Module 3 - Analyse statistique avec R - Séance 6

DUBii 2019

Leslie REGAD (Université Paris Diderot)

2019-02-26

Contents

Le but de ce TP est de Pour ce TP, vous utiliserez les mêmes données que celles que vous avez utilisées pour le TP clustering (séance 4) :

- fichier `BIC_log2-norm-counts_edgeR_DEG_top_1000.tsv.gz` qui correspond au fichier d'expression pour les 1000 gènes (lignes) les plus significatifs pour 819 échantillons (colonnes).
- fichier `BIC_sample-classes.tsv.gz` qui contient les étiquettes des 819 échantillons.

Sur le serveur Rstudio de l'IFB-core-cluster, les données sont dans le répertoire : `/shared/projects/du_bii_2019/data/modul`

Etape 1 : Préparation des données

1. Ouvrez le fichier d'expression des gènes en utilisant la commande `read.table()`. Stockez ce data.frame dans l'objet `BIC.expr`. Vérifiez la taille du data.frame généré en utilisant la commande `dim()`.

```
data.folder="data/BIC/"
BIC.expr.file <- file.path(data.folder, "BIC_log2-norm-counts_edgeR_DEG_top_1000.tsv.gz")
BIC.expr <- read.table(file = BIC.expr.file, header = TRUE)
dim(BIC.expr)
```

```
[1] 1000  819
```

2. Ouvrez le fichier qui contient les étiquettes des échantillons en utilisant la commande `read.table()`. Stockez ce data.frame dans l'objet `BIC.sample.classes`.

```
BIC.sample.classes <- read.table(file.path(data.folder, "BIC_sample-classes.tsv.gz"),header = TRUE)
```

- Vérifier le nombre d'échantillons disponibles dans ce jeu de données.

```
dim(BIC.sample.classes)
```

```
[1] 819   4
```

- Déterminer le type de variables disponibles dans ce jeu de données en utilisant la fonction `summary()`.

```
summary(BIC.sample.classes)
```

```
      cancer.type        ER1            PR1            Her2
 Basal.like  :131   Negative:184   Negative:267   Negative:631
 HER2pos     : 41   Positive:635   Positive:552   Positive:188
 Luminal.A   :422
 Luminal.B   :118
 Unclassified:107
```

Comme vous pouvez le voir, il y a 5 types de cancer, dont un type qui est `Unclassified`. Lors de la prédiction du type de cancer, ce type risque de biaiser les résultats.

3. Supprimer les échantillons correspondant au type `Unclassified` dans les deux data.frames.

- En utilisant la fonction `which()` identifier les lignes du data.frame `BIC.sample.classes` qui correspondent au type `Unclassified`. Vérifiez que vous avez bien sélectionné 107 individus.

```
ind.uncl <- which(BIC.sample.classes[,"cancer.type"]=="Unclassified")
length(ind.uncl)
```

```
[1] 107
```

- Supprimer ces individus dans le data.frame `BIC.sample.classes` en utilisant l'indexation négative. Vérifiez la taille du nouveau data.frame.

```
BIC.sample.classes <- BIC.sample.classes[-ind.uncl,]
dim(BIC.sample.classes)
```

```
[1] 712   4
```

- Supprimer les échantillons correspondant au type `Unclassified` du data.frame `BIC.expr`. Attention, dans le data.frame `BIC.expr` les échantillons sont présentés en colonnes.

```
BIC.expr <- BIC.expr[, -ind.uncl]
dim(BIC.expr)
```

```
[1] 1000  712
```

- Pour construire les modèles, vous n'allez pas utiliser les quatre types de cancers, mais seulement deux :
  - Luminal.A,
  - non Luminal.A. Vous devez donc transformer la colonne `cancer.type` du data.frame `BIC.sample.classes` en une variable qualitative à deux classes : "Luminal.A" ou "no.Luminal.A".
- Créer le vecteur `new.type` qui contient 712 fois (`nrow(BIC.sample.classes)`) la valeur "Luminal.A".

```
new.type <- rep("Luminal.A", length = nrow(BIC.sample.classes))
```

- Identifier les individus qui ne contiennent pas "Luminal.A" dans la première colonne (colonne `cancer.type`) du data.frame `BIC.sample.classes`.

```
ind.noLA <- which(BIC.sample.classes[,"cancer.type"]!="Luminal.A")
```

- Pour ces individus assigner la valeur "no.Luminal.A" au vecteur `new.type`.

```
new.type[ind.noLA] = "no.Luminal.A"
```

- remplacer la colonne `cancer.type` du data.frame `BIC.sample.classes` par le vecteur `new.type` que vous aurez transformé en facteur (fonction `as.factor()`)

```
BIC.sample.classes[,"cancer.type"] = as.factor(new.type)
```

4. Suppression des variables corrélées

La première étape du nettoyage du jeu de données correspond à supprimer les variables (ici les gènes) corrélés. Pour cela, vous allez utiliser la fonction `findCorrelation()` du package `caret`. Pour utilisez cette fonction, il faut lui donner en entrée la matrice de corrélation entre les variables, et le seuil de corrélation à partir duquel on considère que deux variables sont corrélées.

- Calculer la matrice de corrélation entre les gènes différentiellement exprimés. Comme dans le data.frame `BIC.expr` les gènes sont en lignes, pensez à transposer votre data.frame lors.

```
mat.cor <- cor(t(BIC.expr))
```

- Identifier les gènes à supprimer en utilisant un seuil de corrléation de 0.8 et la fonction `findCorrelation()`. Combien de gènes allez vous supprimer.

```
library(caret)
var.supp = findCorrelation(mat.cor, cutoff = 0.8)
length(var.supp)
```

```
[1] 36
```

- Supprimer ces gènes du data.frame `BIC.expr`.

```
BIC.expr <- BIC.expr[-var.supp,]
dim(BIC.expr)
```

```
[1] 964 712
```

5. Pour créer les modèles de prédiction, il est nécessaire d'avoir un data.frame qui regroupe les gènes et les différents types de cancer.

- Créer le data.frame `df.data` qui contient les échantillons en lignes et les gènes ainsi que le type de cancer en colonne. Pour cela, utilisez la commande `data.frame()`.
  - La première colonne de votre data.frame `df.data` doit correspondre aux types de cancer.
  - les colonnes 2 à 965 doivent correspondre aux gènes.

```
df.data <- data.frame(BIC.sample.classes[,1], t(BIC.expr))
```

- Assigner "cancer.type" comme nom de colonne à la première colonne de `df.data`.

```
colnames(df.data)[1] <- "cancer.type"
```

Préparation des échantillons d'apprentissage et de test

- L'échantillon d'apprentissage est l'échantillon qui va permettre d'apprendre les modèles. Il sera constitué de 2/3 des échantillons sélectionnés de manière aléatoire.

- L'échantillon de test est l'échantillon qui va permettre d'estimer les performances non biaisées des modèles. Il sera constitué du 1/3 des échantillons restant.

1. En utilisant la fonction `sample()` créez un vecteur qui contient le numéro de 2/3 des individus choisis aléatoirement. Ces individus vont constituté les individus du jeu d'apprentissage.

```
ind.app <- sample(1:nrow(df.data), size=2/3*nrow(df.data))
```

2. Créer le data.frame `mat.app` qui contient les valeurs des valeurs d'expression de gènes pour les individus choisis aléatoirement. Cette matrice correspond au jeu d'apprentissage.

```
mat.app <- df.data[ind.app,]
dim(mat.app)
```

```
[1] 474 965
```

3. Créer le data.frame `mat.test` qui contient les valeurs d'expression de gènes pour les individus restant (1/3). Cette matrice correspond au jeu test

```
mat.test <- df.data[-ind.app,]
dim(mat.test)
```

```
[1] 238 965
```

Validation des deux échantillons.

1. Calculer une Analyse en composantes principale en utilisant le data.frame `df.data`. Pour cela, utilisez la fonction `PCA()` de la librairie `FactoMineR`.

```
library(FactoMineR)
pca.res <- PCA(df.data[,-1], graph=FALSE)
```

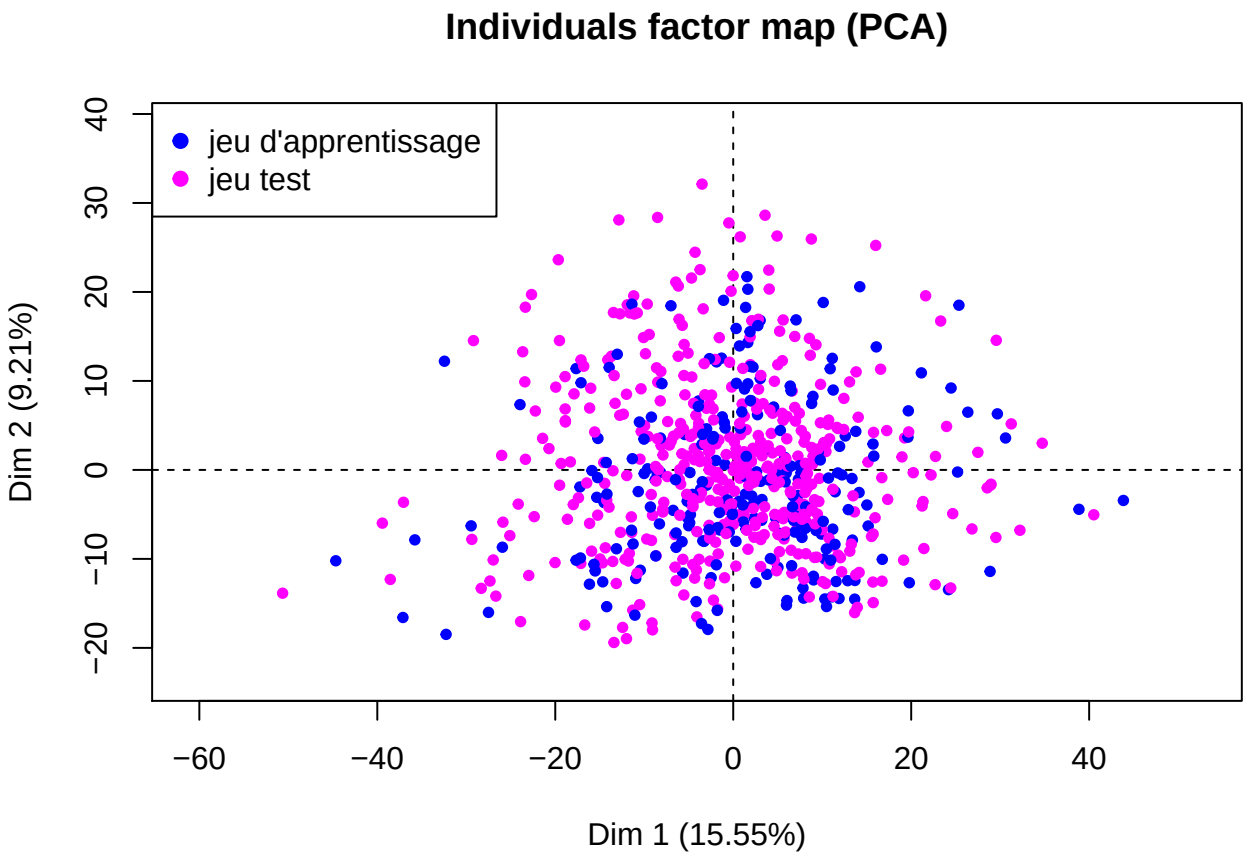
2. Créer un vecteur `vcol.set` qui contiendra les couleurs associés à chaque indidivu suivant s'il appartient au jeu d'apprentissage (en magenta) ou au jeu test (en bleu). Pour cela, suiver les étapes suivantes :

- créer le vecteur `vcol.set` qui contient 712 fois (`nrow(df.data)`) la couleur bleu.
- Dans ce vecteur, remplacer la couleur bleu des individus appartenant à l'échantillon d'apprentissage (`ind.app`) par magenta.

```
vcol.set <- rep("blue", length = nrow(df.data))
vcol.set[ind.app] <- "magenta"
```

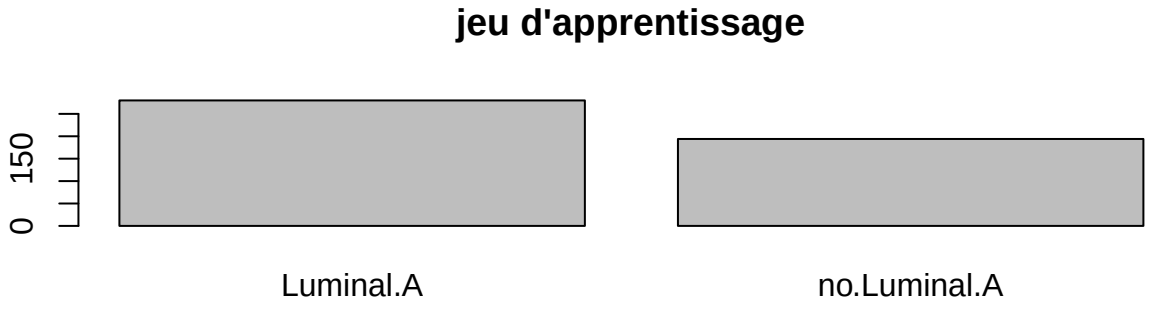
3. Représentez la projection des individus sur les 2 premières composantes principales en colorant les individus suivant l'échantillon auquel ils appartiennent.

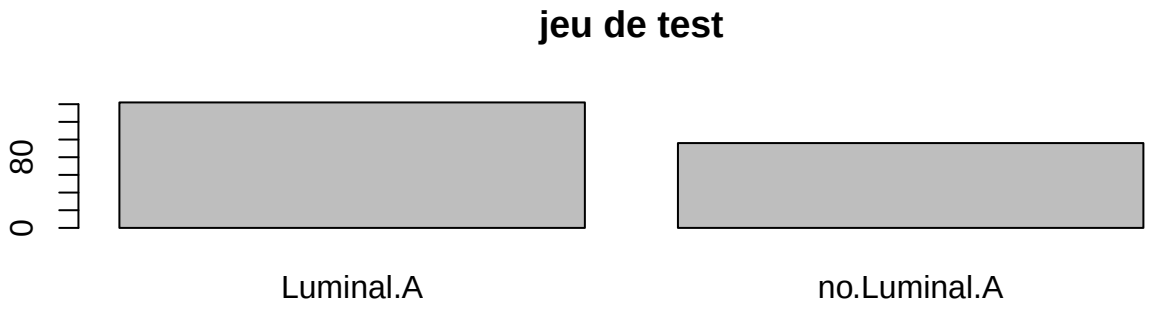
```
plot(pca.res, col.ind = vcol.set, label="none")
legend("topleft", legend=c("jeu d'apprentissage", "jeu test"), col=c("blue","magenta"), pch=19)
```



4. Déterminer la répartition des deux types de cancer dans les échantillons d'apprentissage et test.

```
par(mfrow=c(2,1))
barplot(table(mat.app[,"cancer.type"]), main="jeu d'apprentissage")
barplot(table(mat.test[,"cancer.type"]), main="jeu de test")
```





Prédiction du type de cancer en utilisant une approche de CART

1. A partir des données de l'échantillon d'apprentissage, construisez un modèle de CART permettant de prédire le type de cancer en fonction de l'expression des gènes. Pour cela, utilisez la fonction `rpart()` du package `rpart`.

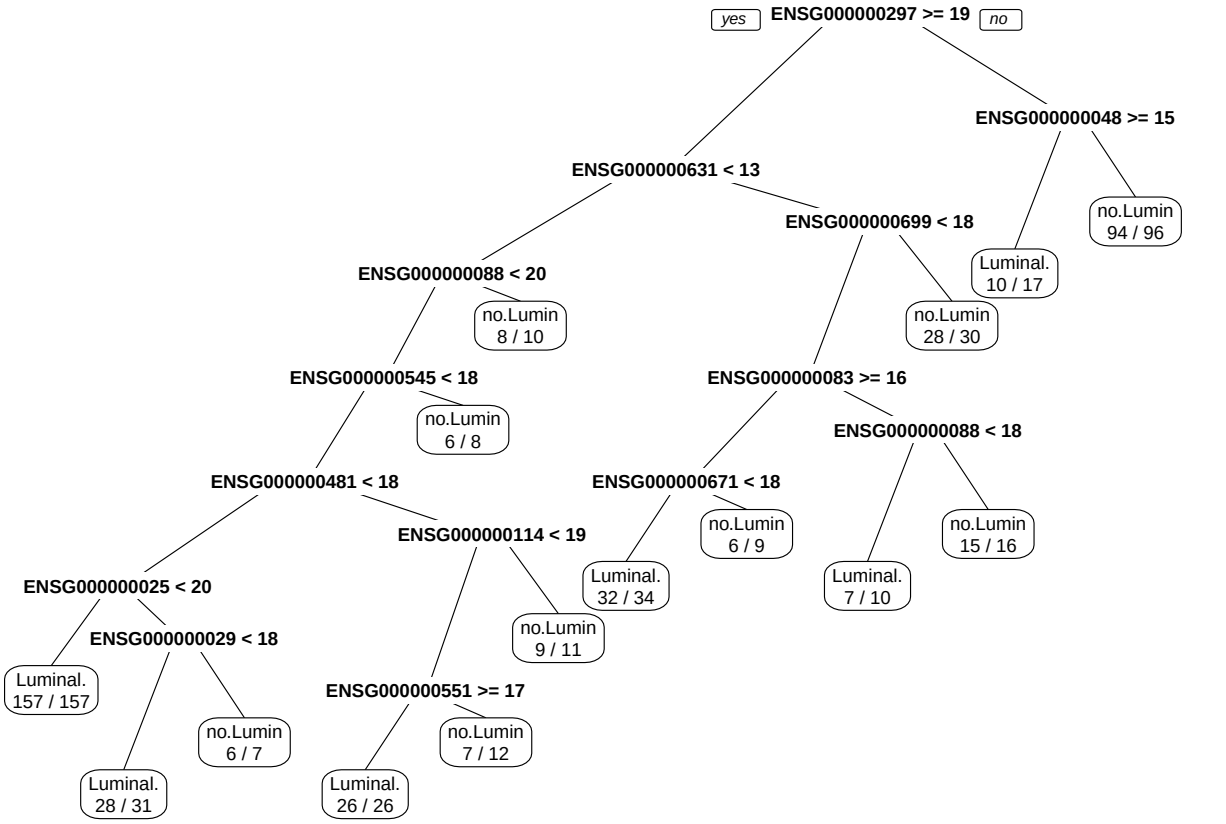
```
library(rpart)
library(rpart.plot)

rpart.fit <-rpart(cancer.type~., data = mat.app)
rpart.fit
```

```
n= 474

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 474 194 Luminal.A (0.59071730 0.40928270)
    2) ENSG00000029725.16>=18.9784 361  93 Luminal.A (0.74238227 0.25761773)
      4) ENSG00000063127.15< 13.09271 262  39 Luminal.A (0.85114504 0.14885496)
        8) ENSG00000008838.18< 20.09066 252  31 Luminal.A (0.87698413 0.12301587)
         16) ENSG00000054598.6< 17.8197 244  25 Luminal.A (0.89754098 0.10245902)
           32) ENSG00000048162.20< 17.99457 195   9 Luminal.A (0.95384615 0.04615385)
             64) ENSG00000002549.12< 19.67172 157   0 Luminal.A (1.00000000 0.00000000) *
             65) ENSG00000002549.12>=19.67172 38   9 Luminal.A (0.76315789 0.23684211)
              130) ENSG00000002919.14< 17.85516 31   3 Luminal.A (0.90322581 0.09677419) *
              131) ENSG00000002919.14>=17.85516 7   1 no.Luminal.A (0.14285714 0.85714286) *
           33) ENSG00000048162.20>=17.99457 49  16 Luminal.A (0.67346939 0.32653061)
```

```
        66) ENSG00000011454.16< 18.72203 38   7 Luminal.A (0.81578947 0.18421053)
         132) ENSG00000055163.19>=16.85609 26   0 Luminal.A (1.00000000 0.00000000) *
         133) ENSG00000055163.19< 16.85609 12   5 no.Luminal.A (0.41666667 0.58333333) *
        67) ENSG00000011454.16>=18.72203 11   2 no.Luminal.A (0.18181818 0.81818182) *
     17) ENSG00000054598.6>=17.8197 8   2 no.Luminal.A (0.25000000 0.75000000) *
    9) ENSG00000008838.18>=20.09066 10   2 no.Luminal.A (0.20000000 0.80000000) *
  5) ENSG00000063127.15>=13.09271 99  45 no.Luminal.A (0.45454545 0.54545455)
   10) ENSG00000069998.12< 17.6825 69  26 Luminal.A (0.62318841 0.37681159)
     20) ENSG00000008382.15>=16.20034 43   8 Luminal.A (0.81395349 0.18604651)
       40) ENSG00000067177.14< 17.502 34   2 Luminal.A (0.94117647 0.05882353) *
       41) ENSG00000067177.14>=17.502 9   3 no.Luminal.A (0.33333333 0.66666667) *
     21) ENSG00000008382.15< 16.20034 26   8 no.Luminal.A (0.30769231 0.69230769)
       42) ENSG00000008838.18< 18.4575 10   3 Luminal.A (0.70000000 0.30000000) *
       43) ENSG00000008838.18>=18.4575 16   1 no.Luminal.A (0.06250000 0.93750000) *
   11) ENSG00000069998.12>=17.6825 30   2 no.Luminal.A (0.06666667 0.93333333) *
 3) ENSG00000029725.16< 18.9784 113  12 no.Luminal.A (0.10619469 0.89380531)
   6) ENSG00000004838.13>=15.44732 17   7 Luminal.A (0.58823529 0.41176471) *
   7) ENSG00000004838.13< 15.44732 96   2 no.Luminal.A (0.02083333 0.97916667) *
```

2. En utilisant la fonction `prp()` de la librairie `rpart.plot()`, représentez le modèle obtenu.

```
prp(rpart.fit, extra=2)
```



3. Sur les données de l'échantillon d'apprentissage calculé le taux de bien prédit.

- En utilisant la fonction `predict()`, prédisez le type de cancer pour chaque individu de l'échantillon d'apprentissage. Stockez ces valeurs prédites dans le vecteur `pred.app`. En fait la fonction `predict()`est une fonction générique. Dans ce cas, vous utilisez la fonction `predict()` appliquée à un objet `rpart`. Pour avoir plus d'information sur les arguments de la fonction, vous devez utiliser l'aide de la fonction

`predict.rpart()`. N'oubliez pas de préciser l'argument `type="class"` pour préciser que vous faites de la classification.

```
pred.app <- predict(rpart.fit, newdata = mat.app, type="class")
```

4. A l'aide de la fonction `table()`, calculez la matrice de confusion entre les données prédites sur l'échantillon d'apprentissage (contenue dans le vecteur `pred.app`) et les vraies valeurs (contenue dans la colonne `cancer.type` du data.frame `mat.app`).

```
tc.rpart.app <- table(pred.app, mat.app[,"cancer.type"])
tc.rpart.app
```

```
pred.app       Luminal.A no.Luminal.A
  Luminal.A          260           15
  no.Luminal.A        20          179
```

5. Calculer le taux de bien prédit ($TBP = \frac{VP+VN}{VP+VN+FP+FN}$) sur les données de l'apprentissage

```
TBP.rpart.app <- sum(diag(tc.rpart.app))/sum(tc.rpart.app)
TBP.rpart.app
```

```
[1] 0.9261603
```

6. En utilisant la même procédure que celle utilisée pour calculer le taux de bien prédit sur le jeu d'apprentissage, calculez le taux de bien prédit sur l'échantillon test

```
pred.test <- predict(rpart.fit, newdata = mat.test, type="class")

tc.rpart.test <- table(pred.test, mat.test[,"cancer.type"])
tc.rpart.test
```

```
pred.test      Luminal.A no.Luminal.A
  Luminal.A          110           45
  no.Luminal.A        32           51
```

```
TBP.rpart.test <- sum(diag(tc.rpart.test))/sum(tc.rpart.test)
TBP.rpart.test
```

```
[1] 0.6764706
```

En comparant les performances obtenues sur l'échantillon d'apprentissage et de test, vous constatez que les performances obtenues sur l'échantillon d'apprentissage (TBP=0.93) sont plus grandes que celles obtenues sur l'échantillon test (TBP=0.68). Ainsi, vous observez un phénomène de sur-apprentissage. Le modèle apprend très bien les données de l'échantillon d'apprentissage, mais prédit moins bien les données du jeu test. Pour diminuer ce sur-apprentissage, une des solutions serait d'élaguer l'arbre, c'est-à-dire d'enlever certaines branches. Ainsi, le nouveau modèle obtenu serait moins performant sur les données d'apprentissage, mais plus performant sur les données du jeu test.

Prédiction du type de cancer en utilisant une approche de CART

1. A partir des données de l'échantillon d'apprentissage, construisez une forêt aléatoire permettant de prédire le type de cancer en fonction de l'expression des gènes.

- Pour cela, utilisez la fonction `randomForest()` du package `randomForest` avec les paramètres `mtry` et `mtree` par défaut. Stockez le modèle dans l'objet `rf.fit`.

```
library(randomForest)
rf.fit <- randomForest(cancer.type~., data = mat.app)
```

- Affichez à l'écran le modèle créé.

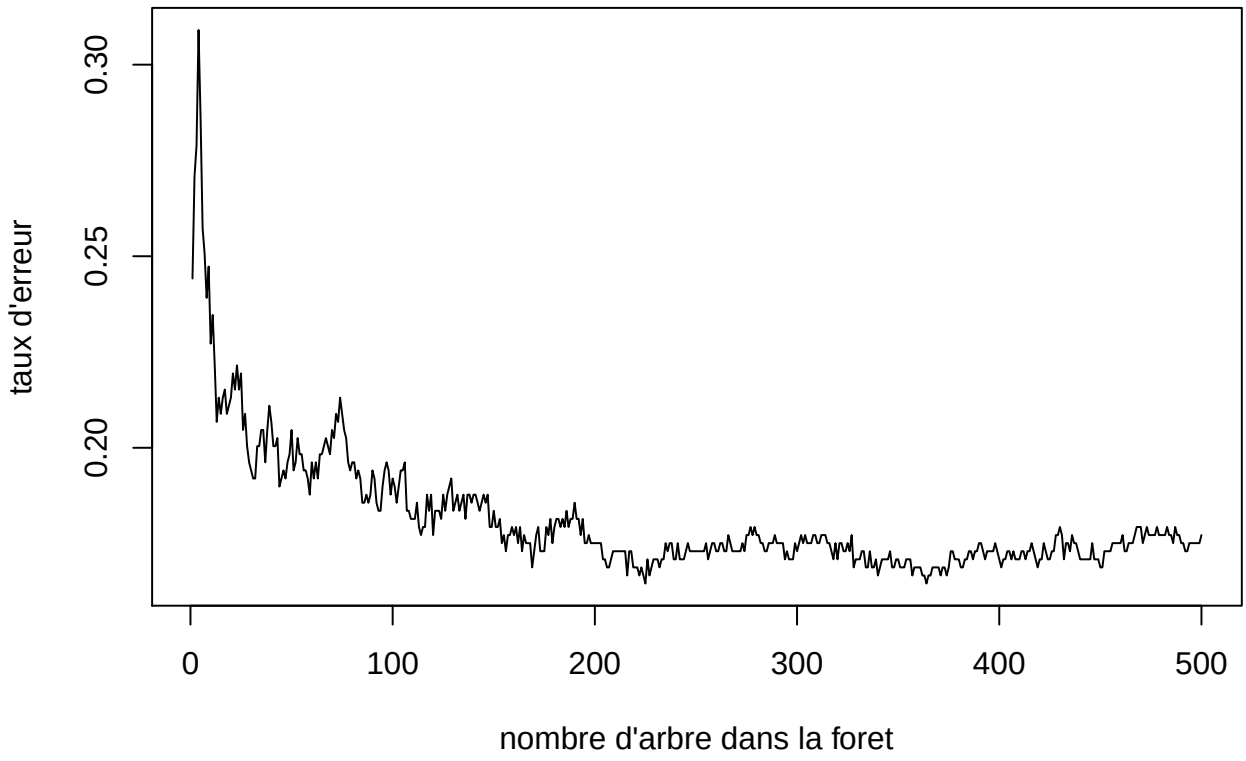
```
rf.fit
```

```
Call:
 randomForest(formula = cancer.type ~ ., data = mat.app)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 31

        OOB estimate of  error rate: 17.72%
Confusion matrix:
             Luminal.A no.Luminal.A class.error
Luminal.A          258           22  0.07857143
no.Luminal.A        62          132  0.31958763
```

2. Choix de la valeur du paramètre `ntree`.

- L'objet `rf.fit` créé est une liste qui contient différent élément. L'élement `err.rate` contient les erreurs commise par le modèle contenant m modèle. Représentez les erreurs pour l'ensemble des arbres

```
plot(rf.fit$err.rate[,1], type="l", xlab="nombre d'arbre dans la foret", ylab="taux d'erreur")
```



- A partir de ce graphique, choisissez le nombre d'arbres optimal, noté $ntree_{opt}$.

```
ntree.opt <- 200
```

3. Choix de la valeur du paramètre `mtry`.

Remarque : cette étape est coûteuse en temps de calculs.

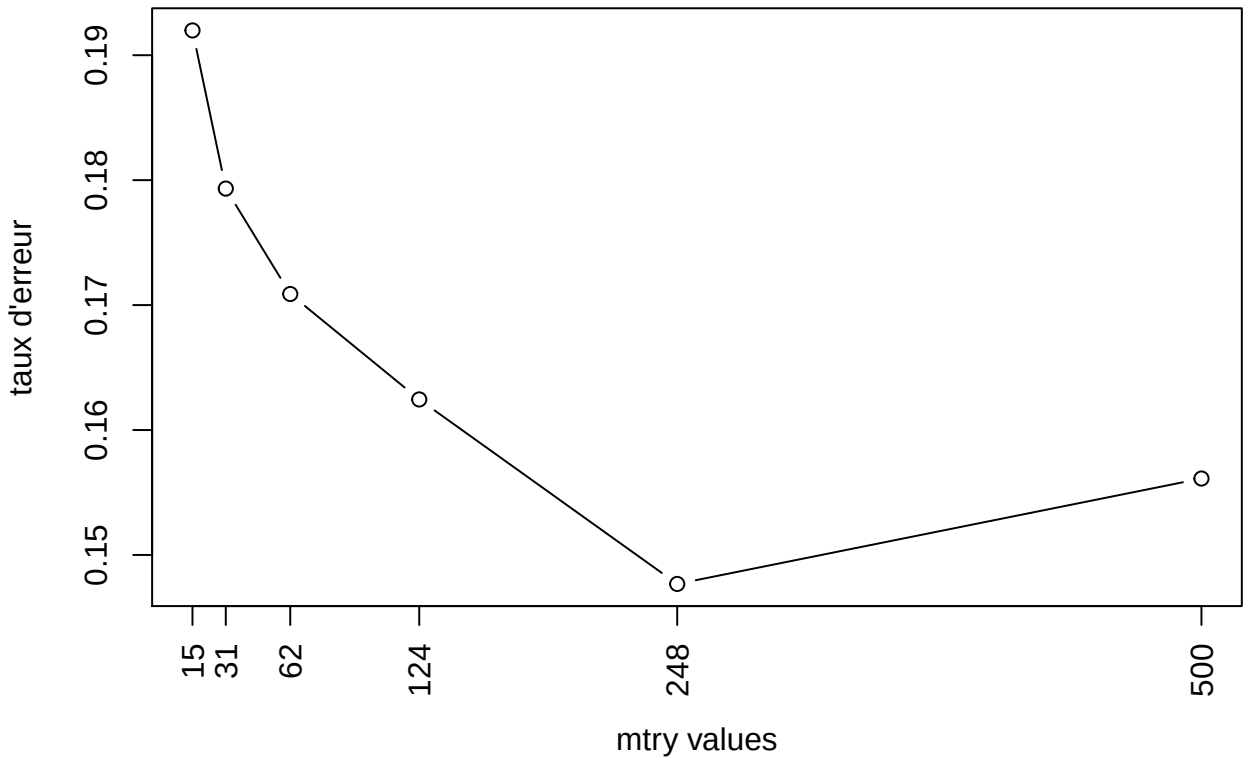
Pour trouver la valeur optimal de `mtry`, vous allez calculer plusieurs modèles en variant la valeur du paramètre `mtry`. Pour chaque modèle l'erreur commise par le modèle sera stockée. La valeur optimale de `mtry` correspondra à la valeur du `mtry` du modèle donnant le plus faible taux d'erreur.

- Calculez les modèles pour des valeurs de `mtry` de 15, 30, 62, 124, 248, 500, et la valeur de `ntree` optimale. Le taux d'erreur de chaque modèle sera stockée dans le vecteur `v.err`.

```
v.err <-  NULL
v.mtry <- c(15, 31, 62, 124, 248, 500)
for(i in v.mtry){
    set.seed(123)
    rf.fit <- randomForest(cancer.type~., data = mat.app, ntree=ntree.opt, mtry=i)
    v.err <- c(v.err, rf.fit$err.rate[ntree.opt,1])
}
```

- Représentez graphiquement les erreurs commises par les modèles en fonction de la valeur de `mtry`.

```
plot(v.mtry, v.err, type ="b", xlab="mtry values", ylab="taux d'erreur", xaxt="n")
axis(1, at=v.mtry, labels=v.mtry, las=2)
```



- A partir de ce graphique, déterminer la valeur optimal de `mtry`.

4. Calcul du modèle avec les paramètres optimaux.

- Calculer à nouveau le modèle sur l'échantillon d'apprentissage en utilisant les valeurs optimales des paramètres `mtry` et `ntree`. Dans la fonction `randomForest()`, ajouter l'argument `importance=TRUE`.

```
rf.fit <- randomForest(cancer.type~., data = mat.app, ntree=ntree.opt, mtry=248, importance=TRUE)
```

- Visualisez le modèle et la table de confusion obtenue sur les échantillons OOB.

```
rf.fit
```

```
Call:
 randomForest(formula = cancer.type ~ ., data = mat.app, ntree = ntree.opt,      mtry = 248, importance
               Type of random forest: classification
                     Number of trees: 200
No. of variables tried at each split: 248

        OOB estimate of  error rate: 14.77%
Confusion matrix:
             Luminal.A no.Luminal.A class.error
Luminal.A          262           18  0.06428571
no.Luminal.A        52          142  0.26804124
```

3. Calcul des performances du modèle *random forest*.

Pour comparer les performances de ce modèle, avec le modèle CART calculé précédemment, il faut calculer les performances du modèle `rf.fit` sur les données de l'échantillon d'apprentissage et de test.

- Calcul du taux de bien prédit sur l'échantillon d'apprentissage.

```
pred.app <- predict(rf.fit, newdata = mat.app, type="class")
```

- Calcul de la matrice de confusion entre les données prédites sur l'échantillon d'apprentissage et les vraies valeurs.

```
tc.rf.app <- table(pred.app, mat.app[,"cancer.type"])
tc.rf.app
```

```
pred.app       Luminal.A no.Luminal.A
  Luminal.A          280            0
  no.Luminal.A         0          194
```

- Calcul du taux de bien prédit ($TBP = \frac{VP+VN}{VP+VN+FP+FN}$) sur les données de l'apprentissage

```
TBP.rf.app <- sum(diag(tc.rf.app))/sum(tc.rf.app)
TBP.rf.app
```

```
[1] 1
```

6. Calcul du taux de bien prédit sur l'échantillon test

```
pred.test <- predict(rf.fit, newdata = mat.test, type="class")

tc.rf.test <- table(pred.test, mat.test[,"cancer.type"])
tc.rf.test
```

```
pred.test      Luminal.A no.Luminal.A
  Luminal.A          134           42
  no.Luminal.A         8           54
```

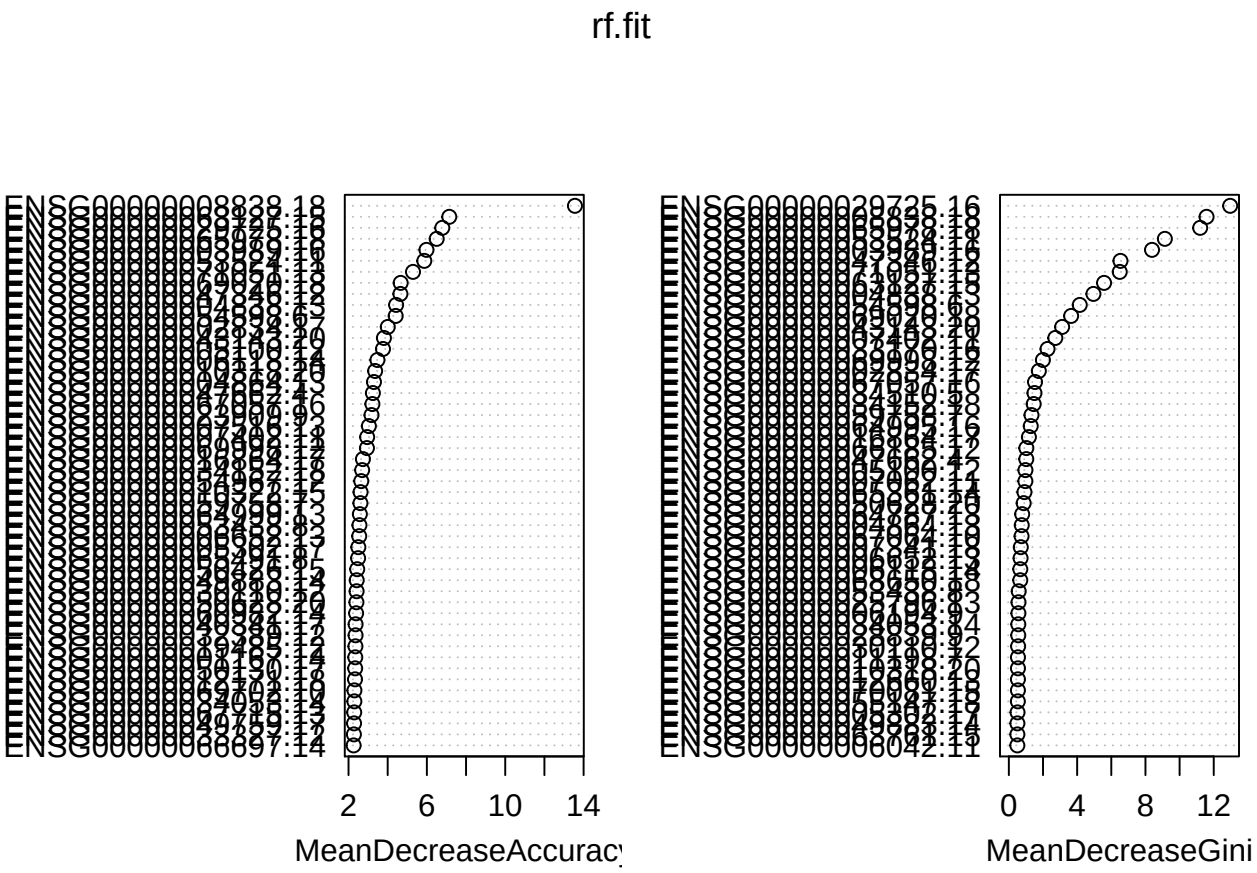
```
TBP.rf.test <- sum(diag(tc.rf.test))/sum(tc.rf.test)
TBP.rf.test
```

```
[1] 0.789916
```

7. Etude de l'importance des descripteurs dans le modèle.

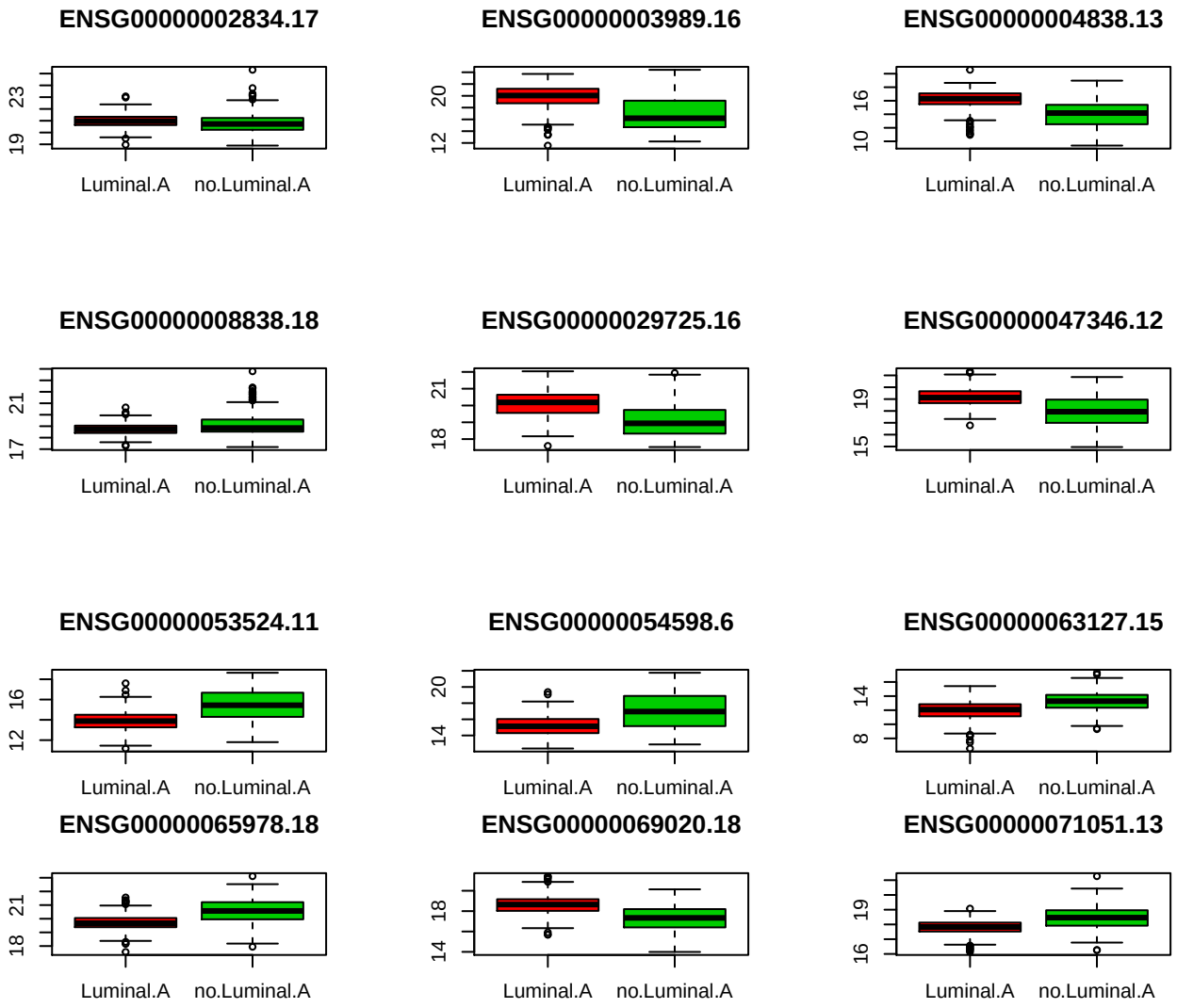
- Représentez graphiquement la valeur d'importance des différents gènes en utilisant la fonction `varImpPlot()`

```
varImpPlot(rf.fit, n.var=50)
```



- Identifier les gènes qui permettent au mieu de différentier le cancer Luminal.A des autres cancers.
- Pour chaque gene sélectionné, tracer la distribution de l'expression du gène chez les patients atteints d'un cancer de type Luminal.A et chez les autres patients d'après les données d'apprentissage.

```
v.imp <- importance(rf.fit)[,"MeanDecreaseAccuracy"]
gene.sel <- names(which(v.imp>4))

par(mfrow=c(3,3))
for(gene in gene.sel){
  boxplot(mat.app[,gene]~mat.app[,1],col=c(2,3), main=gene)
}
```



Prédiction du type de cancer en utilisant les Support Vecteur Machines

- utilisation avec formule R

```
library(e1071)
model=svm(cancer.type~., data=mat.app)
```

- utilisation classique

```
X=subset(mat.app, select = -cancer.type)
y=mat.app$cancer.type
model=svm(X,y)
```

- erreur d'apprentissage avec les paramètres par défaut

```
pred=predict(model, X)
table(pred, mat.app$cancer.type)
```

- surapprentissage avec gaussiennes très resserrées

```
model=svm(X,y,gamma=0.1)
pred=predict(model, X)
table(pred, mat.app$cancer.type)
```

- surapprentissage avec cost élevé

```
model=svm(X,y,cost=100)
pred=predict(model, X)
table(pred, mat.app$cancer.type)
```

- évaluation d'erreur de test (en surapprentissage)

```
Xtest=subset(mat.test, select = -cancer.type)
pred=predict(model, Xtest)
table(pred, mat.test$cancer.type)
```

- évaluation d'erreur de test (avec paramètres par défaut)

```
model=svm(X,y)
pred=predict(model, Xtest)
table(pred, mat.test$cancer.type)
```