 3.234   Mean   : 3.545   Mean   : 3.445
##  3rd Qu.: 4.00   3rd Qu.: 4.000   3rd Qu.: 6.000   3rd Qu.: 5.000
##  Max.   :10.00   Max.   :10.000   Max.   :10.000   Max.   :10.000
##        V9              V10              V11
##  Min.   : 1.00   Min.   : 1.000   Min.   :2.0
##  1st Qu.: 1.00   1st Qu.: 1.000   1st Qu.:2.0
##  Median : 1.00   Median : 1.000   Median :2.0
##  Mean   : 2.87   Mean   : 1.603   Mean   :2.7
##  3rd Qu.: 4.00   3rd Qu.: 1.000   3rd Qu.:4.0
##  Max.   :10.00   Max.   :10.000   Max.   :4.0
```

```
## Q6) Stockez dans la variable X les variables explicatives qui concernent les
## colonnes 2 à 10 (inclus) de D. La variable cible sera stockée dans la variable
## y qui est donnée par la colonne 11 de D.

# Variables explicatives
X <- D[,c(2:10)]
head(X)
```

```
##   V2 V3 V4 V5 V6 V7 V8 V9 V10
## 1  5  1  1  1  2  1  3  1   1
## 2  5  4  4  5  7 10  3  2   1
## 3  3  1  1  1  2  2  3  1   1
## 4  6  8  8  1  3  4  3  7   1
## 5  4  1  1  3  2  1  3  1   1
## 6  8 10 10  8  7 10  9  7   1
```

```
# Variable cible
y <- D[,11]
head(y)
```

```
## [1] 2 2 2 2 2 4
```

```
## Q7) Recodez y de sorte à ce que les valeurs 2 deviennent des 0 (bégnine) et les
## valeurs 4 deviennent des 1 (maligne).

library(dplyr)
```

```
##
## Attachement du package : 'dplyr'
```

```
## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag
```

```
## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
y <- recode(y, '2' = 0 , '4' = 1)
head(y)
```

```
## [1] 0 0 0 0 0 1
```

```
## Q8) Stockez dans la variable benin (resp. malin) les indices des observations
## correspondant à des tumeurs bégnines (resp. maligne). Vous pourrez utiliser
## pour cela la commande which.

# Indices des observations des tumeurs benignes
benin <- which(y == 0)
head(benin)
```

```
## [1] 1 2 3 4 5 7
```

```
# Indices des observations des tumeurs malignes
malin <- which(y == 1)
head(malin)
```

```
## [1]  6 13 15 16 19 21
```

```
## Q9) Nous garderons dans l'ensemble d'entrainement uniquement les 200 premières
## observations bégnines. Stockez dans la variable train_set ces 200 observations.
## Dans l'ensemble de test vous garderez les observations bégnines qui ne sont pas
## dans l'ensemble d'entrainement et toutes les observations malignes. Vous stockerez
## les indices des observations de test dans la variable test_set.

# Indices train_set -> 200 premières observations bégnines
train_set <- benin[1:200]

# Données d'entrainement = 200 premières observations bégnines
Xtrain <- X[train_set,] ; head(Xtrain)
```

```
##   V2 V3 V4 V5 V6 V7 V8 V9 V10
## 1  5  1  1  1  2  1  3  1   1
## 2  5  4  4  5  7 10  3  2   1
## 3  3  1  1  1  2  2  3  1   1
## 4  6  8  8  1  3  4  3  7   1
## 5  4  1  1  3  2  1  3  1   1
## 7  1  1  1  1  2 10  3  1   1
```

```
ytrain <- y[train_set] ; head(ytrain)
```

```
## [1] 0 0 0 0 0 0
```

```
# Indices test_set ->  Indices des observations bégnines restantes + indices
# des observations malignes
test_set <- c(benin[201:length(benin)], malin)

# Données de test = Observations bégnines restantes + observations malignes
Xtest <- X[test_set,] ; head(Xtest)
```

```
##     V2 V3 V4 V5 V6 V7 V8 V9 V10
## 381  1  1  1  1  2  1  1  1   1
## 383  3  2  2  2  2  1  3  2   1
## 384  2  1  1  1  2  1  1  1   1
## 385  2  1  1  1  2  1  1  1   1
## 386  3  3  2  2  3  1  1  2   3
## 388  5  3  3  2  3  1  3  1   1
```

```
ytest <- y[test_set] ; head(ytest)
```

```
## [1] 0 0 0 0 0 0
```

4. One-class SVM

```
## Q10) Chargez la librairie e1071.

# install.packages("e1071")
library(e1071)
```

```
## Q11) Stockez dans la variable oc_svm_fit les résultats de l'estimation du modèle
## à partir de l'ensemble d'entrainement. Vous utiliserez pour cela la commande svm.
## Vous utiliserez un noyau gaussien de paramètre gamma=1/2, vous indiquerez que le
## type de modèle est one-classification.

oc_svm_fit <- svm(Xtrain, ytrain, type='one-classification', gamma=1/2)
summary(oc_svm_fit)
```

```
##
## Call:
## svm.default(x = Xtrain, y = ytrain, 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
##
##
##
```

```
##
## Number of Classes: 1
```

```
## Q12) A l'aide du modèle estimé stocké dans oc_svm_fit, vous prédirez les scores
## des observations de test. Pour cela, utilisez la commande predict et vous
## indiquerez de façon adéquate le paramètre decision.values.
oc_svm_pred_test = predict(oc_svm_fit, decision.values = TRUE, newdata = Xtest)
table(oc_svm_pred_test)
```

```
## oc_svm_pred_test
## FALSE  TRUE
##   383   100
```

```
## Q13) Entrez, exécutez et commentez les commandes suivantes :

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

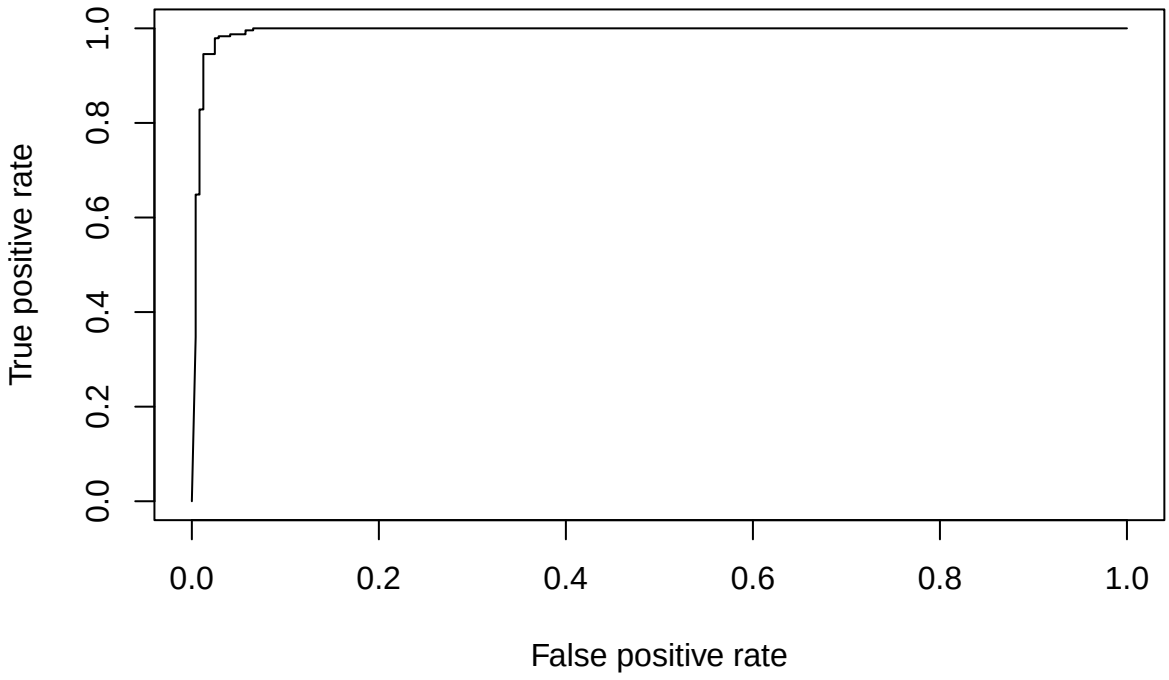
```
## [1] -0.0002052935  4.6938086964 -0.7775610703 -0.7775610703  9.0705985616
## [6]  8.1322527609
```

```
# On recupère les valeurs de decision.values dans l'objet oc_svm_score_test
# (dont on inverse le signe) : Les valeurs négatives sont les outliers et les
# valeurs positives sont les données correctements classés.
```

5. Courbe ROC

```
## Q14) Chargez la librairie ROCR.

# install.packages("ROCR")
library(ROCR)
```

```
## Q15) Entrez, exécutez et commentez les commandes suivantes :

pred_oc_svm = prediction(oc_svm_score_test, y[test_set])
oc_svm_roc = performance(pred_oc_svm, measure = "tpr", x.measure = "fpr")
plot(oc_svm_roc)
```




```
# A l'aide des prédiction effectuées à la question 13, on cherche à visualiser
# les performances de ce modèle à l'aide d'une courbe ROC. Celle-ci compare le
# taux de faux positif en fonction du taux de vrais positifs.
# Plus l'aire sous la courbe ROC est proche de 1, meilleur est le modèle.
```

```
## Q16) Commentez les performances du modèle.

# Calcul de l'AUC
auc_svm <- performance(pred_oc_svm , measure = "auc" )
print(auc_svm@y.values)
```

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

```
# Ici, on remarque que l'aire sous la courbe ROC = 0.9932694 (proche de 1).
# Le moèdle est donc très bon.
# On atteint très vite un taux de vrai positif élevé par rapport aux faux positif.
```

6. Kernel PCA

```
## Q17) Entrez, exécutez et commentez les commandes suivantes :
```

```r
# Chargement de la librairie "kernlab"
library(kernlab)

# Création du kernel
kernel = rbfdot(sigma = 1/8) ; kernel
```

```
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.125
```

```r
# Calcul de la matrice de GRAM sur les données de test
Ktrain = kernelMatrix(kernel, as.matrix(X[train_set,]))
```

```r
## Q18) Entrez, exécutez et commentez les commandes suivantes :

n = nrow(Ktrain)
k2 = apply(Ktrain,1,sum) # Somme de la matrice de GRAM par ligne
k3 = apply(Ktrain,2,sum) # Somme de la matrice de GRAM par colonne
k4 = sum(Ktrain) # Somme de la matrice de GRAM

KtrainCent = matrix(0,ncol=n,nrow=n) # Création de la nouvelle matrice de GRAM vide
for (i in 1:n){ # Pour chaque ligne
  for (j in 1:n){ # Pour chaque colonne
    KtrainCent[i,j]= Ktrain[i,j] - 1/n*k2[i] - 1/n*k3[j] + 1/n^2*k4
    # L'élément (i, j) prend la valeur de la matrice de GRAM initiale - la moyenne
    # de la ligne i - la moyenne de la colonne j + la moyenne de la matrice de GRAM
    # initiale --> vecteurs centrés dans F
  }
}
```

```r
## Q19) Procéder à la décomposition spectrale de la matrice KtrainCent en utilisant
## la commande eigen. Vous stockerez le résultat dans la variable eigen_KtrainCent.

eigen_KtrainCent = eigen(KtrainCent)
```

```r
## Q20) On choisit de garder s = 80 axes principaux. Ainsi instanciez une
## variable s=80. Les coefficients sont obtenus par la ligne de code suivante :

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

```r
## Q21) Entrez, exécutez et commentez l commande suivante :

K = kernelMatrix(kernel, as.matrix(X))
# Calcul de la matrice de GRAM sur l'ensemble des données X :
# - kernel est la fonction à utiliser pour calculer la matrice noyau créé à la question 17.
# - as.matrix(X) est la matrice des donnees X (variables explicatives).
```

```r
## Q22) A partir de la variable K et en vous inspirant des questions (et indications) précédentes,
## instanciez dans les variables p1, p2 et p3 les 3 termes composants l'équation donnée en (4).

p1 <- K
```

```r
p2 <- apply(K[,train_set], 1, sum)
p3 <- sum(K[train_set,train_set])
```

```r
## Q23) A partir des résultats précédents, stockez dans une variable ps le
## vecteur qui pour toute observation des données de test donne la quantité (4).

ps <- NULL
i <- 1

for (z in test_set){
  ps[i] = p1[z,z] -(2/n) * p2[z] + (1/n^2) * p3
  i <- i + 1
}
```

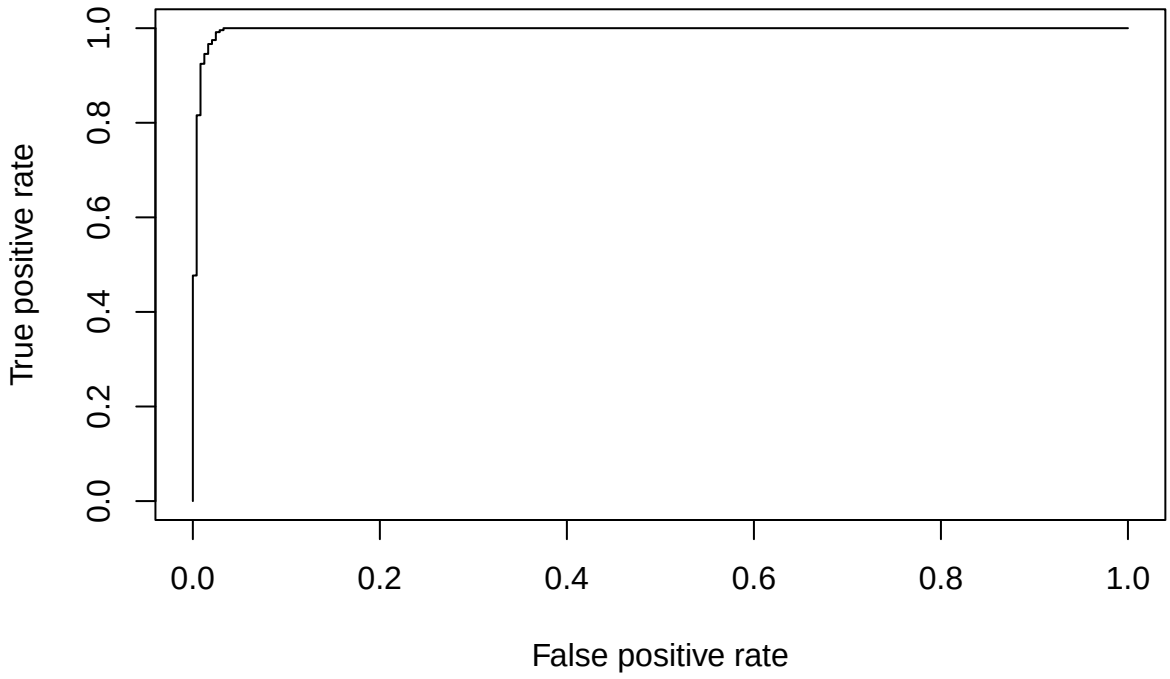
```r
## Q24) A partir de la variable K et en vous inspirant des questions précédentes,
## instanciez dans les variables f1, f2, f3 et f4 les termes successifs de (5).
## Vous remarquerez que certains termes ont déjà été calculés précédemment.

f1 <- K[test_set,train_set]
f2 <- p2[train_set]
f3 <- p2[test_set]
f4 <- p3
```

```r
## Q25) A partir des résultats précédents, stockez dans une variable fl le
## vecteur qui pour toute observation des données de test donne la quantité (5)
## (Remarque : fl représene une matrice dont le nombre de lignes vaut le nombre
## de données tests et le nombre de colonne, le nombre d'axes principaux retenus).

n2 <- length(ytest)
fl <- matrix(0, ncol = s, nrow = n2)

for (m in 1:s){
  i<-0
  for (z in test_set ){
    i <- i + 1
    var_temp <- 0
    for (i2 in 1:n){
      var_temp <- var_temp + (A[i2,m]* (f1[i,i2] - (1/n)*f2[i2] - (1/n)*f3[i] + (1/n^2)*f4))
    }
    fl[i,m] <- var_temp
  }
}
```

```r
## Q26) A partir des résultats précédents, stockez dans une variable kpca_score_test le
## vecteur qui pour toute observation des données de test donne le score défini en (3).

kpca_score_test <- ps - apply(fl^2, 1, sum)
```

```r
## Q27) Écrivez le code qui à partir de la variable kpca_score_test permet
## d'obtenir la courbe ROC. Pour comparer la courbe avec celle du one-class
## SVM vous pourrez ajouter dans la commande plot le paramètre add=TRUE.
## Commentez le graphique obtenu.
```

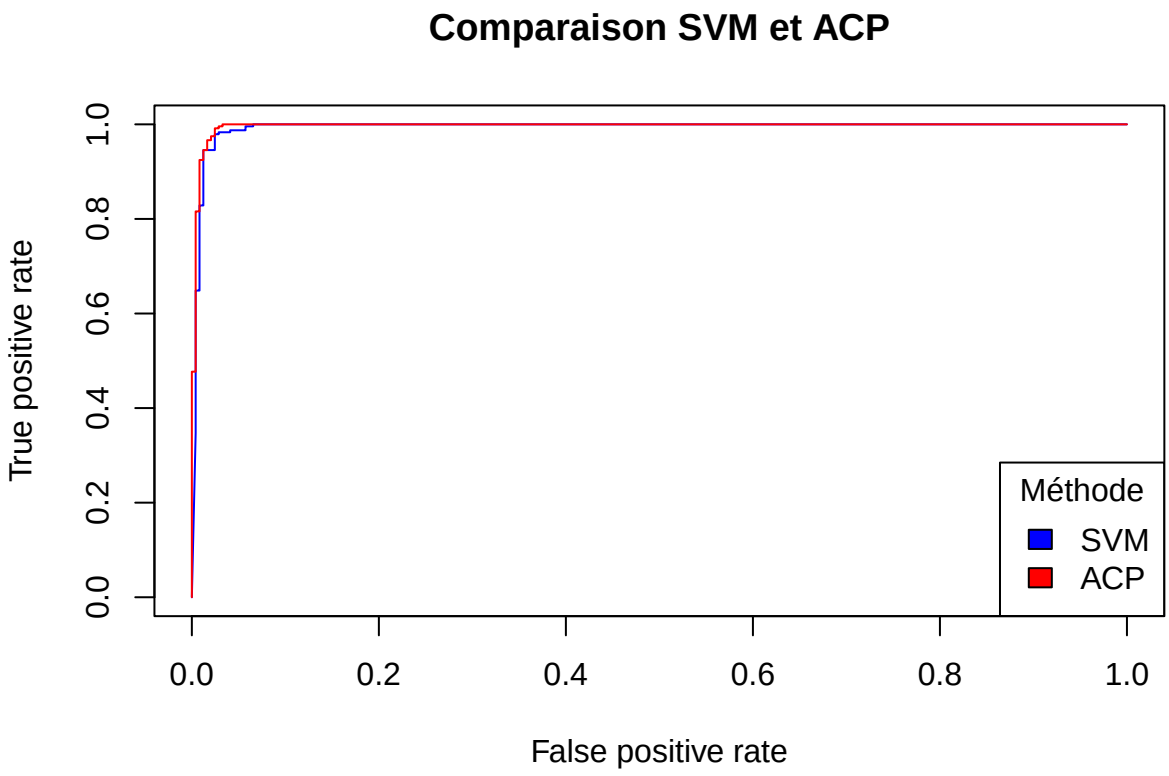
```r
# Afficher la courbe ROC de l'ACP
pred_oc_acp = prediction(kpca_score_test, y[test_set])
oc_acp_roc = performance(pred_oc_acp, measure = "tpr", x.measure = "fpr")
plot(oc_acp_roc)
```




```r
# Calcul de l'AUC de l'ACP
auc_acp <- performance(pred_oc_acp , measure = "auc" )
print(auc_acp@y.values)
```

```
## [[1]]
## [1] 0.9962789
```

```r
# Comparaison SVM et ACP - Courbe ROC
plot(oc_svm_roc, col = "blue", main = "Comparaison SVM et ACP")
plot(oc_acp_roc, add=TRUE, col = "red")
legend("bottomright", title="Méthode", c("SVM","ACP"), fill = c("blue", "red"), horiz=FALSE)
```




```
# Comparaison SVM et ACP - AUC
print(paste0("AUC des SVM = ", auc_svm@y.values))
```

```
## [1] "AUC des SVM = 0.993269428630222"
```

```
print(paste0("AUC de l'ACP = ", auc_acp@y.values))
```

```
## [1] "AUC de l'ACP = 0.996278894300021"
```

```
# Conclusion :
# D'après les AUC et la courbe ROC, dans notre cas la méthode de l'ACP est plus
# performante que la méthode des SVM.
```

Rain Australia

Frintz Elisa, Jacky Madi Corodji, Rives Alexandre

Introduction et présentation du jeu de données

Notre jeu de données présente des relevés météorologiques en fonction de diverses villes en Australie.

Dans un premier temps nous ferons des analyses préliminaires afin d'en sortir des informations pertinentes qui nous permettrons de construire au mieux notre jeu de données.

La variable cible étant "RainTomorrow", nous lancerons plusieurs algorithmes de machine learning afin de prédire s'il pleuvra le lendemain. Nous estimerons notre prédiction avec une matrice de confusion et divers estimateurs tels que la courbe ROC, le taux de reconnaissance et le rappel.

Pour information : RainToday / RainTomorrow = Yes si Rainfall > 1.

Vous trouverez les données au lien ci-contre : https://www.kaggle.com/jsphyg/weather-dataset-rattle-package

Présentation des libraries

- realdxl : permet l'import du fichier .csv.
- tidyverse : permet d'utiliser la sous libraire dplyr pour le filtrage des donnnées.
- PCAmixdata : permet d'utiliser une fonction split pour séparer les données quantitatives et qualitatives.
- ggcorrplot : librairie qui permet d'afficher un graphique en forme de matrice de corrélation.
- ggplot2 : librairie graphique.
- glmnet : package pour la régression logistique.
- caret : package utilisé pour la récupération des résultats.
- tidytable : permet d'utiliser la fonction de recodage des variables.
- e1071 : package pour les SVM.
- neuralnet : package pour les réseaux de neurones.
- pROC : permet l'affichage de la courbe ROC ainsi que de l'AUC.

Présentation des méthodes utilisées

SVM : linéaire, radial et polynomial.

Régression logistique pénalisée : ridge, lasso et elasticnet.

Réseau de neurones : 1 couche cachée avec la fonction d'activation reLU et sigmoid.

Jeux de données

Le jeu de données weatherAUS est une base de données contenant plus de 140 000 observations quotidiennes provenant de plus de 45 stations météorologiques australiennes.

Liste des variables :

- Date
- Location : Ville du relevé météorologique.
- MinTemp : température minimum de la journée.
- MaxTemp : température maximum de la journée.
- Rainfall : quantité de pluie en mm par jour.
- Evaporation : évaporation de l'eau en 24h.
- Sunshine : Le nombre d'heure d'ensolleillement par jour.
- WindGustDir : Direction des rafales de vent dans les 24h.
- WindGustSpeed : Vitesse des rafales dans les 24h.
- Temp9am : température à 9h du matin.
- RelHumid9am : humidité relative en % à 9h.
- Cloud9am : Couverture du ciel par les nuages à 9h.
- WindSpeed9am : Vitesse des vents moyens à 9h.
- Pressure9am : pression atmosphérique à 9h au niveau 0 en hectopascal.
- Temp3pm : température à 15h.
- RelHumid3pm : humidité à 15h.
- Cloud3pm : Couverture du ciel par les nuages à 15h.
- WindSpeed3pm : Vitesse des vents moyens à 15h.
- Pressure3pm : pression atmosphérique à 15h au niveau 0 en hectopascal.
- ChangeTemp : changement de température.
- ChangeTempDir : direction de changement de température.
- ChangeTempMag : magnitude du changement de température.
- ChangeWindDirect : orientation du changement du vent.

- MaxWindPeriod : période de vent maximum.
- RainToday : 1 si précipitation > 1mm dans les 24h sinon 0.
- TempRange : Différence entre la température minimum et maximum en degrés celsius.
- PressureChange : changement de pression atmosphérique.
- RainTomorrow : la variable cible, va-t-il pleuvoir demain ?

Discussion des résultats par algorithme :

SVM

- Linéaire : il s'agit du modèle avec une séparation linéaire, le coût et epsilon sont les hyper-paramètres utilisés.
- Radial : utilisé avec une fonction exponentielle. Le paramètre sigma (gamma) a été utilisé et lorsqu'il atteint une valeur de 1, les données ne collent plus du tout. Les matrices de confusion en sortie ne sont plus du tout représentatives.
- Polynomial : le degré utilisé pour le modèle polynomial est 2. La valeur de gamma, du coût et le coef0 sont les autres paramètres utilisés.

Régression logistique pénalisée

- Ridge : nous avons testé une régression pénalisée avec la norme L1. Pour cela nous mettons le paramètre alpha = 0. Nous utilisons une validation croisée pour trouver le meilleur paramètre lambda pour minimiser l'erreur (Kfold = 5). Nous comparons la valeur obtenue avec la valeur de lambda se trouvant à 1 écart type de celle ci (lambda.1se) en mesurant les performances de prédictions obtenues avec ces 2 valeurs.
- Lasso : nous avons procédé de la même façon pour la norme L2. Le paramètre alpha = 1.
- Elasticnet : enfin, nous avons comparé différentes performances pour un alpha compris entre 0.1 et 0.9. Les valeurs de rappel et de précision obtenues pour la classe positive nous ont permis de sélectionner alpha = 0.5.

De manière générale, les valeurs de lambda étaient petites (<0.1) ce qui signifie que les modèles sont peu pénalisés. La norme L1 permet une meilleure précision mais une performance en rappel moindre que la norme L2. Pour cela nous priviliegions la pénalité Lasso.

Le modèle Elasticnet donne des performances quasi-similaire au modèle Lasso.

A performance égale, il est plus courant de privilégiés Elasticnet dû à la sélection parfois aléatoire des variables avec Lasso lorsque celles-ci sont corrélées entre elle.

Nous choisissons donc la régression logistique Elasticnet avec alpha = 0.5.

Réseau de neurones

Pour les réseaux de neurones nous avons testé les 2 fonctions d'activation suivantes : reLU et Sigmoid, une seule couche cachée est comprise dans le réseau.

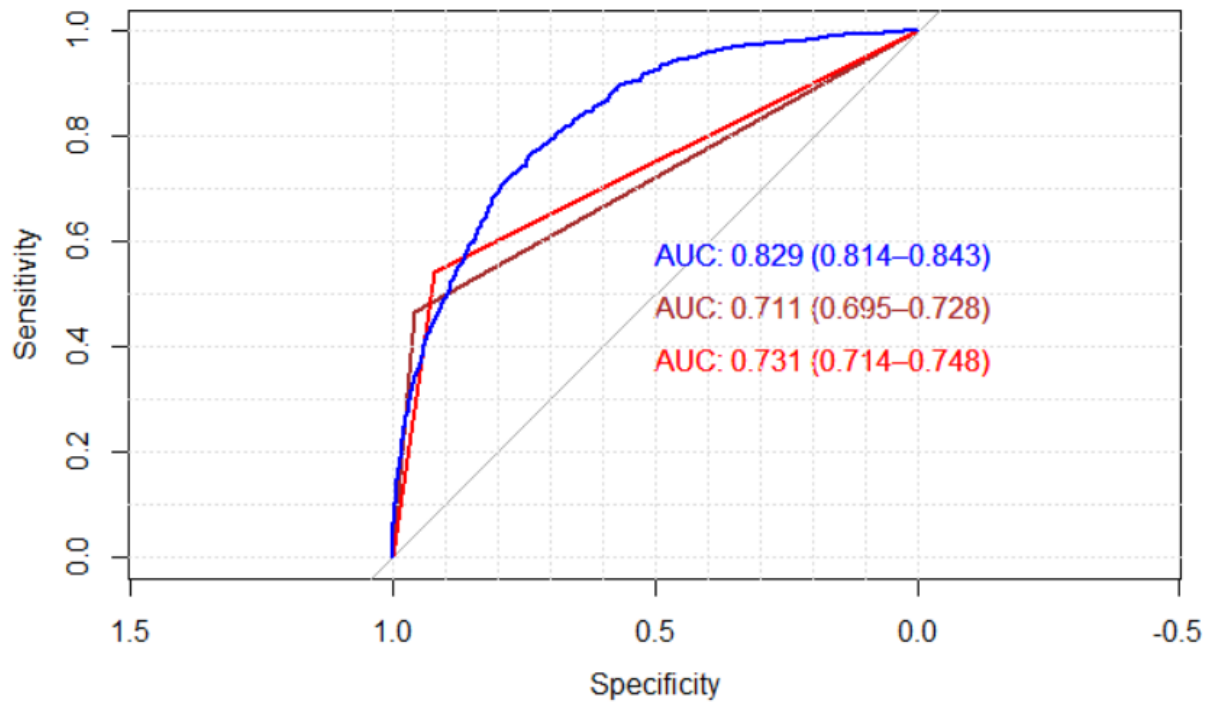
Ces résultats sont les plus performants (taux de reconnaissance, rappel et précision) notamment avec l'utilisation de la fonction d'activation "sigmoïd".

Algorithme le plus performant :

L'algorithme nous apportant les résultats les plus performants sont le réseau de neurones avec un rappel supérieur à 50% ce qui permet de prédire plus d'une fois sur 2 un jour de pluie correctement.

L'accuracy est aussi supérieure.

La courbe ROC montre aussi que ce modèle est celui qui fut le plus robuste car l'air sous la courbe (AUC) est la plus élevée.



Courbe bleu : réseau de neurones

Courbe marron : elasticnet

Courbe rouge : SVM radial

Conclusion :

Prédire la météo en Australie relève du défi. En effet ce pays, considéré comme le plus grand pays du continent de l'Océanie comporte de nombreux climats (désertique, tropical, tempéré…) ce qui rend le contexte des plus difficiles.