Rapport de mini-projet d'apprentissage artificiel

Deux méthodes d'apprentissage artificiel sur la base de données *Internet Advertisement Data Set*

Matthieu DOGNIAUX et Baptiste COLIN

9 juin 2016

Introduction

Pour notre projet, nous avons choisi d'étudier la base de données nommée « Internet Advertisement Data Set », qui peut être trouvée sur le site UCI à l'URL suivante :

`http://archive.ics.uci.edu/ml/datasets/Internet+Advertisements`

La base de données représente un ensemble de possibles publicités sur des pages Internet. Les attributs encodent la géométrie de l'image (si disponible) ainsi que la présence ou non d'expressions contenues dans l'URL de l'image, son alternative textuelle ou son *anchor text*. Le dernier attribut précise si oui ou non l'élément est une publicité. L'enjeu est ici de réussir à prédire, à l'aide d'algorithmes d'apprentissage exercés sur cette base de données, si une image est une publicité ou non.

Il s'agit donc d'un problème de classification, qui va nous faire recourir à des algorithmes d'apprentissage supervisé. Nous allons ici comparer les performances de deux méthodes sur la base de données choisie : celles des forêts aléatoires d'une part, et celles des séparateurs à vaste marge (SVM, *Support Vector Machine*) d'autre part. Ce travail a été réalisé grâce à des algorithmes écrits dans le langage R et exécutés sous RStudio.

Dans ce rapport, nous allons détailler :
— Le pré-traitement effectué pour rendre les données exploitables
— L'implémentation des méthodes avec R
— Nos méthodes de recherche des paramètres optimaux
— Les résultats obtenus et ainsi discuter des possibles améliorations

Pré-traitement des données

28% des entrées de la base de données que nous avons choisie comportent des valeurs manquantes, au niveau des attributs donnant la géométrie de l'image. La première étape est donc de nettoyer la base de données en supprimant les éléments avec des valeurs manquantes.

Dans le fichier texte fournit par l'UCI, les valeurs manquantes sont représentées par un « ? » en lieu et place de la valeur. A l'aide d'un éditeur de texte, nous avons remplacé tous les « ? » par la chaine de caractères « 56743 ». Nous avions bien entendu vérifié au préalable que la chaine de caractères « 56743 » n'apparaissait nulle part ailleurs dans le fichier (aucune donnée n'a donc été écrasée par cette modification).

Les seules valeurs manquantes étant dans les trois premières colonnes (géométrie de l'image), les lignes de code suivantes ont permis de réassigner l'encodage de ces valeurs manquantes, puis de supprimer les lignes qui en contenaient.

```
mydata$V1[mydata$V1==56743] <- NA
mydata$V2[mydata$V2==56743] <- NA
mydata$V3[mydata$V3==56743] <- NA
mydatax <- na.omit(mydata)
```

Nous avons pu ôter les lignes de la base de données contenant des valeurs manquantes, et ainsi créer un *dataframe* nommé `mydatax` qui sera utilisable par la suite. Il contient finalement 2369 éléments.

Implémentation des méthodes dans R

Forêts aléatoires

La bibliothèque `RandomForest` de R permet directement d'utiliser la méthode des forêts aléatoires. Etant donné deux *datasets* `trainData` et `testData`, un bloc d'apprentissage, de prédiction et comparaison des résultats par une matrice de confusion s'écrit de la manière suivante dans R :

```
modelForest <- randomForest(V1559~.,
                            data = trainData,
                            importance=TRUE,
                            keep.forest=TRUE,
    )    # apprendre
predicted <- predict( modelForest, testData ) # prédire
actual <- testData$V1559
confusionMatrix(as.factor(predicted), as.factor(actual)) # comparer
```

Ces quelques lignes de code R permettent à elles seules de mettre en œuvre une méthode d'apprentissage artificiel par une forêt aléatoire.

SVM

La bibliothèque `kernlab` de R permet directement d'utiliser des SVM. Etant donné deux *datasets* `trainData` et `testData`, un bloc d'apprentissage, de prédiction et comparaison des résultats par une matrice de confusion s'écrit de la manière suivante dans R :

```
modelSVM <- ksvm(V1559~., data=trainData, type='C-svc',
                     kernel='vanilladot',
                     C=100, scale=c() ) # apprendre
predicted <- predict( modelSVM, testData ) # prédire
actual <- testData$V1559
confusionMatrix(as.factor(predicted), as.factor(actual)) #comparer
```

Ces quelques lignes de code R permettent à elles seules de mettre en œuvre une méthode d'apprentissage artificiel par des SVM.

Méthode de recherche des paramètres optimaux : validation croisée

Si les deux méthodes d'apprentissage artificiel choisies sont très faciles à déployer avec R, on ne peut pas se contenter d'entraîner un unique modèle sur la base de données choisie, de le tester et d'affirmer avoir atteint la précision maximale réalisable.

Pour avoir une meilleure estimation de la précision de généralisation des modèles d'apprentissage, que ce soit pour la méthode des forêts aléatoires ou des SVM, nous avons décidé d'appliquer la *10 fold cross-validation*. Par ailleurs, nous avonc couplé cette validation croisée avec une exploration de l'espace de certains des hyper-paramètres des méthodes d'apprentissage, afin de déterminer lesquels maximisent les performances de celles-ci.

Voici comment, dans un premier temps, nous avons découpé la base de données. Les lignes sont mélangées puis un premier dixième de celle-ci, `DataTest_absolu`, est mis de côté et sert à tester le modèle gagnant dans l'espace des quelques hyper-paramètres choisis par méthode. Ensuite, les neuf dixièmes restant de `DataValidation` sont délimités en neufs bouts. On peut ensuite appliquer la *10 fold cross-validation*.

```
#Randomly shuffle the data
mydatax<-mydatax[sample(nrow(mydatax)),]

# Splittage des données : on garde un test absolu (1/10)
# et un ensemble de validation (9/10)
test1 <- rep(F, 2369)
test1[1:2132] = T
DataValidation <- mydatax[test1,]
test2 <- rep(F, 2369)
test2[2133:2369] = T
DataTest_absolu <- mydatax[test2,]

#Create 9 equally size folds
folds <- cut(seq(1,nrow(DataValidation)),breaks=9,labels=FALSE)

for(j in 1:9){  #Perform 10 fold cross validation

    #Segement your data by fold using the which() function
```

```
    testIndexes <- which(folds==j,arr.ind=TRUE)
    testData <- DataValidation[testIndexes, ]
    trainData <- DataValidation[-testIndexes, ]

    #Use the test and train data partitions
    ++++++++++++++++++++++++++++++
     apprendre, prédire, comparer, stocker resultats
    ++++++++++++++++++++++++++++++

  } # fin boucle sur les 9 segments de cross-validation
```

A l'issue d'une *10 fold cross-validation*, on considère que la précision de généralisation d'un modèle pour des hyper-paramètres donnés est la moyenne des précisions obtenues pour chacun des 9 tests effectués sur une même validation croisée.

Forêts aléatoires

Pour les forêts aléatoires, nous avons utilisé la *10 fold cross-validation* afin de déterminer quel nombre d'arbres améliore le plus les performances de la méthode. Dans la pratique, la boucle précédente qui réalise la validation croisée est imbriquée dans une autre boucle qui parcourt certaines valeurs de nombre d'arbres : toutes les centaines entre 200 et 1000 inclus [1].

La figure 1 donne les précisions de généralisation calculées par *10 fold cross-validation* sur la base de donnée choisie pour différentes valeurs du nombre d'arbres.

D'après la figure 1, on préfère donc utiliser un apprentissage par forêt aléatoire qui contient 400 ou 500 arbres (égalité parfaite des précisions de généralisation).

Finalement, avec 400 arbres, on obtient une corrélation de 97,88% sur la partition de la base de données `DataTest_absolu`.

SVM

Pour les SVM, nous avons utilisé la *10 fold cross-validation* afin de déterminer le couple noyau-C qui donne la meilleure précision de généralisation. Dans la pratique, la boucle précédente qui réalise la validation croisée est imbriquée dans une boucle qui parcourt des valeurs de C (de $0,01$ à 100, logarithmiquement). Ces deux boucles imbriquées sont parcourues 4 fois, chaque fois pour un noyau différent (`vanilladot` - linéaire, `polydot` - polynomial, `rbfdot` - gaussien, `besseldot` - bessel) [2].

La figure 2 donne les précisions de généralisation calculées par *10 fold cross-validation* sur la base de donnée choisie pour différentes valeurs de l'hyper-paramètre C et pour différents noyaux.

1. Le script entier de l'algorithme utilisé avec la méthode de forêt aléatoire se trouve en annexe
2. Le script entier de l'algorithme utilisé avec les SVM se trouve en annexe

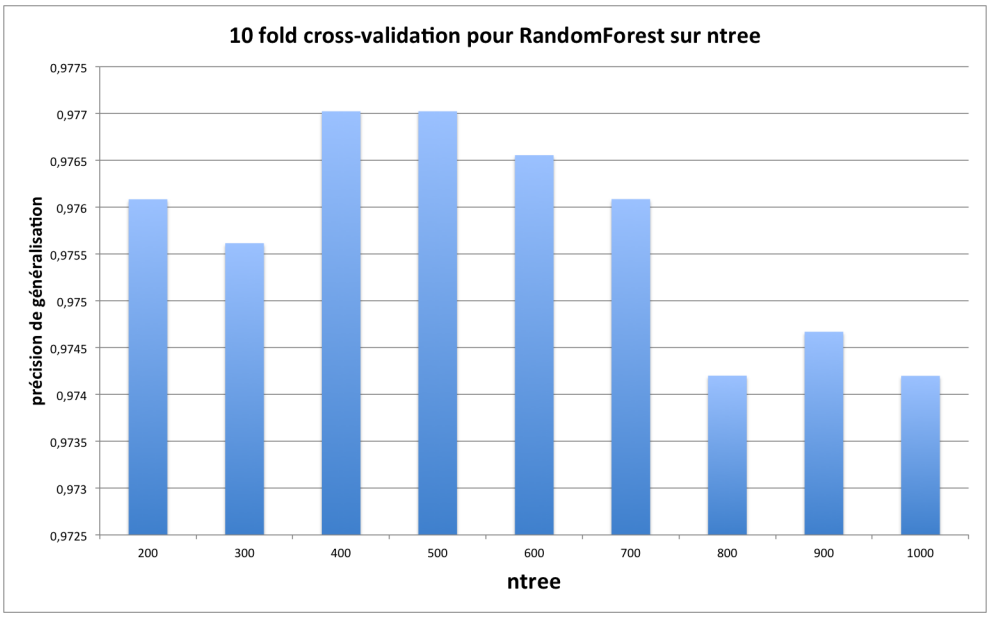

FIGURE 1 – Précision de généralisation pour différentes valeurs du nombre d'arbres pour une méthode de forêt aléatoire

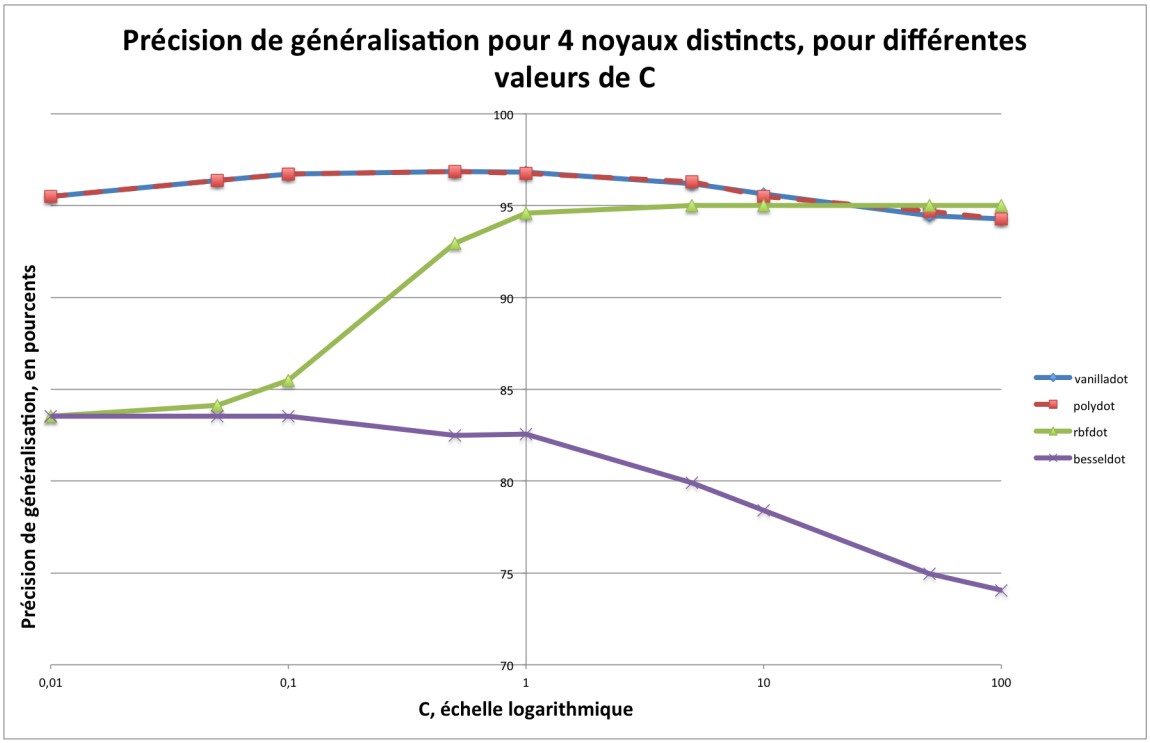

FIGURE 2 – Précision de généralisation pour différentes valeurs de C et pour différents noyaux

D'après la figure 2, on préfère donc utiliser un modèle d'apprentissage à base de SVM avec une valeur de C de 0,5 et un noyau polynomial (`polydot`), de justesse plus performant que le noyau linéaire.

Finalement, avec l'hyper-paramètre C fixé à 0,5 et un noyau polynomial `polydot`, on obtient une corrélation de 97,46% sur la partition de la base de données `DataTest_absolu`.

Conclusion

Pour conclure, que l'on utilise des forêts aléatoires ou des SVM, on obtient des précisions de généralisation équivalentes pour les hyper-paramètres optimaux. Celles-ci sont supérieures à 97%, qui est l'ordre de grandeur mentionné dans la publication [3] citée sur la page Internet de la base de donnée.

Cependant, si l'on considère le temps de calcul pour arriver à ces résultats, on préfèrera l'utilisation des SVM aux forêts aléatoires. En effet, la courbe sur l'influence du nombre d'arbres a requis plus de 15 heures de calcul cumulées (l'apprentissage est très long) tandis que les valeurs du graphe pour les SVM ont été produites en 45 minutes.

Des pistes peuvent également être évoquées pour encore améliorer les résultats. En effet, pour assurer encore plus la précision de généralisation, il est possible d'implémenter un algorithme de *nested cross-validation* qui aurait joué sur les combinaisons possibles entre le `DataTest_absolu` et le `DataValidation`. Par ailleurs, le parti a été pris de supprimer les lignes avec des valeurs manquantes. Utiliser ces lignes avec des méthodes spécifiques aurait permis de garder une base de donnée complète et peut-être de généraliser plus l'apprentissage.

3. N. Kushmerick (1999). « Learning to remove Internet advertisements », 3rd Int Conf Autonomous Agents.

Annexes

Forêt aléatoire : RandomForest_10CrossValid_ntree.R

```
# ======================================================================
# ouvertures du fichier initial, nettoyage des données, initialisations des tableaux
# ======================================================================

setwd("~/Documents/Travail_Mines_de_Paris/2A/Apprentissage_Artificiel/Projet")
mydata <- read.csv("ad5.txt", header=FALSE)
mydata$V1[mydata$V1==56743] <- NA
mydata$V2[mydata$V2==56743] <- NA
mydata$V3[mydata$V3==56743] <- NA
mydatax <- na.omit(mydata)
mydatax$V1559 <- as.factor(mydatax$V1559)

# libraires utilisées
library(randomForest)
library(caret)
library(plyr)

# liste de ntree parcourus
Taille <- list()
Taille[1] = as.integer(200)
Taille[2] = as.integer(300)
Taille[3] = as.integer(400)
Taille[4] = as.integer(500)
Taille[5] = as.integer(600)
Taille[6] = as.integer(700)
Taille[7] = as.integer(800)
Taille[8] = as.integer(900)
Taille[9] = as.integer(1000)

# tableau des scores
Score_accuracy <- array(0.0, c(9, 9))

#Randomly shuffle the data
```

```
mydatax<-mydatax[sample(nrow(mydatax)),]

# Splittage des données : on garde un test absolu (1/10)
et un ensemble de validation (9/10)
test1 <- rep(F, 2369)
test1[1:2132] = T
DataValidation <- mydatax[test1,]
test2 <- rep(F, 2369)
test2[2133:2368] = T
DataTest_absolu <- mydatax[test2,]

#Create 9 equally size folds
folds <- cut(seq(1,nrow(DataValidation)),breaks=9,labels=FALSE)

# ========================================================================
# 10-fold cross-validation sur la taille de la forêt
# ========================================================================

# On va faire de la cross-validaion
for (i in 1:9) {  # Boucle sur les valeurs de Taille (table Taille)

  for(j in 1:9){  #Perform 10 fold cross validation
    #Segement your data by fold using the which() function
    testIndexes <- which(folds==j,arr.ind=TRUE)
    testData <- DataValidation[testIndexes, ]
    trainData <- DataValidation[-testIndexes, ]

    #Use the test and train data partitions
    modelForest <- randomForest(V1559~.,
                          data = trainData,
                          importance=TRUE,
                          keep.forest=TRUE,
                          ntree = as.integer(Taille[i])
    )
    predicted <- predict( modelForest, testData )
    actual <- testData$V1559

    # analyser les resultats
    cm <- confusionMatrix(as.factor(predicted), as.factor(actual))
    # stocker l'accuray
    overall.accuracy <- cm$overall['Accuracy']
    Score_accuracy[i,j] = overall.accuracy
    print(cat(i,j))
    print(Score_accuracy[i,j])
  } # fin boucle sur les 9 segments de cross-validation
```

```
} # fin Boucle sur Taille

# tableau des scores moyennés pour
Score_accuracy_moy <- array(0.0, c(9))

# remplissage du tableau moyenné
for (i in 1:9) {
  for (j in 1:9) {
    Score_accuracy_moy[i] = Score_accuracy_moy[i] + Score_accuracy[i,j]
  }
  Score_accuracy_moy[i] = Score_accuracy_moy[i]/9
}

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# on lit dans la console que le gagnant est ntree = 400 ou 500 (égalité)
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# ========================================================================
# Utilisation du modèle gagnant
# ========================================================================

#Use the test and train data partitions
modelForest <- randomForest(V1559~.,
                            data = DataValidation,
                            importance=TRUE,
                            keep.forest=TRUE,
                            ntree = 400
)
predicted <- predict( modelForest, DataTest_absolu )
actual <- DataTest_absolu$V1559

# analyser les resultats
confusionMatrix(as.factor(predicted), as.factor(actual))
```

SVM : SVM_10CrossValid_4kernels.R

```
# ========================================================================
# ouvertures du fichier initial, nettoyage des données, initialisations des tableaux
# ========================================================================

setwd("~/Documents/Travail_Mines_de_Paris/2A/Apprentissage_Artificiel/Projet")
mydata <- read.csv("ad5.txt", header=FALSE)
mydata$V1[mydata$V1==56743] <- NA
mydata$V2[mydata$V2==56743] <- NA
```

```
mydata$V3[mydata$V3==56743] <- NA
mydatax <- na.omit(mydata)
mydatax$V1559 <- as.factor(mydatax$V1559)

# libraires utilisées
library(kernlab)
library(caret)
library(plyr)

# liste de C parcourus
Cv <- list()
Cv[1] = 0.01
Cv[2] = 0.05
Cv[3] = 0.1
Cv[4] = 0.5
Cv[5] = 1
Cv[6] = 5
Cv[7] = 10
Cv[8] = 50
Cv[9] = 100

# tableau des scores
Score_accuracy <- array(0.0, c(4, 9, 9))

#Randomly shuffle the data
mydatax<-mydatax[sample(nrow(mydatax)),]

# Splittage des données : on garde un test absolu (1/10)
# et un ensemble de validation (9/10)
test1 <- rep(F, 2369)
test1[1:2132] = T
DataValidation <- mydatax[test1,]
test2 <- rep(F, 2369)
test2[2133:2368] = T
DataTest_absolu <- mydatax[test2,]

#Create 9 equally size folds
folds <- cut(seq(1,nrow(DataValidation)),breaks=9,labels=FALSE)

# ========================================================================
# kernel vanilladot
# ========================================================================

# On va faire de la cross-validaion
for (i in 1:9) {  # Boucle sur les valeurs de C (table Cv)
```

```
  for(j in 1:9){  #Perform 10 fold cross validation
    #Segement your data by fold using the which() function
    testIndexes <- which(folds==j,arr.ind=TRUE)
    testData <- DataValidation[testIndexes, ]
    trainData <- DataValidation[-testIndexes, ]

    #Use the test and train data partitions
    modelSVM <- ksvm(V1559~., data=trainData, type='C-svc',
                     kernel='vanilladot',
                     C=Cv[i], scale=c() )
    predicted <- predict( modelSVM, testData )
    actual <- testData$V1559

    # analyser les resultats
    cm <- confusionMatrix(as.factor(predicted), as.factor(actual))
    # stocker l'accuray
    overall.accuracy <- cm$overall['Accuracy']
    Score_accuracy[1,i,j] = overall.accuracy
    print(cat(1,i,j))
    print(Score_accuracy[1,i,j])
  } # fin boucle sur les 9 segments de cross-validation

} # fin Boucle sur C

# tableau des scores moyennés pour
Score_accuracy_moy <- array(0.0, c(4, 9))

# remplissage du tableau moyenné
for (i in 1:9) {
  for (j in 1:9) {
    Score_accuracy_moy[1,i] = Score_accuracy_moy[1,i] + Score_accuracy[1,i,j]
  }
  Score_accuracy_moy[1,i] = Score_accuracy_moy[1,i]/9
}

# ======================================================================
# kernel rbfdot
# ======================================================================

#créer le kernel - valeurs d'hyper-parametres de kernel de l'exemple
rbf <- rbfdot(sigma=0.1)

# On va faire de la cross-validaion
for (i in 1:9) {  # Boucle sur les valeurs de C (table Cv)
```

```
  for(j in 1:9){  #Perform 10 fold cross validation
    #Segement your data by fold using the which() function
    testIndexes <- which(folds==j,arr.ind=TRUE)
    testData <- DataValidation[testIndexes, ]
    trainData <- DataValidation[-testIndexes, ]

    #Use the test and train data partitions
    modelSVM <- ksvm(V1559~., data=trainData, type='C-svc',
                     kernel=rbf,
                     C=Cv[i], scale=c() )
    predicted <- predict( modelSVM, testData )
    actual <- testData$V1559

    # analyser les resultats
    cm <- confusionMatrix(as.factor(predicted), as.factor(actual))
    # stocker l'accuray
    overall.accuracy <- cm$overall['Accuracy']
    Score_accuracy[2,i,j] = overall.accuracy
    print(cat(2,i,j))
    print(Score_accuracy[2,i,j])
  } # fin boucle sur les 9 segments de cross-validation

} # fin Boucle sur C

# tableau des scores moyennés pour
for (i in 1:9) {
  for (j in 1:9) {
    Score_accuracy_moy[2,i] = Score_accuracy_moy[2,i] + Score_accuracy[2,i,j]
  }
  Score_accuracy_moy[2,i] = Score_accuracy_moy[2,i]/9
}

# ========================================================================
# kernel polydot
# ========================================================================

#créer le kernel - valeurs d'hyper-parametres de kernel de l'exemple
pld <- polydot(degree = 1, scale = 1, offset = 1)

# On va faire de la cross-validaion
for (i in 1:9) {  # Boucle sur les valeurs de C (table Cv)

  for(j in 1:9){  #Perform 10 fold cross validation
    #Segement your data by fold using the which() function
```

```
    testIndexes <- which(folds==j,arr.ind=TRUE)
    testData <- DataValidation[testIndexes, ]
    trainData <- DataValidation[-testIndexes, ]

    #Use the test and train data partitions
    modelSVM <- ksvm(V1559~., data=trainData, type='C-svc',
                     kernel=pld,
                     C=Cv[i], scale=c() )
    predicted <- predict( modelSVM, testData )
    actual <- testData$V1559

    # analyser les resultats
    cm <- confusionMatrix(as.factor(predicted), as.factor(actual))
    # stocker l'accuray
    overall.accuracy <- cm$overall['Accuracy']
    Score_accuracy[3,i,j] = overall.accuracy
    print(cat(3,i,j)) #pour vérifier avancement du calcul
    print(Score_accuracy[3,i,j])
  } # fin boucle sur les 9 segments de cross-validation

} # fin Boucle sur C

# tableau des scores moyennés pour
for (i in 1:9) {
  for (j in 1:9) {
    Score_accuracy_moy[3,i] = Score_accuracy_moy[3,i] + Score_accuracy[3,i,j]
  }
  Score_accuracy_moy[3,i] = Score_accuracy_moy[3,i]/9
}

# ========================================================================
# kernel tanhdot
# ========================================================================

#créer le kernel - valeur de sigma = valeur de l'exemple
bsld <- besseldot(sigma = 1, order = 1, degree = 1)

# On va faire de la cross-validaion
for (i in 1:9) {  # Boucle sur les valeurs de C (table Cv)

  for(j in 1:9){  #Perform 10 fold cross validation
    #Segement your data by fold using the which() function
    testIndexes <- which(folds==j,arr.ind=TRUE)
    testData <- DataValidation[testIndexes, ]
    trainData <- DataValidation[-testIndexes, ]
```

```
    #Use the test and train data partitions
    modelSVM <- ksvm(V1559~., data=trainData, type='C-svc',
                     kernel=bsld,
                     C=Cv[i], scale=c() )
    predicted <- predict( modelSVM, testData )
    actual <- testData$V1559

    # analyser les resultats
    cm <- confusionMatrix(as.factor(predicted), as.factor(actual))
    # stocker l'accuray
    overall.accuracy <- cm$overall['Accuracy']
    Score_accuracy[4,i,j] = overall.accuracy
    print(cat(4,i,j)) #pour vérifier avancement du calcul
    print(Score_accuracy[4,i,j])
  } # fin boucle sur les 9 segments de cross-validation

} # fin Boucle sur C

# tableau des scores moyennés pour
for (i in 1:9) {
  for (j in 1:9) {
    Score_accuracy_moy[4,i] = Score_accuracy_moy[4,i] + Score_accuracy[4,i,j]
  }
  Score_accuracy_moy[4,i] = Score_accuracy_moy[4,i]/9
}

# ========================================================================
# finitions
# ========================================================================

#Dilaltion des moyennes
Score_accuracy_moy = 100*Score_accuracy_moy #passage en pourcents
Score_accuracy_moy

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# on lit dans la console que le gagnant est polydot avec C = 0.5
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# ========================================================================
# Utilisation du modèle gagnant
# ========================================================================

#Use the test and train data partitions
modelSVM <- ksvm(V1559~., data=DataValidation, type='C-svc',
```

```
                kernel=pld,
                C=0.5, scale=c() )
predicted <- predict( modelSVM, DataTest_absolu )
actual <- DataTest_absolu$V1559
confusionMatrix(as.factor(predicted), as.factor(actual))
```