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

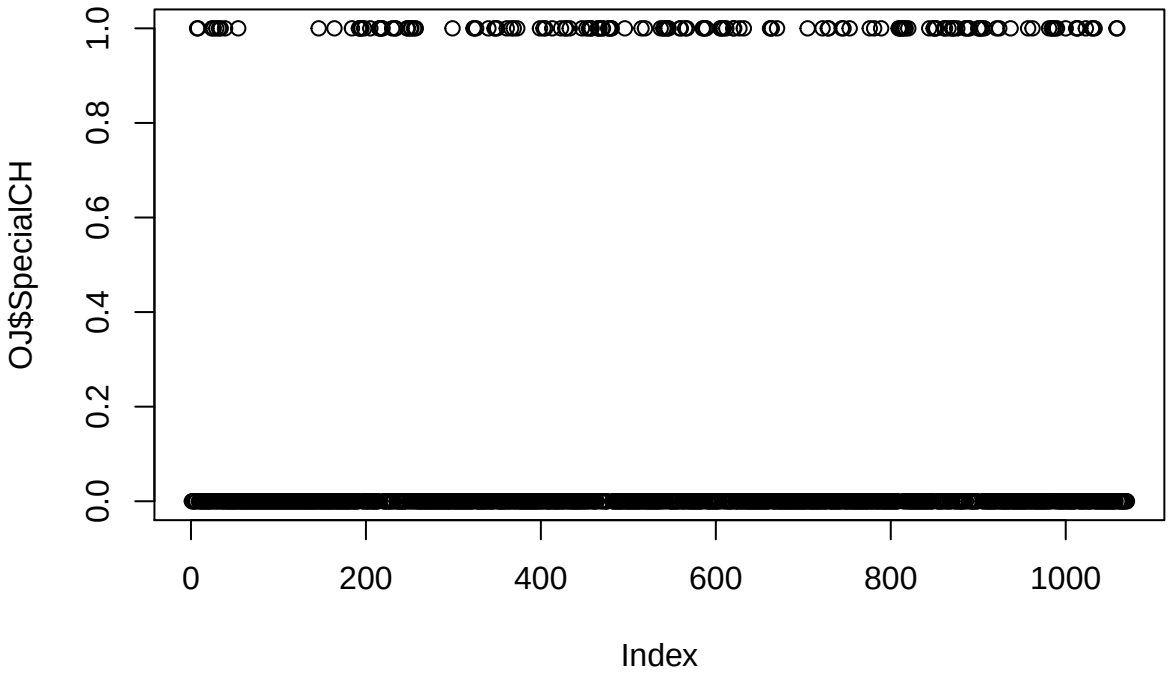
On procéde à une analyse univariée des variables. On se sert de la description des variables ainsi que des commandes `summary`, `plot` et `table`.

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
```

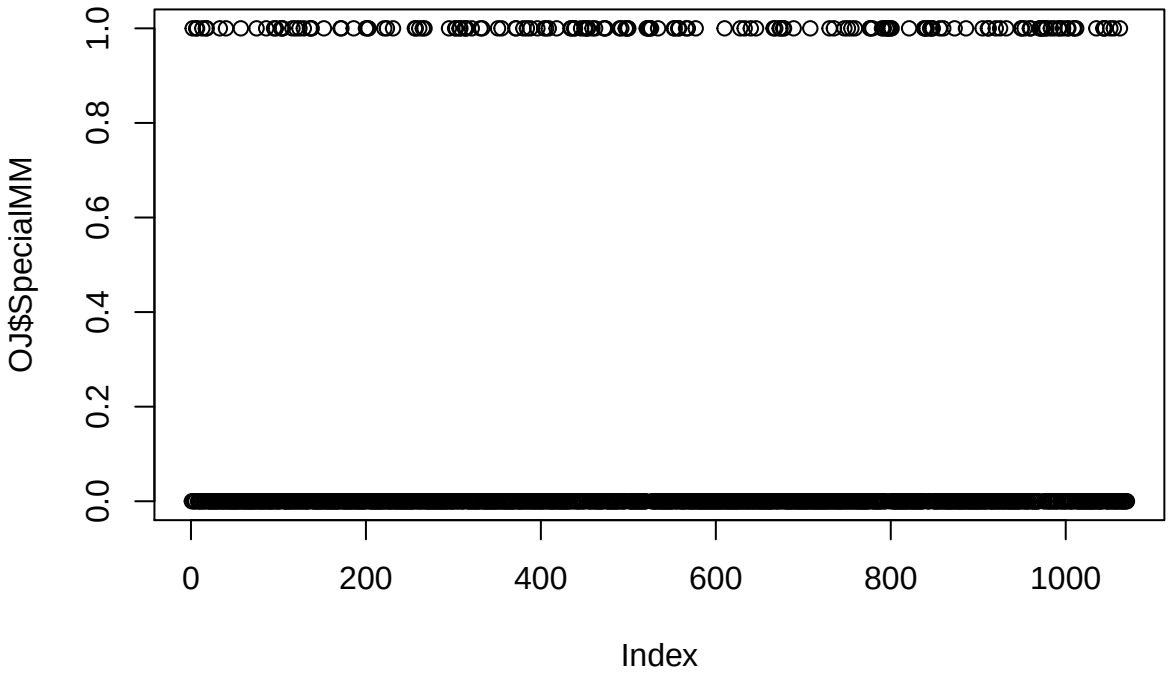
```
plot(OJ$SpecialCH)
```




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

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

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

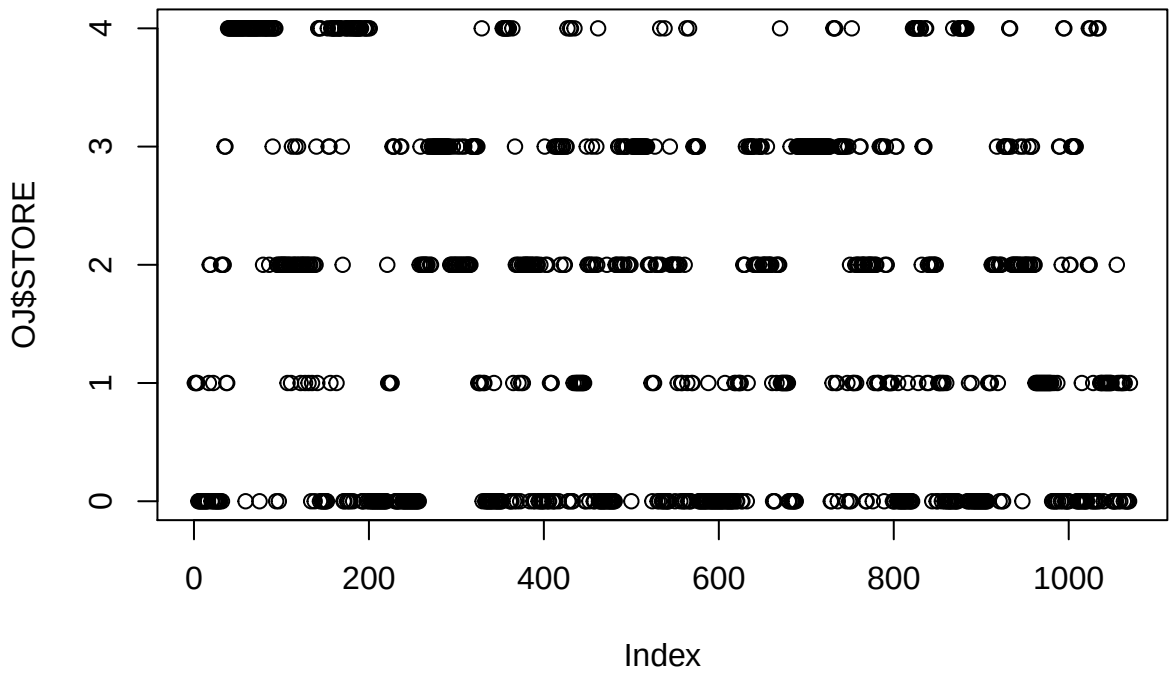


De la même manière `STORE` ne prend que les valeurs entre 0 et 4.

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

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

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



On préfère alors les transformer en variables catégorielles:

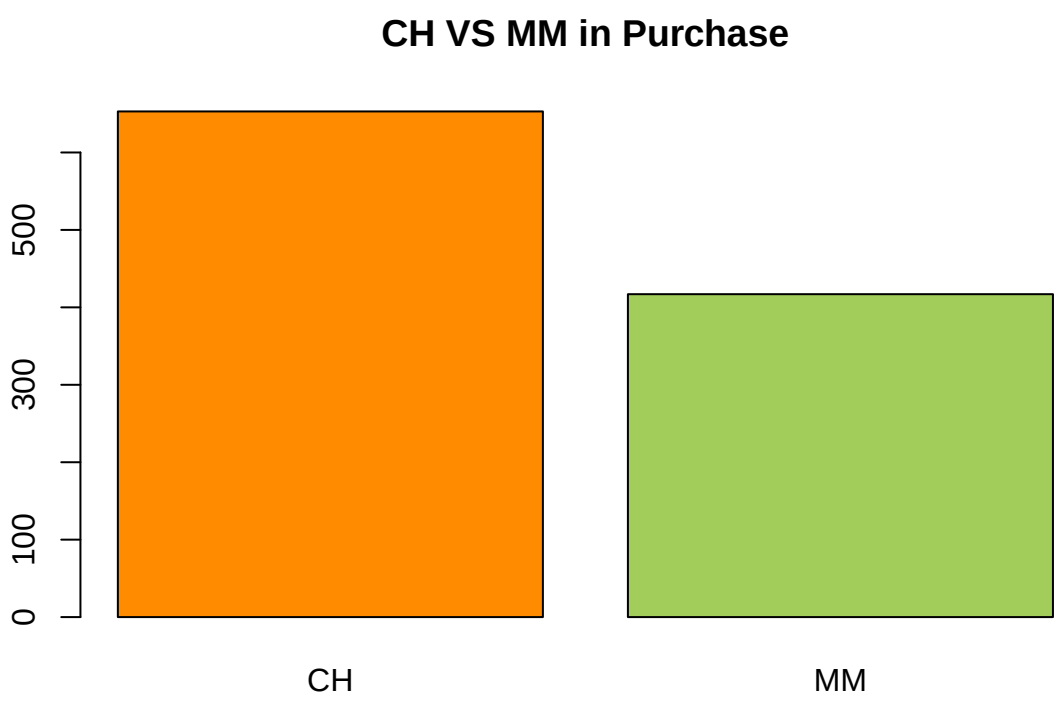
```
OJ$SpecialMM <- as.factor(OJ$SpecialMM)
OJ$SpecialCH <- as.factor(OJ$SpecialCH)
OJ$STORE <- as.factor(OJ$STORE+1) ## On préfère avoir des valeurs entre 1 et 5.
```

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)) ## list comportant l'index de la partie train
```

Question 2

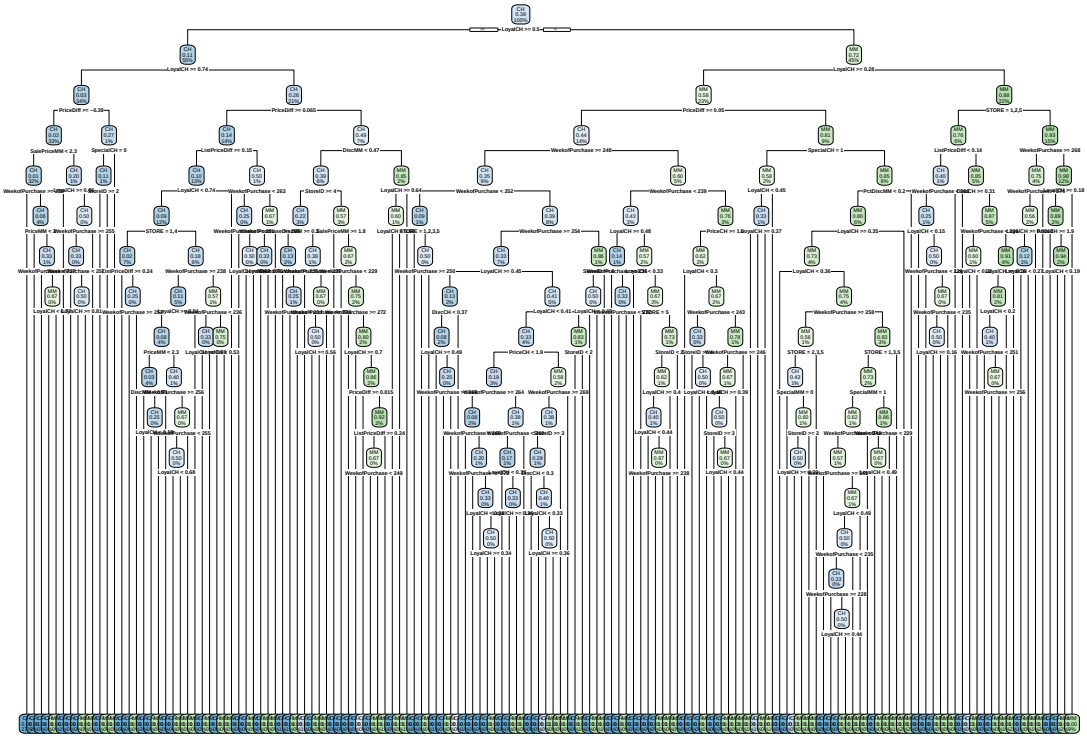
On va utiliser l'apprentissage par arbre de décision pour faire un prédction sur la partie test à partir de la partie train. On utilise d'baord un arbre obtenu sans élagage puis un autre avec élagage, on comparera ensuite la qualité des deux prédictions.

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

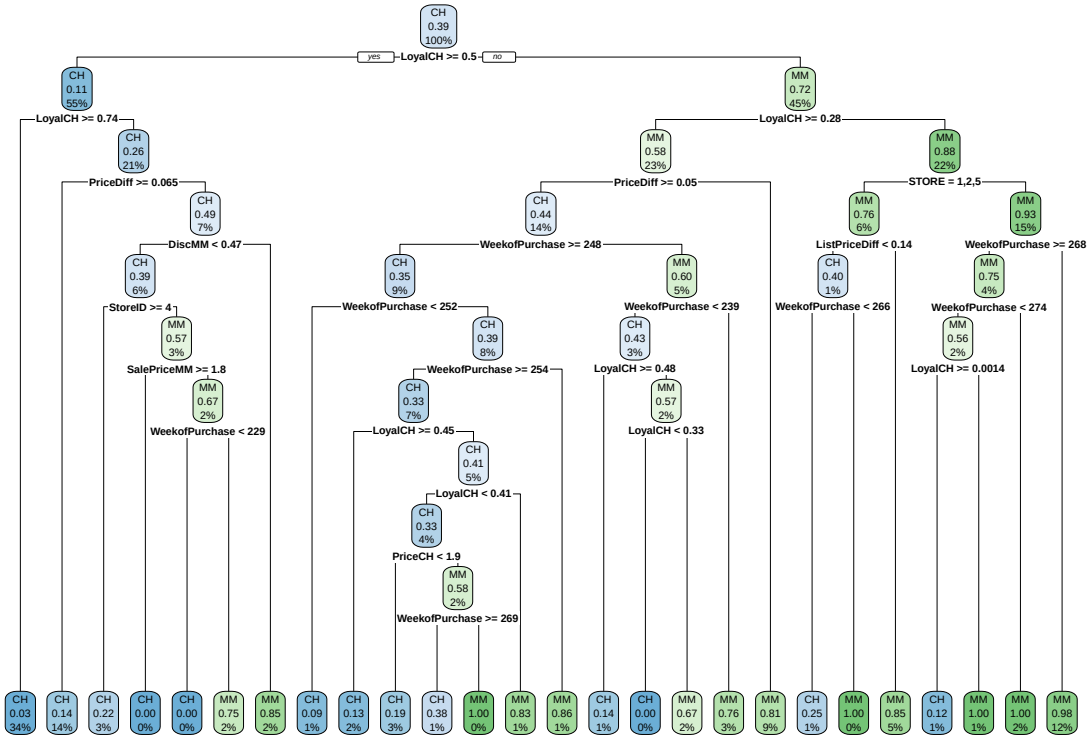
```
## Warning: package 'rpart.plot' was built under R version 3.6.3
```

```
## 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
```

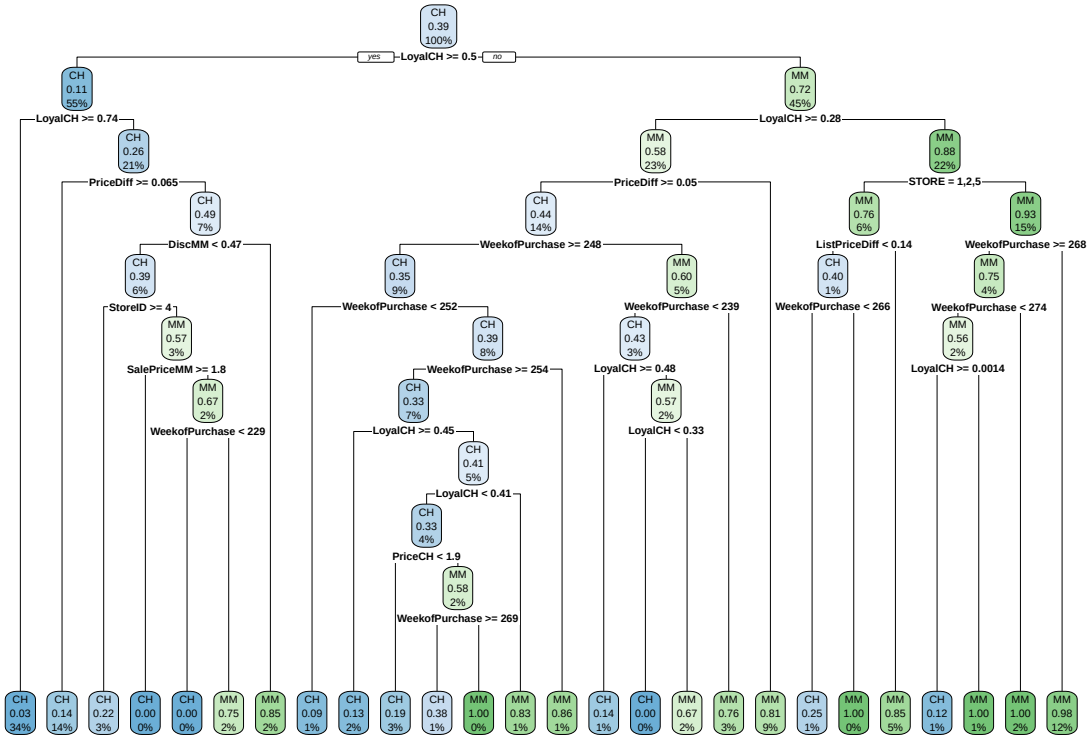
- Pour l'abre sans élagage :
  - 151 CH qui ont correctement été prédits.
  - 22 CH qui ont été incorrectement prédits.
  - 21 MM qui ont été incorrectement prédits.
  - 77 MM qui ont correctement été prédits. Globalement la prediction est assez bonne.

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

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

Question 3

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

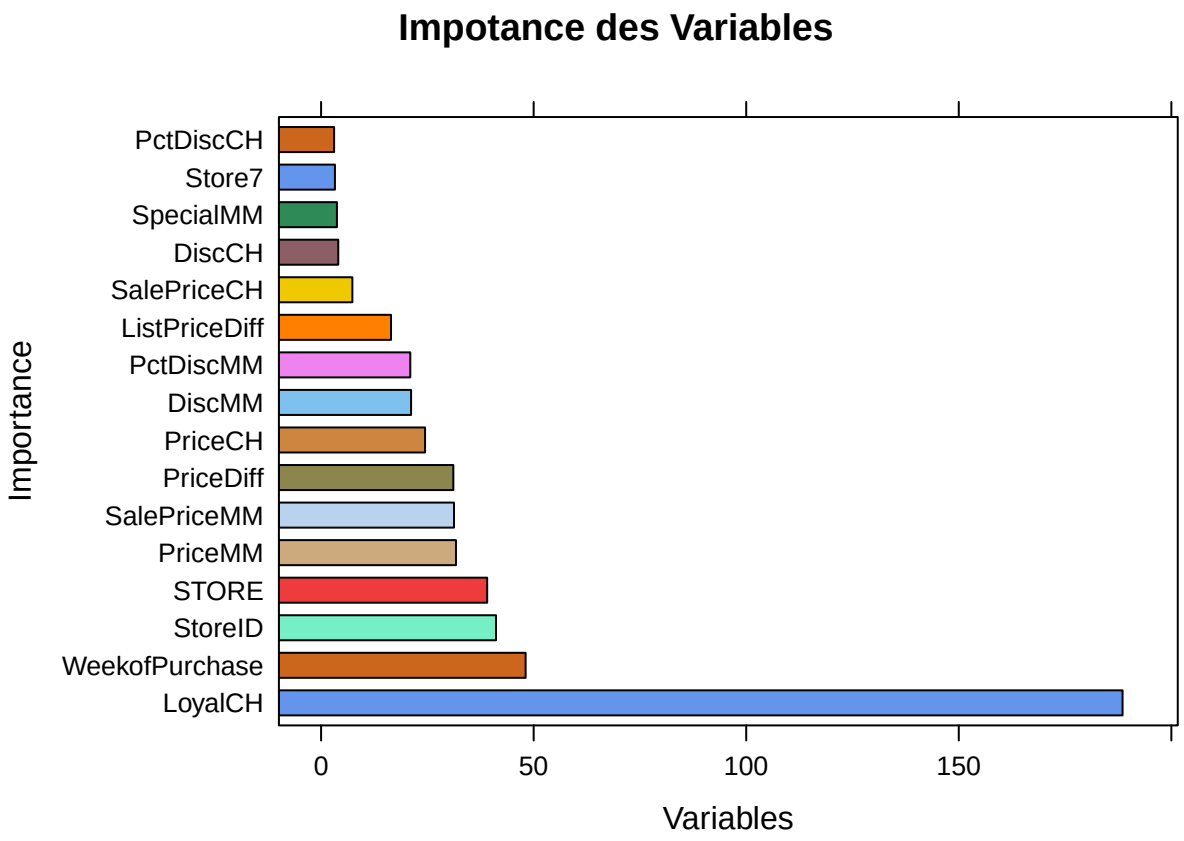


On peut observer l'importance relative des variables.

```
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)
```

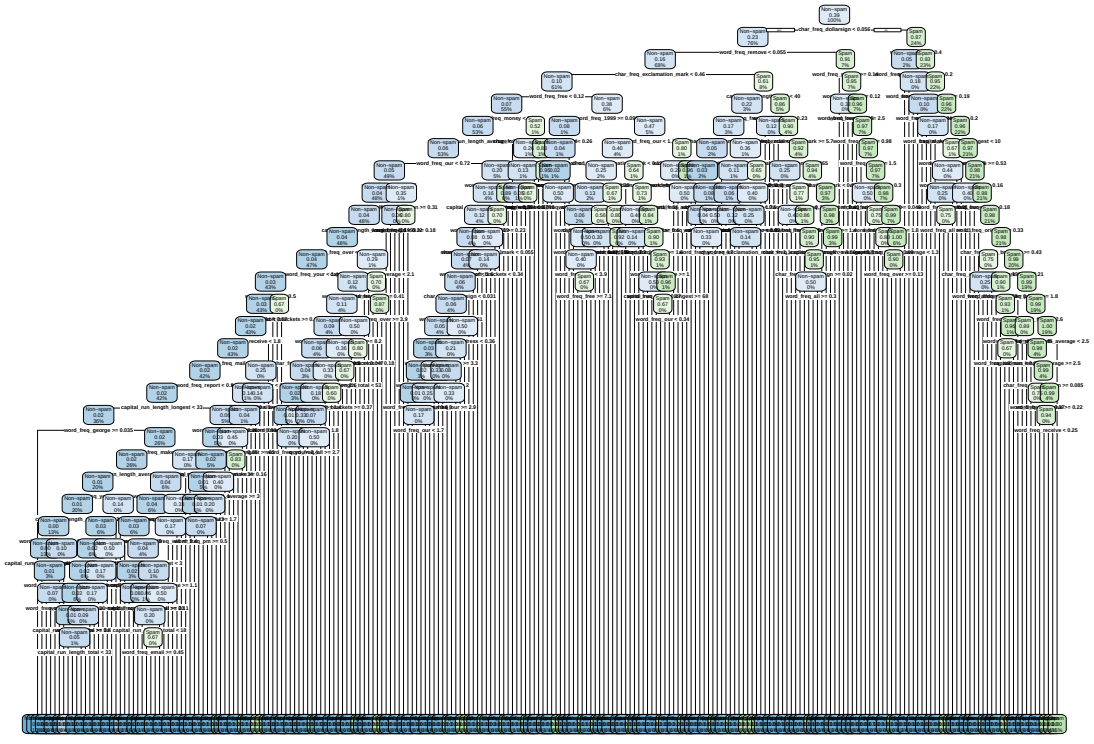
```
N = nrow(email)
set.seed(103)
train = sample(1:N, round(0.75*N))
email.tr = email[train,]
email.te = email[-train,]
```

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)
```

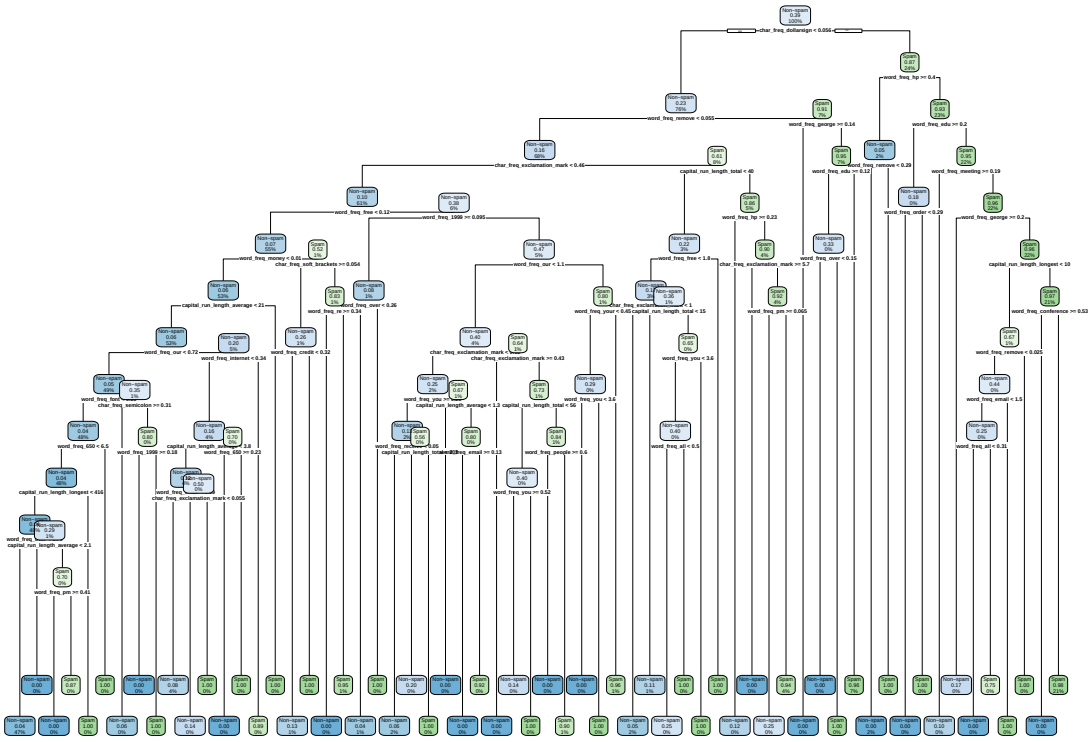
```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Puis élagons cette 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.

Pour

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
```

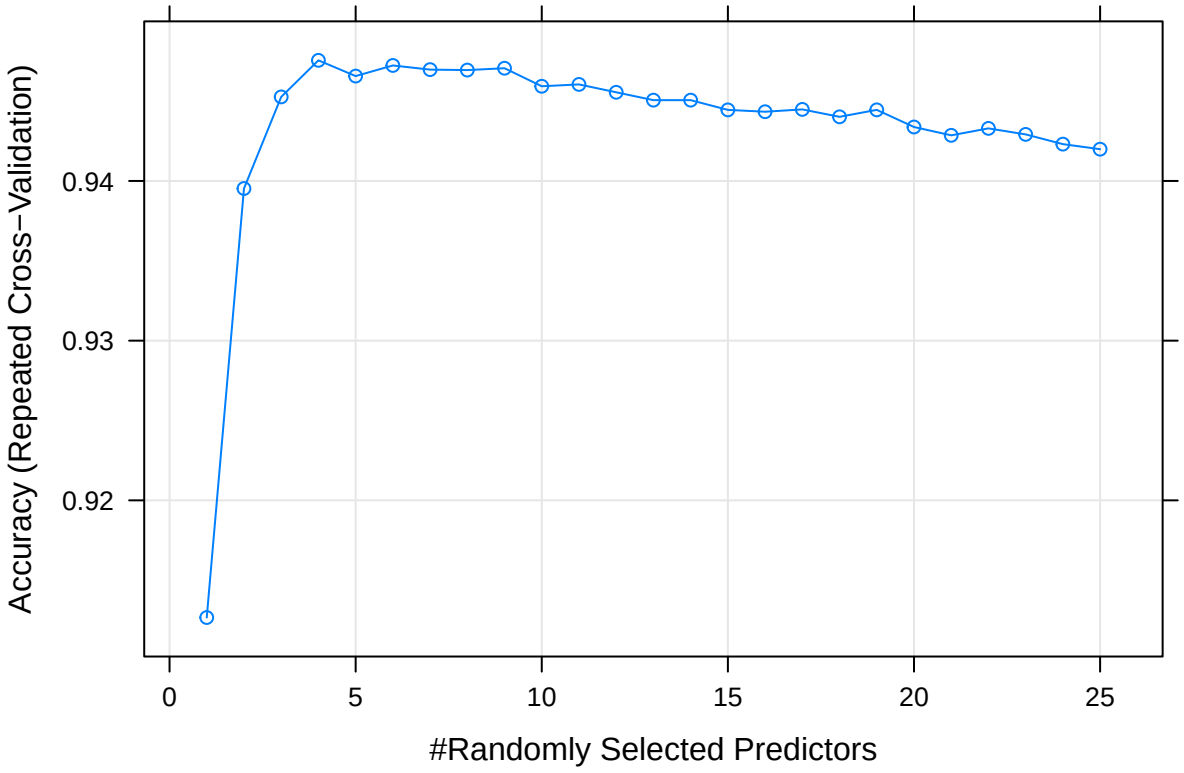
Nous avons donc une erreur de test de 7,5%.

Question 6

Enfin, nous ajustons un modèle random forrest 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 = 1:25)
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] 0.0426087
```

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 assez performant en terme d'erreur de test, et ne demande pas un trop grand temps de calcul. Néanmoins, 100 arbre 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 forrest, nous préfèrerons le bagging même s'il n'y a pas un grand gain de performance avec le modèle Cart.

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