Projet STA203

Alamichel Louise, Koen Aristote

5/3/2020

Introduction

L'objectif de ce projet est de trouver le meilleur modèle et la meilleure méthode parmi la régression logistique, la méthode des K plus proches voisins et une régression ridge sur un problème de classification. Notre jeu de données est un jeu de données sur la musique. L'objectif est de reconnaître un genre de musique à partir de certaines caractéristiques sur les musiques. Le jeu de données ne contient que deux genres de musique, le jazz et la musique classique. C'est donc un problème de classification binaire. Nous commencerons par étudier les résultats obtenus par régression logistique avec différents modèles. Puis, nous étudierons la méthode des K plus proches voisins. Pour finir, nous observerons les performances de la régression ridge sur nos données.

Partie I : Régression logistique

On commence par étudier une régressioin logistique sur notre problème. Nous allons comparer plusieurs modèle et choisir le meilleur.

Question 1

On étudie notre jeu de données, on commence par étudier la dimension et le type de nos variables.

```
# On vérifie qu'il n'y a pas de données manquantes
sum(is.na(music)) # [1] 0 donnée manquante
```

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

```
# Librairie  permettant d'afficher le tableau des types des variables
library(skimr)
types = skim(music)
dim(music) # On a 192 variables et 6447 données
```

```
## [1] 6447  192
```

```
table(types$skim_type)
```

```
##
##  factor numeric
##       1     191
```

```
# factor numeric
#      1     191
df = data.frame(variables = types$skim_variable, type = types$skim_type)
df[df$type == "factor",]
```

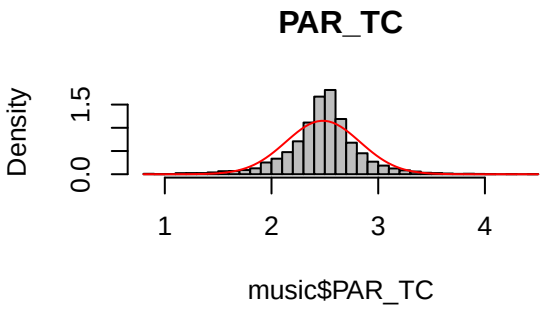
```
##   variables   type
## 1     GENRE factor
```

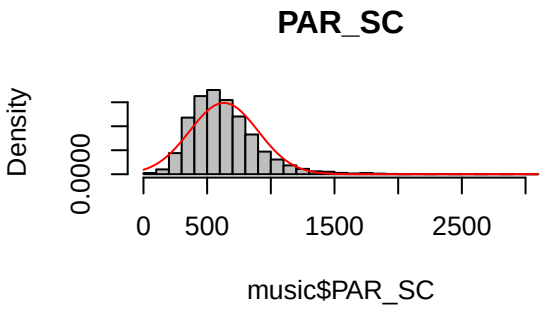
```
#  variables   type
#     GENRE factor
```

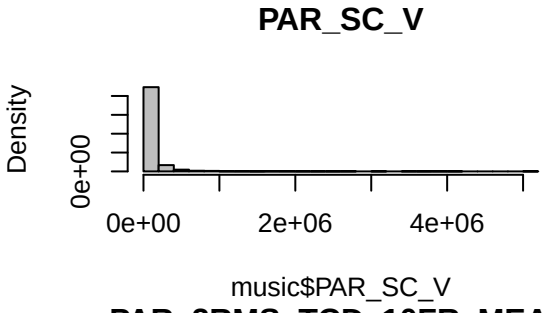
On n'a donc que des variables numériques sauf notre réponse GENRE qui est factorielle et qui indique le genre de chaque morceau. On voit aussi qu'on a 6447 données et 192 variables, par ailleurs, notre jeu de données ne comporte pas de valeur manquante.

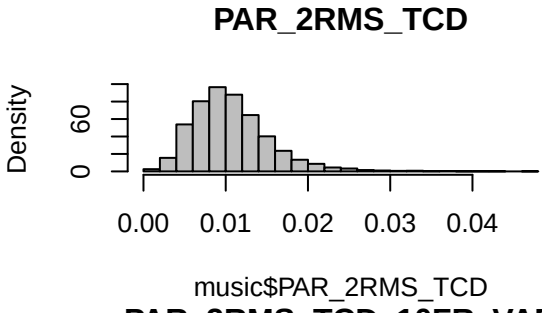
On fait ensuite une analyse descriptive de nos données. On commence par une analyse univariée de nos données. Au vu du nombre de variables, on a choisi de n'étudier qu'une variable par chaque groupe de données décrits dans l'annexe. On a donc 16 variables pour cette analyse descriptive.

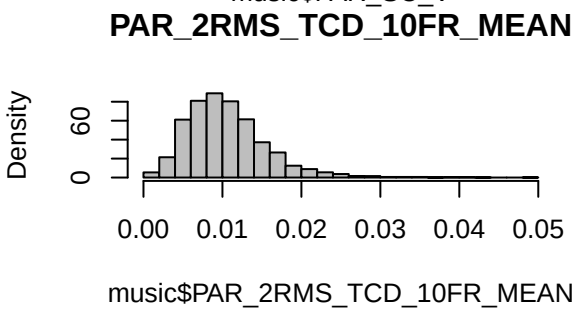
```
##      PAR_TC           PAR_SC          PAR_SC_V        PAR_2RMS_TCD
##  Min.   :0.8377   Min.   :  34.1   Min.   :    605   Min.   :0.0007619
##  1st Qu.:2.3268   1st Qu.: 450.2   1st Qu.:  22714   1st Qu.:0.0073469
##  Median :2.4982   Median : 592.6   Median :  46309   Median :0.0100410
##  Mean   :2.4814   Mean   : 631.4   Mean   : 103732   Mean   :0.0106984
##  3rd Qu.:2.6452   3rd Qu.: 764.4   3rd Qu.:  99464   3rd Qu.:0.0130835
##  Max.   :4.4046   Max.   :3044.4   Max.   :5003700   Max.   :0.0474290
##  PAR_2RMS_TCD_10FR_MEAN PAR_2RMS_TCD_10FR_VAR   PAR_ASC_V
##  Min.   :0.000000       Min.   :0.000e+00     Min.   :0.005926
##  1st Qu.:0.007012       1st Qu.:3.215e-05     1st Qu.:0.165035
##  Median :0.009850       Median :6.996e-05     Median :0.292860
##  Mean   :0.010534       Mean   :1.215e-04     Mean   :0.426270
##  3rd Qu.:0.013091       3rd Qu.:1.463e-04     3rd Qu.:0.532465
##  Max.   :0.048664       Max.   :6.669e-03     Max.   :4.998000
##     PAR_ASC          PAR_ASS_V           PAR_ASS         PAR_ASE_M
##  Min.   :-4.8190   Min.   :0.003518   Min.   :0.3243   Min.   :-0.1689
##  1st Qu.:-1.7795   1st Qu.:0.048588   1st Qu.:0.8576   1st Qu.:-0.1641
##  Median :-1.3186   Median :0.073653   Median :1.0536   Median :-0.1633
##  Mean   :-1.3986   Mean   :0.091978   Mean   :1.0572   Mean   :-0.1632
##  3rd Qu.:-0.9564   3rd Qu.:0.114895   3rd Qu.:1.2392   3rd Qu.:-0.1624
##  Max.   : 1.3843   Max.   :0.769860   Max.   :2.0524   Max.   :-0.1582
##    PAR_ASE_MV          PAR_SFM_M         PAR_SFM_MV          PAR_MFCC1
##  Min.   :6.486e-05   Min.   :0.02342   Min.   :0.0002265   Min.   :-2.6569
##  1st Qu.:3.065e-04   1st Qu.:0.05319   1st Qu.:0.0005785   1st Qu.: 0.7074
##  Median :3.675e-04   Median :0.05919   Median :0.0006406   Median : 1.2294
##  Mean   :3.840e-04   Mean   :0.05865   Mean   :0.0006477   Mean   : 1.3881
##  3rd Qu.:4.419e-04   3rd Qu.:0.06422   3rd Qu.:0.0007116   3rd Qu.: 2.0107
##  Max.   :1.117e-03   Max.   :0.08927   Max.   :0.0012263   Max.   : 4.7966
##    PAR_MFCCV1
##  Min.   :-2.6569
##  1st Qu.: 0.7074
##  Median : 1.2294
##  Mean   : 1.3881
##  3rd Qu.: 2.0107
##  Max.   : 4.7966
```

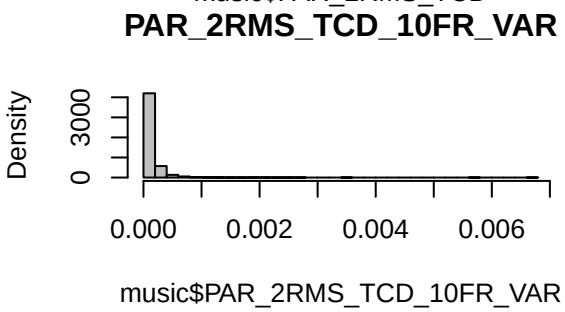

PAR_TC
Density
0.0
1.5
1
2
3
4
music$PAR_TC

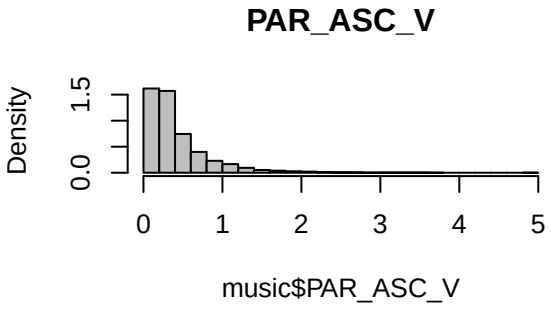

PAR_SC
Density
0.0000
0
500
1500
2500
music$PAR_SC

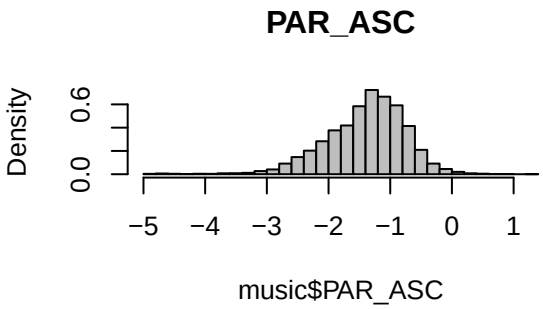

PAR_SC_V
Density
0e+00
0e+00
2e+06
4e+06
music$PAR_SC_V

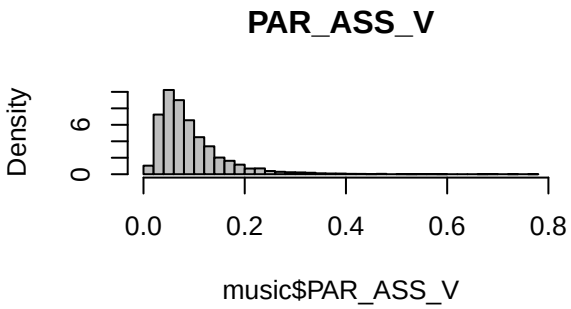

PAR_2RMS_TCD
Density
0
60
0.00
0.01
0.02
0.03
0.04
music$PAR_2RMS_TCD

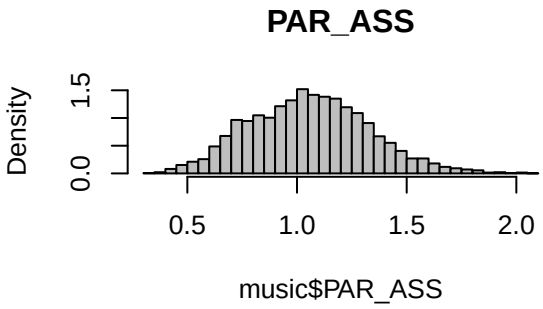

PAR_2RMS_TCD_10FR_MEAN
Density
0
60
0.00
0.01
0.02
0.03
0.04
0.05
music$PAR_2RMS_TCD_10FR_MEAN

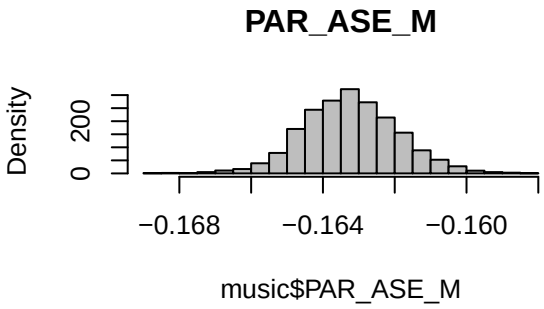

PAR_2RMS_TCD_10FR_VAR
Density
0
3000
0.000
0.002
0.004
0.006
music$PAR_2RMS_TCD_10FR_VAR

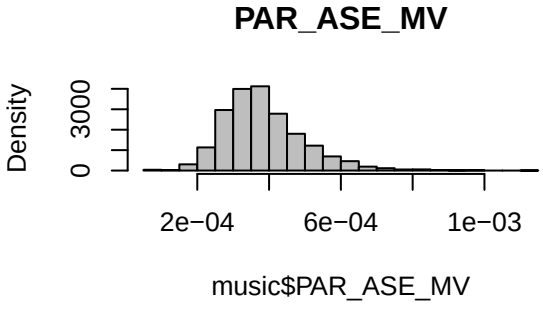

PAR_ASC_V
Density
0.0
1.5
0
1
2
3
4
5
music$PAR_ASC_V

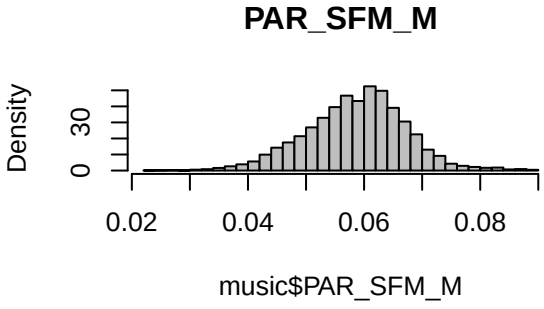

PAR_ASC
Density
0.0
0.6
−5
−4
−3
−2
−1
0
1
music$PAR_ASC

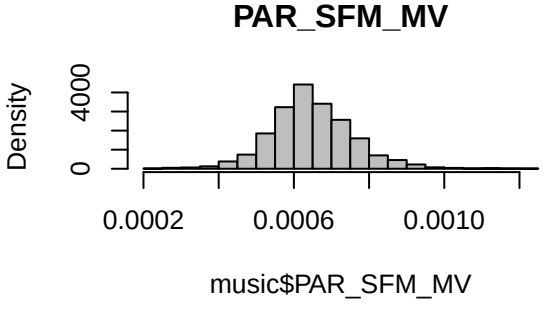

PAR_ASS_V
Density
0
6
0.0
0.2
0.4
0.6
0.8
music$PAR_ASS_V

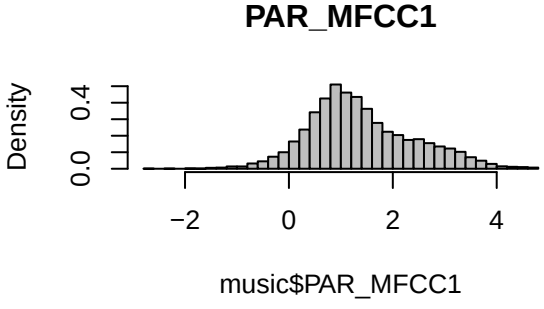

PAR_ASS
Density
0.0
1.5
0.5
1.0
1.5
2.0
music$PAR_ASS


PAR_ASE_M
Density
0
200
−0.168
−0.164
−0.160
music$PAR_ASE_M

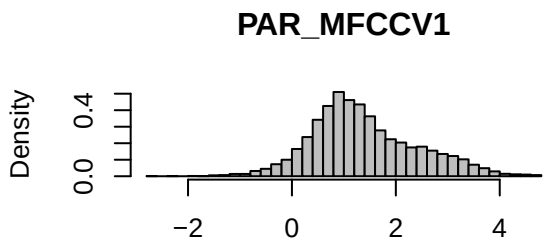

PAR_ASE_MV
Density
0
3000
2e−04
6e−04
1e−03
music$PAR_ASE_MV

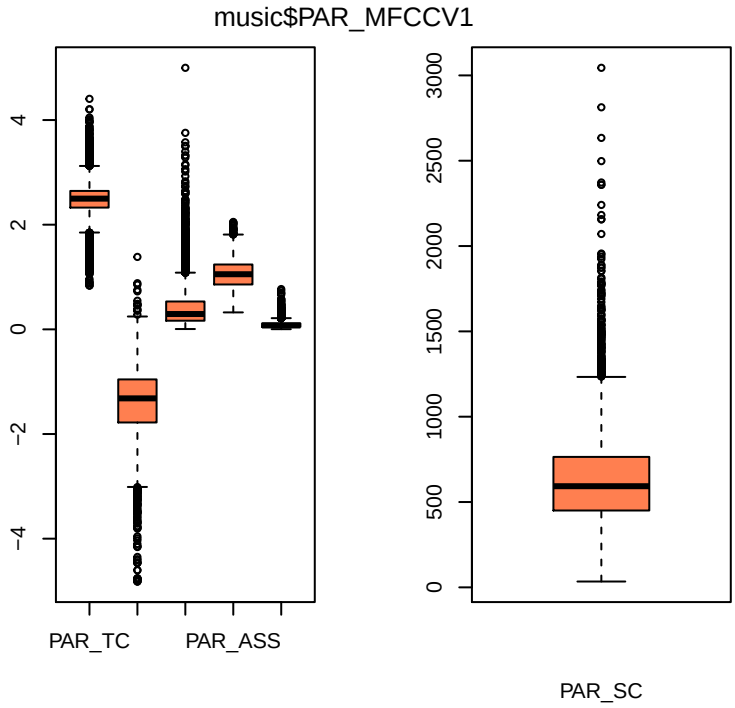

PAR_SFM_M
Density
0
30
0.02
0.04
0.06
0.08
music$PAR_SFM_M

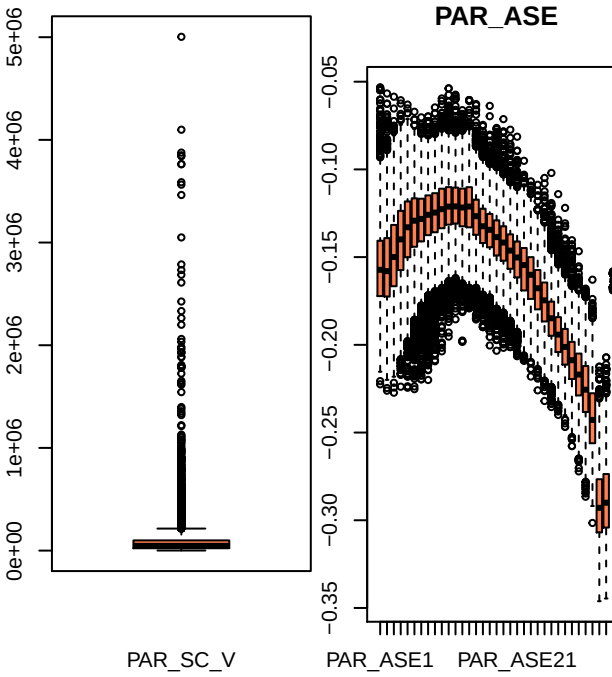

PAR_SFM_MV
Density
0
4000
0.0002
0.0006
0.0010
music$PAR_SFM_MV

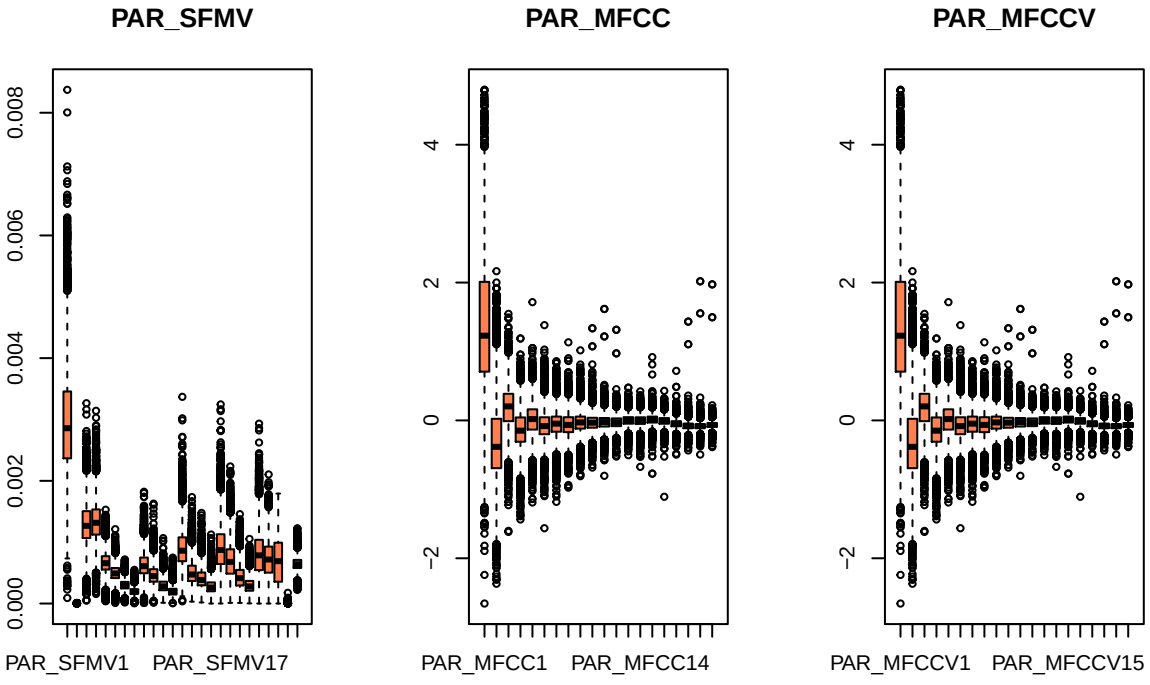

PAR_MFCC1
Density
0.0
0.4
−2
0
2
4
music$PAR_MFCC1

Histogramme d'une variable parmi les principaux groupes de variables





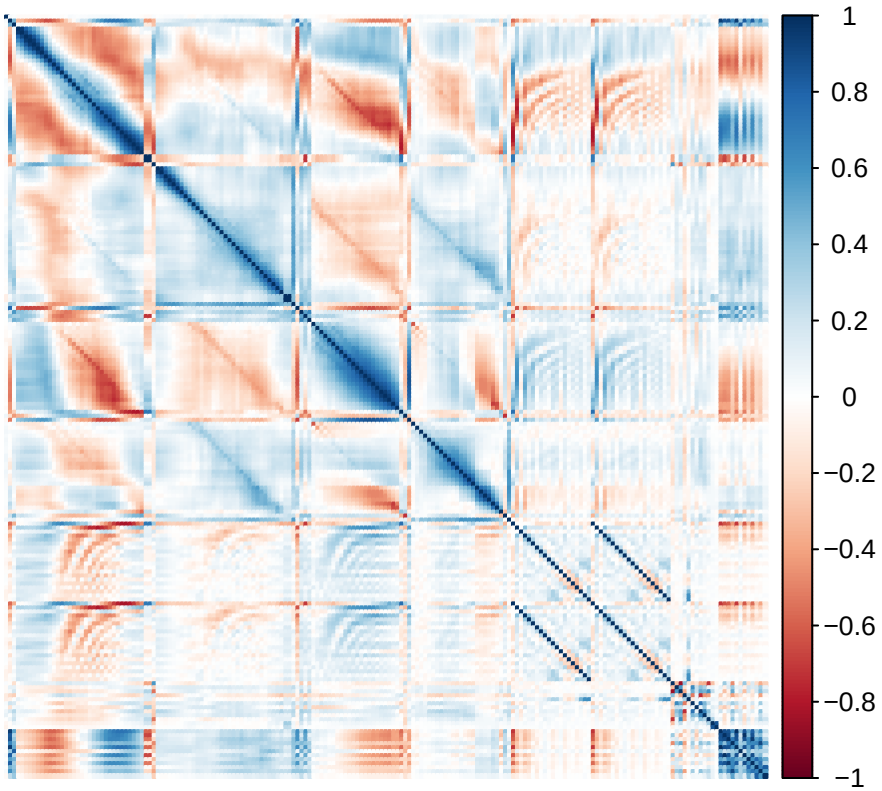




On observe que la plupart de nos données semblent être distribuées selon des lois normales, en dehors de PAR_ASC_V et PAR_SC_V. Sur les boîtes à moustache, on peut voir que les valeurs prises par nos variables varient sur des intervalles très petits en dehors de notre variable PAR_SC_V dont les valeurs varient sur une intervalle beaucoup plus large. On remarque que les variables des deux groupes PAR_MFCC et PAR_MFCCV sont similaires ce que lon étudiera en détail plus tard.

On fait ensuite une analyse bivariée de nos données. On regarde la matrice des corrélations de nos variables.

```
## corrplot 0.84 loaded
```

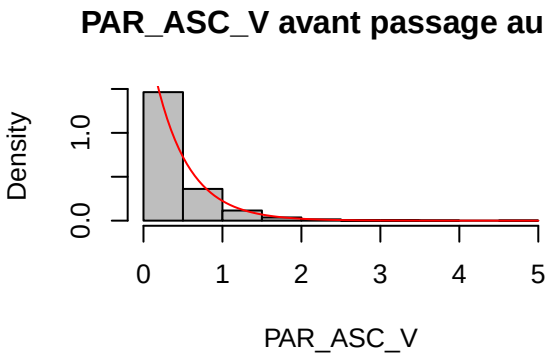


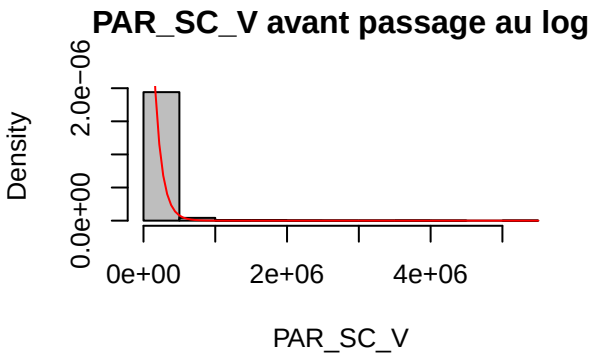
On peut voir sur cette matrice qu'il y a des données assez corrélées et anti-corrélées. On retrouve le fait que les données des groupes PAR_MFCC et PAR_MFCCV sont très corrélés. On va étudier plus précisément les données très corrélées après avoir regardé la proportion de notre variable réponse GENRE dans le jeu de données.

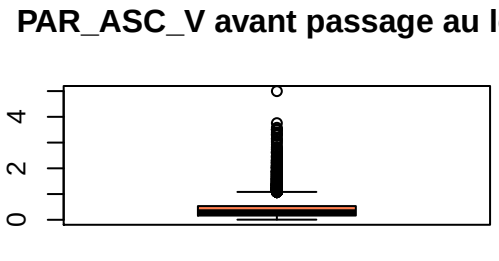
```
##
## Classical      Jazz
##   53.4202   46.5798
```

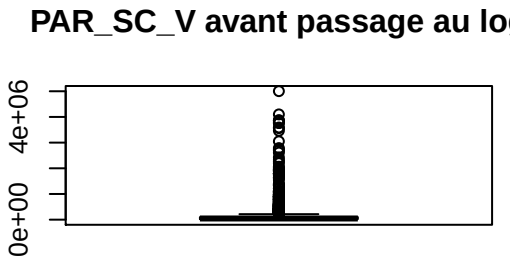
On peut voir que dans notre variable réponse il y a deux genres de musique. Le genre de musique Classical représente 53% du jeu de données et Jazz 47%. On voit que Classical est un peu plus représenté mais notre répartition est plutôt bonne.

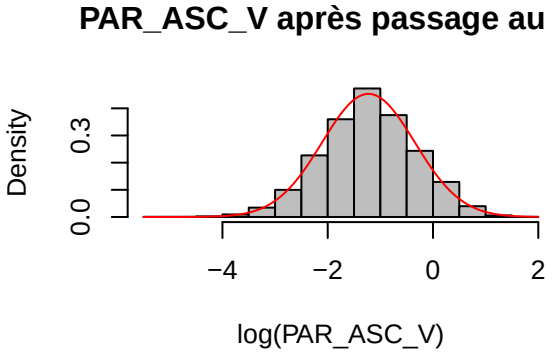
On étudie maintenant les variables PAR_SC_V et PAR_ASC_V, on a déjà vu que la variable PAR_SC_V n'évoluait pas dans le même ordre de grandeur que les autres variables.

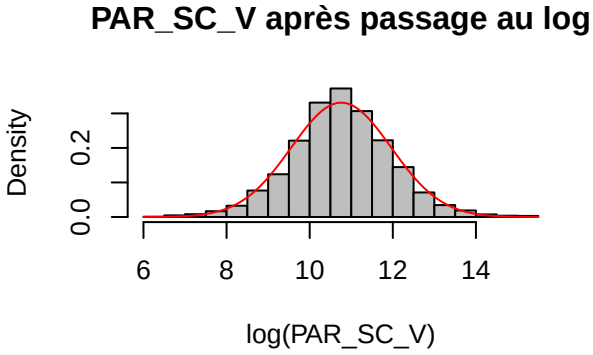


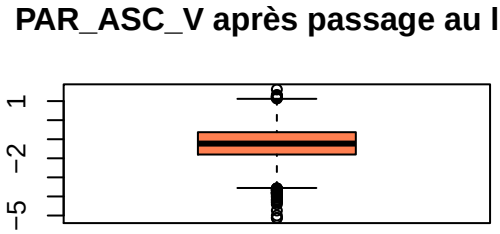


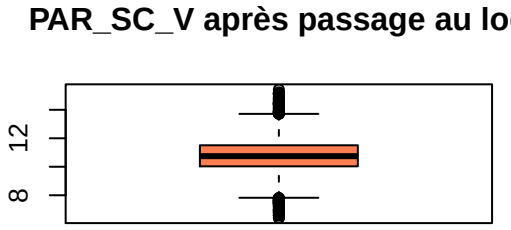













```r
# Nos données semblent maintenant suivre une loi normale
# Test de normalité :
ks.test(log(PAR_ASC_V),pnorm,mean=mean(log(PAR_ASC_V)), sd = sd(log(PAR_ASC_V))*sqrt((n-1)/n))
```

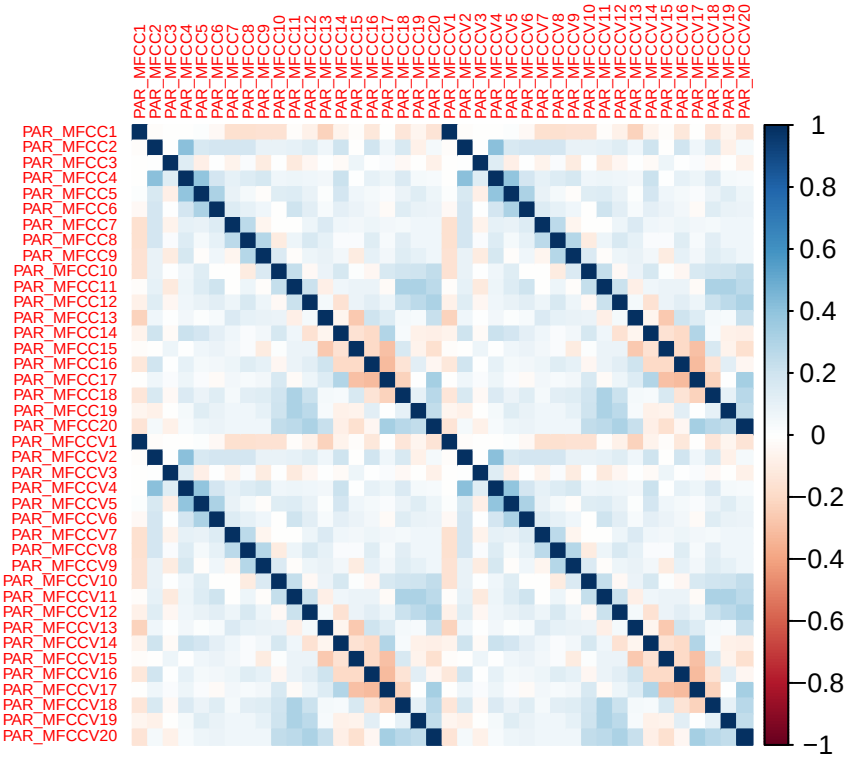
```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  log(PAR_ASC_V)
## D = 0.011981, p-value = 0.313
## alternative hypothesis: two-sided
```

On voit aussi que les deux variables sont très dispersées avec beaucoup de valeurs extrêmes. On décide donc

de passer au log ces variables pour les normaliser. Cela permet aussi de rendre nos deux variables normales.

On observe dans la description des données que certaines variables sont identiques (128-147 sont les mêmes que 148-167), on a d'ailleurs vu qu'il y avait une forte corrélation entre ces deux groupes de variables. On vérifie qu'elles sont bien les mêmes.

Matrice de corrélation des groupes MFCC et MFCCV




```
# La différence des variables vaut bien 0
zeros_ = music[128:147] - music[148:167]
norm(as.matrix(zeros_)) # [1] 0
```

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

On voit qu'il y a une corrélation de 1 entre ces deux groupes de variables. On a aussi étudié la différence des deux groupes de données. On a trouvé une différence de 0, chaque variable d'un groupe a donc son doublon dans l'aute. On choisit donce d'enlever un de ces deux groupes de variables de notre jeu de données.

On s'intéresse au cas des variables très corrélées. On a vu dans notre matrice de corrélation que certaines variables sont très corrélées avec d'autres. On sort ces variables :

```
##                                col
## PAR_ASE34                 PAR_ASE33
## PAR_ASE33                 PAR_ASE34
## PAR_ASEV34               PAR_ASEV33
## PAR_ASEV33               PAR_ASEV34
## PAR_ZCD_10FR_MEAN           PAR_ZCD
## PAR_ZCD           PAR_ZCD_10FR_MEAN
```

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

On voit qu'il n'y a pas de variables anti-corrélées au-delà du seuil -0.99, par contre on a plusieurs variables corrélées au-delà du seuil 0.99. On choisit d'enlever ces variables. En effet, à ce niveau de corrélation, on

peut considérer que l'une de ces variables est totalement expliquée par l'autre. On enlève donc une variable de chaque couple de variables très corrélées.

On s'intéresse maintenant au cas des variables PAR_ASE_M, PAR_ASE_MV, PAR_SFM_M et PAR_SFM_MV. Dans la description des variables, on voit que ces variables correspondent aux moyennes des groupes d), f), j) et l). Ces variables sont donc totalement expliquées par les variables de leur groupe respectif, elles n'apportent pas de nouvelles explications sur la variable réponse. Nous pouvons soit faire le choix d'enlever les groupes de variables, soit d'enlever ces quatres moyennes. Nous avons choisi d'enlever seulement les quatres variables considérées car le fait qu'elles soient totalement expliquée par d'autres variables n'impliquent pas qu'elles apportent toutes les informations des autres variables.

Suite à toutes les modifications sur notre jeu de données, on peut faire l'hypothèse forte que nos données sont indépendantes.

On modélise la variable réponse Y comme un échantillon iid de Bernoulli donc tel que $Y_i \sim \mathcal{B}\rangle\backslash(1, \pi_i(x_i))$ d'espérance $\pi_i(x_i) = \mathbb{P}(Y_i = 1; x_i)$ car nous sommes dans un problème de classification binaire non groupé. Ainsi les Y_i En effet les Y_i suivent bien une loi de Bernoulli où Jazz représente un succès et Classique un échec. Cette probabilité pi dépend des covariables x_i associées à Y_i, c'est-à-dire $p_i = \pi(x_i)$. Dans la suite, nous allons choisir comme fonction de lien la fonction `logit` qui est telle que $\texttt{logit}(\pi(x_i)) = x_i\theta$. On fait donc l'hypothèse que l'espérance de notre variable réponse par notre fonction de lien `logit` peut s'écrire comme un régresseur linéaire.

Question 2

Nous avons défini un échantillon d'apprentissage et donc par défaut un échantillon de test qui correspond aux données qui ne sont pas dans l'échantillon d'apprentissage.

Question 3

Nous allons maintenant définir différents modèles que nous allons estimer sur notre échantillon d'apprentissage.

Nous commençons par définir un modèle Mod0 sur les variables PAR_TC, PAR_SC, PAR_SC_V, PAR_ASE_M, PAR_ASE_MV, PAR_SFM_M et PAR_SFM_MV. Grâce à ce modèle, on va pouvoir voir si notre choix de garder les groupes de variables d), f), j) et l) plutôt que leur moyenne est bon.

On estime ensuite le modèle de toutes les variables retenues précédemment, ModT.

On va ensuite estimer deux modèles correspondant à des sous-modèles de ModT. Le premier, Mod1, correspond aux variables significatives au niveau 5% dans ModT. Le second est composé des variables significatives au niveau 20% dans ModT, il s'appelle Mod2.

```
# On récupère la pvalue des variables du modèle modT
pval= summary_$coefficients[,4]

# On sélectionne les variables de pvalue<0.05 pour créer le modèle mod1
names_5 = names(which(pval < 0.05))
# On sélectionne les variables de pvalue<0.2 pour créer le modèle mod2
names_20 = names(which(pval < 0.2))

formule5 = paste("GENRE~", paste(names_5[-1], collapse = "+"))
formule20 = paste("GENRE~", paste(names_20[-1], collapse = "+"))

# Le modèle mod1
mod1 = glm(formule5, family = "binomial", data = music_modT_train)
summary(mod1)
```

```r
# Le modèle mod2
mod2 = glm(formule20, family = "binomial", data = music_modT_train)
summary(mod1)
```

On obtient un modèle de 70 variables pour Mod1 et de 101 pour Mod2. Pour visualiser les summary des différents modèles merci de vous référer au scripte R.

Enfin, on définit un dernier modèle qui correspond au modèle minimisant le critère AIC. On appelle ce modèle ModAIC. Ce modèle est obtenu par la sélection stepwise des variables sur critère AIC à partir du modèle Mod2. On a choisi de partir de ce modèle car c'est celui qui minimise l'AIC.

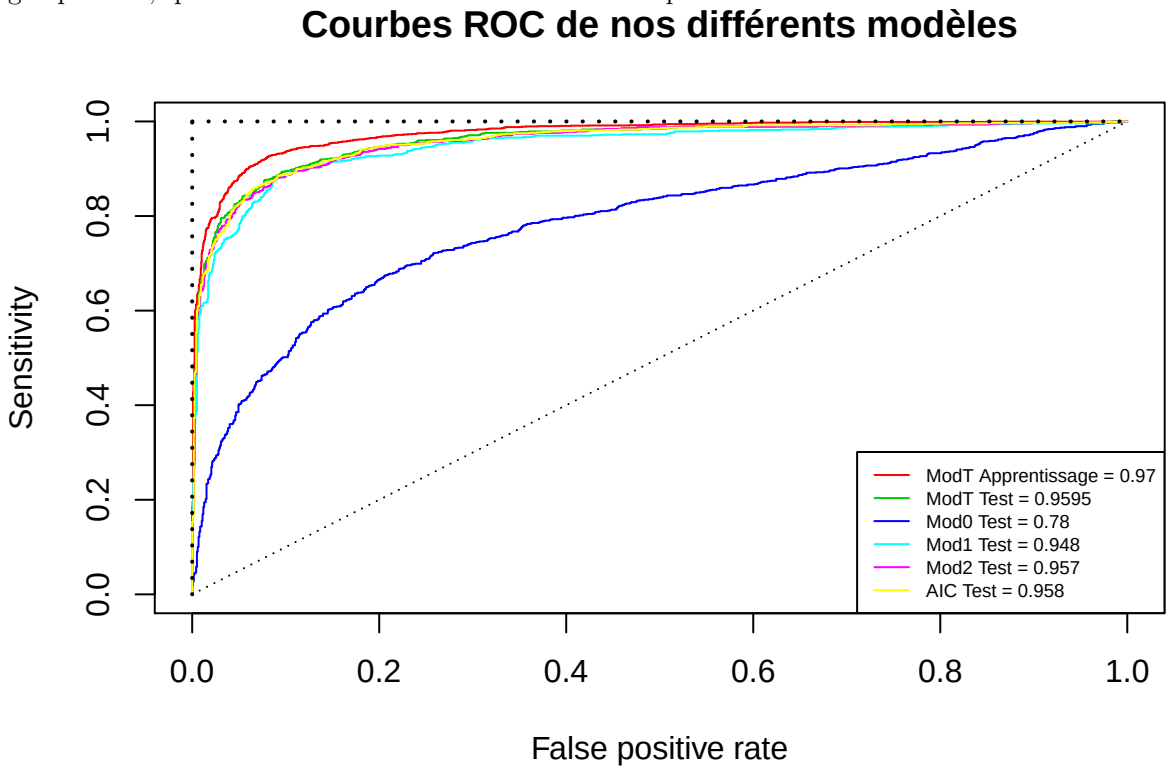
```r
# On crée le modèle modAIC

library(MASS)
library(doParallel)
cl = makePSOCKcluster(4)
registerDoParallel(cl)
modAIC=stepAIC(mod2, direction="both", scope = list(lower = ~1, upper = modT$model))
stopCluster(cl)
summary(modAIC)
```

On obtient un modèle de 99 variables avec un AIC de 2002.

Question 4

Une fois tous nos modèles définis, il nous faut les comparer pour en sélectionner un. Un premier critère de sélection peut être la courbe ROC. On a tracé les courbes ROC de tous nos modèles sur l'échantillon de test, et d'apprenstissage pour ModT. On peut voir en tiret la courbe de la règle parfaite, plus la courbe ROC d'un modèle est proche de cette courbe meilleur est le modèle.



Ici, on peut voir que notre meilleur modèle est ModT appliqué à l'échantillon d'apprentissage. Ce n'est pas une

surprise, en effet un modèle est en général meilleur sur l'échantillon grÃce auquel il a défini sa règle de classification. Par ailleurs, les modèles ModT, Mod1, Mod2 et ModAIC ont une courbe similaire et sont tous les quatres proches de la règle parfaite. Mod1 est légèrement en-dessous des trois autres courbes. On a affiché dans la légende l'aire sous la courbe ROC, l'objectif est d'être le plus proche de 100. Selon ce critère, le meilleur modèle est ModT (à vérifier avec ModAIC) mais les modèles Mod2 et ModAIC sont aussi très bons. Le modèle Mod1 est un tout petit peu moins bon mais vu qu'il est plus simple, l'éliminer ne paraît pas évident sur ce critère. Par contre on peut voir que Mod0 est bienmoins bon que les autres modèles. Sa courbe ROC est plu proche de la règle aléatoire que les autres et son aire sous la courbe est plus basse que celle des autres. Ceci n'est pas non plus une surprise étant donné que ce modèle prend en compte beaucoup moins de variables que les autres.

Question 5

Nous avons vu que le modèle Mod0 est moins bon que les quatre autres qui sont à peu près aussi bons (avec un léger moins bien pour Mod1). Nous allons donc étudier un autre critère pour choisir le meilleur modèle, l'erreur de classification.

```
## modT.probs.train mod1.probs.train mod2.probs.train  AIC.probs.train
##       0.07830762       0.09350164       0.08321646       0.08204769
```

```
## modT.probs.test mod1.probs.test mod2.probs.test  AIC.probs.test
##       0.1009682       0.1051176       0.1051176       0.1037344
```

```
erreur(mod0.probs.train, music_tf_train) # [1] 0.2741935
```

```
## [1] 0.2741935
```

```
erreur(mod0.probs.test, music_tf_test) # [1] 0.264177
```

```
## [1] 0.264177
```

On peut voir que Mod0 a la plus grande erreur que ce soit sur l'échantillon test ou apprentissage. Au vu de tous les résultats sur Mod0, on sait que ce ne sera pas le modèle sélectionné. On voit, par ailleurs, que cette fois encore, le modèle qui ressort comme meilleur selon ce critère est ModT. Toutefois, comme dans la question précédente, les modèles Mod1, Mod2 et ModAIC sont vraiment très proches.

Le choix du meilleur modèle n'est pas évident, le modèle ayant les meilleurs résultats est ModT mais c'est aussi celui avec le plus grand nombre de variables. Mod0 qui est le modèle avec le moins de variables est aussi celui qui a les moins bons résultats. On a aussi quelques modèles avec un nombre de variables intermédaires qui ont de très bon résultats. Au final, on choisit le modèle ModAIC car c'est celui qui a les meilleurs résultats après ModT, un taux d'erreur de $10,4\%$ et un nombre de variables pas trop élevé : 99 variables. On ne pourra pas tester son adéquation, en effet toutes les ethodes que nous avons vu en cours ne nous permettent de tester l'adéquation d'un modèle que si le problème de classification est sur des données groupées. Ici, nos données sont individuelles, chaque donnée correspond à un morceau de musique à classer.

Partie 2 : K-voisins

Question 1

Soit un échantillon d'apprentissage et le vecteur constitué des classes de chaque observation. La méthode des K plus proches voisins consiste à considérer les K observations du jeu d'apprentissage les plus proches d'une une nouvelle entrée x. La classe attribuée à cette entrée sera la classe qui est majoritaire parmis ces K voisins.

Question 2

```
library(class)
# Avec k = 1
knn1 = knn(music_modT_train[,-music_modT_train$GENRE],
           music_modT_test[,-music_modT_test$GENRE],
           cl = music_modT_train$GENRE, k = 1)

summary(knn1)
```

```
##    0    1
## 1276  893
```

```
table(knn1, music_modT_test$GENRE)
```

```
##
## knn1   0   1
##    0 933 343
##    1 239 654
```

```
# Taux d'erreur
1-mean(knn1 == music_modT_test$GENRE) # [1] 0.2683264
```

```
## [1] 0.2683264
```

```
# On cherche le paramètre k permettant la meilleure classification.
# Pour cela, on fait une validation croisée sur différents paramètres k
# génération de B plis
library(pls)
B = 5
folds = cvsegments(nrow(music_modT_train), B, type = "random")
K = c(1,3,5,7,9,11,15,17,23,25,35,45,55,83,101,151) # différents K à tester
nK = length(K)

err_matrix = matrix(data=NA, nrow=B, ncol=nK)
for(b in 1:B){
  subsetb = unlist(folds[[b]])
  for (k in 1:nK){
    music_sub_train = music_modT_train[-subsetb,]
    music_sub_test = music_modT_train[subsetb,]
    Y = music_sub_train$GENRE
    knn_ = knn(music_sub_train[,-Y], music_sub_test[,-music_sub_test$GENRE],
               cl = Y, k = K[k])
    err_matrix[b,k] = 1-mean(knn_ == music_sub_test$GENRE)
  }
}

err = apply(err_matrix, 2, mean)
K_opt = K[which.min(err)]
K_opt; min(err) # On obtient k = [1] 1
```

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

```
## [1] 0.291257
```

```
# Ainsi le k permettant la meilleure classification est 1, ce qui signifie qu'on
# classe chaque morceaux pareil que celui le plus proche
```

```r
#(distance euclidienne dans un espace de dimension 165)

# Comme le paramètre k = 1 ne nous paraissait pas optimal, nous avons voulu vérifier par une manière au
```

Question 3

On observe que la validation croisée 5-fold retient $K = 1$ comme nombre de voisins optimal pour la classification. Nous avons tenté de répéter l'expérience pour des validations croisées à folds différents et le K optimal trouvé restait bien égal à 1. En effet, le fait d'utiliser un seul voisin montre que les données n'ont pas de séparation claire. On observe également qu'avec $K = 1$ le taux de mauvais classement sur le jeu de données test est de $26,8\%$ ce qui est bien plus élevé qu'avec la régression logistique.

Partie 3 : Régression ridge

Question 1

La régression ridge consiste à trouver un estimateur des paramètres à travers la résolution du problème de minimisation du critère des moindres carrés ordinaires sous contrainte sur la norme 2 de θ.

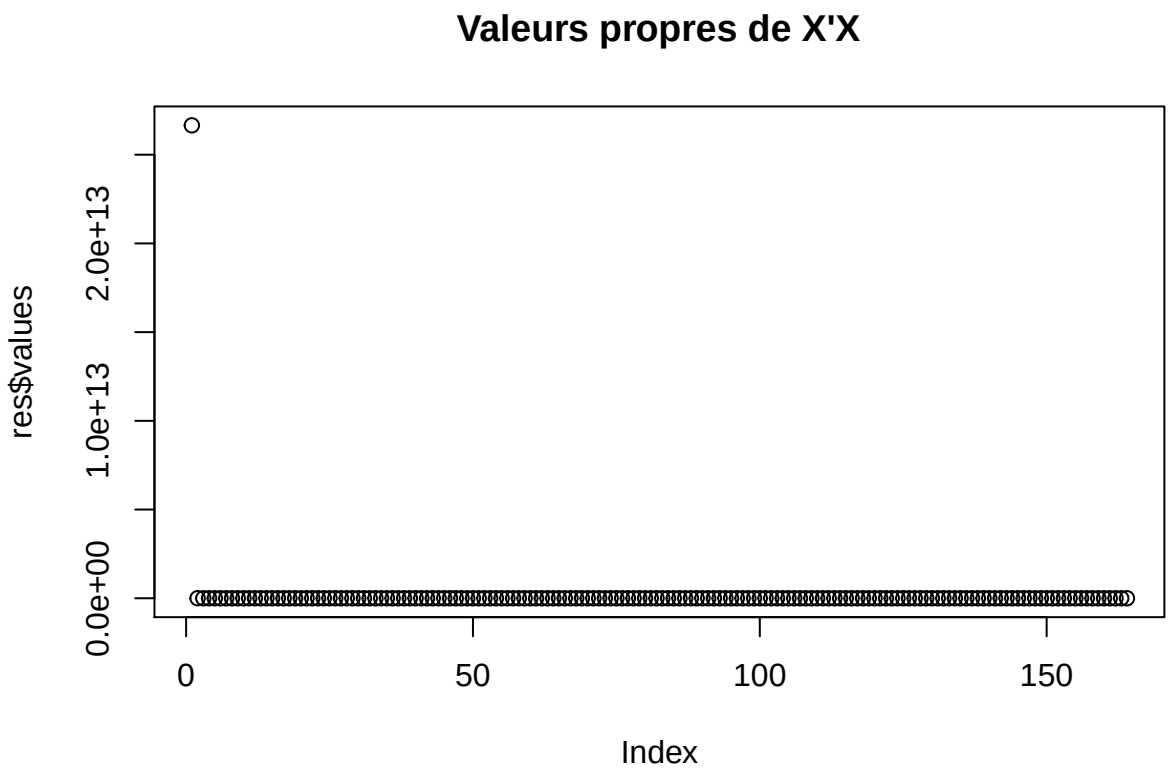
C'est à dire trouver $\hat{\theta}_{\mathcal{R}} = \arg\min_{\theta:\|\theta\|^2<\varepsilon} \|Y - X\theta\| = (X'X + \lambda I_d)^{-1}X'Y$ où λ est le paramètre de pénalité dans le lagrangien du problème.

Ceci peut s'avérer utile dans le cas où la matrice $X'X$ est de rang inférieur au nombre de variables ou lorsque beaucoup de ses valeurs propres sont très proches de zéros ce qui la rend proche d'une matrice non inversible. C'est souvent le cas lorsque le jeu de données contient de nombreuses variables corrélées. En effet, dans ce cas, l'estimateur des moindres carrés $(X'X)^{-1}X'Y$ a une forte variance à cause des faibles valeurs propres de $X'X$. Ainsi, en régularisant, on réduit la variance de cet estimateur qui est inférieure à la variance de l'estimateur des moindres carrés. Cepndant, c'est un estimateur biaisé de θ contrairement à l'EMCO. Ainsi, la régression Ridge permet de réduire la variance de l'estimation mais en faisant un compromis sur le biais.

On peut voir que nous avons certaines variables fortement corrélées dans le jeu de données et que les valeurs propres de notre matrice $X'X$ sont proches de zéro.

```r
# Les valeurs propres de notre matrice X'X
res = eigen(t(as.matrix(music_modT_train[,-music_modT_train$GENRE]))
            %*%as.matrix(music_modT_train[,-music_modT_train$GENRE]))

plot(res$values,main="Valeurs propres de X'X")
```




```
length(res$values[res$values<=1e-3]) # [1] 62
```

```
## [1] 62
```

On observe qu'on a plus de 60 valeurs propres inférieures à 1e-3 ce qui montre que la matrice $X'X$ est peut-être proche d'une matrice non inversible malgré le fait que l'on ait un nombre d'observations bien supérieur au nombre de variables, en effet:

```
solve(t(as.matrix(music_modT_train[,-music_modT_train$GENRE]))
      %*%as.matrix(music_modT_train[,-music_modT_train$GENRE]))

# Error in solve.default(t(as.matrix(music_modT_train[, -music_modT_train$GENRE])) %*% : system is comp
```

Le logiciel considère la matrice $X'X$ comme non inversible. Ainsi, ayant de nombreuses valeurs propres proches de zéro, la variance de l'estimateur des moindres carrés ordinaire $\hat{\theta}_{MCO} = \arg\min_\theta \|Y - X\theta\|^2 = (X'X)^{-1}X'Y$ sera élevée. Ceci est dû à la présence de nombreuses variables fortement corrélées dans la matrice du plan d'expérience qu'on souhaite tout de même considérer dans nos modèles.

On observe, qu'il y a beaucoup de variables avec des corrélations supérieures à 0.90 dans les groupes ASE et ASEV par exemple. Ainsi, il est naturel de se demander si une méthode de régularisation ne serait pas adaptée afin de réduire la variance de l'estimateur.

Question 2

Dans le cas où $\lambda = 10^{-2}$, c'est-à-dire λ quasi nul, on ne pénalise quasiment pas le critère des moindres carrés et donc l'estimation obtenue est proche de celle de l'EMCO. Dans le cas où λ est de l'ordre de 10^{10} on contraint la norme 2 de θ à être quasi nulle.

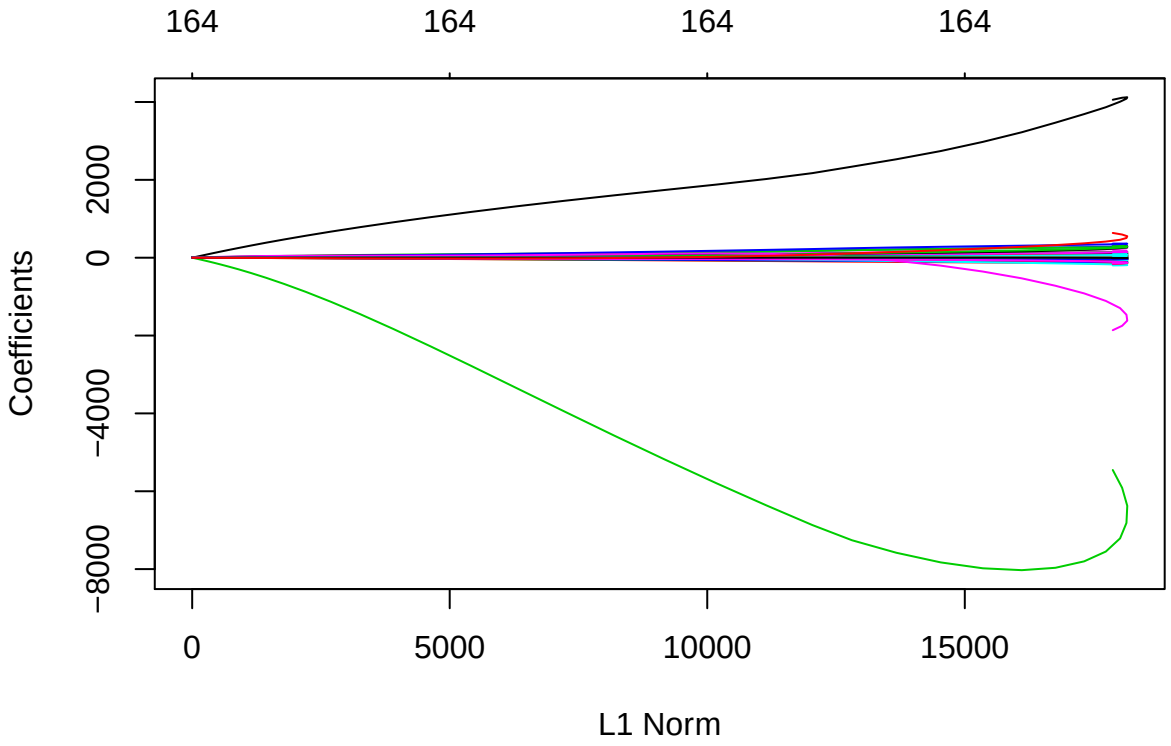
Ainsi, il est indispensable de bien choisir le paramètre de régularisation λ afin d'optimiser le compromis biais variance et obtenir la meilleure modélisation.

```r
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 3.0-2
```

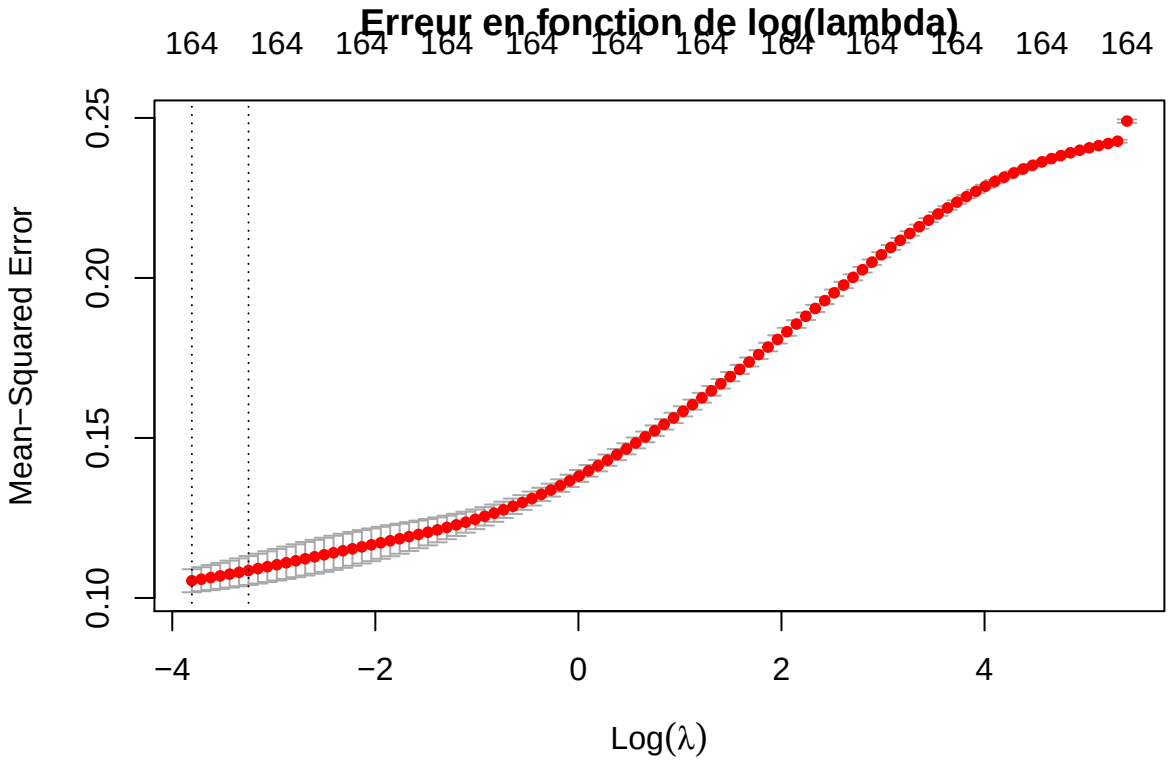
```r
grid = 10^seq(10,-2,length=100) # la grille de lambda
x = model.matrix(GENRE~., music_modT_train)[,-1]
y = music_modT_train$GENRE
ridge.fit = glmnet(x, y, alpha = 0, lambda = grid) # alpha = 0 pour ridge
plot(ridge.fit)
```



Le graphique sorti par la fonction glmnet est un tracé de la valeur de l'estimation de chaque coordonée en fonction de la norme 1 du vecteur $\theta_{\mathcal{R}}$. Ainsi sur l'axe des abcisses, une faible norme correspond à un λ élevé et une norme élevée à une faible valeur de λ. On observe que sans pénalisation, c'est à dire pour un λ très faible et donc pour les fortes valeurs de la norme L1 de $\hat{\theta}_{\mathcal{R}}$, il n'y a que trois coefficients qui ont des valeurs significativement élevées (le reste des coefficients étant proches de zéro). Plus lambda augmente, plus ces trois coefficients décroient. Le lambda à choisir est celui à partir duquel la trajectoire de ces coefficients plonge vers zéro, ce qui est difficile à déterminer graphiquement dans ce cas. Nous procéderons ensuite par validation croisée.

Question 3

```r
# On cherche le meilleur paramètre de régularisation lambda
set.seed(314)
cv.ridge.fit = cv.glmnet(x, y, alpha = 0)
plot(cv.ridge.fit,main="Erreur en fonction de log(lambda)") # On affiche l'évolution de l'erreur en fon
```




```
lam_opt = cv.ridge.fit$lambda.min # [1] 0.02221237 le meilleur lambda

# Performance sur l'échantillon test
xtest = model.matrix(GENRE~., music_modT_test)[,-1]
ytest = music_modT_test$GENRE
ridge.pred = predict(ridge.fit, s = lam_opt, newx = xtest)
mod.pred = ifelse(ridge.pred > .5, 1, 0)
1-mean(mod.pred == ytest) # [1] 0.09727985
```

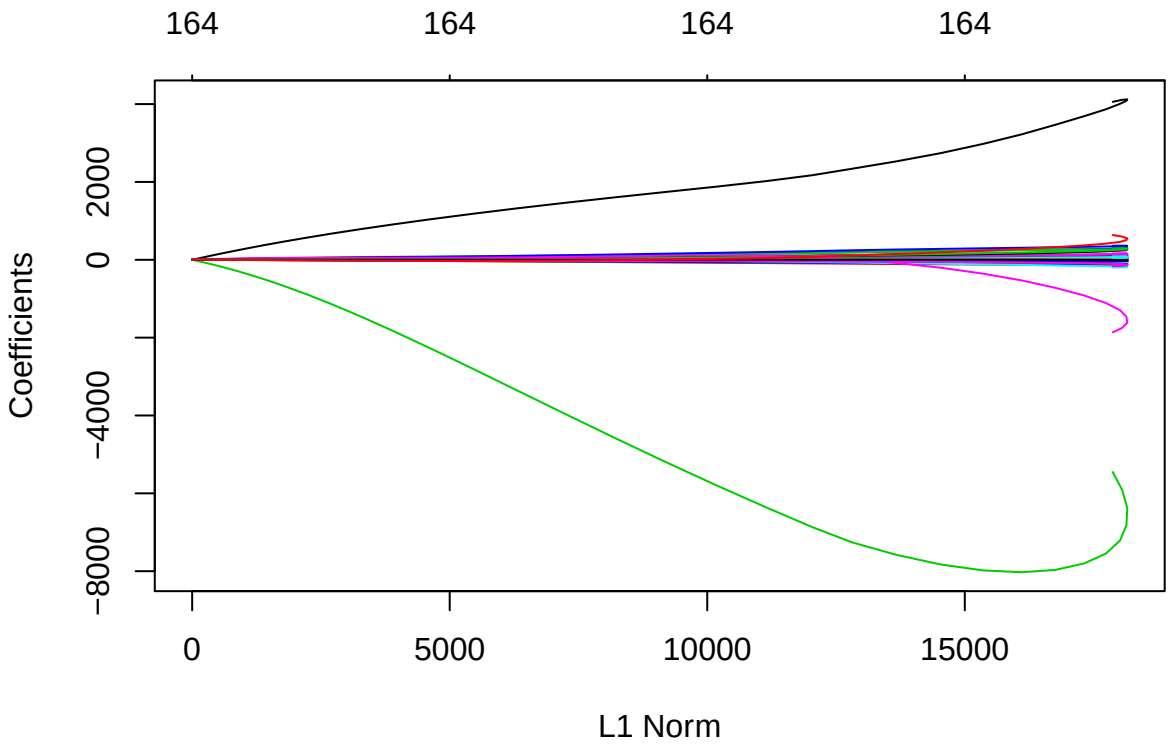
```
## [1] 0.09727985
```

Pour la validation effectuée par cv.glmnet, on donne un vecteur de N valeurs possibles de λ. Le jeu de données d'apprentissage est séparé en 10 sous jeux de données. On choisit successivement un de ces 10 plis considéré comme le jeu test et les 9 autres comme jeu d'apprentissage. Pour chaque pli choisi, l'algorithme calcule l'estimateur de Ridge pour chaque valeur de λ et calcule les erreurs quadratiques de prévision associées à chaque λ Ainsi, une matrice d'erreurs de taille $10 \times N$ est obtenue avec les erreurs de prévision pour chaque λ et pour chaque plis. La moyenne des erreurs associées à chaque λ est effectuée sur les plis et on obtient donc un vecteur de taille N contenant la moyenne des 10 erreurs obtenues pour chaque λ. Le λ à choisir est donc celui associé à l'erreur moyenne la plus faible.

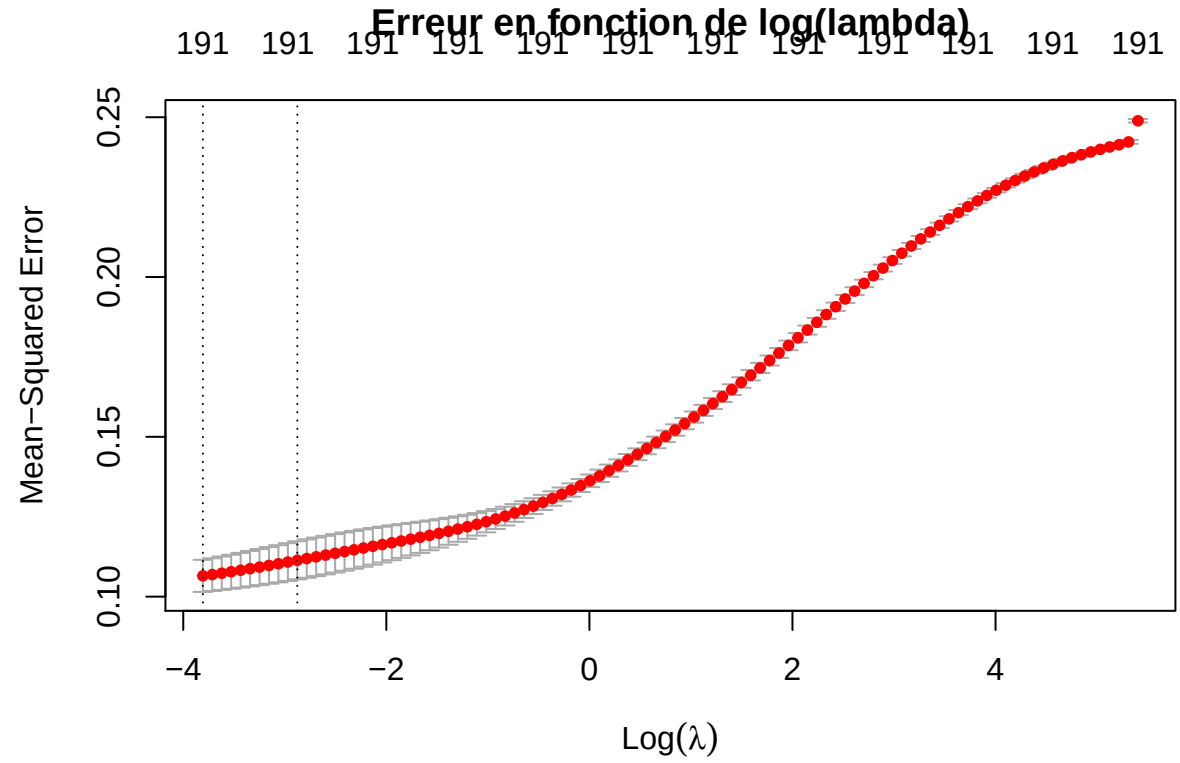
Dans notre cas le λ optimal obtenu par validation croisée est égal à 0,02221237 donc de l'ordre de 10^{-2} ce qui est proche de la plus petite valeur de lambda que nous avions donné dans la grille. Ainsi le lambda optimal régularise très peu voir quasiment pas la matrice $X'X$ et donc l'estimateur de Ridge optimal est proche de celui des moindres carrés ordinaires. Avec ce λ optimal, on obtient une performance assez bonne, en effet on obtient un taux d'erreur d'environ $9,7\%$ sur le jeu de données test des variables sélectionnées en partie 1.

Question 4

On va répliquer l'algorithme expliqué à la question précédente mais sur l'ensemble des données.



On choisit ensuite le λ optimal par validation croisée:




```
condition = ifelse(ridge.pred2 > 0.5, 1, 0)
1 - mean(condition == y2test) # [1] 0.09958506
```

```
## [1] 0.09958506
```

On trouve que le λ minimisant l'erreur par validation croisée vaut environ 0,02221237, donc il n'y a quasiment

pas de pénalité. La performance de la régression ridge appliquée à toutes les variables est d'environ 10%, ce qui est légèrement moins bon que la performance trouvée en question 3.

Conclusion

Dans cette étude, on peut voir en premier que la méthode des K plus proche voisins n'est pas adapté à nos données, en effet le taux d'erreur est d'environ 26% ce qui est beaucoup. Par ailleurs, avec la régression logistique, on obtient un taux d'erreur d'environ $10,4\%$ et avec la régression ridge, on obtient un taux d'erreur d'environ 10%. Ainsi, les deux dernières méthodes semblent toutes deux assez bonnes, le taux d'erreur est presque le même pour les deux, bien que légèrement meilleur en régression ridge. Comme la régression ridge permet aussi un compromis biais-variance, c'est cette méthode que l'on a préféré aux autres dans le cadre de nos données. Nous ne pouvons pas calculer la performance de généralisation de cette méthode, il faudrait pour ça avoir un troisième jeu de données indépendants de ceux d'apprentissage et de test et nous n'avons pas gardé de données pour ce faire.

Bonus

Nous pouvons aussi appliquer d'autres méthodes tels que le gradient boosting qui consiste à construire une suite de règles de classification faibles (un peu meilleures que le hasard) de manière récursive en minimisant une fonction de perte empirique par descente de gradient. Le gradient boosting pour la classification est basé sur les arbres de décision. L'algorithme ajuste un arbre de décision basique sur le jeu d'apprentissage il donne ensuite plus de poids aux observations dont la classe a été mal prédite puis la règle obtenue à l'itération suivante est appliquée sur les données pondérées et ainsi de suite jusqu'à obtenir une règle de classement minimisant la fonction de perte. Voici une implémentation de cet algorithme sur le jeu de données ci-dessous à travers la fonction xgboost. Nous aurions pu donner une grid d'hyperparametres tels que la profondeur des arbres, le nombre d'arbres, le pas de la descente de gradient etc, afin de choisir les paramètres optimaux par validation croisée mais avons préféré utliser la recherche automatique des hyperparamètres optimaux par validation croisée de la fonction qui va beaucoup plus vite. On observe que cette méthode est bien plus précise que toutes les autres méthodes ci dessus on obtient un taux de mauvais classement de 5% seulement.

```
library(caret)
library(xgboost)
fitControl <- trainControl(method = "repeatedcv", number = 10,
                           repeats = 3,  search = "random")

cl = makePSOCKcluster(4)
registerDoParallel(cl)
boosted.cv = train(Xtrain, Ytrain, method = "xgbTree", trControl = fitControl)
stopCluster(cl)
boosted.cv$bestTune
```

```
##   nrounds max_depth       eta    gamma colsample_bytree min_child_weight
## 2     525         6 0.4961538 3.624848        0.4037455                13
##   subsample
## 2 0.5665548
```

```
pred.boost = predict(boosted.cv, Xtest)
1-mean(pred.boost == Ytest) ##[1] 0.05578608
```

```
## [1] 0.06316275
```

Enfin nous avons tenté de classifier les données par séparateurs à vaste marge (support vector machines) consistant à trouver un hyperplan séparant les données en fonction de leur classes en maximisant la somme des distances des points les plus proches de cet hyperplan de chaque côté.

Si les données ne peuvent pas être séparées linéairement, l'espace des données est transofmé en un espace de dimension superieure dans lequel il est possible de les séparer par un hyperplan. Dans ce cas on obtient une taux de mauvais classement encore plus petit, de l'ordre de 3% seulement ce qui est encore mieux que le gradient boosting.

```
library(e1071)
# On implémente une support vector machines via svm
svmmod = svm(GENRE~., data = music[train,])

pred.svm = predict(svmmod, Xtest)
1-mean(pred.svm == Ytest) #[1] 0.03088981
```

```
## [1] 0.03088981
```