TD2 : Sélection par tests multiples

Exercice 1 : Tests Multiples sur un jeu de données simulées

Description du jeu de données

```
load("/Users/bouacha_lazhar/OneDrive/Master MMA/M2 MMA/M2 S3/Apprentissage en Grande Dimension/Partie I
data <- genes$data
condition <- genes$condition
statut <- genes$statut
```

1.

On peut faire un test de comparaison de moyenne avec échantillons appariés :

```
t.test(data[condition == 1, 1], data[condition == 2, 1])
```

```
##
##  Welch Two Sample t-test
##
## data:  data[condition == 1, 1] and data[condition == 2, 1]
## t = 0.029982, df = 57.999, p-value = 0.9762
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -56.59586  58.31703
## sample estimates:
## mean of x mean of y
##  508.2416  507.3810
```

La p-valeur $> 5\%$, le test est donc non significatif. Le gène 1 est non différentiellement exprimé.

2.

```
pval <- apply(data, 2, function(x) t.test(x[condition == 1], x[condition == 2])$p.value)
length(which(pval < 0.05)) # ceux qui sont détéctés positifs
```

```
## [1] 334
```

Sur les 5000 gènes, 334 sont identifiés comme différentiellement exprimés. En tout (voir code suivant), 96 gènes sont réellement différentiellement exprimés.

```
length(which(statut != 0)) # ceux qui sont réellement positifs
```

```
## [1] 96
```

Pour connaître le nombre de faux positifs, on compare la liste de gènes identifiés comme différentiellement exprimés avec ceux qui le sont réellement :

```
FP <- length(intersect(which(statut == 0), which(pval < 0.05)))
FP
```

```
## [1] 241
```

Le nombre de Faux Positifs est très élevé mais cela n'est pas surprenant puisque environ 5% des gènes non différentiellement exprimés sont classés à tort.Cela représente environ $(5000 - 100) * 0.05 = 245$ gènes, où 100 : Vrais Positifs

3.

```
length(which(pval < 0.05/5000))
```

```
## [1] 20
```

```
statut[which(pval < 0.05/5000)]
```

```
##   Genes86  Genes289  Genes408  Genes853  Genes978  Genes985 Genes1152 Genes1592
##         1        -1        -1         1        -1         1        -1        -1
## Genes1968 Genes2000 Genes2222 Genes2685 Genes3009 Genes3121 Genes3156 Genes3997
##         1        -1        -1        -1         1         1         1        -1
## Genes4133 Genes4134 Genes4335 Genes4855
##        -1        -1         1        -1
```

```
FP <- length(intersect(which(statut == 0), which(pval < 0.05/5000)))
FP
```

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

Si on applique la procédure de Bonferroni, on ne garde que 20 gènes (procédure très conservative). Il n'y a aucun faux positif.

4.

```
length(which(pval < (1-(1-0.05)^(1/5000))))
```

```
## [1] 20
```

```
statut[which(pval < (1-(1-0.05)^(1/5000)))]
```

```
##   Genes86  Genes289  Genes408  Genes853  Genes978  Genes985 Genes1152 Genes1592
##         1        -1        -1         1        -1         1        -1        -1
## Genes1968 Genes2000 Genes2222 Genes2685 Genes3009 Genes3121 Genes3156 Genes3997
##         1        -1        -1        -1         1         1         1        -1
## Genes4133 Genes4134 Genes4335 Genes4855
##        -1        -1         1        -1
```

```
FP <- length(intersect(which(statut == 0), which(pval < (1-(1-0.05)^(1/5000)))))
FP
```

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

Si on applique la procédure de Sidàk, on ne garde que 20 gènes identique à Bonferroni. Il n'y a aucun faux positif.

5.

```
holmbonferroni <- function(pval, alpha){ # création d'une fonction holm-bonferroni
  m <- length(pval)
  # 1. ordonne les p-valeurs
  sortedpval <- sort(pval)
  # 2. détermine
  x <- which(sortedpval <= alpha/(m+1-c(1:m)))
```

```
  I <- max(x) # rang à partir duquel il faut couper la liste qu'on renvoie
  # 3. extraire
  selected = c()
  if (I > 1){
    selected = names(sortedpval)[1:I]
  }
  return(selected)
}

holmbonferroni(pval, 0.05)
```

```
##  [1] "Genes289"  "Genes985"  "Genes2000" "Genes86"   "Genes4134" "Genes3121"
##  [7] "Genes1152" "Genes1968" "Genes4133" "Genes4855" "Genes408"  "Genes2222"
## [13] "Genes3997" "Genes853"  "Genes2685" "Genes4335" "Genes978"  "Genes1592"
## [19] "Genes3156" "Genes3009"
```

```
length(holmbonferroni(pval, 0.05))
```

```
## [1] 20
```

Les résultats sont une nouvelle fois les mêmes.

6.

```
benjaminihochberg <- function(pval, alpha){ # création d'une fonction Benjamini-Hochberg
  m <- length(pval)
  # 1. ordonne les p-valeurs
  sortedpval <- sort(pval)
  # 2. détermine
  x <- which(sortedpval <= alpha*(c(1:m)/m))
  I <- max(x) # rang à partir duquel il faut couper la liste qu'on renvoie
  # 3. extraire
  selected = c()
  if (I > 0){
    selected = names(sortedpval)[1:I]
  }
  return(selected)
}

benjaminihochberg(pval, 0.05)
```

```
##  [1] "Genes289"  "Genes985"  "Genes2000" "Genes86"   "Genes4134" "Genes3121"
##  [7] "Genes1152" "Genes1968" "Genes4133" "Genes4855" "Genes408"  "Genes2222"
## [13] "Genes3997" "Genes853"  "Genes2685" "Genes4335" "Genes978"  "Genes1592"
## [19] "Genes3156" "Genes3009" "Genes2440" "Genes3007" "Genes4703" "Genes1684"
## [25] "Genes2250" "Genes3489" "Genes1445" "Genes4084" "Genes4596" "Genes4885"
## [31] "Genes3331" "Genes1641" "Genes1351" "Genes3450" "Genes4940" "Genes3424"
## [37] "Genes3841" "Genes202"  "Genes4891" "Genes1304" "Genes3849" "Genes4645"
## [43] "Genes3281" "Genes171"  "Genes1860" "Genes218"  "Genes3762" "Genes4122"
## [49] "Genes43"   "Genes2539" "Genes4663" "Genes1270" "Genes465"  "Genes4722"
## [55] "Genes1612" "Genes4490" "Genes4259" "Genes1245"
```

```
length(benjaminihochberg(pval, 0.05))
```

```
## [1] 58
```

```
FP <- length(intersect(names(which(statut == 0)), benjaminihochberg(pval,.05)))
Positifs <- length(benjaminihochberg(pval,.05))
FDP <- FP/Positifs
FDP
```

```
## [1] 0.05172414
```

La procédure de Benjamini-Hochberg est moins conservative. On sélectionne plus de gènes, quitte à commettre un peu plus d'erreurs (3 faux positifs ici). Le taux de faux positifs (FDP) est de l'ordre de 5%.

7.

```
padj <- p.adjust(pval, method = "BH")
which(padj < 0.05) # coincide bien
```

```
##   Genes43   Genes86  Genes171  Genes202  Genes218  Genes289  Genes408  Genes465
##        43        86       171       202       218       289       408       465
##  Genes853  Genes978  Genes985 Genes1152 Genes1245 Genes1270 Genes1304 Genes1351
##       853       978       985      1152      1245      1270      1304      1351
## Genes1445 Genes1592 Genes1612 Genes1641 Genes1684 Genes1860 Genes1968 Genes2000
##      1445      1592      1612      1641      1684      1860      1968      2000
## Genes2222 Genes2250 Genes2440 Genes2539 Genes2685 Genes3007 Genes3009 Genes3121
##      2222      2250      2440      2539      2685      3007      3009      3121
## Genes3156 Genes3281 Genes3331 Genes3424 Genes3450 Genes3489 Genes3762 Genes3841
##      3156      3281      3331      3424      3450      3489      3762      3841
## Genes3849 Genes3997 Genes4084 Genes4122 Genes4133 Genes4134 Genes4259 Genes4335
##      3849      3997      4084      4122      4133      4134      4259      4335
## Genes4490 Genes4596 Genes4645 Genes4663 Genes4703 Genes4722 Genes4855 Genes4885
##      4490      4596      4645      4663      4703      4722      4855      4885
## Genes4891 Genes4940
##      4891      4940
```

```
length(which(padj < 0.05))
```

```
## [1] 58
```

On peut essayer avec hochberg, bonferroni :

```
adj <- p.adjust(pval, method = "hochberg")
which(padj < 0.05) # coincide bien
```

```
##   Genes43   Genes86  Genes171  Genes202  Genes218  Genes289  Genes408  Genes465
##        43        86       171       202       218       289       408       465
##  Genes853  Genes978  Genes985 Genes1152 Genes1245 Genes1270 Genes1304 Genes1351
##       853       978       985      1152      1245      1270      1304      1351
## Genes1445 Genes1592 Genes1612 Genes1641 Genes1684 Genes1860 Genes1968 Genes2000
##      1445      1592      1612      1641      1684      1860      1968      2000
## Genes2222 Genes2250 Genes2440 Genes2539 Genes2685 Genes3007 Genes3009 Genes3121
##      2222      2250      2440      2539      2685      3007      3009      3121
## Genes3156 Genes3281 Genes3331 Genes3424 Genes3450 Genes3489 Genes3762 Genes3841
##      3156      3281      3331      3424      3450      3489      3762      3841
## Genes3849 Genes3997 Genes4084 Genes4122 Genes4133 Genes4134 Genes4259 Genes4335
##      3849      3997      4084      4122      4133      4134      4259      4335
## Genes4490 Genes4596 Genes4645 Genes4663 Genes4703 Genes4722 Genes4855 Genes4885
##      4490      4596      4645      4663      4703      4722      4855      4885
## Genes4891 Genes4940
```

```
##      4891      4940
```

```
length(which(padj < 0.05))
```

```
## [1] 58
```

```
adj <- p.adjust(pval, method = "bonferroni")
which(padj < 0.05) # coincide bien
```

```
##   Genes43   Genes86  Genes171  Genes202  Genes218  Genes289  Genes408  Genes465
##        43        86       171       202       218       289       408       465
##  Genes853  Genes978  Genes985 Genes1152 Genes1245 Genes1270 Genes1304 Genes1351
##       853       978       985      1152      1245      1270      1304      1351
## Genes1445 Genes1592 Genes1612 Genes1641 Genes1684 Genes1860 Genes1968 Genes2000
##      1445      1592      1612      1641      1684      1860      1968      2000
## Genes2222 Genes2250 Genes2440 Genes2539 Genes2685 Genes3007 Genes3009 Genes3121
##      2222      2250      2440      2539      2685      3007      3009      3121
## Genes3156 Genes3281 Genes3331 Genes3424 Genes3450 Genes3489 Genes3762 Genes3841
##      3156      3281      3331      3424      3450      3489      3762      3841
## Genes3849 Genes3997 Genes4084 Genes4122 Genes4133 Genes4134 Genes4259 Genes4335
##      3849      3997      4084      4122      4133      4134      4259      4335
## Genes4490 Genes4596 Genes4645 Genes4663 Genes4703 Genes4722 Genes4855 Genes4885
##      4490      4596      4645      4663      4703      4722      4855      4885
## Genes4891 Genes4940
##      4891      4940
```

```
length(which(padj < 0.05))
```

```
## [1] 58
```

Exercice 2 : Tests Multiples sur un jeu de données réelles

Chargement et description du jeu de données

```
#BiocManager::install("golubEsets")
library(golubEsets)
```

```
## Loading required package: Biobase
```

```
## Loading required package: BiocGenerics
```

```
## Warning: package 'BiocGenerics' was built under R version 4.0.5
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
```

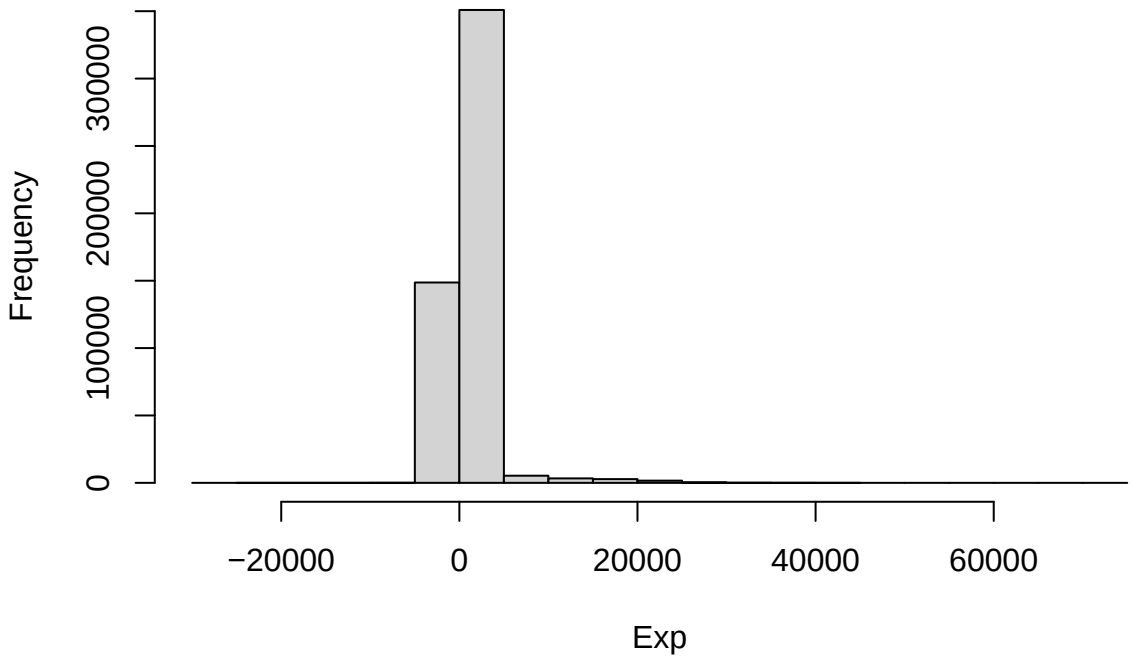
```
##
##     anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##     dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which.max, which.min

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
data("Golub_Merge")
load("/Users/bouacha_lazhar/OneDrive/Master MMA/M2 MMA/M2 S3/Apprentissage en Grande Dimension/Partie I
Exp <- exprs(Golub_Merge)
dim(Exp)
```

```
## [1] 7129   72
```

1.

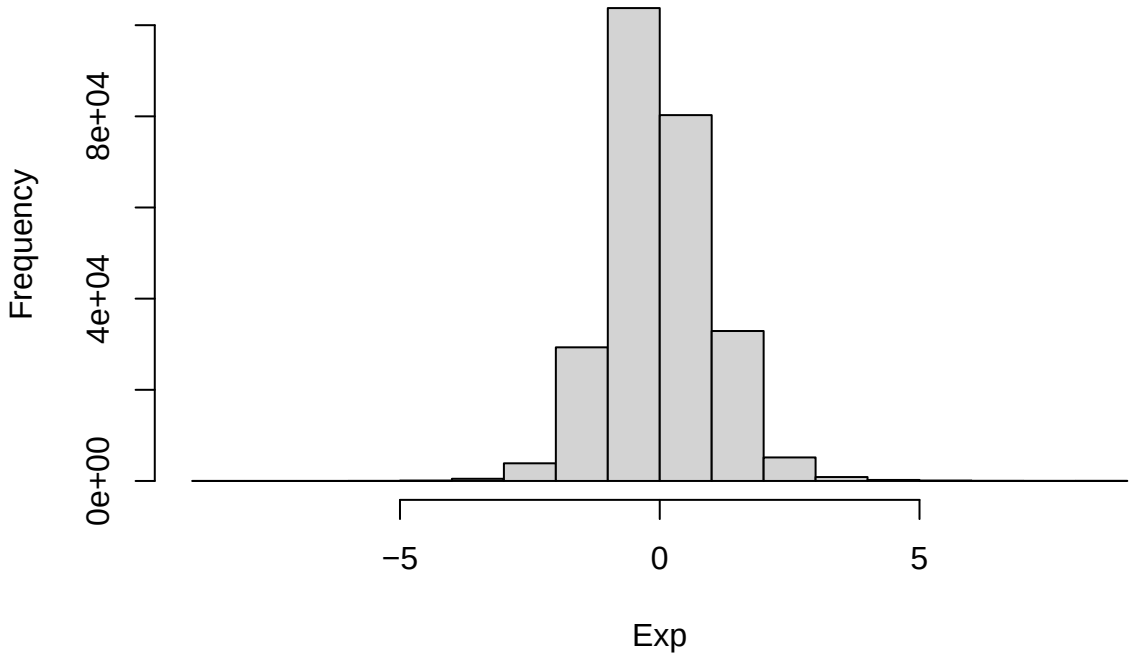
```
hist(Exp, main="")
```




```
Exp[Exp < 100] <- 100
Exp[Exp > 16000] <- 16000

emax <- apply(Exp, 1, max)
emin <- apply(Exp, 1, min)
R <- which(emax/emin > 5)
S <- which(emax - emin > 500)
Exp <- Exp[intersect(R, S),]
dim(Exp)
```

```
## [1] 3571   72
```

```
Exp <- log(Exp,10)
Exp <- scale(t(Exp), center=TRUE, scale=TRUE) # ne pas oublier de transposer la matrice

hist(Exp, main="")
```



2.

```
load("/Users/bouacha_lazhar/OneDrive/Master MMA/M2 MMA/M2 S3/Apprentissage en Grande Dimension/Partie I
ALL <- which(Golub_Merge$ALL.AML == "ALL")
AML <- which(Golub_Merge$ALL.AML == "AML")

pval <- apply(Exp, 2, function(x){
  t.test(x[ALL], x[AML])$p.value
})

pvalBH <- p.adjust(pval,method="BH")
length(which(pvalBH<0.01))
```

```
## [1] 571
```

Sur les 3571 gènes, 571 sont identifiés comme différentiellement exprimés avec un contrôle de la FDR de 1%.

```
DEgenes <- Exp[,which(pvalBH<0.01)]
dim(DEgenes)
```

```
## [1]  72 571
```

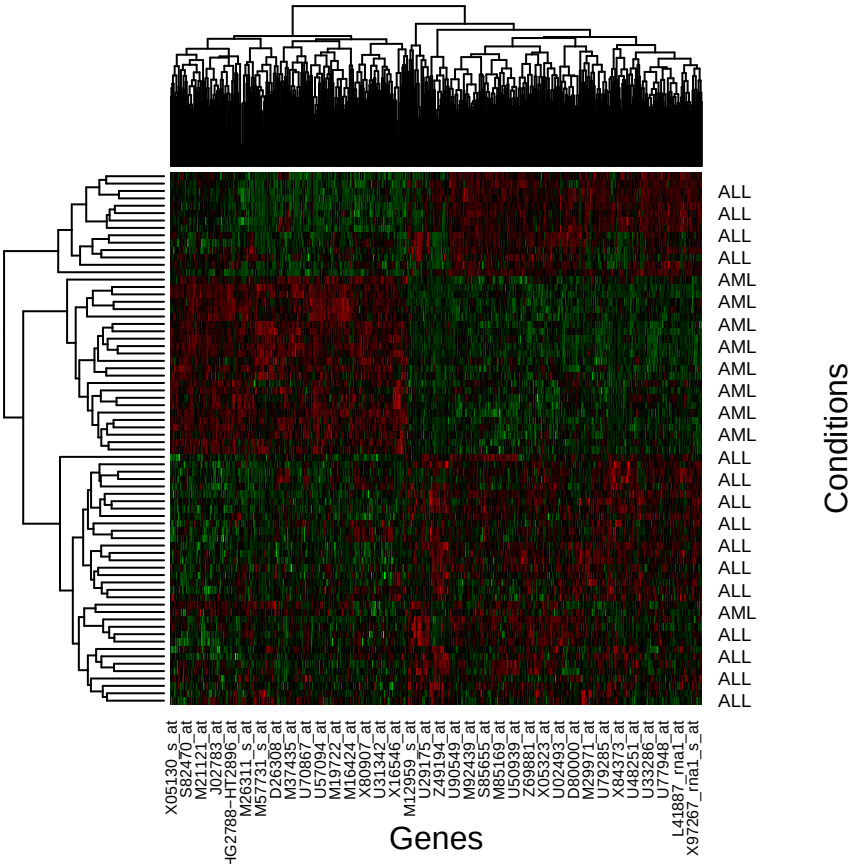
3.

```
library(gplots)
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
rownames(DEgenes) <- Golub_Merge$ALL.AML
heatmap(DEgenes, col=greenred(75), ylab="Conditions", xlab="Genes")
```



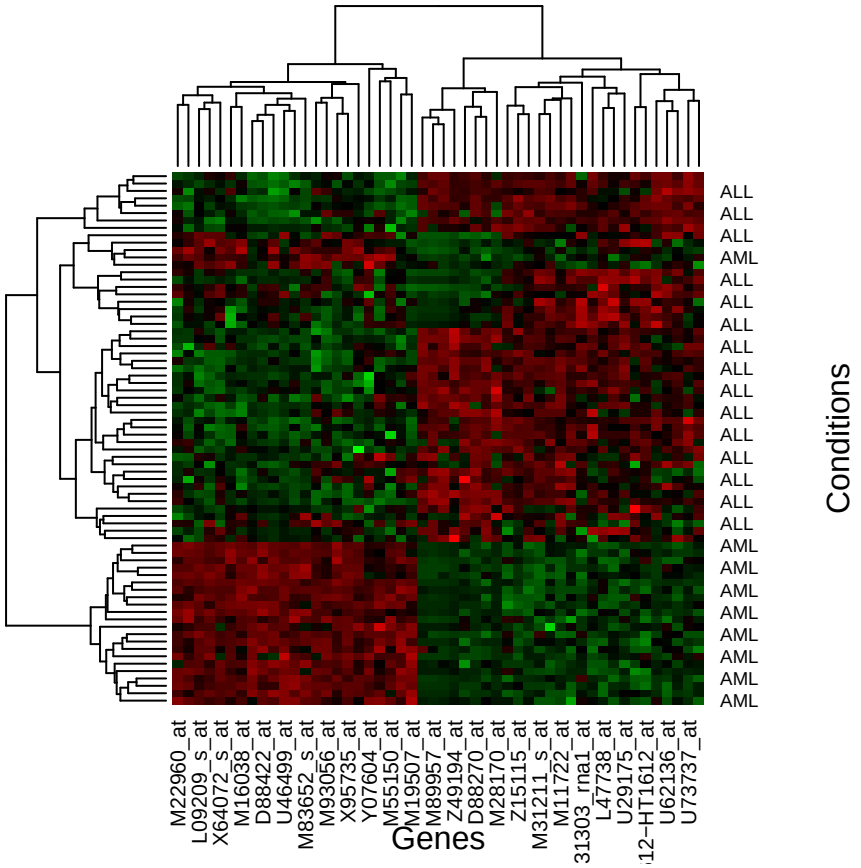
Des groupes de gènes apparaissent distinctement et pourraient permettre de différencier les deux types de leucémie.

4.

```
bestgenes <- order(pvalBH)[1:50]
DEbest <- Exp[,bestgenes]
dim(DEbest)
```

```
## [1] 72 50
```

```
rownames(DEbest) <- Golub_Merge$ALL.AML
heatmap(DEbest, col=greenred(75), ylab="Conditions", xlab="Genes")
```




```
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:Biobase':
##
##     combine
```

```
## The following object is masked from 'package:BiocGenerics':
##
##     combine
```

```
I <- rbinom(nrow(DEbest),1,.8) # pour créer les training et test sets
Train <- which(I==1)
Test <- which(I==0)
rf <- randomForest(x = DEbest[Train,],y=Golub_Merge$ALL.AML[Train],
                   xtest=DEbest[Test,], ytest = Golub_Merge$ALL.AML[Test])
rf
```

```
##
## Call:
##  randomForest(x = DEbest[Train, ], y = Golub_Merge$ALL.AML[Train],      xtest = DEbest[Test, ], ytes
##                Type of random forest: classification
##                      Number of trees: 500
```

```
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##     ALL AML class.error
## ALL  32   0           0
## AML   0  15           0
##                 Test set error rate: 12%
## Confusion matrix:
##     ALL AML class.error
## ALL  15   0         0.0
## AML   3   7         0.3
```

La méthode de classification fonctionne très bien, aucune erreur n'est commise.