Statistics and Classification on Genomic Data: Elements of correction

julien.chiquet@gmail.com

Bioinformatics Summer School - Angers, 2016

This practical aims to provide an overview of a series of statistical methods now routinely used to process genomic data. The statistical tasks at play will concern both supervised and unsupervised classification problems. The classical Golub data set is used to illustrate these methods.

---

1 Preliminaries

1.1 The Golub (Leukemia) data set

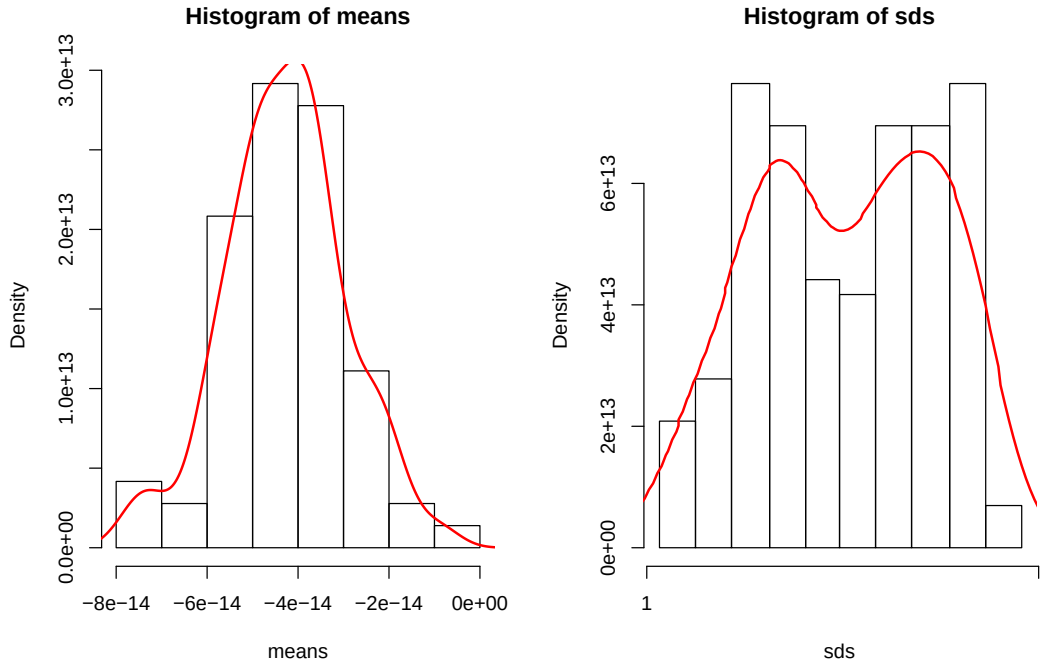
1.2 Loading the data

```
load("data/leukemia.RData")
X <- as.matrix(t(expr))
Y <- as.vector(pheno)
Gene <- colnames(X)
n <- nrow(X) ; p <- ncol(X)
```

1.3 Descriptive statistics

- Plot a histogram / density plot of the mean expression and of the standard deviation of the expression level of each individual. Comment.

```
par(mfrow=c(1,2))
means <- apply(X, 1, mean); sds <- apply(X, 1, sd)
hist(means, breaks=sqrt(n), probability = TRUE)
lines(density(means), col="red" ,lwd=2)
hist(sds, breaks=sqrt(n), probability = TRUE)
lines(density(sds), col="red", lwd=2)
```



The mean expression seem normaly distributed. There is some heterogeneity in the variance.

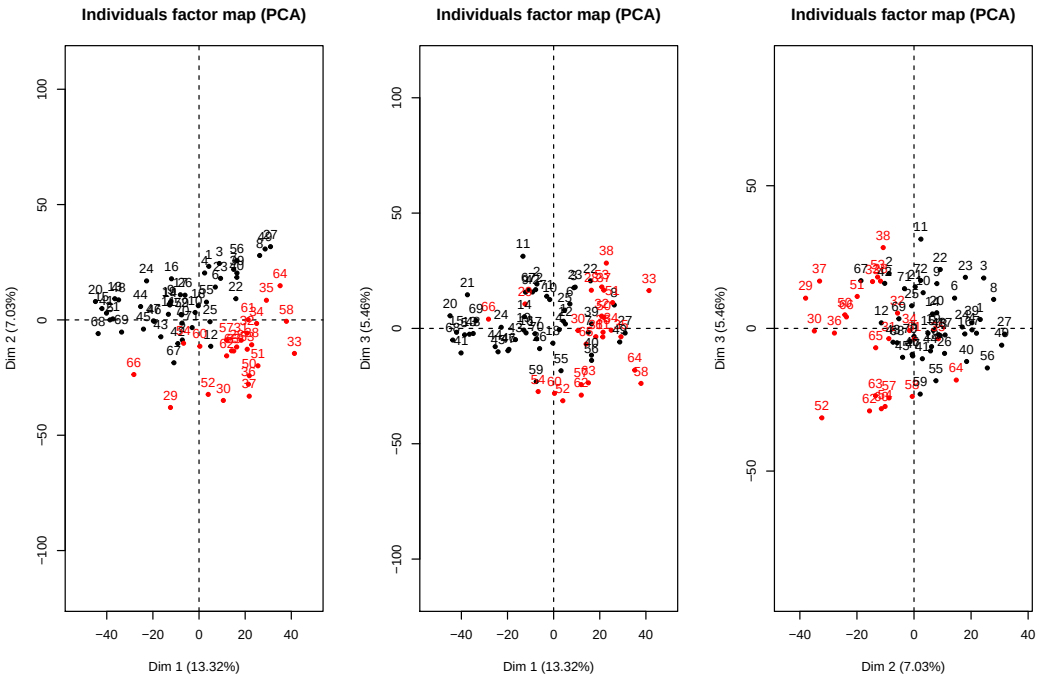
We could also have a look at the qqplot of the gene expression.

```
qqnorm(X); qqline(X)
```

This is not exactly Gaussian, but anyway the matrix of predictor is not suppposed to be in the logistic regression model.

- Do a PCA analysis of the matrix of predictors. Plot the individual factor map for axes (1,2), (1,3), (2,3). Color the individual according to their status (AML/ALL). Comment

```
library(FactoMineR)
pca.out <- PCA(X, graph=FALSE)
par(mfrow=c(1,3))
plot(pca.out, axes = c(1,2), col.ind = 1+Y)
plot(pca.out, axes = c(1,3), col.ind = 1+Y)
plot(pca.out, axes = c(2,3), col.ind = 1+Y)
```



The problem of discriminating the two group seem easy. However, the PCA does that job with a linear combinaison of all the genes : we would like to do create a signature of only a few genes that would do the same job.

1.4 Variable Screening by differential analysis

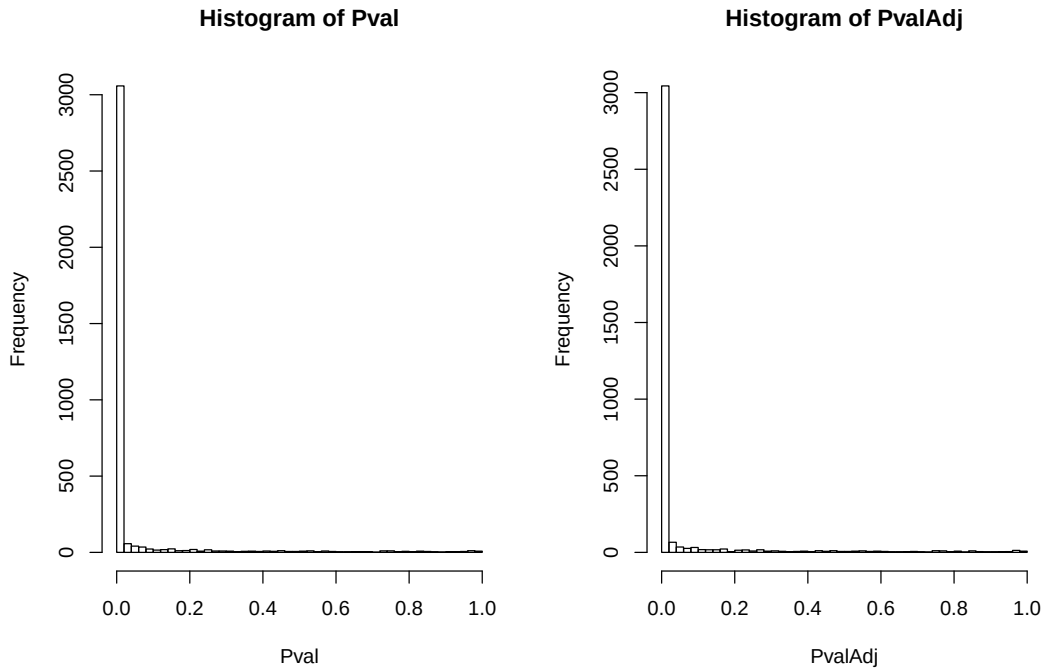
- For each gene, test the difference in the mean between the two groups (ALL/AML), with a t-test or a Wilcoxon test. You can also simply rank the genes by decreasing variance.

This could be doneas followS.

```
Pval <- sapply(as.data.frame(X), function(x) t.test(x,Y)$p.value)
```

- Plot the histogram of the p-values before and after multiple testing correction (Benjamini-Hochberg, Bonferroni: function `p.adjust`).

```
PvalAdj <- p.adjust(Pval, method="BH")
par(mfrow=c(1,2))
hist(Pval, breaks=sqrt(p))
hist(PvalAdj, breaks=sqrt(p))
```




```
length(Gene[which(PvalAdj < .001)])
```

```
## [1] 2816
```

- Only retain the genes with an adjusted p-value smaller than 0.1%.

Anyway, I keep all the genes since ma computer is allright.

If your computer cannot stand it, only retains the first thousand of genes.

2 Supervised classification of the samples

In this part, our goal is to select a small set of genes that do a reasonably good job for discriminating the AML/ALL samples. To this end, we rely on sparse logistic regression.

2.1 Main Analyses

- Why usual logistic regression would not run? (try it with the function `glm`).

We are in a high-dimensional setup: X is not of full rank, the system is underdetermined. The function would not fit.

- Randomly split your data in a training set and a test set. Use 2/3 of the sample for the training set.

```
set.seed(1)
Train  <- sort(sample(1:n, round(2*n/3))); ntrain = length(Train)
Test   <- sort(setdiff(1:n, Train)); ntest = length(Test)
Xtrain <-  X[Train, ]; Ytrain = Y[Train]
Xtest  <-  X[Test, ]; Ytest = Y[Test]
```

- Adjust a logistic regression model regularized by a Lasso penalty ($\alpha = 1$)

```
suppressMessages(library(glmnet))
lasso <- glmnet(Xtrain, Ytrain, family="binomial", alpha=1)
# alpha = 0 -> Ridge
ridge <- glmnet(Xtrain, Ytrain, family="binomial", alpha=0)
```

- Adjust a logistic regression model regularized by an Elastic-net penalty. Use various values, say $\{.25, .5, .75, .95\}$ for the mixture parameter α.

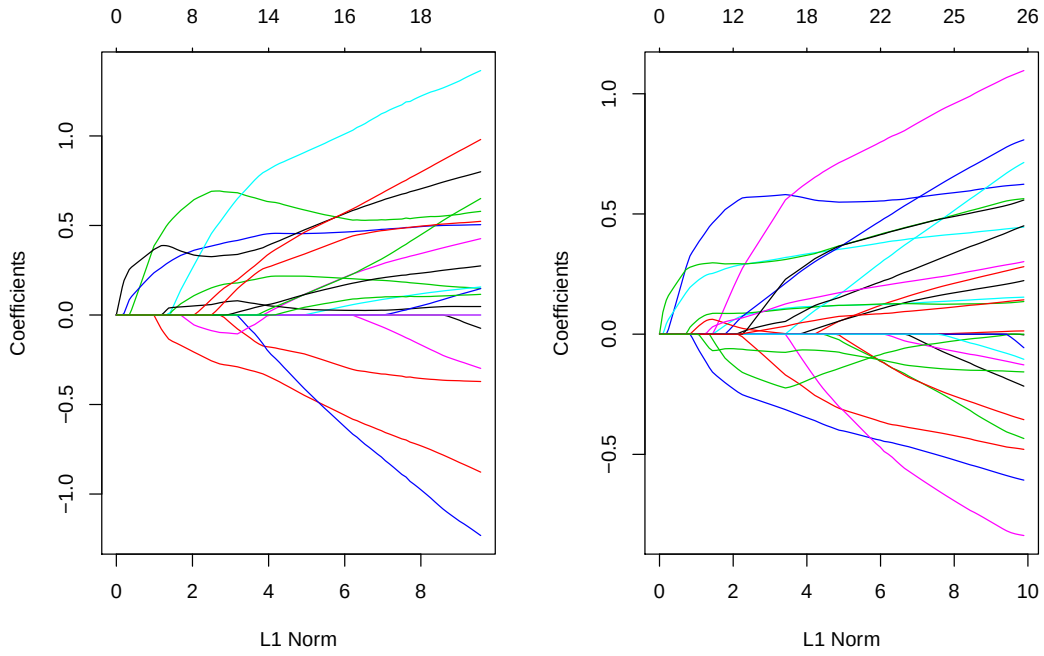
```
enet1 <- glmnet(Xtrain, Ytrain, family="binomial", alpha=.25)
enet2 <- glmnet(Xtrain, Ytrain, family="binomial", alpha=.5)
enet3 <- glmnet(Xtrain, Ytrain, family="binomial", alpha=.75)
enet4 <- glmnet(Xtrain, Ytrain, family="binomial", alpha=.9)
```

- For each model (i.e. each value of α), choose the final estimator (that is, the final λ by cross-validation of the classification error.

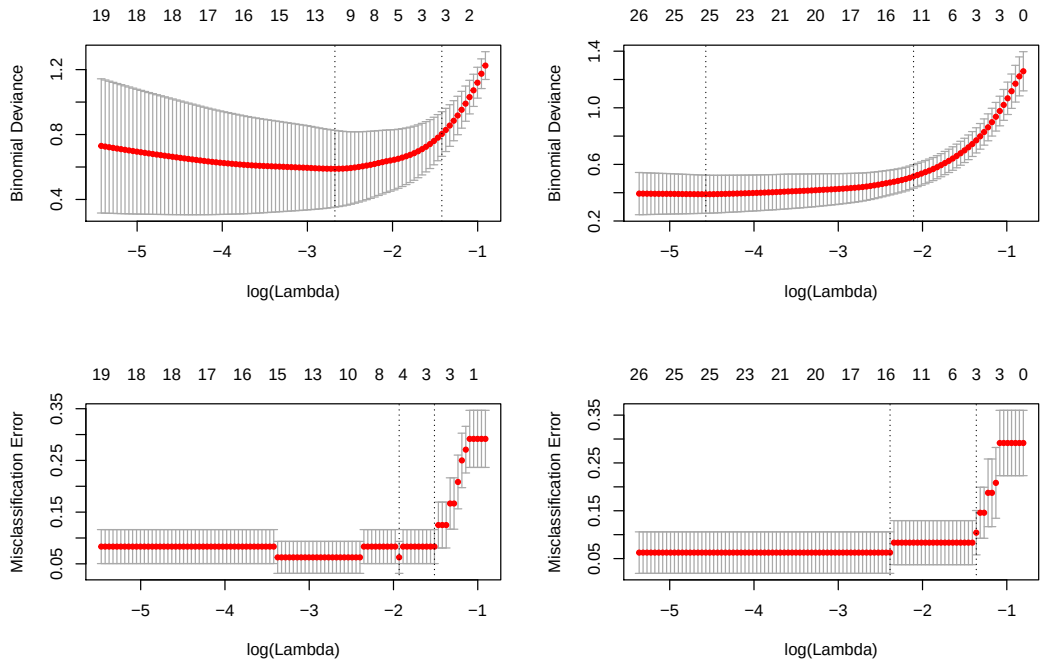
I also cross-validate the deviance (calls are only shown for the LASSO, similar for Elastic-Net)

```
cv.lasso.dev <- cv.glmnet(Xtrain, Ytrain, family="binomial", alpha=1)
cv.lasso.cla <- cv.glmnet(Xtrain, Ytrain, family="binomial",
                          type.measure="class", alpha=1)
```

```
par(mfrow=c(1,2))
plot(lasso); plot(enet4);
```




```
par(mfrow=c(2,2))
plot(cv.lasso.dev); plot(cv.enet4.dev)
plot(cv.lasso.cla); plot(cv.enet4.cla)
```



- Compute the classification error on the test set

I first extract the prediction for each methods + the selected features (I only show the code for the LASSO, this goes the same way for the other ones)

```
yhat.lasso.dev <- as.numeric(predict(cv.lasso.dev$glmnet.fit, Xtest,
                                     s=cv.lasso.dev$lambda.min, type="class"))
yhat.lasso.cla <- as.numeric(predict(cv.lasso.cla$glmnet.fit,
                                     Xtest, s=cv.lasso.cla$lambda.min, type="class"))

sel.lasso.dev <- predict(cv.lasso.dev$glmnet.fit, Xtest,
                         s=cv.lasso.dev$lambda.min, type="nonzero")
sel.lasso.cla <- predict(cv.lasso.cla$glmnet.fit, Xtest,
                         s=cv.lasso.cla$lambda.min, type="nonzero")
```

Here are the errors of classification :

```
error.class <- setNames(c(sum(abs(Ytest - yhat.lasso.dev)),
                          sum(abs(Ytest - yhat.lasso.cla)),
                          sum(abs(Ytest - yhat.enet1.dev)),
                          sum(abs(Ytest - yhat.enet1.cla)),
                          sum(abs(Ytest - yhat.enet2.dev)),
                          sum(abs(Ytest - yhat.enet2.cla)),
                          sum(abs(Ytest - yhat.enet3.dev)),
                          sum(abs(Ytest - yhat.enet3.cla)),
                          sum(abs(Ytest - yhat.enet4.dev)),
                          sum(abs(Ytest - yhat.enet4.cla))),
                        c("lasso (deviance)", "lasso (classif)",
                          "enet4 (deviance)", "enet4 (classif)",
                          "enet3 (deviance)", "enet3 (classif)",
                          "enet2 (deviance)", "enet2 (classif)",
                          "enet1 (deviance)", "enet1 (classif)"))
print(error.class)
```

```
## lasso (deviance)  lasso (classif) enet4 (deviance)  enet4 (classif)
##                1                3                0                2
## enet3 (deviance)  enet3 (classif) enet2 (deviance)  enet2 (classif)
##                0                2                0                2
## enet1 (deviance)  enet1 (classif)
##                0                1
```

Here is the adjusted Rand Index

```
library(mclust)
```

```
## Package 'mclust' version 5.2
```

```
## Type 'citation("mclust")' for citing this R package in publications.
```

```
error.adjRI <- setNames(c(adjustedRandIndex(Ytest, yhat.lasso.dev),
                          adjustedRandIndex(Ytest, yhat.lasso.cla),
                          adjustedRandIndex(Ytest, yhat.enet1.dev),
                          adjustedRandIndex(Ytest, yhat.enet1.cla),
                          adjustedRandIndex(Ytest, yhat.enet2.dev),
                          adjustedRandIndex(Ytest, yhat.enet2.cla),
                          adjustedRandIndex(Ytest, yhat.enet3.dev),
                          adjustedRandIndex(Ytest, yhat.enet3.cla),
                          adjustedRandIndex(Ytest, yhat.enet4.dev),
                          adjustedRandIndex(Ytest, yhat.enet4.cla)),
                        c("lasso (deviance)", "lasso (classif)",
                          "enet4 (deviance)", "enet4 (classif)",
                          "enet3 (deviance)", "enet3 (classif)",
                          "enet2 (deviance)", "enet2 (classif)",
                          "enet1 (deviance)", "enet1 (classif)"))
print(error.adjRI)
```

```
## lasso (deviance)  lasso (classif) enet4 (deviance)  enet4 (classif)
##        0.8332458        0.5446737        1.0000000        0.6814104
## enet3 (deviance)  enet3 (classif) enet2 (deviance)  enet2 (classif)
##        1.0000000        0.6814104        1.0000000        0.6814104
## enet1 (deviance)  enet1 (classif)
##        1.0000000        0.8332458
```

And finally the number of selected features

```
sparsity <- setNames(c(nrow(sel.lasso.dev),nrow(sel.lasso.cla),
                       nrow(sel.enet4.dev),nrow(sel.enet4.cla),
                       nrow(sel.enet3.dev),nrow(sel.enet3.cla),
                       nrow(sel.enet2.dev),nrow(sel.enet2.cla),
                       nrow(sel.enet1.dev),nrow(sel.enet1.cla)),
                     c("lasso (deviance)", "lasso (classif)",
                       "enet4 (deviance)", "enet4 (classif)",
                       "enet3 (deviance)", "enet3 (classif)",
                       "enet2 (deviance)", "enet2 (classif)",
                       "enet1 (deviance)", "enet1 (classif)"))
print(sparsity)
```

```
## lasso (deviance)  lasso (classif) enet4 (deviance)  enet4 (classif)
##               12                5               25               16
## enet3 (deviance)  enet3 (classif) enet2 (deviance)  enet2 (classif)
##               33               16               64               21
## enet1 (deviance)  enet1 (classif)
##              144               40
```

- Compare the genes selected by the different methods.

Some examples :

```
gene.lasso.cla <- Gene[sel.lasso.cla[[1]]]
gene.enet4.dev <- Gene[sel.enet4.dev[[1]]]
```

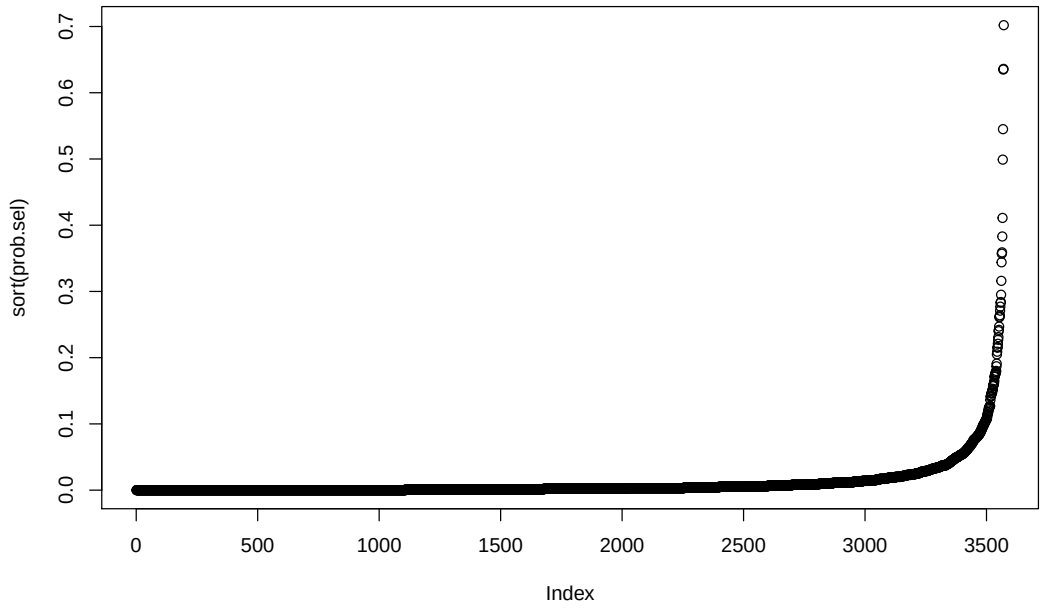
2.2 Additional analyses

Use parallel computing to

- Estimate the classification error for the Lasso by resampling the training/test sets a hundred of times. Plot the boxplot of the estimated classification error.
- Select the more stable genes by subsampling the training set, a.k.a. perform stability selection.

Let me treat the problem of stability selection

```
library(parallel)
stabsel <- function(i) {
  cat("+")
  b_sort <- sort(sample(1:ntrain,floor(ntrain/2)))
  out <- glmnet(Xtrain[b_sort,],Ytrain[b_sort], penalty.factor = 1/runif(ncol(Xtrain),
                lambda=cv.enet4.dev$lambda.min*0.1, alpha=.9)
  return(tabulate(which(out$beta[,ncol(out$beta)]!=0),p))
}

nrep <- 1e3
mc.cores <- 10
res.cum <- Reduce("+", mclapply(1:nrep, stabsel, mc.cores=mc.cores))
prob.sel <- res.cum/nrep
plot(sort(prob.sel))
```



```
thres <- .3
gene.stabsel <- Gene[prob.sel > thres]
```

I refit the logistic model only on the selected features in order to perform prediction

```
learn <- data.frame(y = Ytrain, Xtrain[,prob.sel > thres])
test <- data.frame(y = Ytest, Xtest[,prob.sel > thres])
fit <- glm(y~., data=learn, family="binomial")
yhat.stabsel <- 1*(predict(fit, test, type="response")>.5)
error.stabsel <- sum(abs(Ytest - yhat.stabsel))
error.stabsel
```

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

Perform a simple unsupervised clustering *of the samples*

- Apply spectral clustering (see function of the web page) with 2 groups on the correlation matrix of the sample
- check adequation with the true classification by computing the adjusted Rand Index between the two classifications.

Let's do this

```
library(mclust)
source("code/func_spectralClustering.R")
cl.sp <- SpectralClustering(cor(t(X)), 2) -1
sum(abs(Y - cl.sp)) ##  check label switching !
```

```
## [1] 3
```

```
adjustedRandIndex(Y,cl.sp)
```

```
## [1] 0.8371862
```

3 Unsupervised classification of the genes

In this part, we would like to regroup the predictors (i.e. the genes) together. The problem is that we do not have a clue on the classes we are looking for (contrary to the supervised problem, where there were two classes corresponding to AML/ALL). Here, we do not even know the number of groups we are looking for. We rather speak of clusters than classes for unsupervised classification.

3.1 Main analysis

- Compute the correlation matrix between all the genes. Plot this matrix (into a `jpeg` file to save time !). You can use the image method of the package `Matrix`, with option `useRaster=TRUE`.

Here I use a code available from the <web page (function `mat.plot`)

```
source("code/func_matplot.R")
corX <- cor(X)
print(mat.plot(corX))
```

- The BIC/ICL criterion for the k-means is defined by

$$BIC(k) = -2\log \ell(\mathbf{X}) + 2(kp+1)\log(n),$$

where k is the number of clusters and p the number of genes. The log-likelihood corresponds to a mixture of Gaussian distribution with spherical covariances, such that

$$\log \ell(\mathbf{X}) = -\frac{1}{2}(np + np\log(WSS/p)) - n\log(k) + \frac{np}{2}\log(2\pi).$$

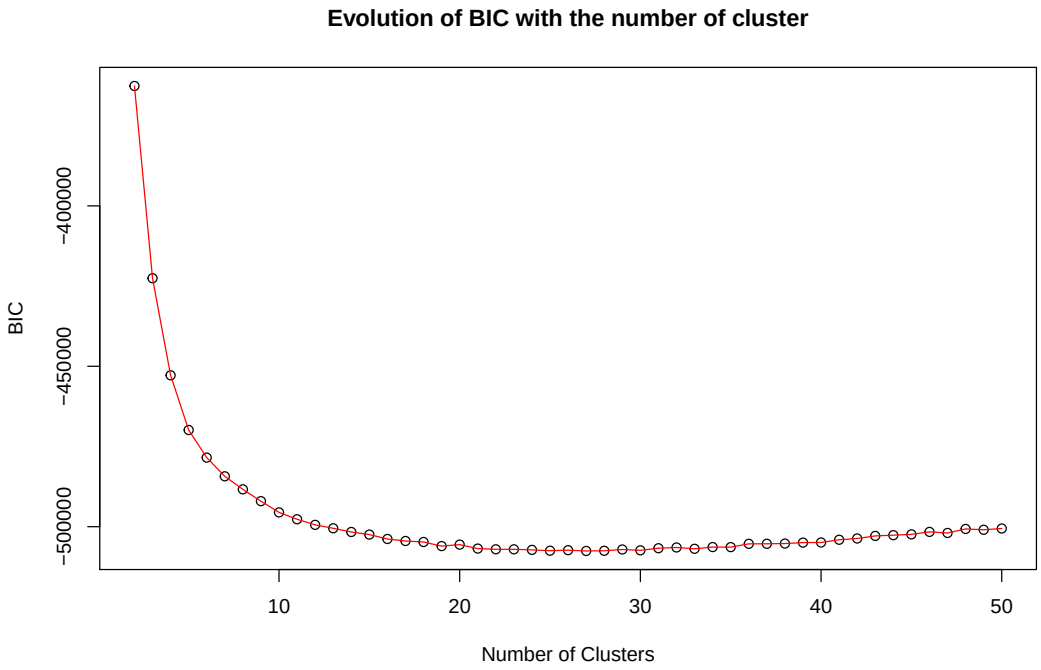
The total within sum of squares WSS is sent back by the `kmeans` function. If you fail in implementing the BIC, you can use the function available on the website. Use this criterion to choose the number of cluster, by performing k-means clustering for a number of groups varying from 2 to 50. Redo the plot

Here is the function

```
BIC <- function(reskmeans) {

  n<-length(reskmeans$cluster)
  k<-dim(reskmeans$centers)[1]
  p<-dim(reskmeans$centers)[2]
  traceS<-reskmeans$tot.withinss / n

  logLc=-0.5*(n*p + n*p*log(traceS/p))-n*log(k)+(n*p/2) *log(2*pi)

  nu<- k*p+1

  bic<- -2 *logLc + 2*nu*log(n)
  return(bic)
}
```

No go for a bunch or value of K

```
library(parallel)
Q <- 50
BICs <- mclapply(2:Q, function(k) {
  return(BIC(kmeans(t(X),k,nstart = 10,iter.max=30)))
}, mc.cores=10)

plot(2:Q, unlist(BICs),xlab="Number of Clusters",ylab="BIC",main="Evolution of BIC wit
lines(2:Q, unlist(BICs),col="red")
```



The plot associated with the minimal value is the following

```
k.min <- (2:Q)[which.min(unlist(BICs))]
cl <- kmeans(t(X),k.min,iter.max = 100)$cl
print(mat.plot(corX[order(cl), order(cl)]))
```

3.2 Additional Analysis

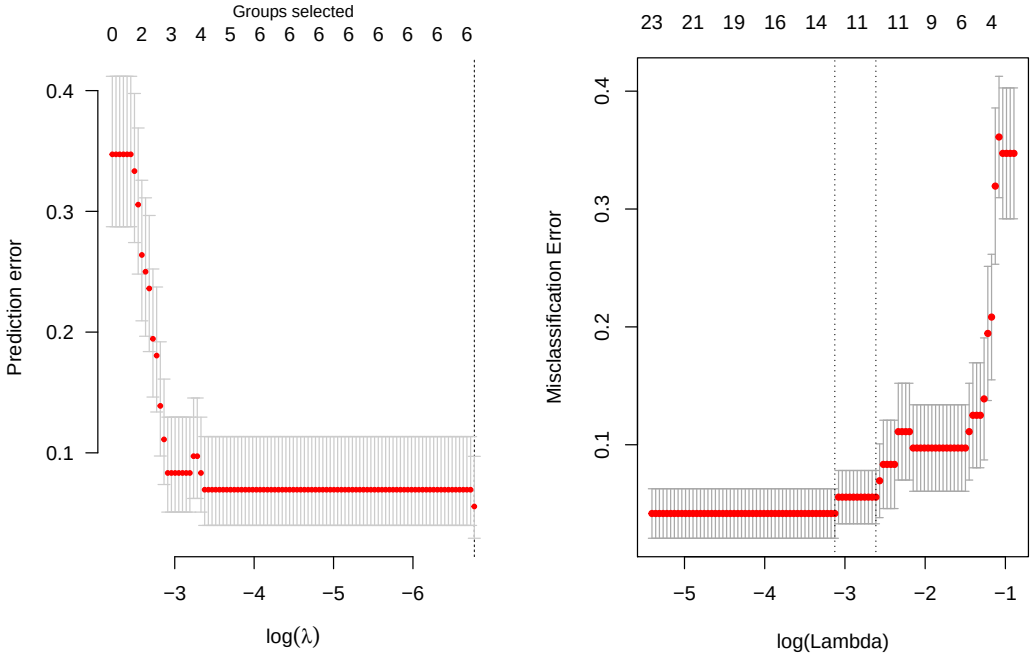
Now that we have a clustering of the genes, we can take it into account in our original supervised classification problem:

- Create a matrix of expression for the "meta-genes" by computing the average expression in each cluster of gene for each sample. Use sparse logistic regression to select the most relevant clusters of genes.

```
X.mean <- sapply(lapply(split(1:ncol(X), cl), function(ind) X[, ind, drop = FALSE]),ro
lasso <- glmnet(X.mean, Y, family = "binomial", lambda.min=1e-6)
cv.lasso <- cv.glmnet(X,Y, family = "binomial", type.measure = "class")
```

- Apply the logistic group-Lasso (package **grpreg**) using the grouping obtained above on the genes. Compare the groups selected by this method with the meta genes selected by the Lasso.

```
library(grpreg)
grplog <- grpreg(X,Y,group=cl, penalty="grLasso", family = "binomial", lambda.min = 0.
cv.grplog <- cv.grpreg(X,Y,group=cl, penalty="grLasso", family = "binomial", lambda.mi
par(mfrow=c(1,2))
plot(cv.grplog, type="pred")
plot(cv.lasso)
```




```
genes.grp <- Gene[which(coef(cv.grplog)[-1]  !=0)]
sel.grp <- sort(unique(cl[which(coef(cv.grplog)[-1]  !=0)]))
genes.grp
```

```
##   [1] "x.6"    "x.7"    "x.8"    "x.16"   "x.72"   "x.354"  "x.377"
##   [8] "x.435"  "x.436"  "x.490"  "x.510"  "x.592"  "x.608"  "x.623"
##  [15] "x.634"  "x.635"  "x.751"  "x.874"  "x.878"  "x.907"  "x.918"
##  [22] "x.926"  "x.927"  "x.956"  "x.979"  "x.1014" "x.1048" "x.1049"
##  [29] "x.1062" "x.1067" "x.1146" "x.1161" "x.1182" "x.1205" "x.1211"
##  [36] "x.1225" "x.1234" "x.1244" "x.1249" "x.1260" "x.1356" "x.1652"
##  [43] "x.1699" "x.1768" "x.1805" "x.1819" "x.2032" "x.2049" "x.2051"
##  [50] "x.2065" "x.2133" "x.2136" "x.2145" "x.2175" "x.2195" "x.2204"
##  [57] "x.2220" "x.2226" "x.2246" "x.2285" "x.2337" "x.2360" "x.2404"
##  [64] "x.2481" "x.2593" "x.2663" "x.2664" "x.2699" "x.2700" "x.2875"
##  [71] "x.2971" "x.2983" "x.2984" "x.2993" "x.2994" "x.3005" "x.3024"
##  [78] "x.3043" "x.3108" "x.3114" "x.3117" "x.3126" "x.3127" "x.3129"
##  [85] "x.3185" "x.3199" "x.3216" "x.3218" "x.3232" "x.3276" "x.3292"
##  [92] "x.3311" "x.3327" "x.3368" "x.3394" "x.3401" "x.3416" "x.3417"
##  [99] "x.3418" "x.3419" "x.3441" "x.3449" "x.3466" "x.3488" "x.3516"
## [106] "x.3517" "x.3571"
```

```
genes.lass <- Gene[predict(cv.lasso$glmnet.fit, Xtest, s=cv.lasso$lambda.min, type="no
genes.lass
```

```
##  [1] "x.456"  "x.626"  "x.672"  "x.956"  "x.979"  "x.1182" "x.1219"
##  [8] "x.1652" "x.1946" "x.2481" "x.2888" "x.3098" "x.3158" "x.3441"
```

The Lasso tends to get good representant from many groups

```
intersect(genes.lass, genes.grp)
```

```
## [1] "x.956"  "x.979"  "x.1182" "x.1652" "x.2481" "x.3441"
```

```
sort(unique(cl[names(cl) %in% genes.lass])) ## corresponding group number
```

```
## [1]  3  4  8 10 12 16 18 19 24
```

```
sel.grp
```

```
## [1] 10 11 12 17 21 25
```