Apprentissage à partir de 3 jeux de données réels

Lucie BECHERI & Huiling SHAO

Ce TP a pour but de proposer les meilleurs prédicteurs possibles pour traiter le problème de classification des données : - expression_train.txt, contenant les niveaux de gris d'une image associés à une expression du visage parmi joie, surprise, tristesse, dégoût, colère, peur; - characters_train.txt, composé d'exemples issus de 26 classes correspondant aux 26 lettres de l'alphabet; - parole_train.txt, dont chaque observation correspond à prononciation d'un phonème parmi "sh", "dcl", "iy", "aa" et "ao"

Nous allons donc comparer sur ces jeux de données des qualités de prévision de plusieurs modèles.

1 Données expression_train.txt

La suite du travail nécessite un échantillon d'apprentissage pour estimer les modèles et un échantillon de test pour comparer les erreurs de prévision. Nous séparons donc le jeu de données en un ensemble de test comportant un quart des données et un ensemble d'apprentissage composé des données restantes.

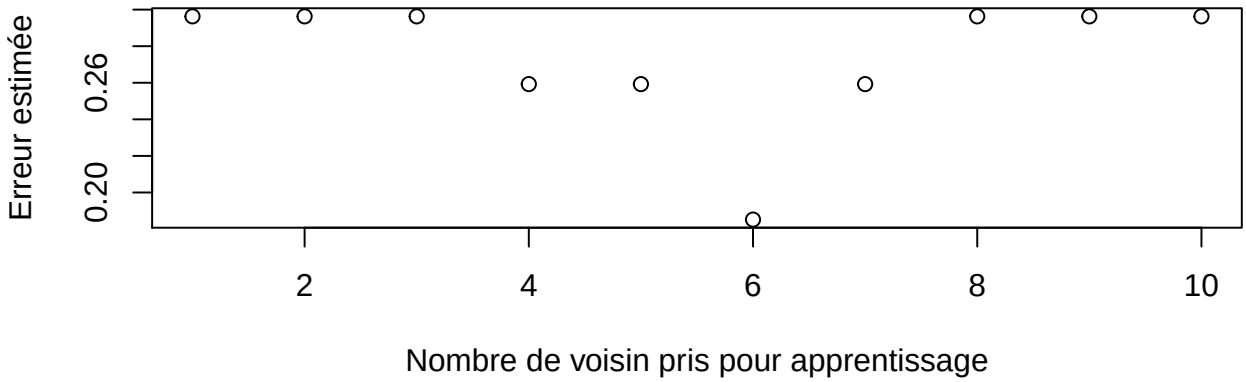
Nous remarquons un certain nombre de problèmes sur ce jeu de données.

Tout d'abord, il existe des colonnes entièrement nulles qui génèrent des erreurs lors de l'application des méthodes d'apprentissage. Il s'agit des prédicteurs correspondant aux coins au bas des images. Nous avons donc choisi de supprimer ces derniers puisqu'ils n'apportent aucune information. De plus, les images sont similaires (même cadrage du visage, ...) : les variables de ce jeu de données sont donc fortement corrélées.

Il y a également un très grand nombre de prédicteurs comparé au nombre d'observations. Nous avons donc choisi d'effectuer une réduction de dimension par FDA.

```
#Remove column full of 0
cleanData.train<-train[,colSums(abs(train[,1:ncol(train)-1])) !=0]
cleanData.test<-test[,colSums(abs(test[,1:ncol(test)-1])) !=0]

#FDA (with train data)
library("MASS")
lda_data<- lda(y~.,data=cleanData.train)
U<-lda_data$scaling
X<-as.matrix(cleanData.train[,1:ncol(cleanData.train)-1])
Z<-X%*%U
Z<-as.data.frame(Z)
y<-cleanData.train$y
trainFDA<-cbind(Z,y)

#Apply FDA on test data
X<-as.matrix(cleanData.test[,1:ncol(cleanData.test)-1])
Z<-X%*%U
Z<-as.data.frame(Z)
y<-cleanData.test$y
testFDA<-cbind(Z,y)
```

1.1 K plus proches voisins



Selon le graphique, on obtient l'erreur minimale de 0.18 quand k = 6. La méthode des K plus proches voisins a généralement de mauvais résultats en grande dimension puisque les voisins se retrouvent en réalité très loin (fléau de la dimension). Le fait d'avoir effectué au préalable une réduction de dimension a permis d'obtenir de bons résultats, donc une bonne généralisation et pas d'overfitting.

1.2 Analyse Discriminante

Nous regarderons dans un premier temps les résultats des classifieurs obtenus par analyse discriminante linéaire et quadratique et du classifieur Bayésien naïf. Nous chercherons ensuite un modèle optimal en appliquant le principe de régularisation.

1.2.1 LDA

```
lda_data<- lda(y~.,data=trainFDA)
pred<-predict(lda_data,newdata=testFDA)
table<-table(testFDA$y,pred$class)
error<-1-sum(diag(table))/nTst
```

Nous obtenons un taux d'erreur de test de 0.14. Des frontières linéaires donnent donc un résultat satisfaisant.

1.2.2 QDA

```
qda_data<- qda(y~.,data=trainFDA)
pred<-predict(qda_data,newdata=testFDA)
table<-table(testFDA$y,pred$class)
error<-1-sum(diag(table))/nTst
```

Nous obtenons un taux d'erreur de test très élevé : 0.40. Ce classifieur est probablement trop flexible pour nos données, il est donc à exclure.

1.2.3 Classifieur Bayésien naif

```
library(klaR)
naivB_data<-NaiveBayes(y~.,data=trainFDA)
pred<-predict(naivB_data,newdata=testFDA)
```

```
table<-table(testFDA$y,pred$class)
error<-1-sum(diag(table))/nTst
```

Nous obtenons un taux d'erreur de 0.25. Ce dernier est donc moins bon que celui obtenu avec la LDA (0.14).

1.2.4 Analyse discriminante régularisée

```
rda_data <- rda(y~.,data=trainFDA, crossval = TRUE)
pred<-predict(rda_data,newdata=testFDA)
table<-table(testFDA$y,pred$class)
error<-1-sum(diag(table))/nTst
```

Le taux d'erreur est 0.18. Il est relativement proche de celui obtenu avec la LDA. Son erreur est comprise entre l'erreur de LDA(0.14) et QDA(0.4). Ceci est normal puisque l'analyse discriminante régularisée est un compromis entre la LDA et la QDA.

1.3 Régression Logistique et modèles additifs généralisés (GLM & GAM)

Ce problème de classification n'est pas une classification binaire. On ne fera donc pas de régression logistique car la méthode de régression logistique réalisée par GLM et GAM (qui est une extension de GLM) est souvent moins performante que l'analyse discriminante (LDA, QDA...) pour les classifications avec plus de deux classes.

Nous avons de plus trop de prédicteurs pour tester de manières exhaustives les différents modèles additifs généralisés possibles. Ces derniers servent d'ailleurs d'avantage à expliquer les effets des différents prédicteurs, ce qui n'est pas le but de notre étude.

De ce fait ces méthodes ne seront pas non plus utilisées pour les jeux de données "parole" et "characters"

1.4 Mixture Discriminant Analysis

```
library(mclust)
ind_y = 6
MclustDa_data <- MclustDA(trainFDA[,1:ind_y-1],trainFDA[,ind_y])
#general covariance structure selected by BIC
summary(MclustDa_data, newdata = testFDA[,1:ind_y-1], newclass = testFDA[,ind_y])
```

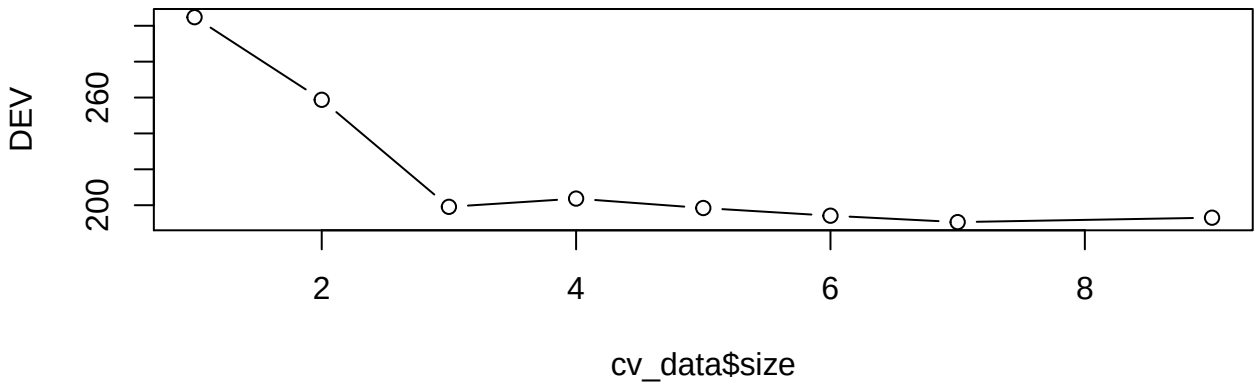
Le summary nous montre le nombre de clusters (G) et le modèle Parsimonious (Model) dans chaque classe en utilisant la méthode de modèle de sélection (critère BIC).

Malgré une erreur d'apprentissage nulle, l'erreur de test est extrêmement élevée (0.59). Ce modèle, comme la QDA, colle donc probablement beaucoup trop au données d'apprentissage (sur apprentissage). Il faut donc privilégier pour ces données des classifieurs plus simples qui vont pouvoir extraire des tendances plus générales.

1.5 Arbres

1.5.1 Arbre de décision

```
#install.packages("tree")
library(tree)
#Full tree
tree_data = tree(as.factor(y)~., trainFDA)
#plot(tree_data)
#text(tree_data, pretty = 0)

#Cross validation
size<-cv.tree(tree_data)$size
DEV<-rep(0, length(size))

for (i in (1:10))
{
  cv_data = cv.tree(tree_data)
  DEV<-DEV+cv_data$dev
}

DEV <- DEV/10

plot(cv_data$size, DEV, type = 'b')
```




```
#Pruning
prune_data = prune.tree(tree_data, best = 7)
#plot(prune_data)
#text(prune_data, pretty = 0)

#Test Error
y_pred = predict(prune_data, newdata = testFDA, type = 'class')
table<-table(y_pred, testFDA$y)
error<-1-sum(diag(table))/nTst
```

L'erreur obtenue est 0.44. Elle n'est pas acceptable. Nous allons tenter de l'améliorer en utilisant les méthodes de Bagging et de Random Forest.

1.5.2 Bagging

```
#install.packages("randomForest")
library(randomForest)
#m =p = 5
bag_data = randomForest(y~., data=trainFDA, mtry=5)
```

```
ypred = predict(bag_data, newdata=testFDA, type = 'response')
table<-table(ypred, testFDA$y)
error<-1-sum(diag(table))/nTst
```

On obtient l'erreur 0.29.

1.5.3 Random Forest

```
#m = sqrt(p) = p/2
rdForest_data = randomForest(y~., data=trainFDA,mtry=2)
ypred = predict(rdForest_data, newdata=testFDA, type = 'response')
table<-table(ypred, testFDA$y)
error<-1-sum(diag(table))/nTst
```

On obtient l'erreur 0.29. Ici on prend mtry = sqrt(p) = p/2 = 2.

Les erreurs obtenues par bagging et random forest sont plus faibles que celle obtenues initialement par l'arbre élagué. Néanmoins, ces classifieurs n'égalent pas l'analyse discriminante linéaire.

1.6 Support Vector Machine

```
#install.packages("e1071")
library(e1071)
#Kernel = radial
tune.out = tune(svm, y~., data = trainFDA, kernel = "radial",
                range = list(cost=c(0.01, 0.1, 1, 10, 100),
                             gamma = c(0.1, 1, 10)))
summary(tune.out)
svm_data<-svm(y~., data = trainFDA, kernel = "radial", gamma = 0.1, cost = 1)
ypred = predict(svm_data, newdata=testFDA)
table<-table(ypred, testFDA$y)
error<-1-sum(diag(table))/nTst
```

On obtient l'erreur 0.22.

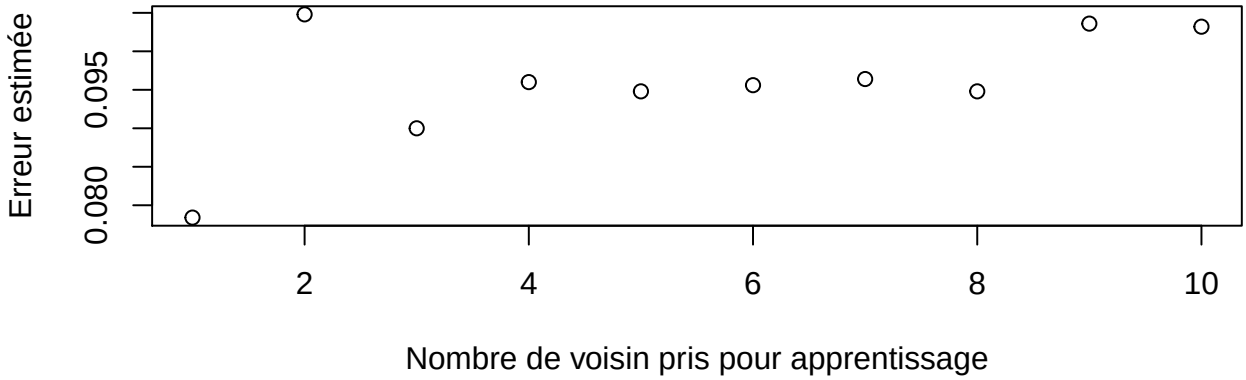
Le classifieur obtenu par LDA est donc le meilleur pour "expressions".

Pour les données suivantes, nous ne préciserons plus les codes pour calculer les erreurs de test en raison de la contrainte sur la taille du rapport.

2 Données characters_train.txt

Comme nous disposons d'un grand nombre d'observations (1000) et d'un nombre de variables relativement petit (17), il ne nous parait pas nécessaire de procéder à une réduction de dimension. Nous partageons simplement les données en un ensemble de test comportant un quart des données et un ensemble d'apprentissage composé des données restantes.

2.1 K plus proches voisins



Nous obtenons, par cross validation, une erreur minimale de 0.0784. Elle correspond cependant à 1 seul voisin. Pour les data_characteurs on a 17 prédicteurs et 10000 obsercations (p<<N). Donc c'est normal que la méthode avec le plus de flexibilité (k = 1) soit la plus performante.

2.2 Analyse Discriminante

Nous regarderons dans un premier temps les résultats des classifieurs obtenus par analyse discriminante linéaire et quadratique et du classifieur Bayésien naïf. Nous chercherons ensuite un modèle optimal en appliquant le principe de régularisation .

2.2.1 LDA

```
lda_data<- lda(Y~.,data=train)
pred<-predict(lda_data,newdata=test)
```

Nous obtenons un taux d'erreur de 0.3. Un classifieur offrant une frontière linéaire ne semble donc pas adapté.

2.2.2 QDA

```
qda_data<- qda(Y~.,data=train)
pred<-predict(qda_data,newdata=test)
```

Nous obtenons un taux d'erreur de 0.12. Il est relativement faible : la QDA donne donc de bon résultats. Cela était prévisible puisque nous possédons un grand nombre d'observations. On peut également déduire que nos données posèdent un bruit assez faible.

2.2.3 Classifieur Bayésien naif

```
library(klaR)
naivB_data<-NaiveBayes(Y~.,data=train)
pred<-predict(naivB_data,newdata=test)
```

Nous obtenons un taux d'erreur de 0.36. Ce dernier est donc moins bon que celui obtenu avec la QDA (0.12)

2.2.4 Analyse discriminante régularisée

```
rda_data <- rda(Y~.,data=train, crossval = TRUE)
pred<-predict(rda_data,newdata=test)
```

Le taux d'erreur de test est de 0.12. Il est relativement proche de celui obtenu avec la QDA. On peut donc penser que la régularisation a favorisé une matrice de covariance propre à chaque classe.

2.3 Mixture Discriminant Analysis

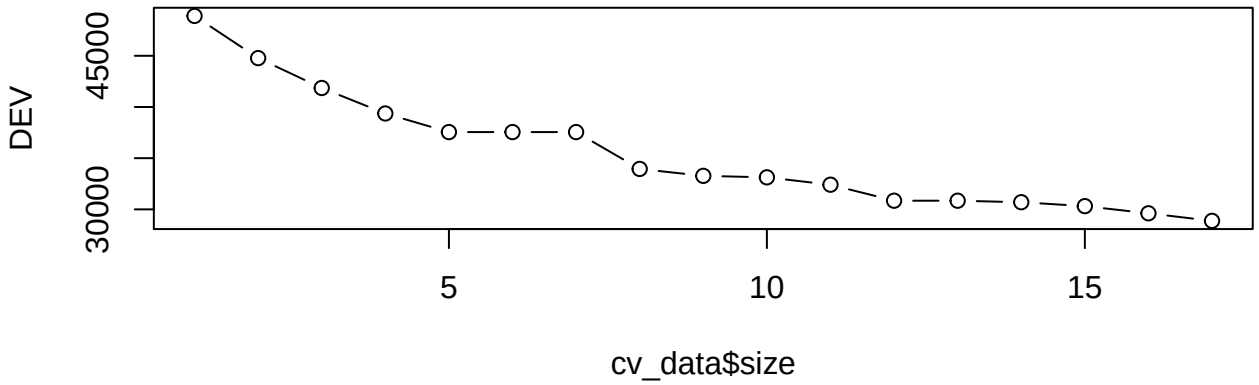
```
library(mclust)
ind_y = 1
MclustDa_data <- MclustDA(train[,(ind_y+1):ncol(train)],train[,ind_y])
#general covariance structure selected by BIC
summary(MclustDa_data, newdata = test[,(ind_y+1):ncol(train)], newclass = test[,ind_y])
```

Nous obtenons de bon résultats : une erreur d'apprentissage de 0.03 et une erreur de test de 0.08. Ce classifieur est donc meilleur que celui obtenu par QDA. On peut donc déduire que la distribution des données à l'intérieur des classes se rapproche plus d'un mélange de Gausiennes que d'une seule.

2.5 Arbres

2.5.1 Arbre de décision

```
#install.packages("tree")
library(tree)
#Full tree
tree_data = tree(as.factor(Y)~., train)
#plot(tree_data)
#text(tree_data, pretty = 0)

#Cross validation
size<-cv.tree(tree_data)$size
DEV<-rep(0, length(size))

for (i in (1:10))
{
  cv_data = cv.tree(tree_data)
  DEV<-DEV+cv_data$dev
}

DEV <- DEV/10

plot(cv_data$size, DEV, type = 'b')
```




```
#Pruning
prune_data = prune.tree(tree_data, best = 17)
#plot(prune_data)
#text(prune_data, pretty = 0)
y_pred = predict(prune_data, newdata = test, type = 'class')
```

Nous obtenons la plus petite erreur, par cross validation, pour un arbre non élagué : on fait donc probablement du surapprentissage.

Cela est confirmé par une erreur de test élevée : 0.63. Cette erreur n'est donc pas acceptable.

2.5.2 Bagging

```
#install.packages("randomForest")
library(randomForest)
#m =p = 16
bag_data = randomForest(Y~., data=train, mtry=16)
ypred = predict(bag_data, newdata=test, type = 'response')
```

On obtient l'erreur de test 0.0756. Elle est donc nettement meilleure que celle obtenue auparavant.

2.5.3 Random Forest

```
#m = sqrt(p)
rdForest_data = randomForest(Y~., data=train,mtry=4)
ypred = predict(rdForest_data, newdata=test, type = 'response')
```

On obtient l'erreur de test 0.058. Ici on prend mtry = sqrt(p) = 4. C'est jusqu'a maintenant le meilleur résultat obtenu, même si celui du bagging en est extrêment proche.

2.6 Support Vector Machine

L'execution de la fonction tune est trop longue. Nous n'avons pas pu run la fonction tune suffisamment longtemps pour obtenir un résultat.

Donc le classifieur obtenu par Random Forest est le meilleur pour "characters".

3 Données parole_train.txt

Le jeu de données possède 257 variables. Afin de savoir s'il est préférable de procéder à une réduction de dimension nous allons comparer les erreurs de test d'un modèle estimé avec un traitement préalable par FDA des données et sans. Pour la QDA, l'amélioration est flagrante :

```
## [1] "Erreur sans FDA"
```

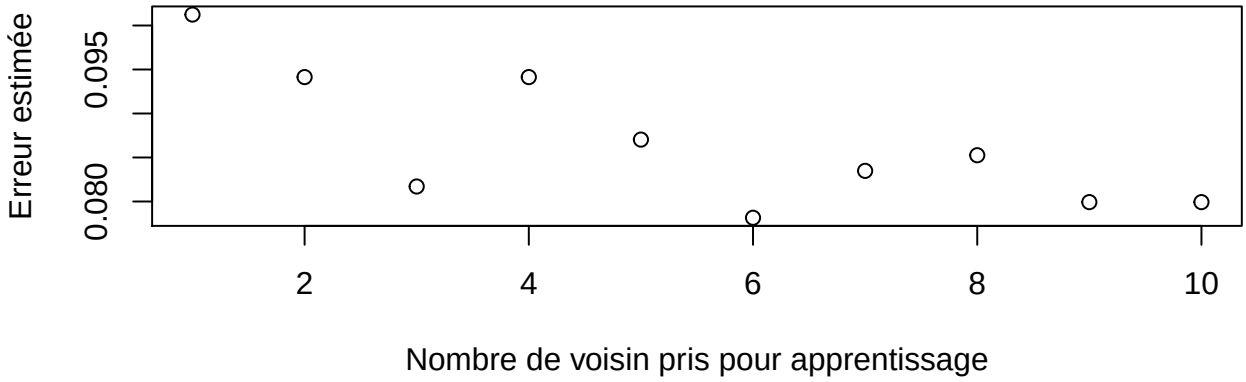
```
## [1] 0.3889876
```

```
## [1] "Erreur avec FDA"
```

```
## [1] 0.08348135
```

L'erreur avec FDA (0.08) est significativement plus petite que l'erreur sans FDA (0.39). Nous choisissons donc d'appliquer tout de même la FDA.

3.1 K plus proches voisins



Selon le graphique, on obtient l'erreur minimale de 0.078 quand k = 6.

3.2 Analyse Discriminante

Nous regarderons dans un premier temps les résultats des classifieurs obtenus par analyse discriminante linéaire et quadratique et du classifieur Bayésien naïf. Nous chercherons ensuite un modèle optimal en appliquant le principe de régularisation .

3.2.1 LDA

```
lda_data<- lda(y~.,data=trainFDA)
pred<-predict(lda_data,newdata=testFDA)
```

Nous obtenons un taux d'erreur de 0.078. Des frontières linéaires donnent donc un résultat satisfaisant.

3.2.2 QDA

```
qda_data<- qda(y~.,data=trainFDA)
pred<-predict(qda_data,newdata=testFDA)
```

Nous obtenons un taux d'erreur de 0.08.

3.2.3 Classifieur Bayésien naif

```
library(klaR)
naivB_data<-NaiveBayes(y~.,data=trainFDA)
pred<-predict(naivB_data,newdata=testFDA)
```

Nous obtenons un taux d'erreur de 0.07.

L'erreurs de QDA, LDA et Bayésien naif sont très proches. Donc on peut déduire que les matrices de covariance de chaque classe sont similaires et diagonales.

3.2.4 Analyse discriminante régularisée

```
rda_data <- rda(y~.,data=trainFDA, crossval = TRUE)
pred<-predict(rda_data,newdata=testFDA)
```

Le taux d'erreur est 0.07.

Les différentes méthodes d'analyse discriminantes donnent donc des erreurs de test similaires. Cette approche semble plutôt bien marcher sur ces données.

3.3 Mixture Discriminant Analysis

```
library(mclust)
ind_y = 5
MclustDa_data <- MclustDA(trainFDA[,1:ind_y-1],trainFDA[,ind_y])
#general covariance structure selected by BIC
summary(MclustDa_data, newdata = testFDA[,1:ind_y-1], newclass = testFDA[,ind_y])
```

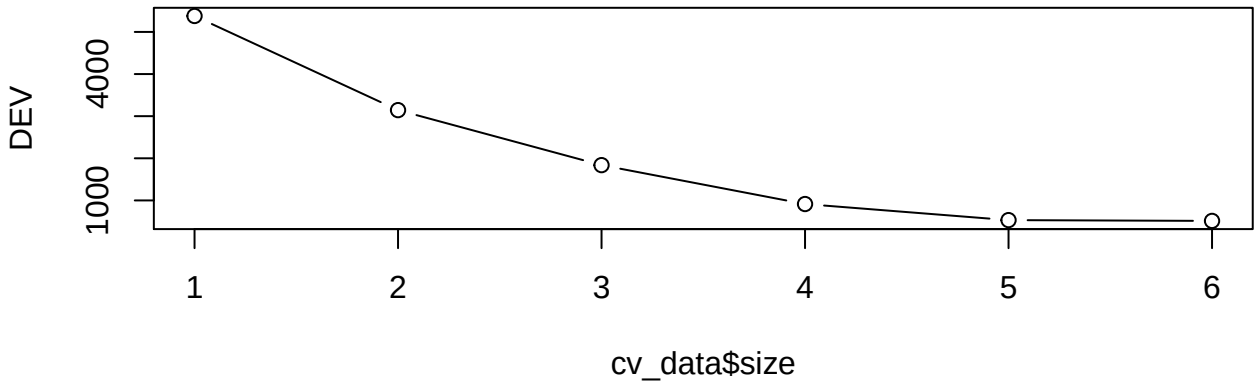
L'erreur d'apprentissage est de 0.04, l'erreur de test de 0.08. Ce modèle donne également de bon résultats.

3.4 Arbres

3.4.1 Arbre de décision

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

#Full tree
tree_data = tree(as.factor(y)~., trainFDA)
#plot(tree_data)
#text(tree_data, pretty = 0)

#Cross validation
size<-cv.tree(tree_data)$size
DEV<-rep(0, length(size))

for (i in (1:10))
{
  cv_data = cv.tree(tree_data)
  DEV<-DEV+cv_data$dev
```

```
}

DEV <- DEV/10

plot(cv_data$size, DEV, type = 'b')
```




```
#Pruning
prune_data = prune.tree(tree_data, best = 6)
#plot(prune_data)
#text(prune_data, pretty = 0)

y_pred = predict(prune_data, newdata = testFDA, type = 'class')
```

L'erreur de test obtenue est 0.087. Elle est plutôt faible et est obtenue sans élaguage pour 6 feuilles. Cela est probablement lié au fait que l'on ait effectué une réduction de dimension, ce qui limite la complexite de l'arbre initial.

3.4.2 Bagging

```
#install.packages("randomForest")
library(randomForest)
#m =p = 4
bag_data = randomForest(y~., data=trainFDA, mtry=4)
ypred = predict(bag_data, newdata=testFDA, type = 'response')
```

On obtient l'erreur 0.09.

3.4.3 Random Forest

```
#m = sqrt(p) = p/2
rdForest_data = randomForest(y~., data=trainFDA,mtry=2)
ypred = predict(rdForest_data, newdata=testFDA, type = 'response')
```

On obtient l'erreur 0.08. Ici on prend mtry = sqrt(p) = p/2 = 2.

3.5 Support Vector Machine

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

#Kernel = radial
tune.out = tune(svm, y~., data = trainFDA, kernel = "radial",
                range = list(cost=c(0.01, 0.1, 1, 10, 100),
                             gamma = c(0.1, 1, 10)))
summary(tune.out)

svm_data<-svm(y~., data = trainFDA, kernel = "radial", gamma = 0.1, cost = 0.1)
ypred = predict(svm_data, newdata=testFDA)
```

On obtient l'erreur 0.08 pour un cout de 0.1 et un gamma de 0.1.

Le classifieur Bayésien naïf est donc le meilleur pour "parole". Il faut néanmoins noter que globalement, toutes les méthodes ont donné de bon résultats sur ce jeu de données.

Conclusion

En comparant les différents taux d'erreur de test, nous avons trouvé que la LDA offrait les meilleurs résultats pour les données "expression". Nous avons choisi pour ce jeu de données, comme pour les suivants, d'effectuer une division des données en un ensemble de test et un ensemble d'apprentissage. Cependant, pour celui-ci, nous avions également pensé à la possibilité de procéder par cross validation afin de pouvoir apprendre les modèles sur d'avantage de données. En effet, le nombre d'observations disponible est assez faible. Néanmoins, nous n'avons pas implémenté cette méthode car les résultats obtenus nous semblaient satisfaisants.

Pour les données "characters", c'est le classifieur Random Forest que nous avons choisi.

Enfin, pour les données "parole", nous avons tranché en faveur du classifieur Bayésien Naïf. Il faut tout de même noter que pour ce dernier jeu, les résultats obtenus étaient similaires peu importe la méthode.