

TP : STA212

Kais Cheikh, Bechir Trabelsi

Les bibliothéques qu'on va utiliser le long du projet.

```
require(MASS)
require(rpart)
require(rpart.plot)
require(caret)
require(doParallel)
require(randomForest)
require(xgboost)
require(ipred)
require(purrr)
require(ada)
```

Arbre de décision unique

Question 1 :

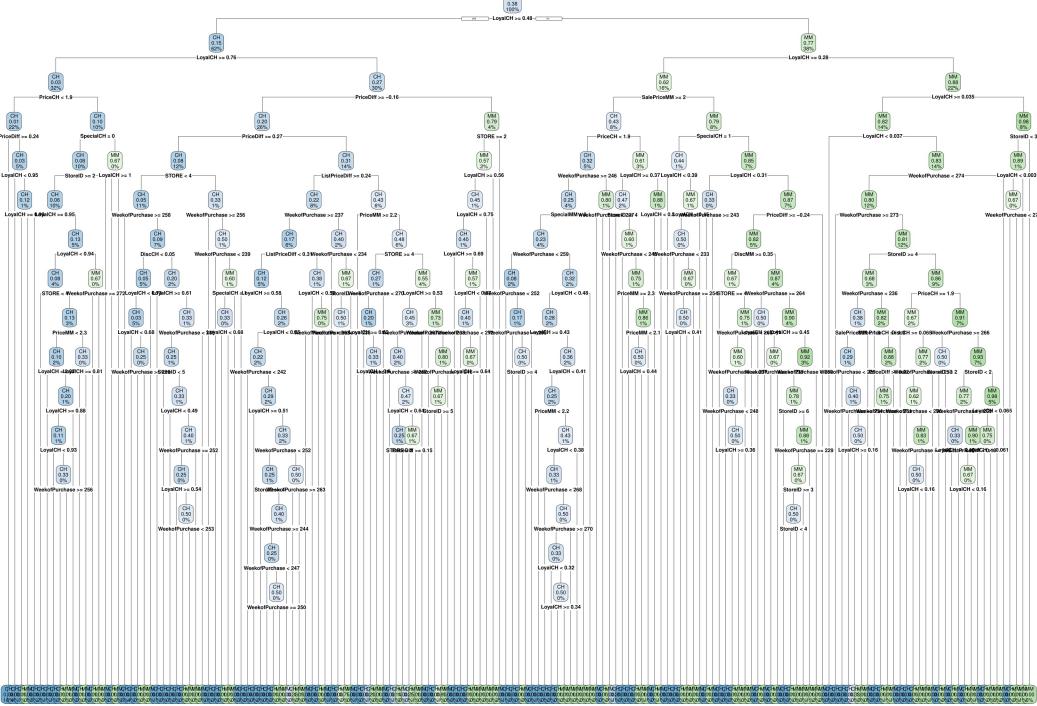
On va utiliser la fonction sample pour avoir 800 indice aléatoire pour créer les données d'apprentissage et de test. On va fixer la graine à 103 pour avoir les mêmes résultats d'une exécution à l'autre.

```
oj<-read.csv("oj.csv", header = TRUE)
set.seed(103)
train<-sample(c(1:dim(oj)[1]), 800)
oj_train<-oj[train,]
oj_Test<-oj[-train,]
```

Question 2 :

L'arbre de décision sans élagage :

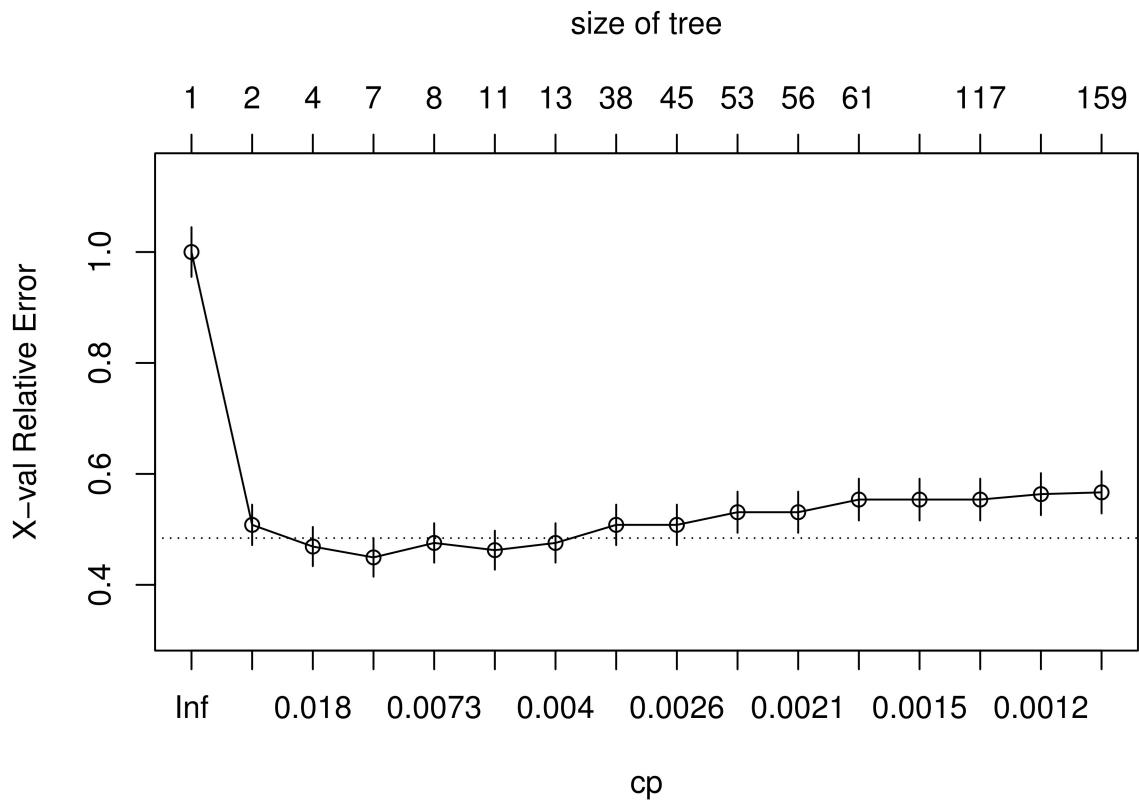
```
mod1<-rpart(Purchase~.,
              data=oj_train,
              control=rpart.control(minsplit=1, cp=0, xval=5))
rpart.plot(mod1)
```



```
pred.1 <- predict(mod1, obj_Test, type="class")
errTree = mean(obj_Test$Purchase != pred.1)
table(obj_Test$Purchase, pred.1)
```

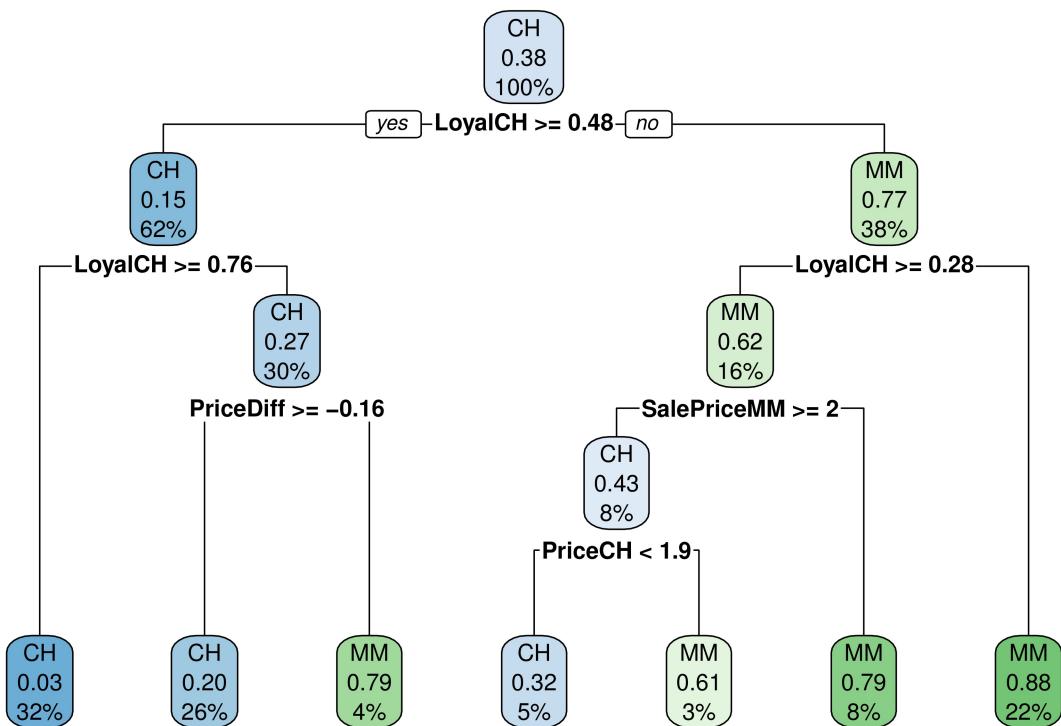
```
##      pred.1
##      CH  MM
##      CH 126 34
##      MM 34 76
```

```
plotcp(mod1)
```



L'arbre de décision avec élagage :

```
mod1.pruned <- prune(mod1, cp = mod1$cptable[which.min(mod1$cptable[, "xerror"]),"CP"])
rpart.plot(mod1.pruned)
```



```

pred1.pruned <- predict(mod1.pruned, obj_Test, type="class")
errTree.pruned = mean(obj_Test$Purchase != pred1.pruned)
table(obj_Test$Purchase, pred1.pruned)

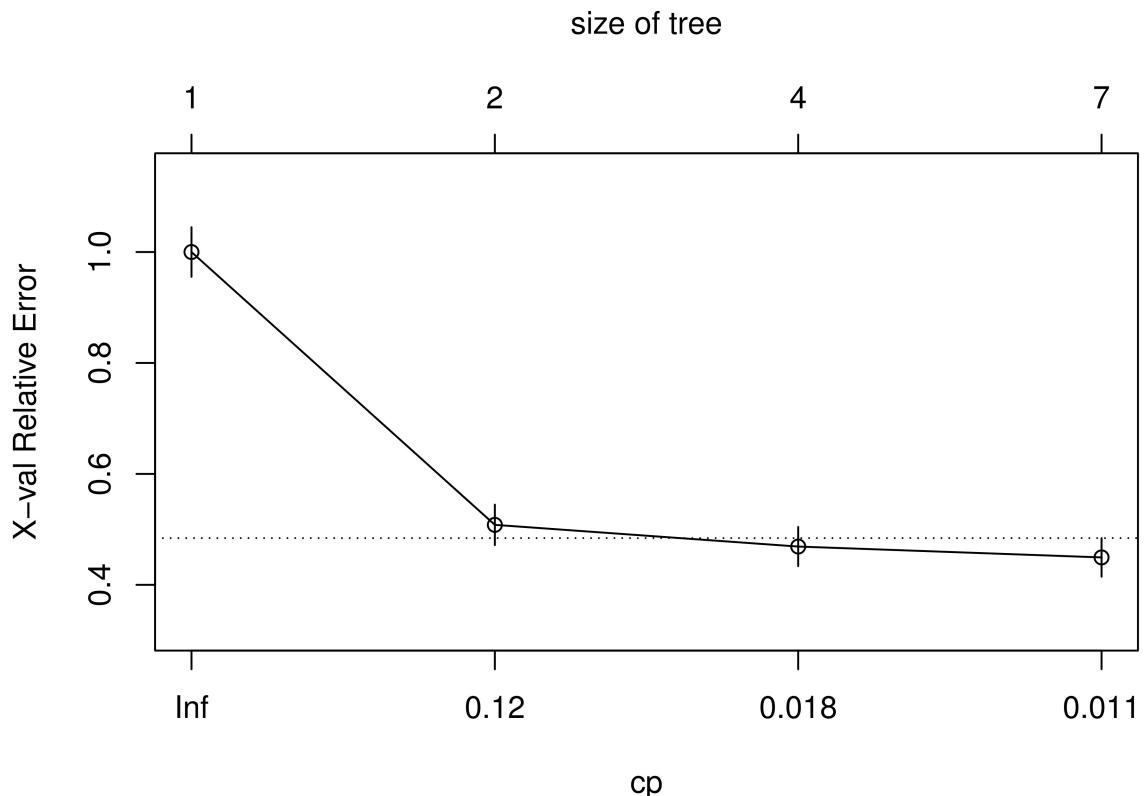
```

```

##      pred1.pruned
##      CH   MM
##  CH 134  26
##  MM  31  79

```

```
plotcp(mod1.pruned)
```

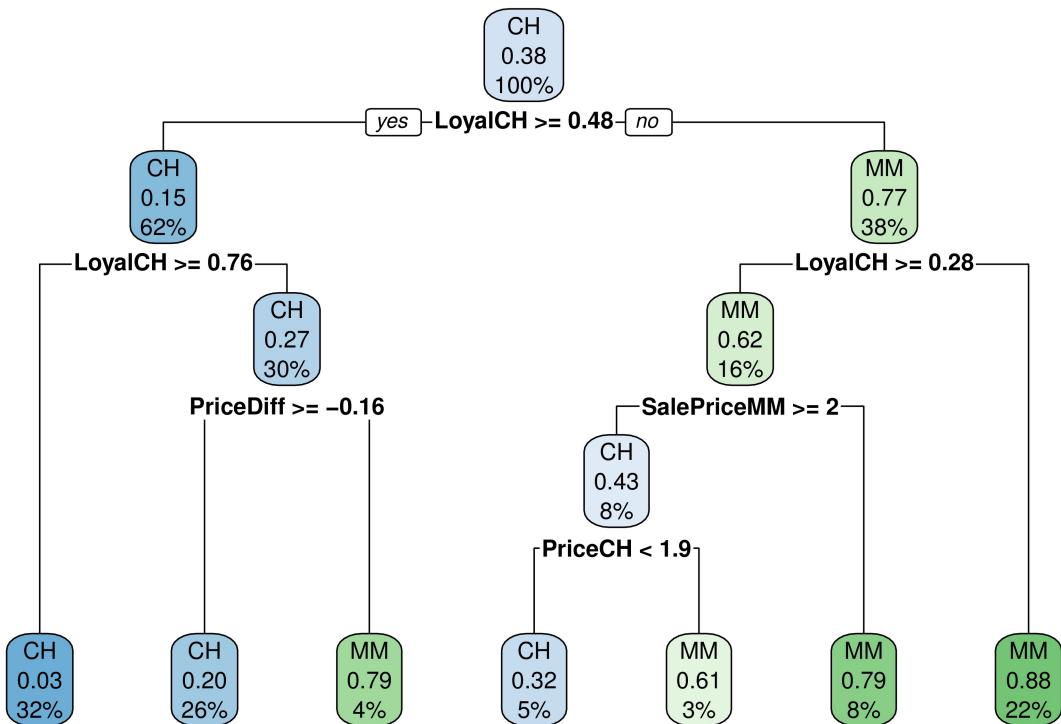


l'erreur sur le modèle avec élagage 0.2518519 est supérieur à celle du modèle élagé 0.2111111.

Question 3 :

Cette figure présente notre arbre élagé.

```
rpart.plot(mod1.pruned)
```



On choisit la feuille à l'extrême gauche. Les variables les plus importantes sont LoyalCH, PriceDiff et SalePriceMM. avec un écart très important entre LoyalCH et les autres variables.
 le noeud CH 0.03 32% on a 32 % de chance d'appartenir à cette feuille et elle caractérisé par les deux critères (LoyalCh ≥ 0.48) et (LoyalCh ≥ 0.76). Donc cette feuille est caractérisé par (LoyalCh ≥ 0.76) et elle donne le label CH

```
mod1.pruned$variable.importance
```

```
##          LoyalCH      PriceDiff      SalePriceMM       StoreID      DiscMM
##    169.2827058   24.4422831   19.4902289   14.2892052  12.9282060
##  PctDiscMM      PriceMM WeekofPurchase      PriceCH      STORE
##    12.9282060   12.0703423   8.2464386   5.6287177   2.4133300
##  SalePriceCH ListPriceDiff      DiscCH      PctDiscCH
##    2.2664990     1.2965684   0.6707225   0.6707225
```

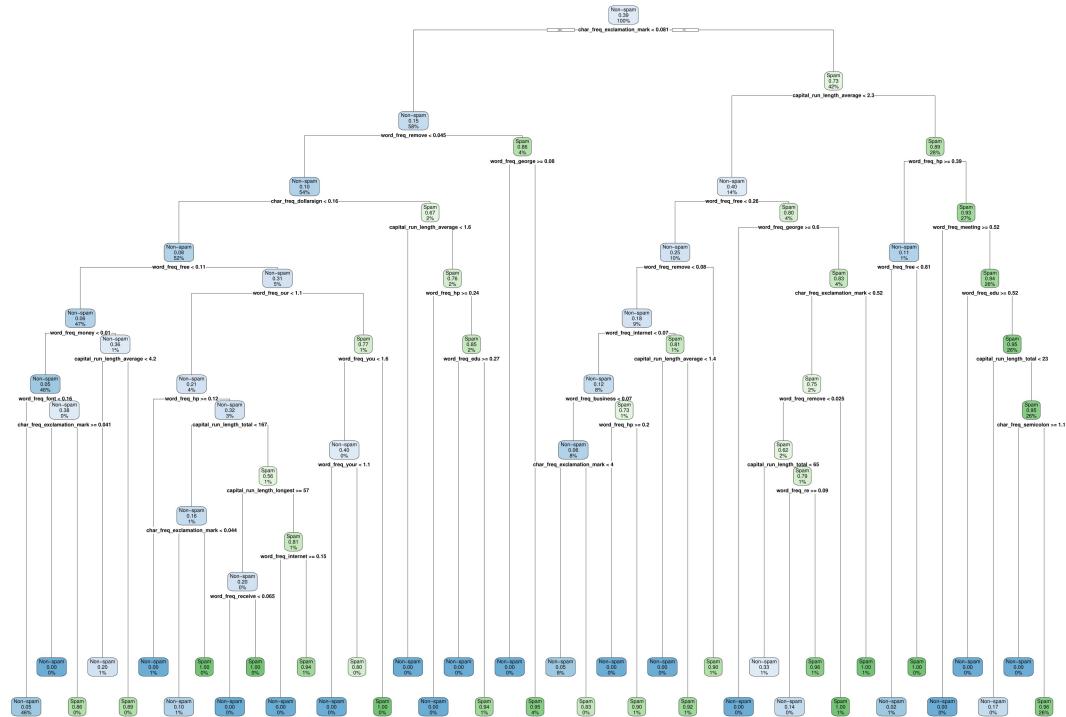
Forêt alétoires

On fixe encore une fois la graine à 103. On divise les données en 3/4 apprentissage et 1/4 pour le test.

```
mail<-read.csv("email.csv",header = TRUE)
set.seed(103)
train<-sample(c(TRUE,FALSE),dim(mail)[1],rep=TRUE,prob=c(0.75,0.25))
mail.train<-mail[train,]
mail.test<-mail[!train,]
```

Question 4 :

```
mod2<-rpart(Class~.,
             data=mail.train,
             control=rpart.control(minsplit=1,cp=0, xval=5))
mod2.pruned <- prune(mod2, cp = mod2$cptable[which.min(mod2$cptable[, "xerror"]),"CP"])
rpart.plot(mod2.pruned)
```



```
pred2.pruned <- predict(mod2.pruned, mail.test, type="class")
## erreur
err.mail = mean(mail.test$Class!=pred2.pruned)
table(mail.test$Class,pred2.pruned)
```

```
##          pred2.pruned
##          Non-spam Spam
##    Non-spam      658   42
##    Spam         57  396
```

l'erreur trouvé est 0.085863.

Question 5 :

On va utiliser deux méthodes pour déterminer un modèle se basant sur le bagging. Le premier avec la fonction *bagging* et le deuxième est avec *randomForest* car bagging est un cas particulier de randomForest.

```

bag.class =bagging(Class~,data=mail ,subset =train ,
                   nbagg=100,control=rpart.control(minsplit=1,cp=0, xval=5))

bag.class2<- randomForest(Class~,data=mail ,subset =train ,
                            mtry=57, importance =TRUE,ntree=100)

bag.pred<-predict(bag.class,mail.test,type="class")
mean(mail.test$Class!=bag.pred) ## erreur

## [1] 0.06678231

bag.pred2<-predict(bag.class2,mail.test,type="class")
mean(mail.test$Class!=bag.pred2) ## erreur

## [1] 0.0633131

```

Question 6 :

```

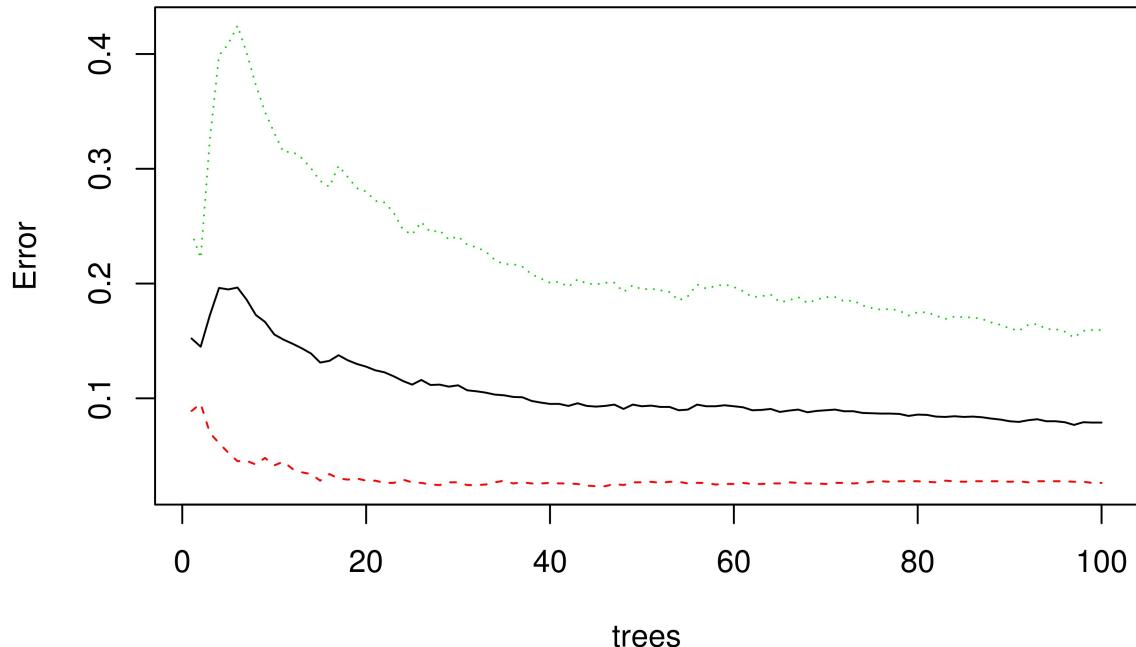
cl <- makePSOCKcluster(2)
registerDoParallel(cl)

RF.model =randomForest(Class~,data=mail ,subset =train ,
                        mtry=1:57, importance =TRUE,ntree=100)

stopCluster(cl)
plot(RF.model)

```

RF.model



```
pred.RF.caret <- predict(RF.model, mail.test, type="class")
#erreur de classification
mean(mail.test$Class != pred.RF.caret)
```

```
## [1] 0.08673027
```

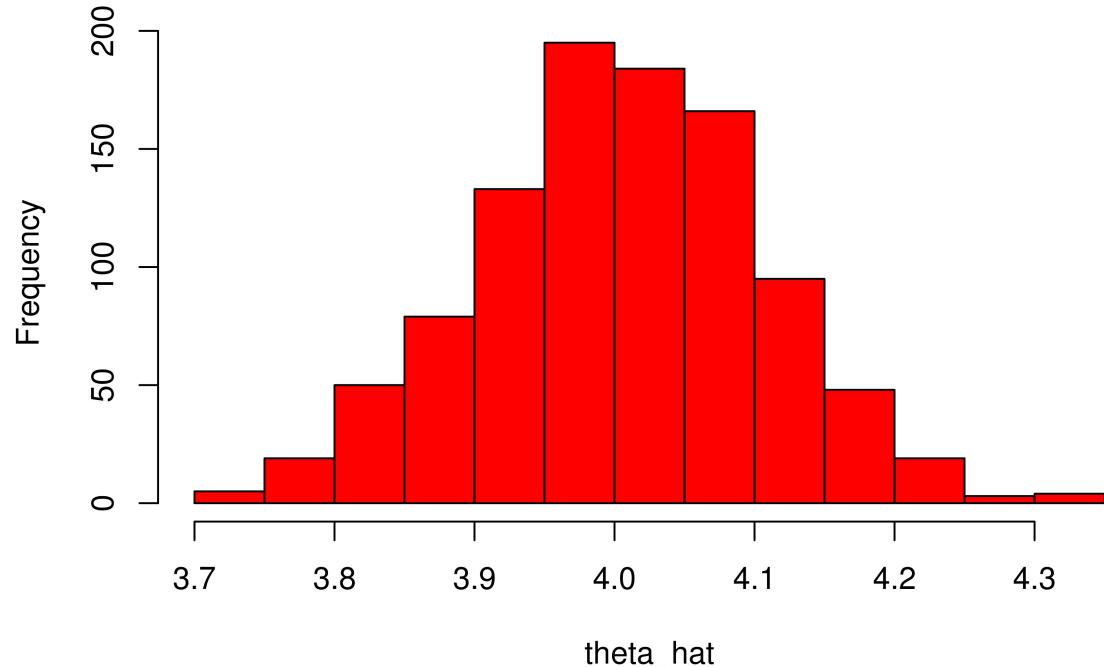
Bootstrap

Question 8 :

On sait que Y suit une loi $\mathcal{N}(\theta, 1)$ car $Y = \theta + \epsilon$ avec ϵ suit une $\mathcal{N}(0, 1)$. Ainsi on veut estimer la moyenne d'une variable aléatoire normale avec maximum de vraisemblance. Son expression est donnée par $\hat{\theta} = (X'X)^{-1}XY$ avec $X = (1, \dots, 1)$

```
M = 1000
n = 100
X = rep(1,n)
theta_hat<-c()
for ( i in 1:M){
  Y = rnorm (n,mean= 4 , sd = 1)
  theta_hat = c(theta_hat, solve((t(X) *%*% X)) *%*% t(X) *%*% Y )
}
hist(theta_hat, col= "red")
```

Histogram of theta_hat

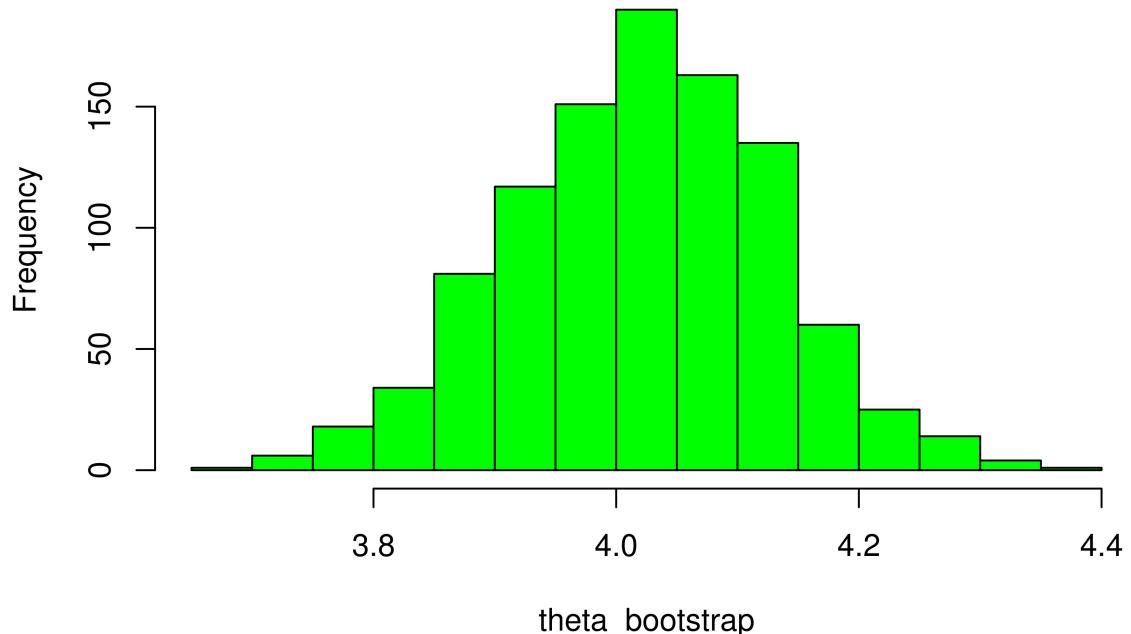


Question 9 :

Dans ce cas on se limite à un seul réalisation $n = 100$ des y_1, \dots, y_n et on tire d'une façon aléatoire et avec remise n échantillon bootstrap y_1^*, \dots, y_n^*

```
# le vecteur de réalisation originale
Y=rnorm(n,mean=4,sd=1)
X=rep(1,n)
theta_bootstrap = c()
for (i in 1:M){
  #tirage aléatoire des indices
  u = rdunif(n,1,n)
  theta_bootstrap =c(theta_bootstrap , solve((t(X) %*% X)) %*% t(X) %*% Y[u] )
}
hist(theta_bootstrap,col='green')
```

Histogram of theta_bootstrap



On remarque que les deux histogrammes admettent un pic en 4. Or que la distribution de $\hat{\theta}^*$ ressemble la distribution d'une loi normale que celle générée par $\hat{\theta}$.

Autour de l'algorithme Adaboost

Question 10 :

On a $\hat{y}(x) = \mathbb{I}\{\hat{c}(x) \geq 0\} - \mathbb{I}\{\hat{c}(x) < 0\}$, donc $\hat{y}(x)=1$ si $\hat{c}(x) \geq 0$ et -1 sinon

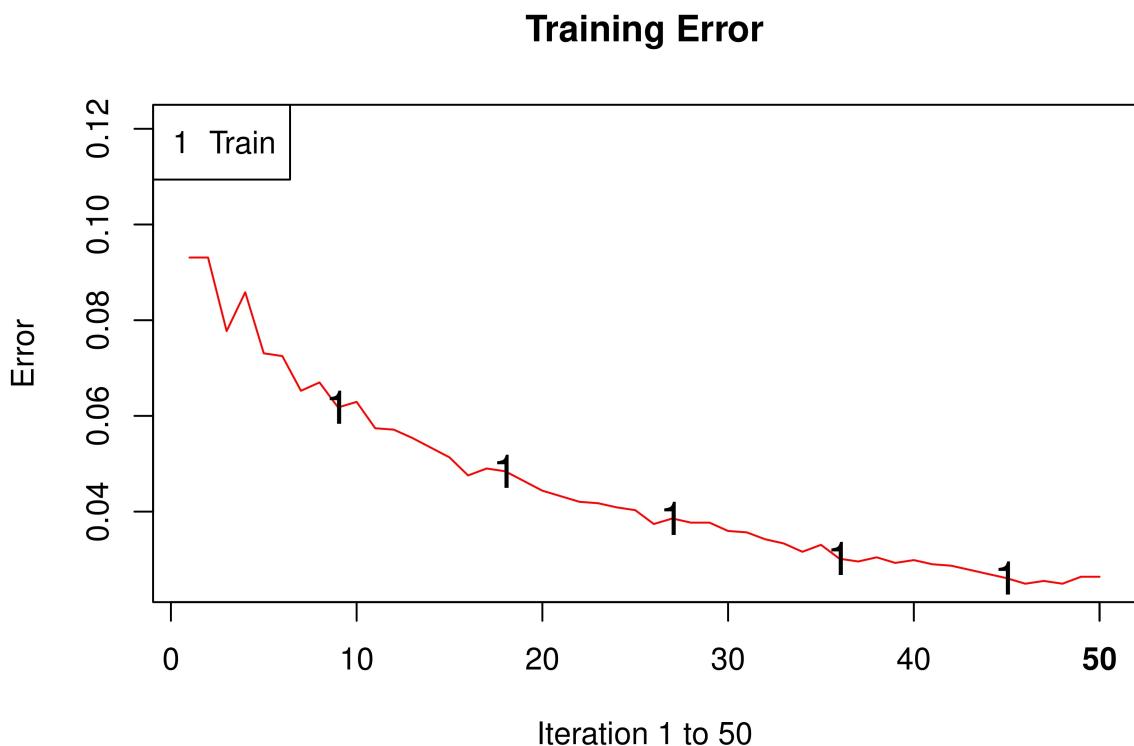
```
c_hat = c(0.3, -0.2, 1.5, -4.3)
y_star = c(-1, -1, 1, 1)
y_hat = ifelse(c_hat>=0 , 1, -1)
perre_exp = exp(-y_hat*c_hat)
perre_bin = ifelse(y_star!=y_hat, 1, 0)
Df = data.frame(c_hat,y_hat,perre_exp,perre_bin,y_star)
Df
```

```
##   c_hat y_hat  perre_exp perre_bin y_star
## 1  0.3     1  0.74081822      1     -1
## 2 -0.2    -1  0.81873075      0     -1
## 3  1.5     1  0.22313016      0      1
## 4 -4.3    -1  0.01356856      1      1
```

l'erreur exp est plus importante si $Y^* = -1$ donc elle est plus informative sur la valeur de Y^* , en effet plus l'erreur est importante plus Y^* a de chance de valoir -1 l'erreur binaire renseigne sur la différence donc elle est plus informative sur l'estimation

Question 11 :

```
x.train = mail.train[- dim(mail)[2]]  
y.train = mail.train$Class  
x.test = mail.test[- dim(mail)[2]]  
y.test = mail.test$Class  
n = dim(mail)[1]  
Ada = ada(x.train, y.train, x.test, y.test, iter=50, model.coef = TRUE, loss ="exponential")  
plot(Ada)
```



```
yhat_ada = predict(Ada, x.test)  
  
# calculate misclassification rate  
mean(yhat_ada != y.test)  
  
table(yhat_ada,y.test)
```

Pour reproduire la version originale de l'algorithme AdaBoost.M1 il faut utiliser l'erreur logistique binaire on doit donner un poids qui vaut $1/n$ pour chaque observations et l'arbre doit comporter que deux feuilles, 2

Question 12 :

On va utiliser les données *Boston* incluses dans *MASS* pour ajuster un modèle de régression par gradient boosting.

```
library(ISLR)
library(MASS)
attach(Boston)
head(Boston)
library(xgboost)
length(Boston$medv)
train<-sample(1:506,350)
bost.tr<-Boston[train,]
bost.te<-Boston[-train,]

cl <- makePSOCKcluster(3)
registerDoParallel(cl)
boosted = xgboost(data = as.matrix(bost.tr), label = bost.tr$medv,
                   objective = "reg:squarederror" , # la fonction de perte quadratique L2
                   booster = "gbtree",
                   #max.depth = 100,
                   eta = 1,
                   nrounds = 2000)
stopCluster(cl)

pred.boost = predict(boosted , as.matrix(bost.te))
pred.boosttr = predict(boosted , as.matrix(bost.tr))
#erreur sur l'echantillon apprentissage:
mean((pred.boosttr -bost.tr$medv)^2)

## [1] 3.12573e-07

#erreur sur l'echantillon test:
mean((pred.boost -bost.te$medv)^2)

## [1] 0.2933699
```

On voit que l'erreur sur l'échantillon d'apprentissage est quasiment nulle, alors que sur l'échantillon test l'erreur est assez importante (25%), c'est le phénomène de sur-apprentissage.