Compte Rendu Agrégation de modèles

Boubacar TRAORE (SID)

10 décembre 2019

```
library(magrittr) #pour utiliser les pipes
library(plotly)
library(reshape2) #pour utiliser melted
library(ggplot2) #viz
library(caret) #confusion matrix, etc.
#On utilise la librairie e1071 pour tester différents hyper paramètres...
library(e1071)
library(gbm)
#library(webshot)
```

1. Analyse des données

Nous allons étudier les données provenant de Spotify. L'objectif de l'analyse est de prédire si un client va 'liker' ou non une musique. En cela, nous avons une variable à prédire binaire où 1 veut dire like et 0 veut dire dislike. La base contient 16 variables. Nous ne toucherons pas aux variables "NumId" (identifiant dans la base), "Song" et "Band" (sont des variables catégorielles d'une variété trop importante).

```
data = read.csv2("Spotify.csv")
```

Les données ne sont pas au bon format. Nous allons mettre les variables quantitatives au bon format et lkes variables qualitatives au bon format.

```
set_to_numeric_var    = c('acousticness', 'danceability', 'energy', 'instrumentalness',
                          'liveness', 'loudness', 'speechiness', 'tempo', 'time_signature',
                          'valence')
set_to_categorial_var = c('key', 'mode', 'like')

data[set_to_numeric_var]    <- data[set_to_numeric_var]    %>% lapply(as.numeric)
data[set_to_categorial_var] <- data[set_to_categorial_var] %>% lapply(as.factor)

data %>% summary()
```

```
##      NumId       acousticness     danceability      duration
##  Min.   :   1   Min.   :   1.0   Min.   :  1.0   Min.   :  16042
##  1st Qu.: 505   1st Qu.: 404.0   1st Qu.:236.0   1st Qu.: 200015
##  Median :1009   Median : 755.0   Median :353.0   Median : 229261
##  Mean   :1009   Mean   : 714.4   Mean   :343.3   Mean   : 246306
##  3rd Qu.:1513   3rd Qu.:1005.0   3rd Qu.:459.0   3rd Qu.: 270333
##  Max.   :2017   Max.   :1394.0   Max.   :632.0   Max.   :1004627
##
##      energy    instrumentalness      key         liveness
##  Min.   :  1   Min.   :   1     1      :257   Min.   :  1.0
##  1st Qu.:298   1st Qu.:   1     0      :216   1st Qu.:319.0
##  Median :445   Median : 280     7      :212   Median :410.0
##  Mean   :425   Mean   : 378     9      :191   Mean   :411.4
##  3rd Qu.:573   3rd Qu.: 719     11     :187   3rd Qu.:523.0
##  Max.   :719   Max.   :1107     2      :184   Max.   :793.0
##                                 (Other):770
```

```
##     loudness       mode      speechiness         tempo
##  Min.   :   1.0   0: 782   Min.   :  1.0   Min.   :   1.0
##  1st Qu.: 486.0   1:1235   1st Qu.:127.0   1st Qu.: 482.0
##  Median : 929.0            Median :286.0   Median : 944.0
##  Mean   : 919.3            Mean   :336.7   Mean   : 955.7
##  3rd Qu.:1360.0            3rd Qu.:565.0   3rd Qu.:1436.0
##  Max.   :1808.0            Max.   :792.0   Max.   :1919.0
##
##  time_signature     valence      like                song
##  Min.   :1.000   Min.   :  1.0   0: 997   Jack          :   3
##  1st Qu.:3.000   1st Qu.:243.0   1:1020   River         :   3
##  Median :3.000   Median :426.0            1-800-273-8255:   2
##  Mean   :2.969   Mean   :429.7            Acamar        :   2
##  3rd Qu.:3.000   3rd Qu.:613.0            Alright       :   2
##  Max.   :4.000   Max.   :853.0            Annie         :   2
##                                           (Other)       :2003
##               band
##  Drake           :  16
##  Rick Ross       :  13
##  Disclosure      :  12
##  Backstreet Boys:  10
##  WALK THE MOON   :  10
##  Crystal Castles:   9
##  (Other)         :1947
```

Nous pouvons constater maintenant que les données sont au bon format et il n'y a pas besoin de faire des traitements de valeurs manquantes.

La variable à expliquer (like/dislike) semble bien proportionné dans l'échantillon, il n'y a donc pas besoin d'avoir recours à des méthodes de traitement spécifiques au données mal balancées.

Nous allons procéder à des analyses bivarirées des différents regresseurs avec la variable cible. Nous allons également faire des graphiques pour avoir une idée de l'impact des covariables X sur la variable à prédire Y.

Dans un premier temps, nous faisons des tests. Les variables explicatives quantitatives passent des tests de student d'égalité de moyenne et les variables qualitatives des tests de d'indépendance de Chi-2 par rapport à la variable Y.

```
plots = list()
N_numeric = length(set_to_numeric_var)
N_categ   = length(set_to_categorial_var) - 1
for(i in 1:N_numeric){
  var = set_to_numeric_var[i]
  plots[[i]] = plot_ly(y = data[,var], color = data$like, type = 'box') %>%
    layout(title = set_to_numeric_var[i])

  #test de student de difference  de moyenne par groupe de (like ou non like)
  test_stat = t.test(data[,var] ~ like, data = data)
  print(paste(var, " : p-value = ", test_stat$p.value))
}
```

```
## [1] "acousticness  : p-value =  4.58082376389948e-10"
## [1] "danceability  : p-value =  6.12556562339599e-17"
## [1] "energy  : p-value =  0.782724576190012"
## [1] "instrumentalness  : p-value =  7.44231321645048e-36"
## [1] "liveness  : p-value =  0.820732311528584"
## [1] "loudness  : p-value =  6.26214574989217e-16"
```

```
## [1] "speechiness  : p-value =  2.1384485690501e-11"
## [1] "tempo  : p-value =  2.89203188748171e-05"
## [1] "time_signature  : p-value =  0.0810612858199225"
## [1] "valence  : p-value =  1.17580254225447e-06"
```

Remarque : Seule la variable "time_signature" semble ne pas vraiment varier significativement selon la variable catégorique, elle a une p-value > 0.05. Nous pouvons dire que nous rejetons tojours l'hypothèse nulle d'égalité de moyenne dans les deux classes (Y=0 vs Y=1) pour toutes les autres variables au seuil de 5%. Les régresseurs quantitatifs semblent donc pertinents pour expliquer Y.

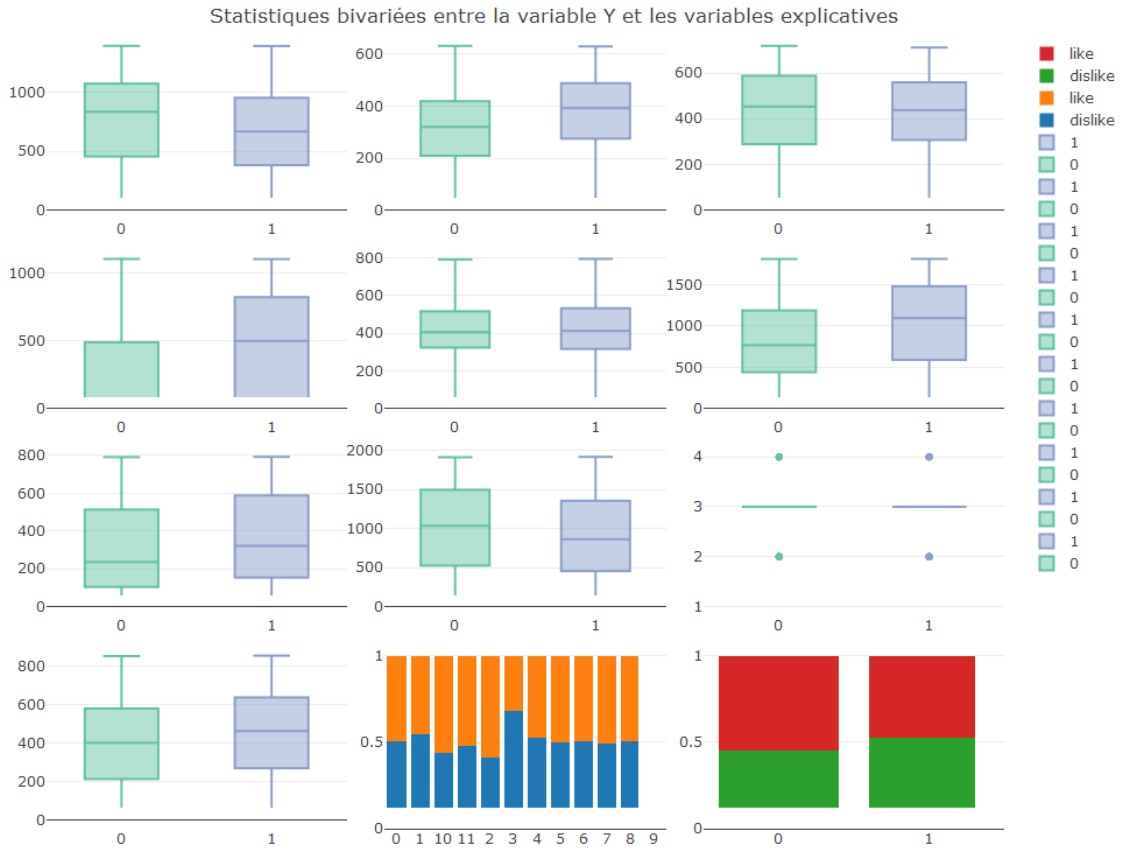
```
for(j in 1:N_categ){
  var = set_to_categorial_var[j]
  freq.table <- table(data$like, data[,var])
  df <- apply(freq.table, 1, function(x) x/freq.table%>% colSums()) %>% t() %>% round(3) %>% t() %>% da
  df[var] = data[,var] %>% levels()
  plots[[j+N_numeric]] = plot_ly(df, x=df[,var], y=~X0, type = 'bar', name = 'dislike') %>%
    add_trace(y = ~X1, name = 'like')  %>%
    layout(yaxis = list(title = 'Count'), barmode = 'stack')

  #test
  #test de proportion
  print(freq.table)
  chi.test = chisq.test(freq.table)
  print(paste(var, " : p-value = ", chi.test$p.value))
}
```

```
##
##       0   1   2   3   4   5   6   7   8   9  10  11
##   0 109 140  75  43  55  83  80 105  69  87  62  89
##   1 107 117 109  20  50  83  79 107  67 104  79  98
## [1] "key  : p-value =  0.0356648266693088"
##
##       0   1
##   0 351 646
##   1 431 589
## [1] "mode  : p-value =  0.00136008111804573"
```

La répartition des effectifs est significativement différente selon les likes/dislikes... On ne peut donc pas dire qu'il n'existe pas de lien d entre ces variables catégoriques et la variable Y.

Nous affichons maintenant les représentations graphiques.

```
p = subplot(plots, nrows = 4) %>% layout(title='Statistiques bivariées entre la variable Y et les varial


tmpFile <- tempfile(fileext = ".png")
export(p, file = tmpFile)
```



Nous affichons des boxplots pour les variables quantitatives selon la classe. Elles sont affichées selon l'odre présent dans la base de données (i.e. 'acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo', 'time_signature','valence').

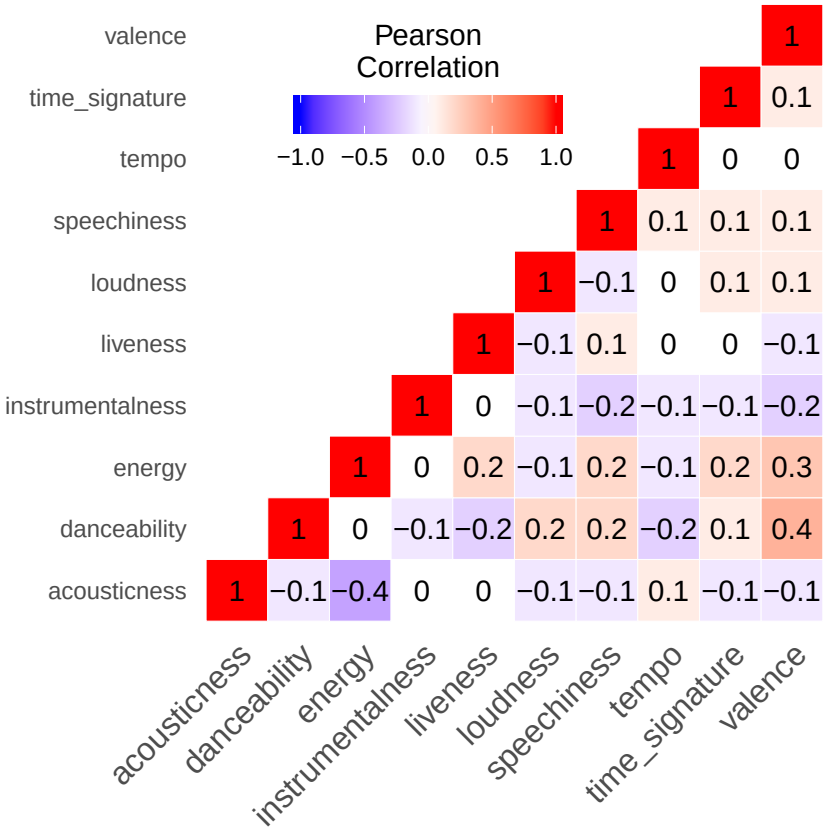
Quant aux variables variables catégorielles, nous affichons des diagrammes en bar normalisées (ce sont les proportions qui sont représentées). L'avant dernier graphique montre ainsi la répartition de like et dislike selon la clé de la chanson("key" qui voient ses valeurs annotées comme des entiers allant de 0 à 12). Nous remarquons que la variation de la proportion de like n'est pas très importante (visuellement parlant) lorsque la clé change. Nous pouvons soupconner un lien assez faible entre cette variable et la variable Y.

Nous voyons également sur le graphique la variable mode aussi ne fluctue pas beaucoup.

Nous nous proposons de faire un heatmap des corrélations entre les variables.

```
get_upper_tri <- function(cormat){
  cormat[lower.tri(cormat)]<- NA
  return(cormat)
}

melted_cormat = data[, set_to_numeric_var] %>%
  cor() %>% round(1) %>% get_upper_tri() %>% melt(na.rm = T)

#melted_cormat %>% plot_ly(x = ~Var1, y=~Var2, z=~value, type = 'heatmap')

# Créer un ggheatmap
```

```
ggheatmap <- ggplot(melted_cormat, aes(Var2, Var1, fill = value))+
  geom_tile(color = "white")+
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1,1), space = "Lab",
                       name="Pearson\nCorrelation") +
  theme_minimal()+ # minimal theme
  theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                   size = 12, hjust = 1))+
  coord_fixed() +
  geom_text(aes(Var2, Var1, label = value), color = "black", size = 4) +
  theme(
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    panel.grid.major = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    axis.ticks = element_blank(),
    legend.justification = c(1, 0),
    legend.position = c(0.6, 0.7),
    legend.direction = "horizontal")+
  guides(fill = guide_colorbar(barwidth = 7, barheight = 1,
                               title.position = "top", title.hjust = 0.5))

# Afficher heatmap
print(ggheatmap)
```



Nous constatons que les corrélations entre variables quantitatives sont toutes assez faibles. Nous pouvons supposer ne pas avoir de problème de multicolinéarité plus tard lors de la modélisation que nous abordons dans la section suivante.

2. Modélisation

Nous proposons deux modèles dans cette section : le RandomForest et le Boosting. Ces deux modèles sont appliquées sur nos données et nous selectionnons le meilleur à la fn.

```
#ON PREND uniquement les variables dont on a besoin
data = data[,c(set_to_numeric_var, set_to_categorial_var)]
```

Il faut commencer d'abord par découper notre base de données en entrainement (que nous allons appeler 'train') et en test (appelé 'test'). Seule la base d'entrainement sera choisie pour entrainer les modèles e trouver quelques bons hyper-paramètres. La partie 'test' servira à comparer les différents modèles.

Voici la fonction que nous allons fréquemment utiliser pour le découpage.

```
train.test.split = function(data, train.size, seed){
  set.seed(seed)
  sample <- sample.int(n = nrow(data), size = floor(train.size*nrow(data)), replace = F)
  train  <- data[sample, ]
  test   <- data[-sample, ]
  return(list('train'=train, 'test'=test))
}
```

Nous l'appliquons aux données.

```
splited.data = train.test.split(data, train.size = 0.8, seed = 7)
train = splited.data$train
test  = splited.data$test
train$like %>% summary()
```

```
##   0   1
## 797 816
```

```
test$like %>% summary()
```

```
##   0   1
## 200 204
```

On voit bien que les "Y" sont bien réparties entre entrainement et test.

Principe utilisée dans cette section:

1. Pour chaque méthode, on choisit d'abord les paramètres simples par défaut dont on va comparer les performances via une méthode de découpage train/validation sur laquelle nous allons évaluer rapidement les sorties de la méthode et donner une interprétation rapide de la performance générale du modèle.
2. En second lieu, nous faisons de la validation croisée sur tout le train (et non le inner.train) pour choisir les meilleurs paramètres que l'on va garder pour la partie test.

Pour ce faire, on va d'abord découper la partie entrainement encore en 2 parties. La première sera la base d'entrainement interne (nommé "inner.train") et la seconde sera la base de validation (nommé "validation"). Nous choisissons encore un taux de 20% pour la partie validation.

```
splited.data = train.test.split(train, train.size = 0.8, seed = 2)
inner.train  = splited.data$train
validation   = splited.data$test
inner.train$like %>% summary()
```

```
##   0   1
## 637 653
```

```
validation$like %>% summary()
```

```
##   0   1
## 160 163
```

Il y a égalementune bonne répartition des 'Y'.

2.1. Une méthode de bagging, le randomforest

2.1.1. Découpage 80/20

```
set.seed(123)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

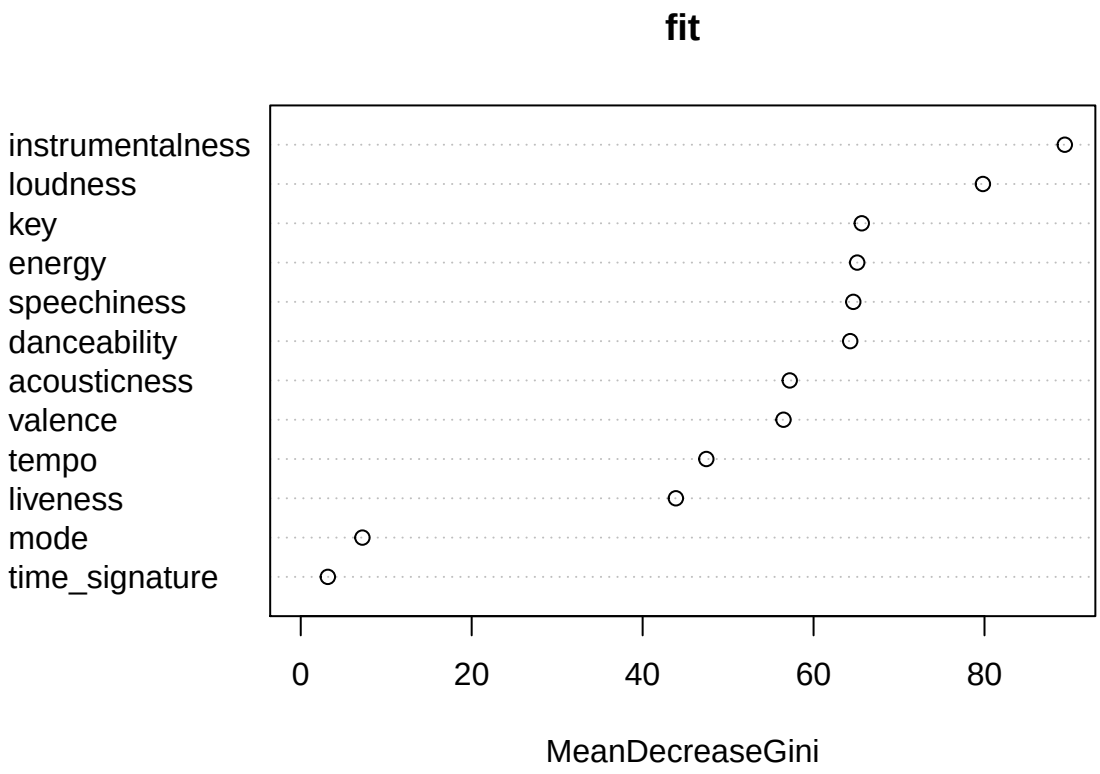
```
fit <- randomForest(like ~ ., data = inner.train)
fit
```

```
##
## Call:
##  randomForest(formula = like ~ ., data = inner.train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 24.34%
## Confusion matrix:
##     0   1 class.error
## 0 461 176   0.2762951
## 1 138 515   0.2113323
```

Par defaut il y a 500 arbres construits et p=3 pour chaque division. Sur l'entrainement on voit une erreur globale de 24% par la méthode OOB (Out Of Bag)

Jettons un coup d'oeil à l'importance des variables.

```
varImpPlot(fit)
```



Ce plot nous montre que la variable 'instrumentalnes' a une tres grande importance dans l'explication des likes/dislikes.

Nous allons maintenant voir les performances générales en prédiction de l'algorithme su rla base de validation (on rappelle que train = inner.train + validation et que seul 'inner.train' a été utilisé pour l'entrainement).

```
prediction <-predict(fit, validation)
confusionMatrix(prediction, validation$like)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 119  28
##          1  41 135
##
##                Accuracy : 0.7864
##                  95% CI : (0.7376, 0.8298)
##     No Information Rate : 0.5046
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5724
##  Mcnemar's Test P-Value : 0.1486
##
##             Sensitivity : 0.7438
##             Specificity : 0.8282
##          Pos Pred Value : 0.8095
```

```
##          Neg Pred Value : 0.7670
##              Prevalence : 0.4954
##          Detection Rate : 0.3684
##    Detection Prevalence : 0.4551
##       Balanced Accuracy : 0.7860
##
##        'Positive' Class : 0
##
```

On retrouve une performance de 78%. Ce qui n'est pas mal.

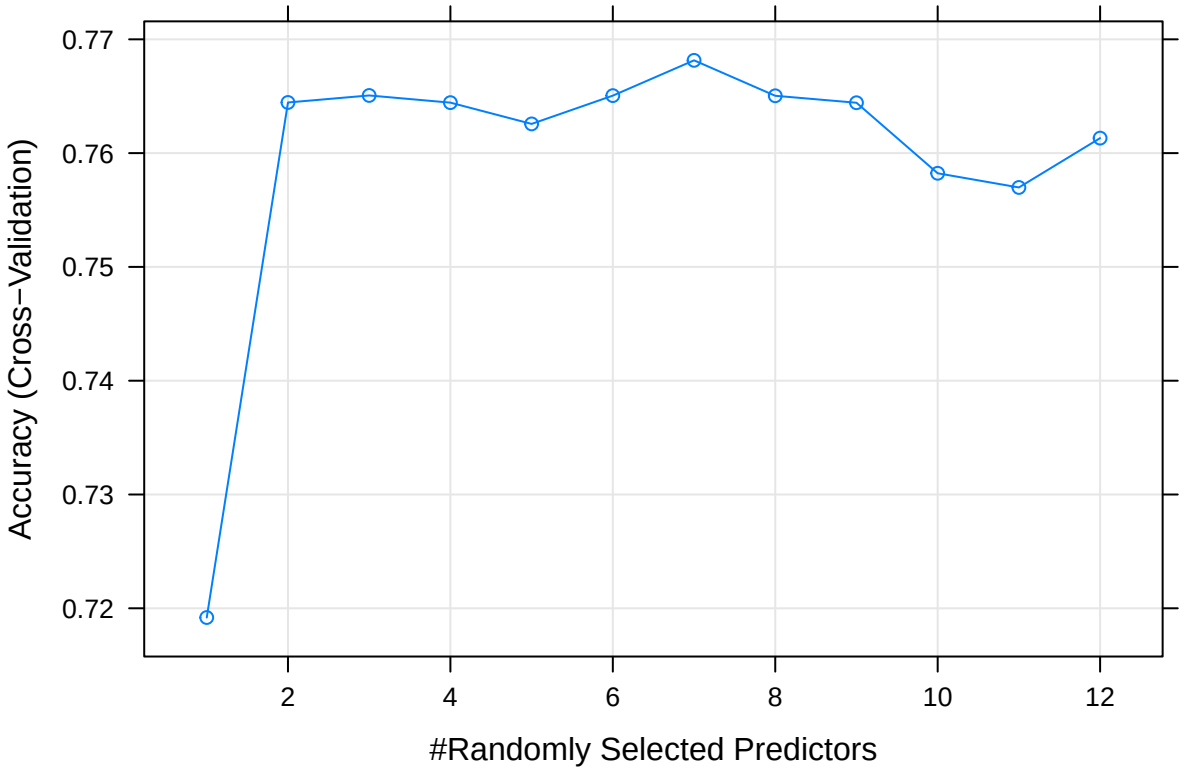
2.1.2. Validation croisée et sélection d'hyper-paramètre

On sait que les 12 premieres variables sont nos regresseurs X.

```
set.seed(123)
control  <- trainControl(method = "cv", number = 5)
tunegrid <- expand.grid(mtry = 1:12)
set.seed(123)
custom <- train(like~., data=train, method="rf", ntree=500,
                tuneGrid=tunegrid, trControl=control)
summary(custom)
```

```
##                 Length Class      Mode
## call               5   -none-     call
## type               1   -none-     character
## predicted       1613   factor     numeric
## err.rate        1500   -none-     numeric
## confusion          6   -none-     numeric
## votes           3226   matrix     numeric
## oob.times       1613   -none-     numeric
## classes            2   -none-     character
## importance        22   -none-     numeric
## importanceSD       0   -none-     NULL
## localImportance    0   -none-     NULL
## proximity          0   -none-     NULL
## ntree              1   -none-     numeric
## mtry               1   -none-     numeric
## forest            14   -none-     list
## y               1613   factor     numeric
## test               0   -none-     NULL
## inbag              0   -none-     NULL
## xNames            22   -none-     character
## problemType        1   -none-     character
## tuneValue          1   data.frame list
## obsLevels          2   -none-     character
## param              1   -none-     list
```

```
plot(custom)
```

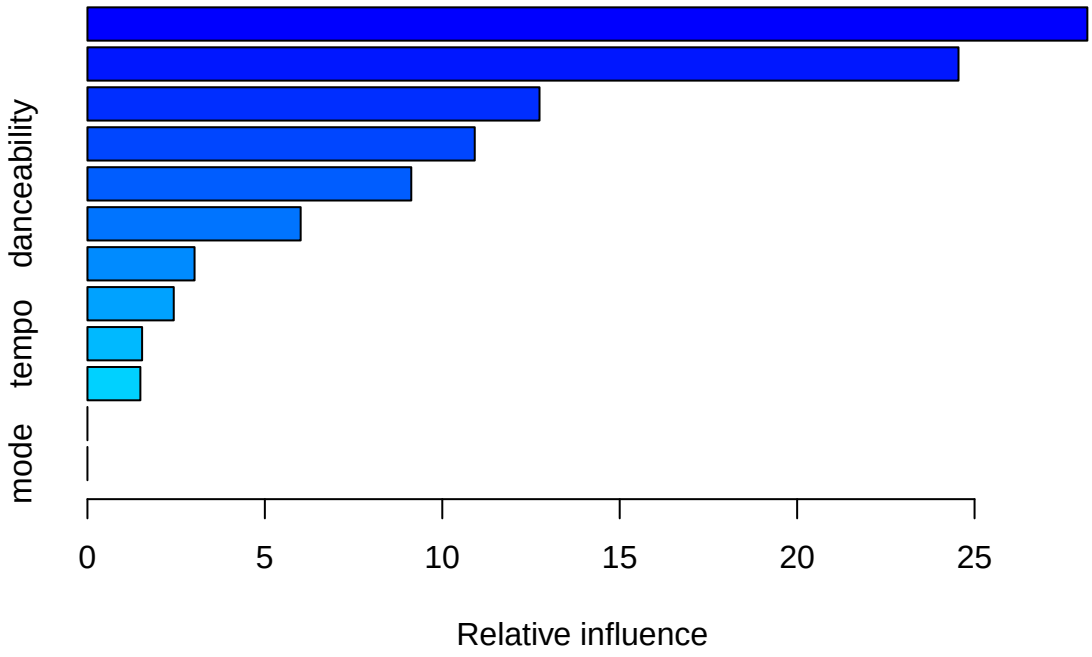


Nous trouvons une valeur optimale pour 'mtry'=7. Nous allons garder cette valeur pour la partie test.

2.2 Le Boosting

2.2.1. Découpage 80/20

```
set.seed(123)
boosting = gbm(as.character(like)~., data=inner.train, distribution = "bernoulli")
summary(boosting)
```




```
##                               var   rel.inf
## instrumentalness instrumentalness 28.178444
## loudness                 loudness 24.547432
## speechiness           speechiness 12.740061
## energy                     energy 10.914879
## danceability         danceability  9.126180
## acousticness         acousticness  6.008728
## key                           key  3.018940
## liveness                 liveness  2.432639
## tempo                       tempo  1.540732
## valence                   valence  1.491965
## time_signature     time_signature  0.000000
## mode                         mode  0.000000
```

Ce graphe montre l'influence des variables dans la performance de ce modèle. Nous retrouvons la même variable en tête que dans le RandomForest.

Que donne la prédiction ?

```
set.seed(123)
prediction2 <-predict.gbm(boosting, validation, n.trees=100, type="response")
prediction2 = prediction2 %>% sapply(function(x) ifelse(x>.5, 1, 0))
confusionMatrix(as.factor(prediction2), validation$like)
```

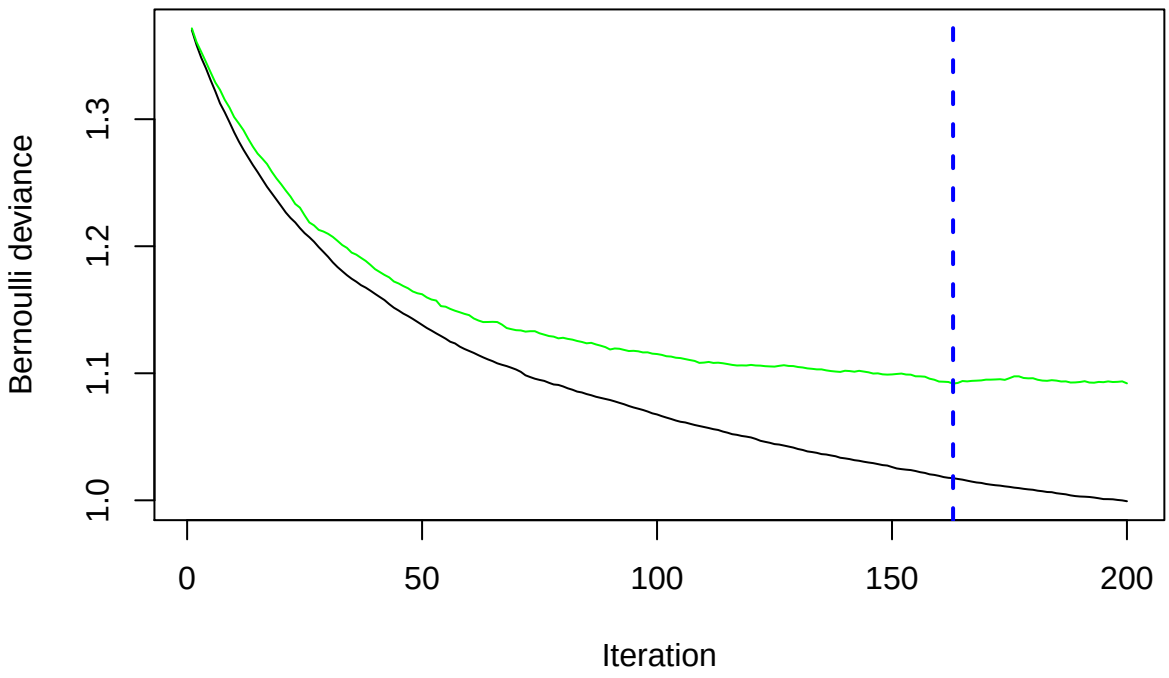
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

```
##          0 116  48
##          1  44 115
##
##                Accuracy : 0.7152
##                  95% CI : (0.6626, 0.7638)
##     No Information Rate : 0.5046
##     P-Value [Acc > NIR] : 1.109e-14
##
##                   Kappa : 0.4304
##  Mcnemar's Test P-Value : 0.7545
##
##             Sensitivity : 0.7250
##             Specificity : 0.7055
##          Pos Pred Value : 0.7073
##          Neg Pred Value : 0.7233
##              Prevalence : 0.4954
##          Detection Rate : 0.3591
##    Detection Prevalence : 0.5077
##       Balanced Accuracy : 0.7153
##
##        'Positive' Class : 0
##
```

Nous obtenons une précision de 71% avec 100 arbres construits, un peu moins que le RandomForest.

2.2.2. Nombre optimal d'arbres à construire

```
set.seed(123)
boost.cv = gbm(as.character(like)~., data=train, distribution = "bernoulli", n.trees = 200, cv.folds = 5

set.seed(123)
gbm.perf(boost.cv)
```




```
## [1] 163
```

Ce resultat nous indique une valeur optimale B=163. C'est ce que nous prendrons pour la partie test.

2.3. Choix final du modèle

Dans cette partie, nous prenons les meilleurs paramètres obtenus pour enfin utiliser la base de données test.

```
# selection finale de modèles
final.rf = randomForest(like ~ ., data = train, mtry=7)
rf.pred  = final.rf %>% predict(test)
confusionMatrix(rf.pred, test$like)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 141  37
##          1  59 167
##
##                Accuracy : 0.7624
##                  95% CI : (0.7178, 0.8031)
##     No Information Rate : 0.505
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5242
##  Mcnemar's Test P-Value : 0.03209
##
```

```
##             Sensitivity : 0.7050
##             Specificity : 0.8186
##          Pos Pred Value : 0.7921
##          Neg Pred Value : 0.7389
##              Prevalence : 0.4950
##          Detection Rate : 0.3490
##    Detection Prevalence : 0.4406
##       Balanced Accuracy : 0.7618
##
##        'Positive' Class : 0
##
```

```r
set.seed(123)
final.boost = gbm(as.character(like)~., data=train, distribution = "bernoulli", n.trees = 163)
final.boost <-predict.gbm(boosting, test, n.trees=163, type="response")
final.boost = prediction2 %>% sapply(function(x) ifelse(x>.5, 1, 0))
confusionMatrix(as.factor(final.boost), validation$like)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 116  48
##          1  44 115
##
##                Accuracy : 0.7152
##                  95% CI : (0.6626, 0.7638)
##     No Information Rate : 0.5046
##     P-Value [Acc > NIR] : 1.109e-14
##
##                   Kappa : 0.4304
##  Mcnemar's Test P-Value : 0.7545
##
##             Sensitivity : 0.7250
##             Specificity : 0.7055
##          Pos Pred Value : 0.7073
##          Neg Pred Value : 0.7233
##              Prevalence : 0.4954
##          Detection Rate : 0.3591
##    Detection Prevalence : 0.5077
##       Balanced Accuracy : 0.7153
##
##        'Positive' Class : 0
##
```

Nous finissons par chosir le RandomForest qui arrive à a voir une meilleure capacité de généralisation avec une meilleure performance sur les données test par rapport au Boosting.