SY19 projet 2

Dataset Phonemes

Tout d'abord, je traiterai d'abord la base de données. J'ai effectué une analyse en composantes principales sur les données et sélectionné les 100 principaux facteurs pour former une nouvelle base de données. Ensuite, j'ai normalisé les deux ensembles de données et les ai divisés en ensembles d'entra??nement et de test selon un rapport de 2:1.

```
data_phonemes <- read.csv("phoneme_train.txt",header = TRUE,sep=" ",row.names = NULL)
data_phonemes <- data_phonemes[2:length(data_phonemes)]
# générer 2 base de donnée pour train
x<-data_phonemes
pca <- prcomp(x[,-257])
X<-pca$x[,1:100]
data_pca <- data.frame(X)
data_pca['y']<-data_phonemes['y']
#scale
data_phonemes[,-257]<-apply(data_phonemes[,-257], 2, scale)
data_pca[,-101]<-apply(data_pca[,-101], 2, scale)
n <- nrow(data_phonemes)
train<-sample(1:n,round(2*n/3))
data_train <- data_phonemes[train,]
data_test <- data_phonemes[-train,]
data_pca_train <- data_pca[train,]
data_pca_test <- data_pca[-train,]
ntest<-nrow(data_test)
```

Ensuite, nous avons utilisé de nombreux modèles pour apprendre et tester sur les données. Pour les données originales, nous avons utilisées : KNN, Naive Bayes, LDA, Random Forests, MDA. Pour les données de l'analyse en composantes principales, nous avons utilisé les modèles suivants: KNN, LDA, QDA, Random Forests.

```
#knn
library(FNN)
ERR.knn<-matrix(0,20,2)
for(k in 1:20){
  knn.class<-knn(data_train[,-257],
                 data_test[,-257],data_train$y,k=k)
  knn.class_1<-knn(data_pca_train[,-101],
                   data_pca_test[,-101],data_pca_train$y,k=k)
  ERR.knn[k,1]<-mean(data_test$y != knn.class)
  ERR.knn[k,2]<-mean(data_pca_test$y != knn.class_1)
}

err.knn.min=min(ERR.knn)
print(err.knn.min)
```

```
## [1] 0.09333333
```

```r
k_min=which(ERR.knn[,1]==min(ERR.knn[,1]),arr.ind=TRUE)

# Naive Bayes
library(naivebayes)
fit.nb<- naive_bayes(as.factor(y)~.,data=data_train)
pred.nb<-predict(fit.nb,newdata=data_test,type="class")
perf <-table(data_test$y,pred.nb)
print(perf)
```

```
##      pred.nb
##        aa  ao dcl  iy  sh
##   aa   90  35   0   0   0
##   ao   33 124   0   0   0
##   dcl   0   0 114  12   0
##   iy    0   0   0 193   1
##   sh    0   0   0   0 148
```

```r
err.nb <-1-sum(diag(perf))/ntest
print(err.nb)
```

```
## [1] 0.108
```

```r
# LDA
library(MASS)
fit.lda<- lda(y~.,data=data_train)
pred.lda<-predict(fit.lda,newdata=data_test)
perf <-table(data_test$y,pred.lda$class)
print(perf)
```

```
##
##        aa  ao dcl  iy  sh
##   aa   93  32   0   0   0
##   ao   17 140   0   0   0
##   dcl   0   0 122   4   0
##   iy    0   0   1 193   0
##   sh    0   0   0   0 148
```

```r
err.lda <- 1-sum(diag(perf))/ntest
print(err.lda)
```

```
## [1] 0.072
```

```r
# LDA_pca
fit.lda_pca<- lda(y~.,data=data_pca_train)
pred.lda_pca<-predict(fit.lda_pca,newdata=data_pca_test)
perf_pca <-table(data_pca_test$y,pred.lda_pca$class)
print(perf_pca)
```

```
##
##        aa  ao dcl  iy  sh
##   aa   93  32   0   0   0
##   ao   19 138   0   0   0
##   dcl   0   0 123   3   0
##   iy    0   0   1 193   0
##   sh    0   0   0   0 148
```

```r
err.lda_pca <- 1-sum(diag(perf_pca))/ntest
print(err.lda_pca)
```

```
## [1] 0.07333333
```

```r
# QDA
fit.qda<- qda(y~.,data=data_pca_train)
pred.qda<-predict(fit.qda,newdata=data_pca_test)
perf <-table(data_pca_test$y,pred.qda$class)
print(perf)
```

```
##
##        aa  ao dcl  iy  sh
##   aa   53  70   0   1   1
##   ao   17 139   0   1   0
##   dcl   0   0 115  11   0
##   iy    0   0   1 193   0
##   sh    0   0   0   0 148
```

```r
err.qda <-1-sum(diag(perf))/ntest
print(err.qda)
```

```
## [1] 0.136
```

```r
#performance très mal


# Random forests
library(randomForest)
fit.RF<-randomForest(as.factor(y) ~ .,data=data_train,mtry=16)
pred.RF<-predict(fit.RF,newdata=data_test,type="class")
err.RF<-1-mean(data_test$y==pred.RF)
print(err.RF)
```

```
## [1] 0.08
```

```r
#Random forests en pca (performance mieux)
fit.RF.pca<-randomForest(as.factor(y) ~ .,data=data_pca_train,mtry=10)
pred.RF.pca<-predict(fit.RF.pca,newdata=data_pca_test,type="class")
err.RF.pca<-1-mean(data_pca_test$y==pred.RF.pca)
print(err.RF.pca)
```

```
## [1] 0.08533333
```

```r
#mda
library(mda)
fit.mda<- mda(y~.,data=data_train)
pred.mda<-predict(fit.mda,newdata=data_test)
perf <-table(data_test$y,pred.mda)
print(perf)
```

```
##      pred.mda
##        aa  ao dcl  iy  sh
```

```
##   aa   92  33   0   0   0
##   ao   25 132   0   0   0
##   dcl   0   0 123   3   0
##   iy    0   0   1 193   0
##   sh    0   0   0   0 148
```

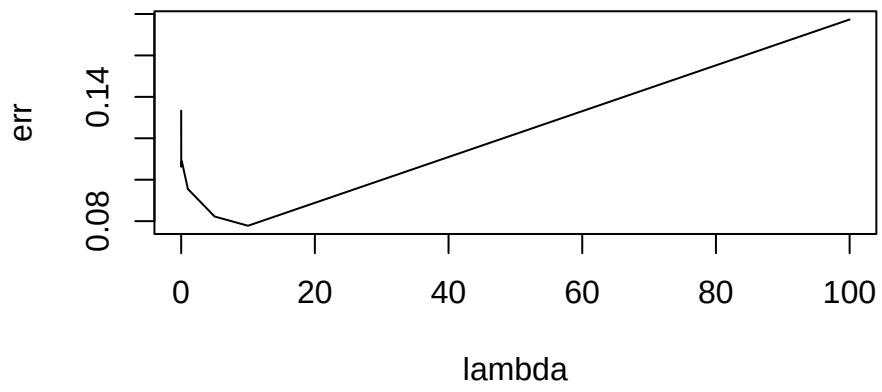
```
pred.mda <-1-sum(diag(perf))/ntest
print(pred.mda)
```

```
## [1] 0.08266667
```

Parmi eux, nous pouvons voir que KNN, LDA, LDA en PCA, MDA, Random forests ont une plus grande précision.

Ensuite, nous apprenons et testons sur le modèle 'Neural Network'. Nous devons déterminer le meilleur choix pour le paramètre "decay", nous utilisons donc une 5-fold validation croisée. Dans ce processus, nous avons sélectionné une valeur "size" plus petite pour accélérer la vitesse d'apprentissage.

```
library(nnet)
library("caret")
K<-5
ntrain<-nrow(data_phonemes)
lambda<-c(0,0.01,0.1,1,5,10,100)
N<-length(lambda)
err<-matrix(0,N)
folds<-createFolds(y=data_phonemes[,257],k=K)
for(i in (1:N)){
  for(k in (1:K)){
    nn<- nnet(as.factor(y) ~ ., data=data_phonemes[-folds[[k]],],size=5,
              decay=lambda[i],trace=FALSE,MaxNWts = 100000,maxit = 1000)
    pred<-predict(nn,newdata=data_phonemes[folds[[k]],],type="class")
    err[i]<-err[i]+ (1-mean(data_phonemes[folds[[k]],]$y==pred))
  }
  err[i]<-err[i]/K
}
plot(lambda,err,type='l')
```



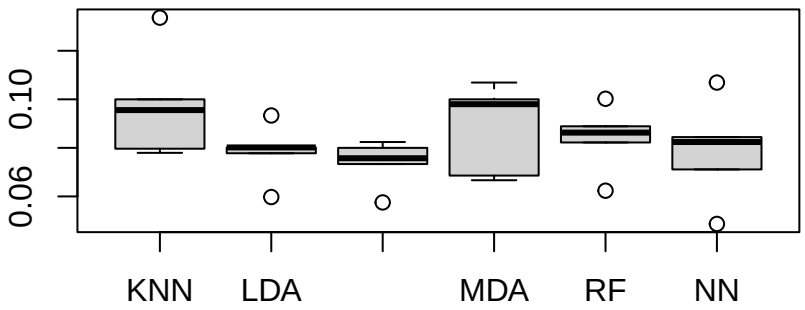
```r
lambda.opt<-lambda[which.min(err)] # Best decay coefficient
lambda.opt
```

```
## [1] 10
```

Par la suite, nous avons effectué une 5-fold validation croisée avec les modèles précédemment sélectionnés avec de faibles taux d'erreur et 'Neural Network' pour sélectionner le modèle final à utiliser.

```r
K<-5
ntrain<-nrow(data_phonemes)
err<-matrix(0,6,K)
folds<-createFolds(y=data_phonemes[,257],k=K)
for(k in (1:K)){
  #knn
  knn.class<-knn(data_phonemes[-folds[[k]],-257],
                 data_phonemes[folds[[k]],-257],data_phonemes[-folds[[k]],257],k=k_min)
  err[1,k]<-mean(data_phonemes[folds[[k]],257] != knn.class)

  # LDA
  fit.lda<- lda(y~.,data=data_phonemes[-folds[[k]],])
  pred.lda<-predict(fit.lda,newdata=data_phonemes[folds[[k]],])
  err[2,k]<-mean(data_phonemes[folds[[k]],257] != pred.lda$class)

  # LDA_pca
  fit.lda.pca<- lda(y~.,data=data_pca[-folds[[k]],])
  pred.lda.pca<-predict(fit.lda.pca,newdata=data_pca[folds[[k]],])
  err[3,k]<-mean(data_pca[folds[[k]],101] != pred.lda.pca$class)

  #mda
  fit.mda<- mda(y~.,data=data_phonemes[-folds[[k]],])
  pred.mda<-predict(fit.mda,newdata=data_phonemes[folds[[k]],])
  err[4,k]<-mean(data_phonemes[folds[[k]],257] != pred.mda)

  #random-foret
  fit.RF<-randomForest(as.factor(y) ~ .,data=data_phonemes[-folds[[k]],],importance=TRUE,mtry=16)
  pred.RF<-predict(fit.RF,newdata=data_phonemes[folds[[k]],],type="class")
  err[5,k]<-mean(data_phonemes[folds[[k]],257] != pred.RF)

  #nn
  nn<- nnet(as.factor(y) ~ ., data=data_phonemes[-folds[[k]],],size=10,
            decay=lambda.opt,trace=FALSE,MaxNWts = 100000,maxit = 300)
  pred<-predict(nn,newdata=data_phonemes[folds[[k]],],type="class")
  err[6,k]<-mean(data_phonemes[folds[[k]],257] != pred)
}
ERR <- data.frame(t(err))
names(ERR) <- c("KNN","LDA","LDA_PCA","MDA","RF","NN")
```

```r
boxplot(ERR)
```



On peut voir que LDA en PCA, Random Forests et Neural Network ont une plus grande précision. étant donné que Neural Network peut continuer à augmenter le nombre d'unités dans la couche cachée pour améliorer la précision du modèle, nous avons choisi d'utiliser Neural Network comme modèle final et augmenté la valeur du paramètre 'size'.

Dataset Robotics

Faire avec ksvm

On va analyser le dataset Robotics. Il y a 4000 observations avec 8 prédicteurs et 1 variable de réponse. On peut essayer de réduire le nombre de prédicteurs afin d'obtenir un modèle plus simple.

```
library(ggplot2)
library(e1071)
library(splines)
library(kernlab)
library(nnet)
library(dplyr)
library(caret)
set.seed(123)
data <- read.table("robotics_train.txt", header = TRUE, sep = " ")
X <- subset(data, select = -y)
y <- data$y
fit <- lm(y ~ .,data)
summary(fit)
```

On remarque que tous les prédicteurs semblent signifiants pour prédire y. On a pu le confirmer avec regsubsets le R2 et le BIC n'est pas assez important pour supprimer des prédicteurs en backward et forward.

Après cette première analyse du dataset, on peut s'intéresser aux méthodes que l'on va utiliser pour prédire de nouvelles valeurs. Afin de faire une première analyse rapidement, on va sélectionner un base d'apprentissage et une base de tests par l'approche hold-out sur plusieurs base d'apprentissage différentes et de tailles différentes.

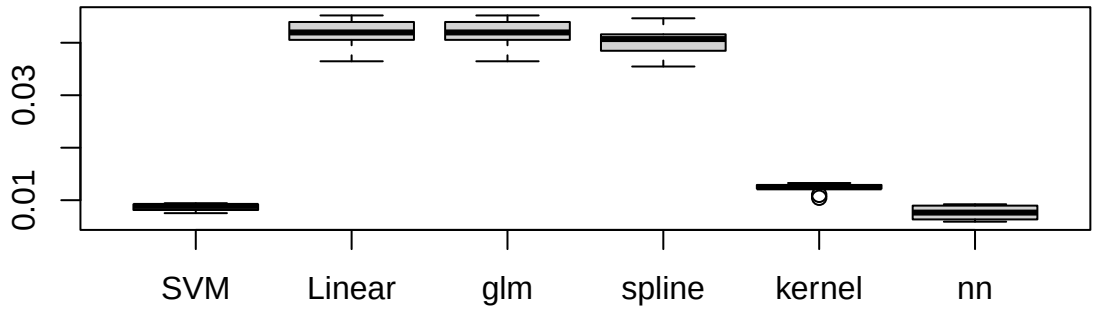
```
# Diviser les données en un ensemble d'entraînement et un ensemble de test
train_index <- sample(1:nrow(data), 0.8 * nrow(data))
data.train <- data[train_index, ]
data.test <- data[-train_index, ]
```

```r
X_train <- X[train_index, ]
y_train <- y[train_index]
X_test <- X[-train_index, ]
y_test <- y[-train_index]
```

Les méthodes choisis pour ce dataset sont : la régression linéaire, la régression généralisée, les SVM, les B-splines cubiques, les KSVM et un modèle de réseaux de neurones. On va maintenant faire une cross-validation sur ces méthodes.

```r
X <- subset(data, select = -y)
y <- data$y
k <- 10
folds <- createFolds(y, k = k)
err<-matrix(0,6,k)
for (i in 1:k) {
  # Sélectionner le pli de données de test
  test_index <- folds[[i]]
  X_test <- X[test_index, ]
  y_test <- y[test_index]
  data.test <- data[test_index, ]

  # Sélectionner les données d'entraînement
  train_index <- unlist(folds[-i])
  X_train <- X[train_index, ]
  y_train <- y[train_index]
  data.train <- data[train_index, ]

  # Entraîner le modèle SVM sur les données d'entraînement
  model_svm <- svm(x = X_train, y = y_train, kernel = "radial")
  model_linear <- lm(y_train ~ ., data = X_train)
  model_glm <- glm(y_train ~ ., data = X_train, family = gaussian)
  model_spline <- lm(y ~ bs(X1,df=5) + bs(X2,df=5) + bs(X3,df=5) + bs(X4,df=5) +
                     bs(X5,df=5) + bs(X6,df=5) + bs(X7,df=5) + bs(X8,df=5),
                     data = data[train_index, ])
  model_kernel <- ksvm(y_train ~ ., data = X_train, kernel = "rbfdot", type = "eps-bsvr")
  model_nn <- nnet(y_train ~ ., data = X_train, size = 50,trace=FALSE)

  #Prédire les cibles sur les données de test
  predictions_SVM <- predict(model_svm, newdata = X_test)
  predictions_linear <- predict(model_linear, X_test)
  predictions_glm <- predict(model_glm, X_test, type = "response")
  predictions_spline <- predict(model_spline, X_test)
  predictions_kernel <- predict(model_kernel, X_test)
  predictions_nn <- predict(model_nn, newdata=X_test)

  for (n in 1:1:length(test_index)) {
  if (predictions_nn[[n]] > 0.92) {
    predictions_nn[[n]] <- predictions_SVM[[n]]
  }
}


  # Calculer l'erreur quadratique moyenne (MSE) sur les données de test
  err[1,i] <- mean((y_test - predictions_SVM) ^ 2)
  err[2,i] <- mean((y_test - predictions_linear) ^ 2)
  err[3,i] <- mean((y_test - predictions_glm) ^ 2)
  err[4,i] <- mean((y_test - predictions_spline) ^ 2)
  err[5,i] <- mean((y_test - predictions_kernel) ^ 2)
```

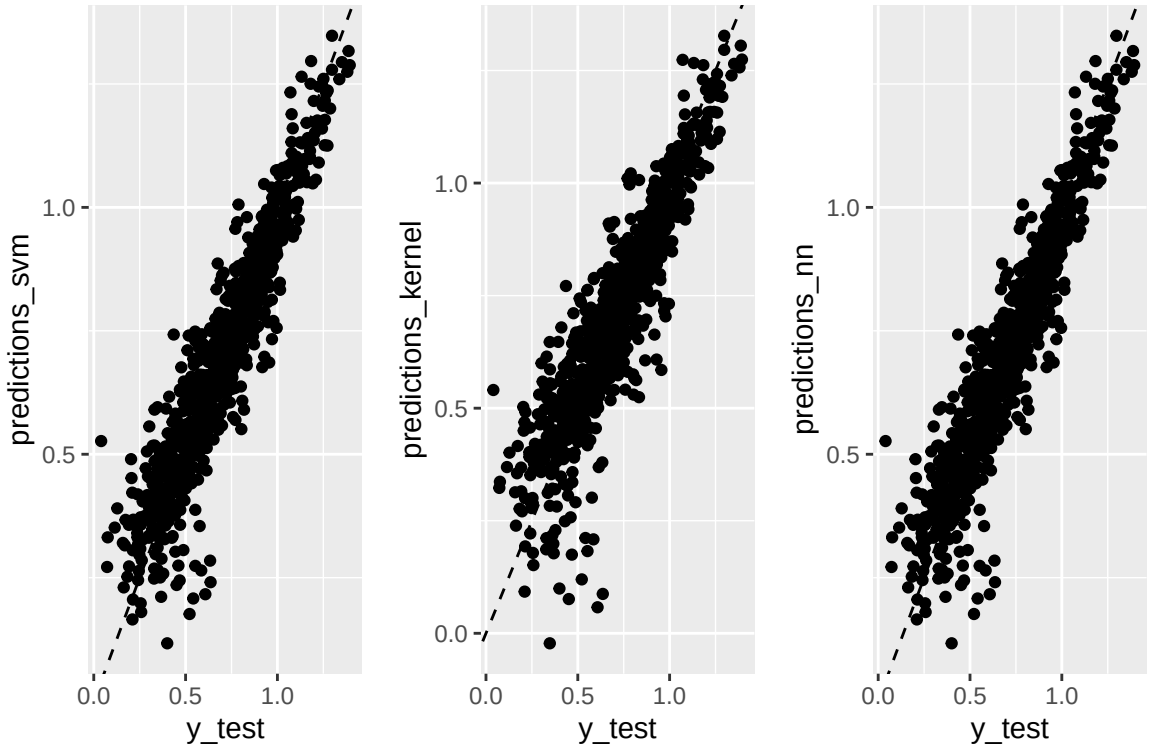
```
  err[6,i] <- mean((y_test - predictions_nn) ^ 2)

}
ERR <- data.frame(t(err))
names(ERR) <- c("SVM","Linear","glm","spline","kernel","nn")
boxplot(ERR)
```



Nous remarquons que pour les valeurs prédites supérieurs à 0.9 tendent doucement vers 1. De plus il n'y as pas de valeur supérieure à 1. Pour contrer ce résultat, nous utilisons la prédiction de la méthode SVM (la méthode avec le MSE le plus faible). Un autre avantage est que le MSE du NN sera au maximum égal à celui de la méthode SVM même lorsque la méthode ne converge pas. Nous avons donc décider de garder 3 méthodes pour les tests :le SVM, le KSVM et un réseau de neurones associé au SVM pour les valeurs prédites supérieur à 0.92.

```
train_index <- sample(1:nrow(data), 0.8 * nrow(data))
X_train <- X[train_index, ]
y_train <- y[train_index]
X_test <- X[-train_index, ]
y_test <- y[-train_index]

model_svm <- svm(y_train ~ ., data = X_train, kernel = "radial")
model_kernel <- ksvm(y_train ~ ., data = X_train, kernel = "rbfdot", type = "eps-bsvr")
model_nn <- nnet(y_train ~ ., data = X_train, size = 50, trace = FALSE)

predictions_svm <- predict(model_svm, X_test)
predictions_kernel <- predict(model_kernel, X_test)
predictions_nn <- predict(model_nn, X_test)

for (i in 1:1:(nrow(data)-length(train_index))) {
  if (predictions_nn[[i]] > 0.92) {
    predictions_nn[[i]] <- predictions_svm[[i]]
  }
}

ggplot(data = data.frame(y_test, predictions_svm), aes(x = y_test, y = predictions_svm)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed")
```

```
ggplot(data = data.frame(y_test, predictions_kernel), aes(x = y_test, y = predictions_kernel)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed")
ggplot(data = data.frame(y_test, predictions_nn), aes(x = y_test, y = predictions_nn)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed")
```



Troisième dataset