STA212 - Méthodes de rééchantillonnage

Enseignant: Mohammed Sedki

Devoir : aspects pratiques

Romin DURAND
Loukman Eltarr

May 10, 2020

Arbre de décision unique

```
setwd('~/Cours/STA212/STA212DM')
#setwd("/home/lokmen/Documents/ENSTA/STAT/STA212/STA212DM")
rm(list = objects())
graphics.off()
OJ=read.csv("oj.csv", header = TRUE)
#View(OJ)
```

On regarde la nature de nos données. On a 1070 observations pour 18 variables différentes. Les variables categorielles sont `Purchase` qui admet deux niveaux, et `Store 7` qui admet aussi deux niveaux. Les autres sont numériques.

```
str(OJ)
```

```
## 'data.frame':    1070 obs. of  18 variables:
##  $ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...
##  $ WeekofPurchase: int  237 239 245 227 228 230 232 234 235 238 ...
##  $ StoreID       : int  1 1 1 1 7 7 7 7 7 7 ...
##  $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99 ...
##  $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
##  $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
##  $ SpecialCH     : int  0 0 0 0 0 0 1 1 0 0 ...
##  $ SpecialMM     : int  0 1 0 0 0 1 1 0 0 0 ...
##  $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
##  $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
##  $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
##  $ Store7        : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...
##  $ PctDiscMM     : num  0 0.151 0 0 0 ...
##  $ PctDiscCH     : num  0 0 0.0914 0 0 ...
##  $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
##  $ STORE         : int  1 1 1 1 0 0 0 0 0 0 ...
```

Analyse Univariée

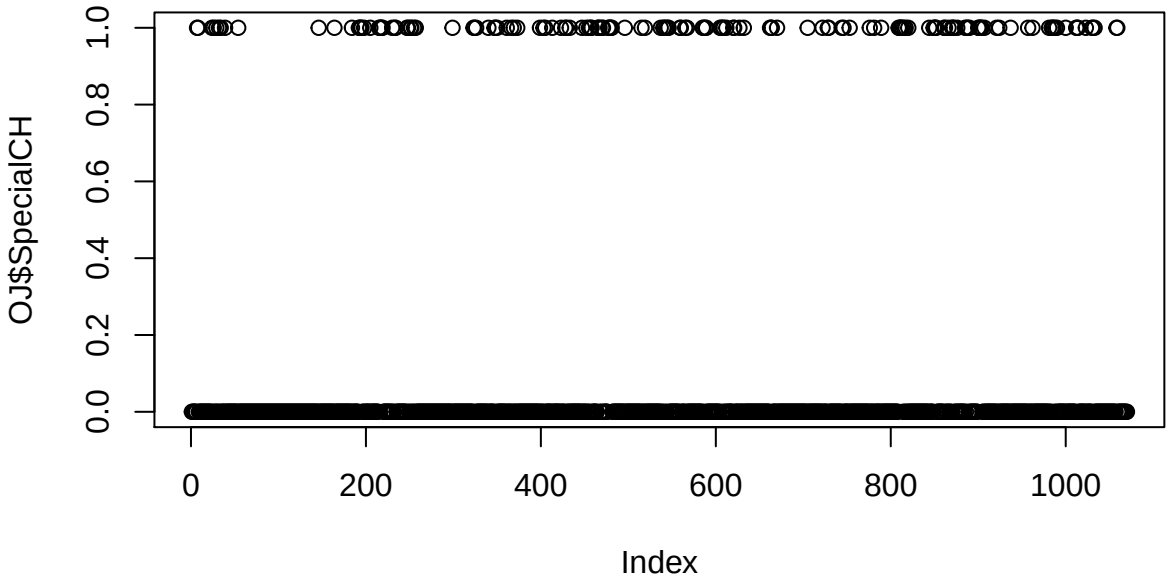
On procéde à une analyse univariée des variables. On se sert de la description des variables ainsi que des commandes `summary`, `plot` et `table`.

Par exemple, on peut voir que les variables `SpecialCH` et `SpecialMM` prennent seulement les valeurs 0 et 1.

```
table(OJ$SpecialCH)
```

```
##
##   0   1
## 912 158
```

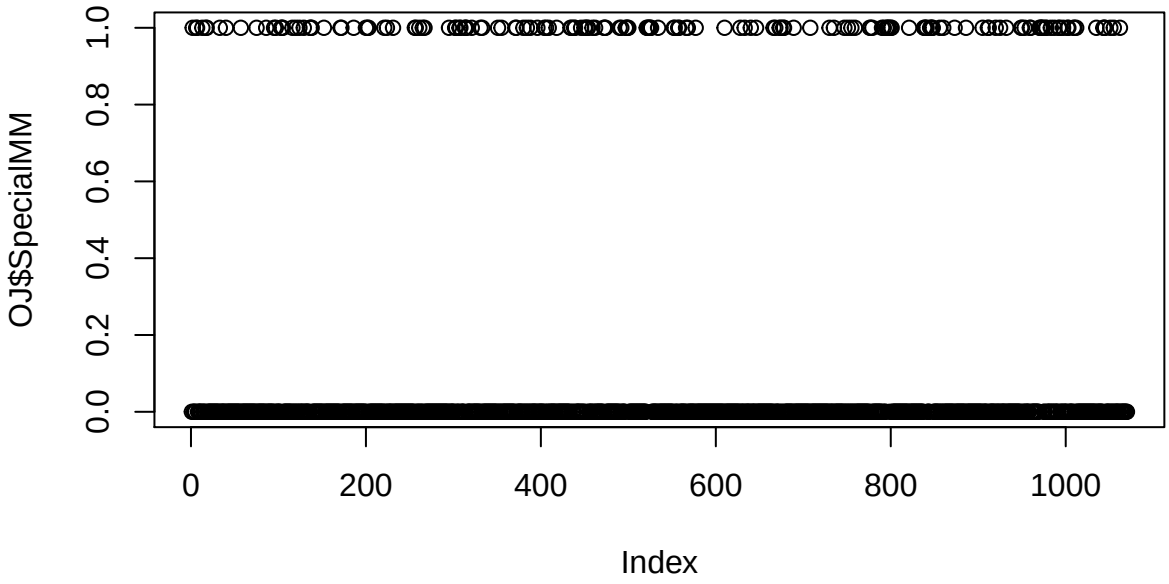
```
plot(OJ$SpecialCH)
```




```
table(OJ$SpecialMM)
```

```
##
##   0   1
## 897 173
```

```
plot(OJ$SpecialMM)
```

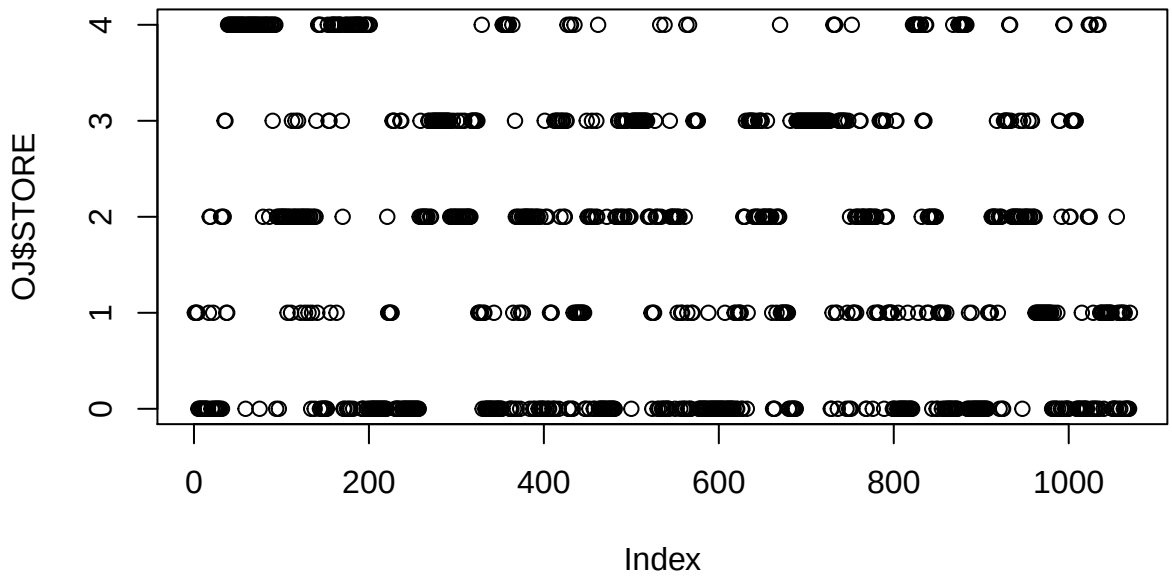


De la même manière `STORE` ne prend que les valeurs entre 0 et 4.

```
table(OJ$STORE)
```

```
##
##   0   1   2   3   4
## 356 157 222 196 139
```

```
plot(OJ$STORE)
```



On préfère alors les transformer en variables catégorielles:

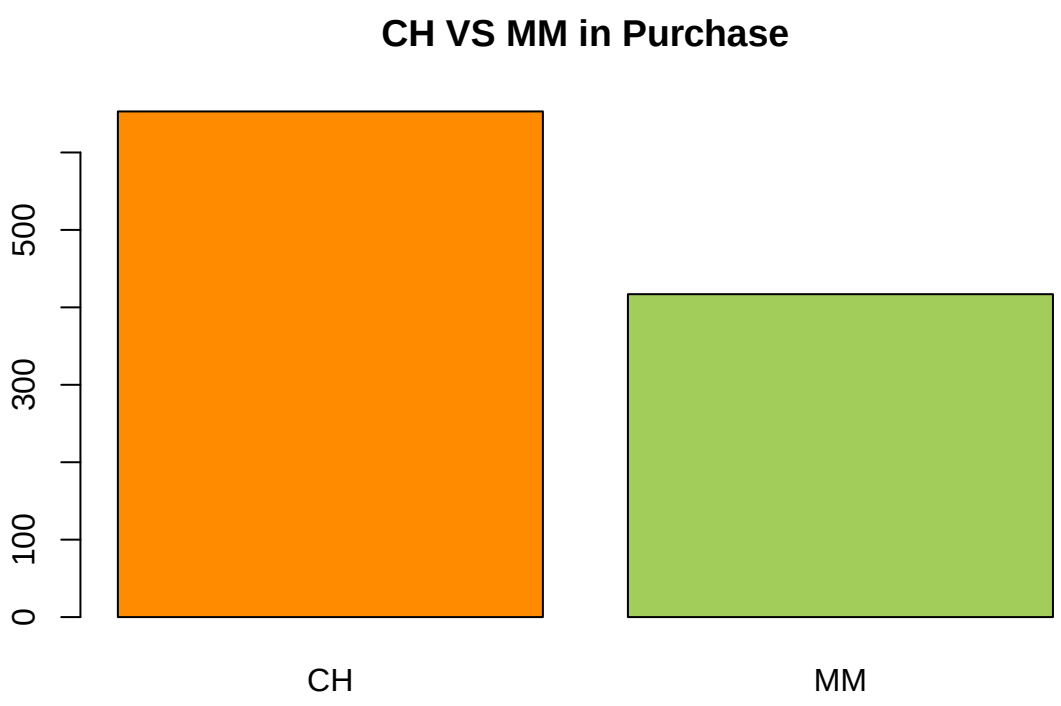
```
OJ$SpecialMM <- as.factor(OJ$SpecialMM)
OJ$SpecialCH <- as.factor(OJ$SpecialCH)
OJ$STORE <- as.factor(OJ$STORE+1) ## On préfère avoir des valeurs entre 1 et 5.
```

On regarde la proportion de "MM" par rapport à celle de "CH". Il ya plus de CH que de MM qui ont été commendés. La proportion est de 61%-39%.

```
table(OJ$Purchase)/nrow(OJ)
```

```
##
##        CH        MM
## 0.6102804 0.3897196
```

```
plot(OJ$Purchase, main ="CH VS MM in Purchase",col=c("darkorange","darkolivegreen3"))
```



Question 1

On divise d'abord notre jeu de donnée en une partie **train** et une partie **test**

```
set.seed(2) ## On précise la graine afin d'avoir la reproductibilité
size_sample=800 ## tailer de notre echantillon train
train <- sample(c(1:nrow(OJ)), size=floor(size_sample)) ## liste comportant l'index de la partie train
```

Question 2

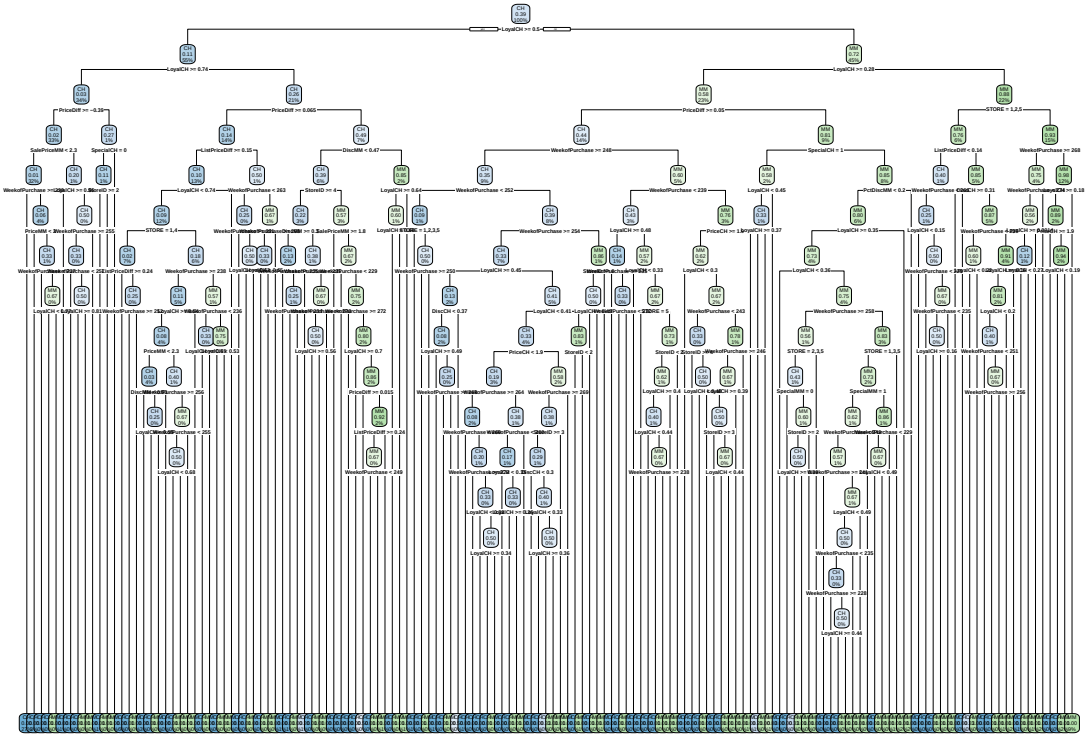
On va utiliser l'apprentissage par arbre de décision pour faire un prédction sur la partie test à partir de la partie train. On utilise d'baord un arbre obtenu sans élagage puis un autre avec élagage, on comparera ensuite la qualité des deux prédictions.

```
require(rpart)
require(rpart.plot)
```

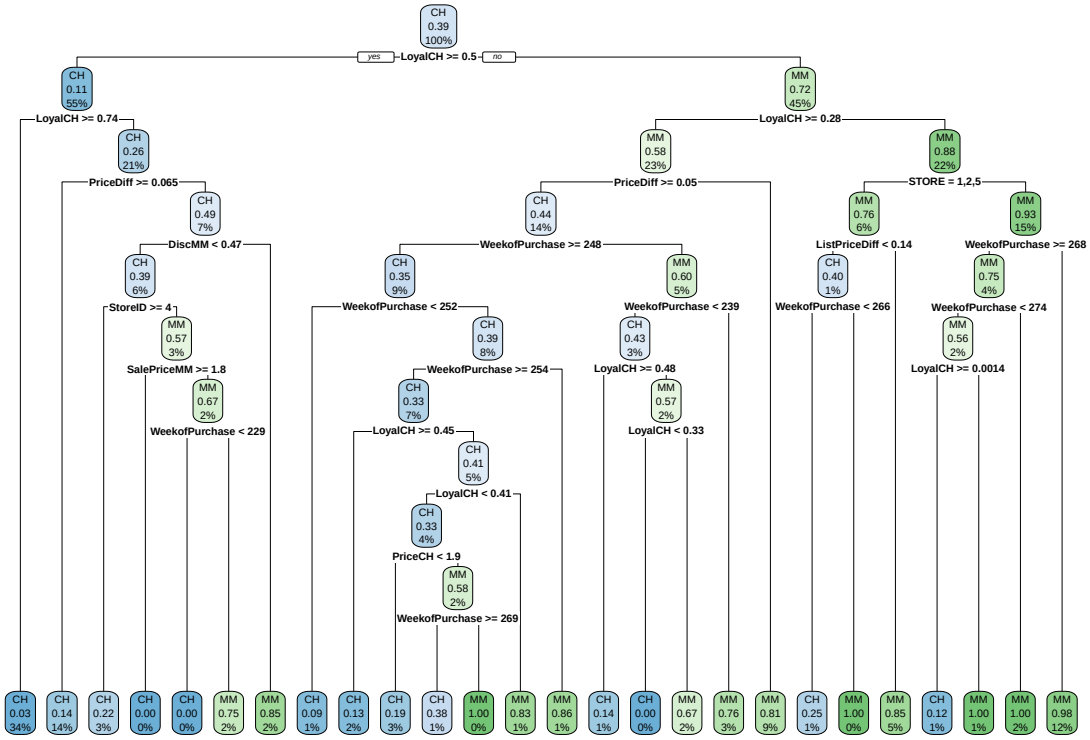
```
## Arbre sans élagage
#   # Avec minsplit =1 on continue à explorer un noeau tant qu'il y a plus d'une seule variable
#   # cp = 0 indique qu'on explore un noeau même si il n'apporte pas de précision supplémentaire
#   # xval= 10 pour le nombre de fold dans notre validation croisée
three.0 <- rpart(Purchase~.,
                 data=OJ[train,],
```

```
                control=rpart.control(minsplit=1,cp=0, xval=10),model =TRUE)
rpart.plot(three.0)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```
## Arbre avec élagage
three.1 <- prune(three.0, cp = three.0$cptable[which.min(three.0$cptable[,"xerror"]),"CP"],model =TRUE)
rpart.plot(three.1)
```



On voit que le graphique obtenu sans élagage est difficilement lisible ou interpretable. Il présente aussi probablement un problèmes d'affichage au vu du de la trop importante information qu'il contient. Au contraire, pour l'arbre pourlequel on a utilisé l'élagage, on voit clairement les noeuds internees et les feuilles ainsi que les proportions associées.

Erreurs de Prédiction

On compare ensuite leurs erreurs de prediction.

```
pred.0 <- predict(three.0, OJ, type="class")
mean(OJ$Purchase[-train]!=pred.0[-train])
```

```
## [1] 0.2222222
```

```
pred.1 <- predict(three.1, OJ, type="class")
mean(OJ$Purchase[-train]!=pred.1[-train])
```

```
## [1] 0.2259259
```

On note que notre taux d'erreur est autour de 0.21 pour le premier arbre. Ell est légerement inférieur à 0.16 pour le deuxième. Le second est donc un légèrement meilleur.

Matrice de Confusion

On dresse alors une matrice de confusion. On a :

- Pour l'abre sans élagage :
  - 142 CH qui ont correctement été prédits.
  - 31 CH qui ont été incorrectement prédits.
  - 26 MM qui ont été incorrectement prédits.
  - 71 MM qui ont correctement été prédits. Globalement la prediction est assez bonne.

```
table(OJ$Purchase[-train],pred.0[-train],dnn = c("Purchase", "Prediction"))
```

```
##          Prediction
## Purchase  CH  MM
##       CH 138  25
##       MM  35  72
```

- Pour l'abre sans élagage :
  - 151 CH qui ont correctement été prédits.
  - 22 CH qui ont été incorrectement prédits.
  - 21 MM qui ont été incorrectement prédits.
  - 77 MM qui ont correctement été prédits. Globalement la prediction est assez bonne.

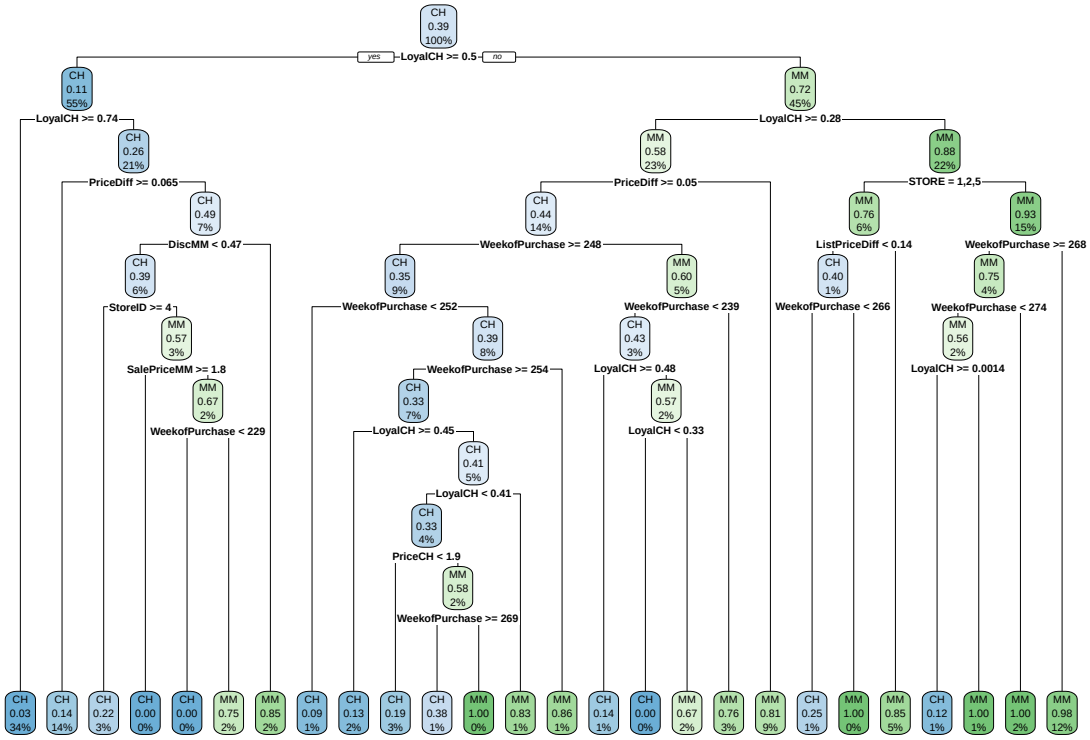
```
table(OJ$Purchase[-train],pred.1[-train],dnn = c("Purchase", "Prediction"))
```

```
##          Prediction
## Purchase  CH  MM
##       CH 134  29
##       MM  32  75
```

Question 3

Ici on commente l'apparition de la feuille la plus à gauche. On peut voir que depuis la racine il ya une séparation selon la valeur de `LoyalCH` par deux fois. En effet, l'algorithme a d'abord distingué les valeurs supérieures et inférieures à 0.5 puis identiquement pour la valeurs 0.74. La zone correspondant à `LoyalCH` $>$ 0.74 contient donc 34% des obseevations.

```
rpart.plot(three.1)
```

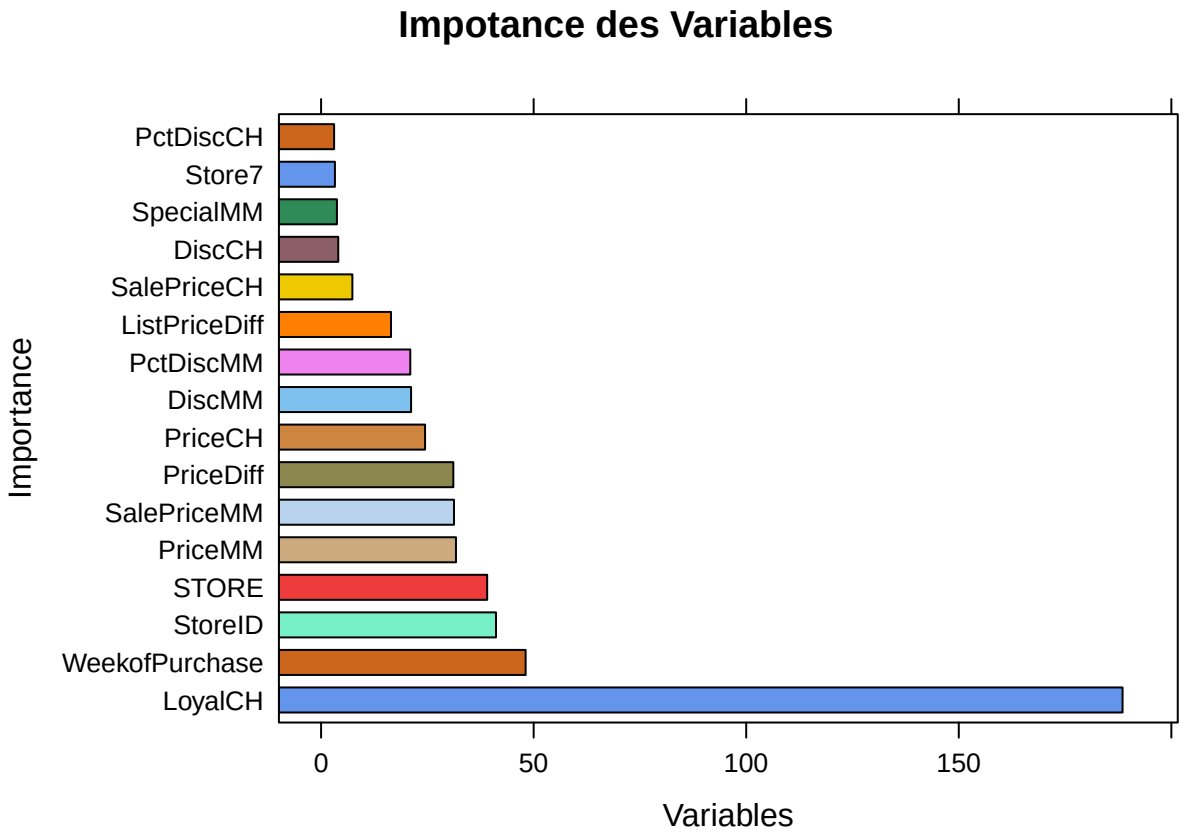


On peut observer l'importance relative des variables.

```
three.1$variable.importance
```

```
##        LoyalCH WeekofPurchase         StoreID           STORE        PriceMM
##     188.526325      48.116729      41.142199      39.069757      31.727448
##    SalePriceMM      PriceDiff        PriceCH        DiscMM      PctDiscMM
##      31.264888      31.110619      24.459878      21.170419      20.986853
##  ListPriceDiff    SalePriceCH        DiscCH      SpecialMM         Store7
##      16.450657       7.363646       4.044643       3.730935       3.259171
##      PctDiscCH
##       3.062500
```

```
require(lattice)
barchart(three.1$variable.importance,xlab = "Variables", ylab = "Importance", main="Impotance des Varial
```



Forêt aléatoires

Il faut tout d'abord changer la variable Class en variable factor :

```
email$Class = as.factor(email$Class)
```

Question 4

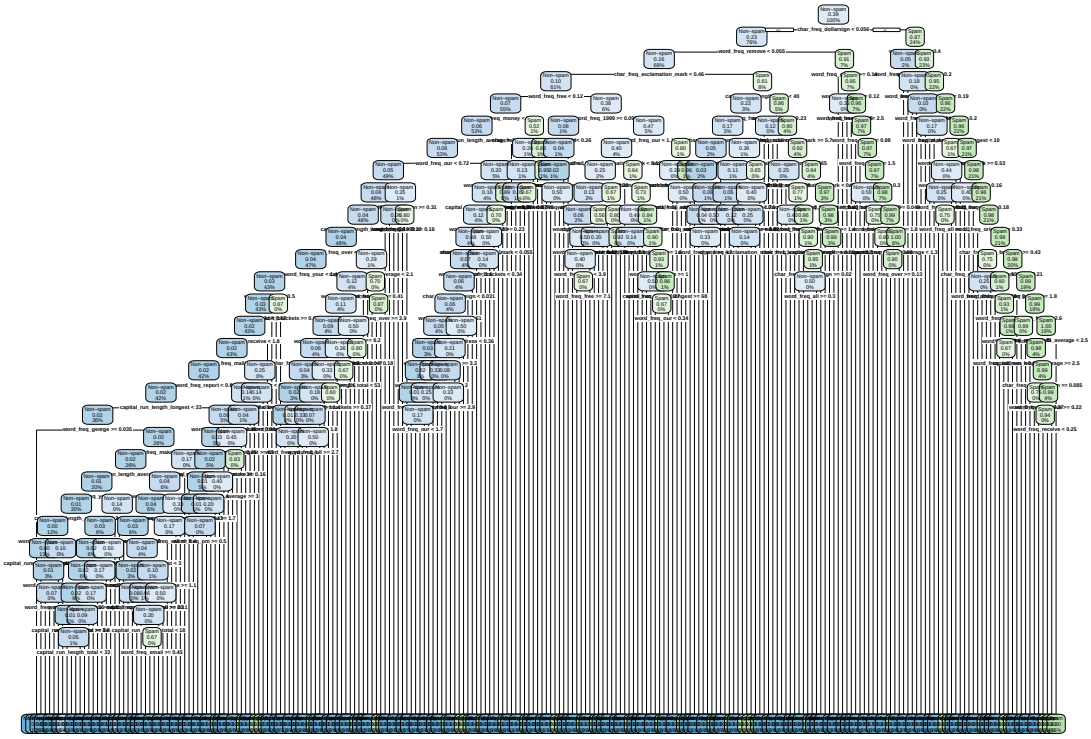
```
require(rpart)
require(rpart.plot)
require(ipred)
require(caret)
require(randomForest)
require(doParallel)
require(JOUSBoost)
require(xgboost)
```

```
N = nrow(email)
set.seed(103)
train = sample(1:N, round(0.75*N))
email.tr = email[train,]
email.te = email[-train,]
```

Ajustons tout d'abord un arbre sans élagage :

```
cart.0 <- rpart(Class~.,
                data=email.tr,
                control=rpart.control(minsplit=1,cp=0, xval=30)) # minsplit=1,cp=0
                                                                 # pour avoir un arbre le
                                                                 # plus profond possible
rpart.plot(cart.0)
```

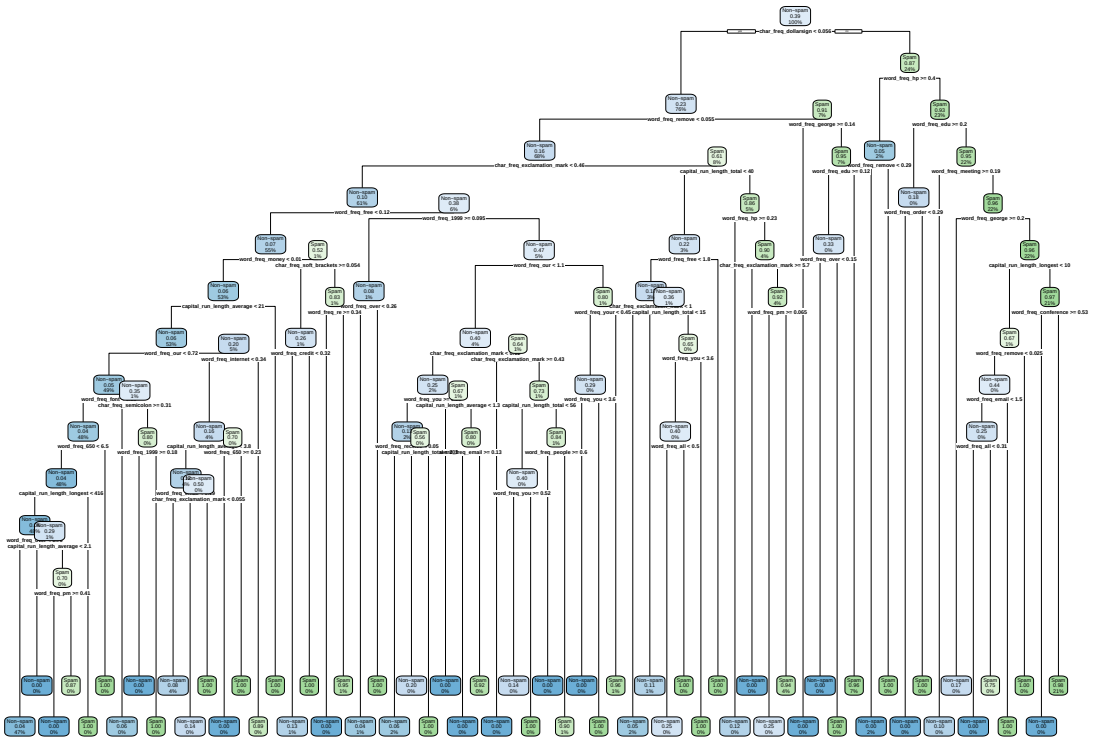
```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Puis élagons cet arbre :

```
cart.pruned <- prune(cart.0, cp = cart.0$cptable[which.min(cart.0$cptable[,"xerror"]),"CP"])
rpart.plot(cart.pruned)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Enfin on calcul l'erreur de test (taux de mauvais classement), en appliquant la règle de Bayes :

```
pred.pruned <- predict(cart.pruned, email.te)
mean(abs(ifelse(email.te$Class == "Spam", 1,0) - ifelse(pred.pruned[,2] >.5, 1,0)))
```

```
## [1] 0.0773913
```

Nous atteignons donc un taux de mauvais classement de 8% avec ce modèle.

Question 5

Nous réalisons un bagging 100 arbres de décisions à l'aide de la fonction bagging de la librairie ipred :

```
bag.email <- bagging(Class~., data=email.tr, mfinal=100)
```

Puis on peut évaluer l'erreur de test :

```
pred.bag <- predict(bag.email, email.te)
mean(abs(ifelse(email.te$Class == "Spam", 1,0) - ifelse(pred.bag == "Spam", 1,0)))
```

```
## [1] 0.04695652
```

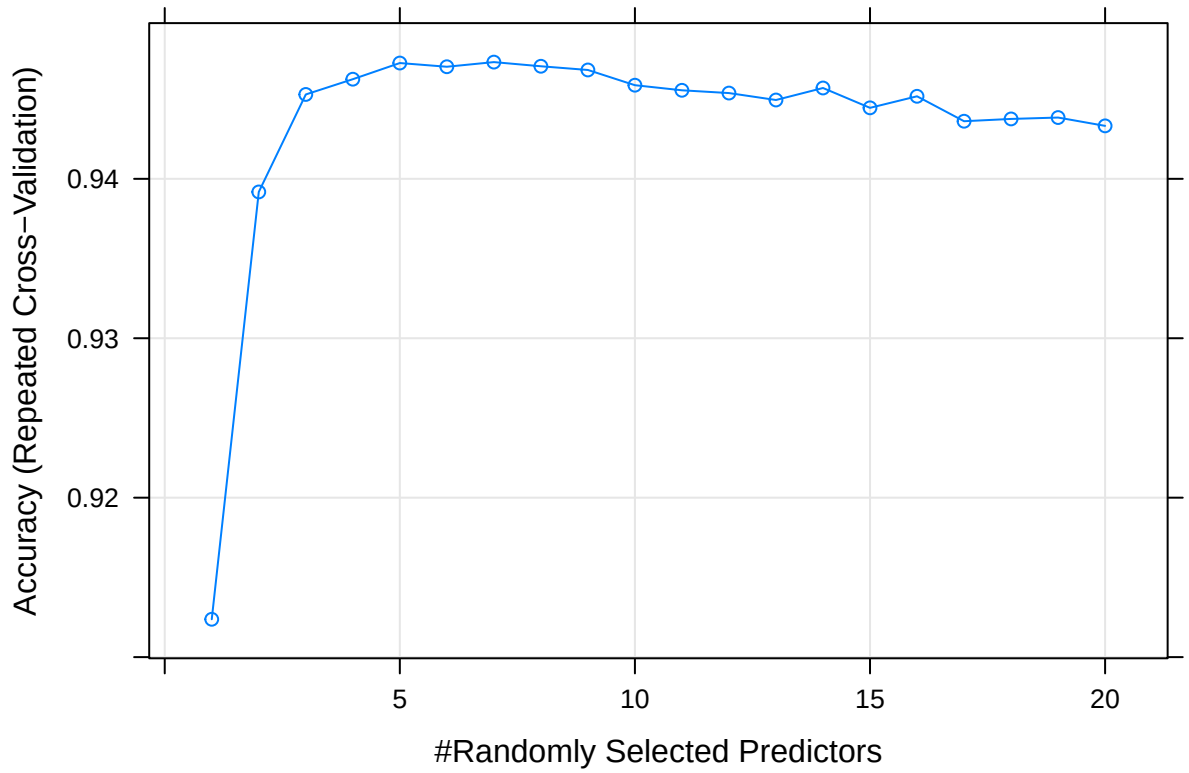
Nous avons donc une erreur de test de 7,5%.

Question 6

Enfin, nous ajustons un modèle random forest à 100 arbres en choisisant le mtry, (c'est à dire le nombre de variable que l'on prend pour chaque arbre), par validation croisée.

```
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

control <- trainControl(method="repeatedcv", number=5, repeats=10)
rfGrid <-  expand.grid(mtry = 1:20)
RFmodel <- train(Class~., data=email.tr, method="rf",
                 trControl=control,
                 ntree=100,
                 tuneGrid = rfGrid,
                 verbose=FALSE)
stopCluster(cl)
plot(RFmodel)
```



Le choix de mtry fait par validation croisée n'est pas reproductible en pratique. Cela demande trop de temps de calcul pour réactualiser le mtry à chaque fois que les donnée changent.

Puis on peut évaluer l'erreur de test :

```
pred.rf <- predict(RFmodel, email.te)
mean(abs(ifelse(email.te$Class == "Spam", 1,0) - ifelse(pred.rf == "Spam", 1,0)))
```

```
## [1] 0.04347826
```

On a une erreur de test de 6%.

Question 7

Le modèle random forrest est le plus performant en terme d'erreur de test. Cependant, il est aussi le plus couteux en terme de temps de calcul, et n'est pas reproductible en pratique. Le modèle avec le bagging est assez performant en terme d'erreur de test, et ne demande pas un trop grand temps de calcul. Néanmoins, 100 arbres avec bagging ne font pas beaucoup mieux qu'un seul arbre. En conclusion si on doit manipuler une trop grande base de données d'email, la complexité est bien trop grande pour utiliser le random forest, nous préfèrerons le bagging même s'il n'y a pas un grand gain de performance avec le modèle Cart.

Bootstrap

Question 8

Ici, on considère une variable aléatoire Y qui suit une loi normal de moyenne θ et de variance 1. On souhaiterait estimer son espérance par estimateur du maximum de vraisemblance.

```
y = rnorm(100,4,1)
```

On a pour la vraisemblance

$$\begin{aligned} L(y_1,\dots,y_n \mid \theta) &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{(y_i-\theta)^2/2} \\ &= \frac{1}{\sqrt{2\pi}^n} e^{\sum_{i=1}^{n}(y_i-\theta)^2/2} \end{aligned}$$

En passant au log

$$\Longrightarrow$$

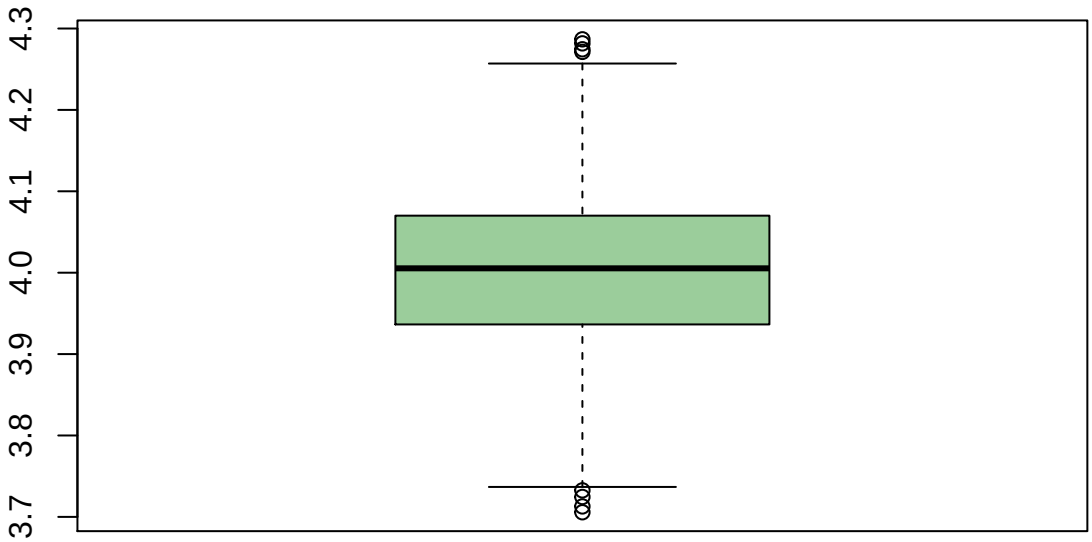
$$log(L(y_1,\dots,y_n \mid \theta)) = log(\frac{1}{\sqrt{2\pi}^n}) + \sum_{i=1}^{n}(y_i-\theta)^2/2$$

Maximisier la vraisemblance revient à maximiser $\frac{1}{2}\sum_{i=1}^{n}(y_i-\theta)^2$. On en déduit que l'EMV ici est

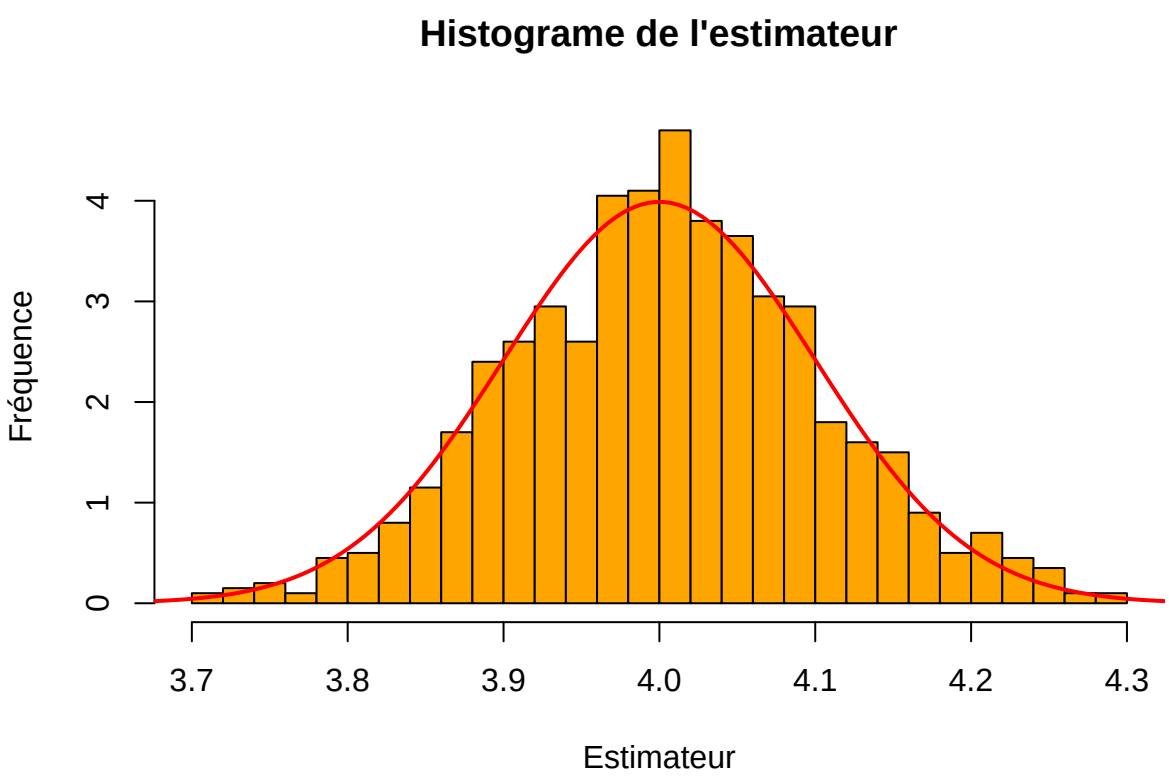
$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} y_i.$$

Il coïncide avec l'estimateur des moments et avec la moyenne empirique. On va produire plusieurs valeurs de $\hat{\theta}$:

```
n <- 100
N <- 1000
set.seed(11)
theta_hat <- rowMeans(matrix(rnorm(N*n,4,1), N,n))
boxplot(theta_hat,col = 'darkseagreen3')
```



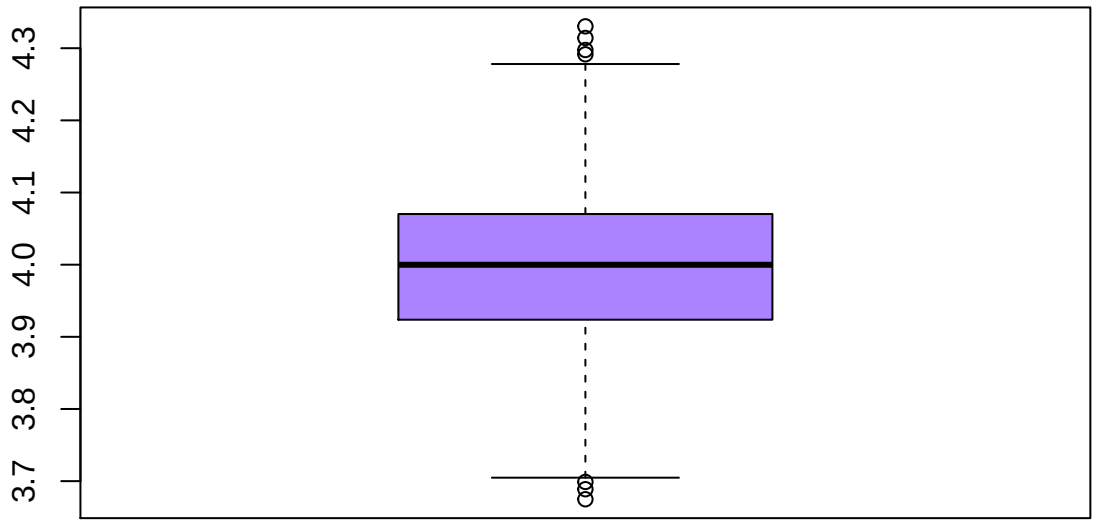

```r
hist(theta_hat, ylab="Fréquence", xlab="Estimateur",main="Histograme de l'estimateur", col="orange",brea
x <-seq(3.5, 4.5, le=200)
z <- 1/sqrt(n)
points(x,dnorm(x, 4, z), type="l", lwd = 2, col="red")
```



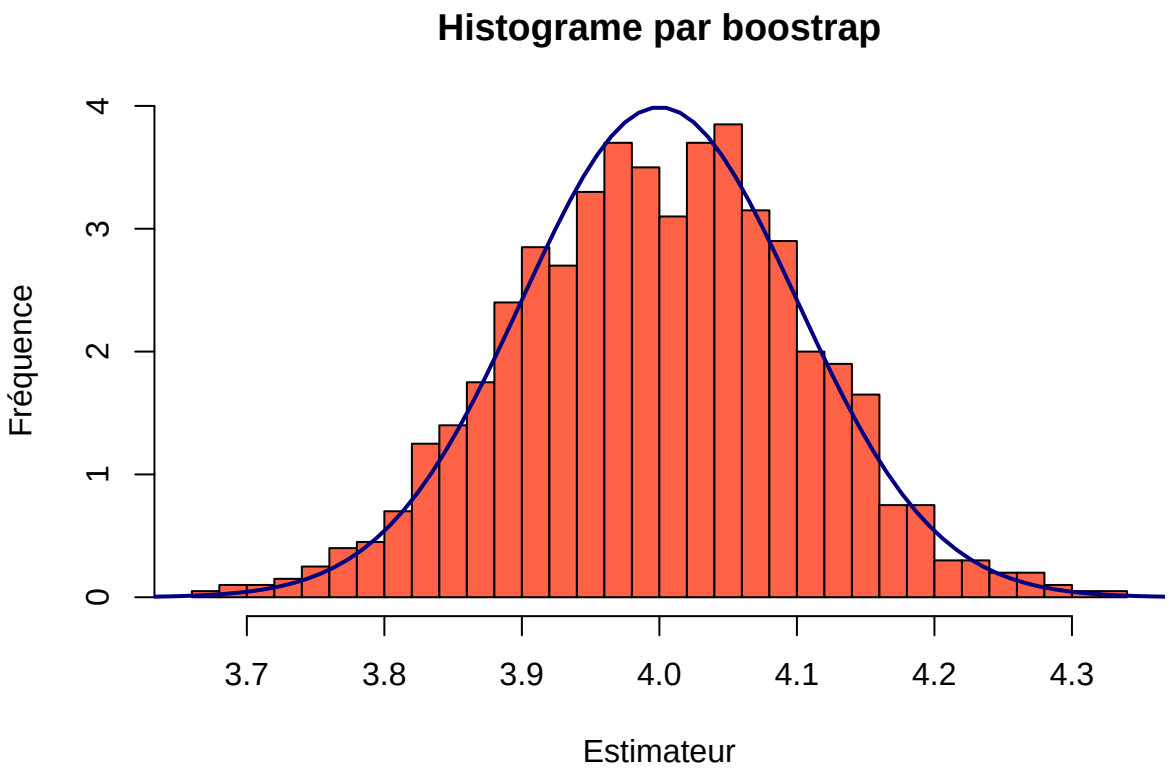
Question 9

On se place maintenant dans un cas plus réaliste, c'est à dire qu'on ne dispose pas de nombreuses observations pour notre variable aléatoire. On se limite alors à un échantillon de taille $n = 100$. La méthode bootsrtap se prête particulièrement à ce cas. On tire aléatoirement $n \times N$ avec remise dans notre ensemble de réalisation y. pour obtenir N échcantillons de taille n. On prend alors la moyenne pour chaque échantillon.

```
set.seed(11)
Y = y[sample(c(1:n),n*N,replace =T)] ## Replace =TRUE permet d'effectuer un tirage avec remise
Y_matrix = matrix(Y,byrow=TRUE, ncol = n) ## on a 1000 lignes avec 100 réalisations obtenues par tirage
theta_hat <- rowMeans(Y_matrix)
boxplot(theta_hat,col = 'mediumpurple1')
```




```
hist(theta_hat, ylab="Fréquence", xlab="Estimateur",main="Histograme par boostrap", col="tomato",breaks
x <-seq(3, 5, le=200)
z <- 1/sqrt(n)
points(x,dnorm(x, 4, z), type="l", lwd = 2, col="navy")
```



On peut remarquer que l'histograme obtenu par bootstrap est centré autour de 4.1 alors que le précédant l'est autour de 4. La méthode bootsrap a donc été assez précise et se rapproche d'une manière respectable de ce qu'on désirait. De plus on peut voir que la distribution des valeurs est quasiment symétrique. On remarque qu'il y a quelques valeurs qui dépasse le maximum dans les deux cas. Grâce aux lignes que l'on a tracé, on se rend facilement compte de l'écart ou de l'erreur de notre approximation. L'histogramme est légèrement décalé vers les valeurs plus hautes contrairement à celui du premier cas qui se confond pratiquement avec sa ligne.

On aurait pu faire les mêmes observations depuis le box-plot. On voit que les valeurs obtenues par bootstrap sont centrées autour de 4.1 et qu'on la distribution est symétrique. Il y a un peu mois de valeurs abhérentes pour le bootsrap que pour le premier cas.

On peut conclure que notre approximtion a été correcte et que le réchantillonage dit bootstrap peut palier aux problème de manque de données. On aurait aussi pu démontrer que le bootstraping était utile lorsqu'on on ne connait pas du tout la distribution des observations et peut permettre d'identifier des intervalles de confiance mais ce n'était pas là l'objet de l'exercice.

Autour de l'algorithme Adaboost

Question 10

$\hat{c}(x_*)$	$\hat{y}_*$	**Perte exponentielle** $exp(-y_*\hat{c}(x_*))$	**Perte binaire** $1(y_* \neq \hat{y}_*)$	y_*
0.3	1	1.35	1	-1
-0.2	-1	0.8187	0	-1
1.5	1	0.2231	0	1
-4.3	-1	73.7	1	1

Si y_* et $\hat{c}(x_*)$ sont de signe différents, c'est à dire $exp(-y_*\hat{c}(x_*)) > 1$, alors $1(y_* \neq \hat{y}_*) = 1$, et réciproquement. Donc en comparant $exp(-y_*\hat{c}(x_*))$ à 1 on connait entièrement $1(y_* \neq \hat{y}_*)$, c'est à dire les erreurs. Mais en plus, $exp(-y_*\hat{c}(x_*))$ nous dit à quel point on se trompe. On le voit à la dernière ligne du tableau, $exp(-y_*\hat{c}(x_*)) = 73.7$ alors que l'on se trompe de façon moins importante à la première ligne : $exp(-y_*\hat{c}(x_*)) = 1.35$, on n'est pas pas loin de détecter la bonne réponse. Cela peut être utilisé dans le cadre du boosting : à la prochaine étape, on va donner plus de poids aux individus ayant une plus grande perte exponentielle, pour corriger le modèle là où il fait le plus d'erreurs.

Question 11

Il faut prendre tree_depth = 2 car AdaBoost.M1 exécute à chaque étapes des règles de classifcation faible à partir d'arbres à deux feuilles. De plus n_rounds = 50, car l'énoncé nous demande 50 itérations d'entraînement.

```
ada.email = adaboost(as.matrix(email.tr[,-58]), ifelse(email.tr$Class == "Spam", 1, -1), tree_depth = 2
  control = NULL)
```

Et l'erreur de test :

```
pred.ada <- predict(ada.email, email.te)
mean(abs(ifelse(email.te$Class == "Spam", 1,0) - ifelse(pred.ada == 1, 1,0)))
```

```
## [1] 0.05478261
```

On fait donc une erreur de 6.5 %.

Question 12

Avec la library xgboost :

On reprend la classification d'email avec les mêmes données et la même partition. On opte pour un model de regression logistique binaire, en appliquant la règle de Bayes pour faire la prédiction. On peut ainsi évaluer l'erreur de test en fonction du nombre d'itérations dans le boosting :

```
erreurtest_xgboost = function(N)
{
  xgb.email = xgboost(data = as.matrix(email.tr[,-58]),
                      label = ifelse(email.tr$Class == "Spam", 1, 0),
                      objective = "binary:logistic",
```

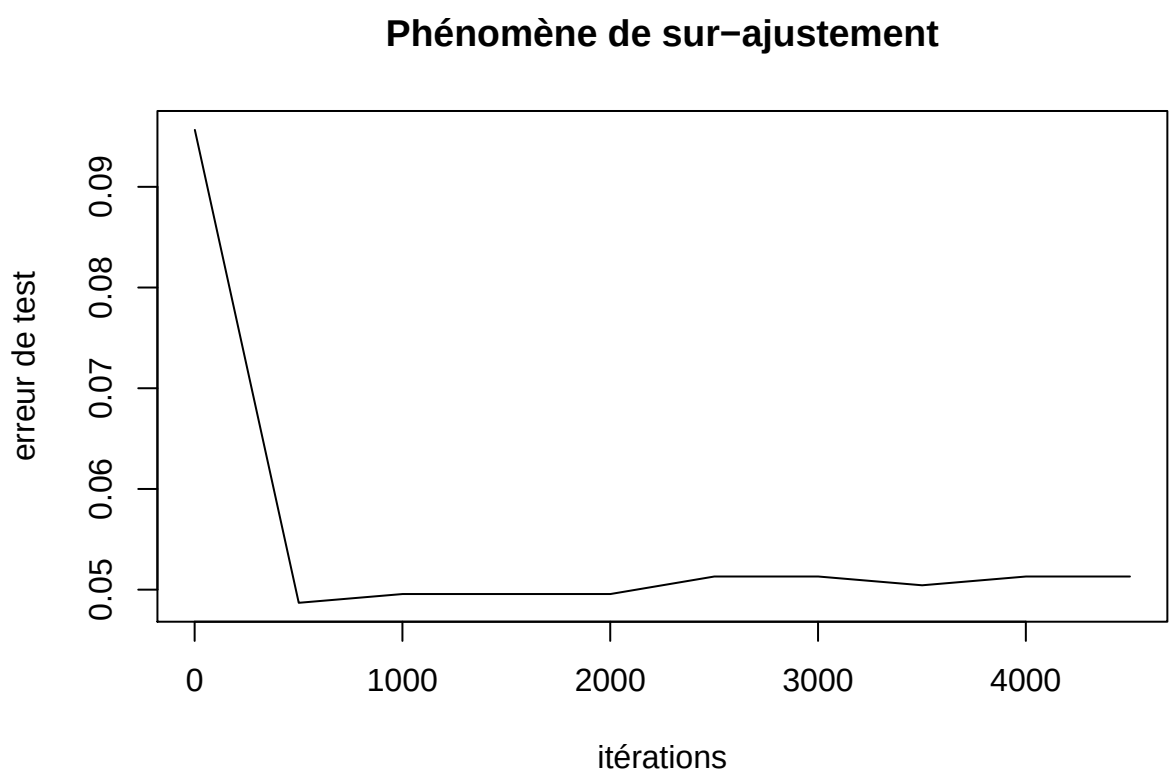
```
                    booster = "gbtree",
                    nrounds = N,
                    verbose = FALSE)

  pred.xgb <- predict(xgb.email, as.matrix(email.te[,-58]))
  pred.xgb <- ifelse(pred.xgb >.5, 1, 0)
  return (mean(abs(ifelse(email.te$Class == "Spam", 1,0) - ifelse(pred.xgb == 1, 1,0))))
}
```

On affiche ainsi l'erreur de test en fonction du nombre d'itérations :

```
surajustement = sapply(seq(from = 1, to = 5000, by = 500), erreurtest_xgboost)

plot(seq(from = 1, to = 5000, by = 500), surajustement,
     type = "l",
     xlab = "itérations",
     ylab = "erreur de test",
     main = "Phénomène de sur-ajustement")
```



On voit bien le phénomène de sur-ajustement qui apparaît, l'erreur de test augmente car on veut trop coller aux données.

Conclusion

Pour ce travail pratique, on a pu en particulier aborder des exercices de classification binaire pour deux jeux de données différents. On a d'abord pu comparer l'efficacité et l'intéprétabilité d'un arbre de décision avec et sans élagage. Ensuite on a confronté une méthode faisaint appel à un bagging pour 100 arbres de décision avec la méthodes des fôrets aléatoires. Puis, on a vu que les méthodes de bootstrap pouvaient être très précises lorsqu'on est dans une situation où on manque de données. Finalement, on a pu découvrir et se familisariser avec l'algorithme adaboost. Il présente lui aussi des avantages intéressants.