STA 212: Partie théorique

Kahale Abdou Cadmos, Koen Aristote

5/10/2020

Partie I

Question 1

```
rm(list = objects())
df <- read.csv("oj.csv")
```

```
set.seed(123)
train_index <- sample(1:nrow(df), size = 800, replace = FALSE )
train <- df[train_index,]
test <- df[-train_index,]
library(corrplot)
```

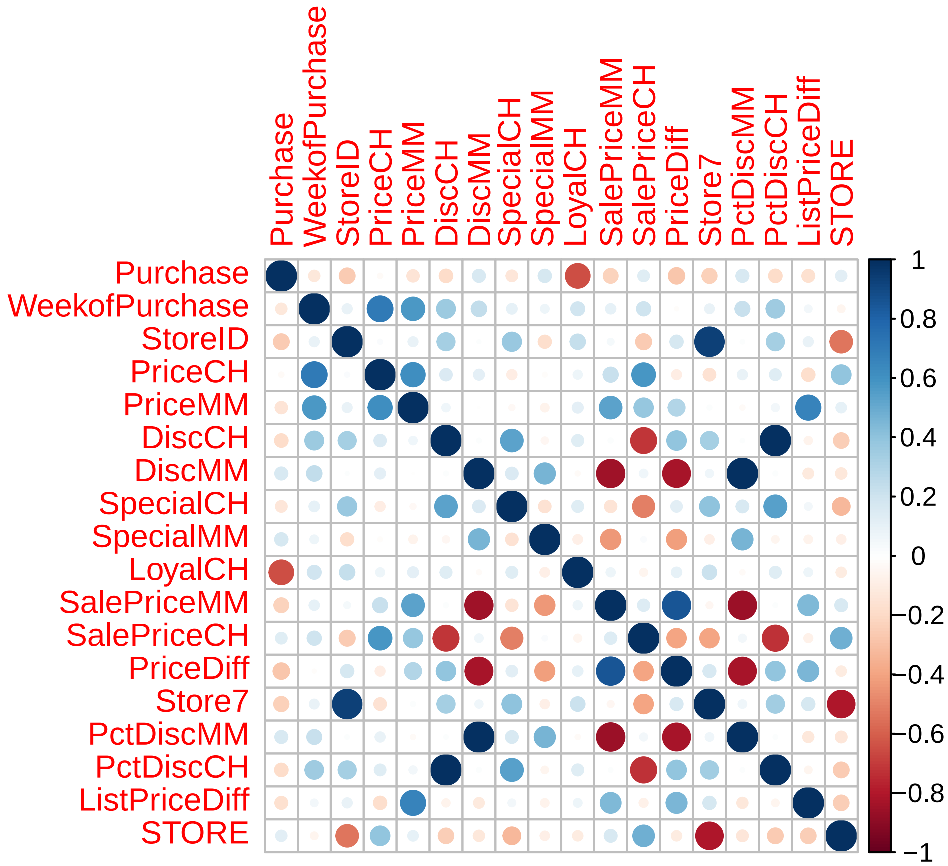
```
## corrplot 0.84 loaded
```

```
df_cor= df
df_cor$Purchase=as.numeric(df_cor$Purchase)
df_cor$Store7=as.numeric(df_cor$Store7)
summary(train)
```

```
##  Purchase WeekofPurchase     StoreID        PriceCH        PriceMM
##  CH:487   Min.   :227.0   Min.   :1.000   Min.   :1.69   Min.   :1.690
##  MM:313   1st Qu.:241.0   1st Qu.:2.000   1st Qu.:1.79   1st Qu.:2.090
##           Median :257.0   Median :3.000   Median :1.86   Median :2.090
##           Mean   :254.7   Mean   :3.936   Mean   :1.87   Mean   :2.086
##           3rd Qu.:269.0   3rd Qu.:7.000   3rd Qu.:1.99   3rd Qu.:2.180
##           Max.   :278.0   Max.   :7.000   Max.   :2.09   Max.   :2.290
##      DiscCH             DiscMM          SpecialCH        SpecialMM
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.00000   Median :0.0000   Median :0.0000   Median :0.0000
##  Mean   :0.05249   Mean   :0.1257   Mean   :0.1487   Mean   :0.1613
##  3rd Qu.:0.00000   3rd Qu.:0.2400   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :0.50000   Max.   :0.8000   Max.   :1.0000   Max.   :1.0000
##     LoyalCH          SalePriceMM     SalePriceCH      PriceDiff       Store7
##  Min.   :0.000011   Min.   :1.190   Min.   :1.390   Min.   :-0.6700   No :537
##  1st Qu.:0.320000   1st Qu.:1.690   1st Qu.:1.750   1st Qu.: 0.0000   Yes:263
##  Median :0.600000   Median :2.090   Median :1.860   Median : 0.2300
##  Mean   :0.565767   Mean   :1.961   Mean   :1.817   Mean   : 0.1435
##  3rd Qu.:0.843226   3rd Qu.:2.180   3rd Qu.:1.890   3rd Qu.: 0.3200
##  Max.   :0.999947   Max.   :2.290   Max.   :2.090   Max.   : 0.6400
```

```
##    PctDiscMM        PctDiscCH       ListPriceDiff        STORE
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.1400   1st Qu.:0.000
##  Median :0.0000   Median :0.0000   Median :0.2400   Median :2.000
##  Mean   :0.0604   Mean   :0.0277   Mean   :0.2167   Mean   :1.635
##  3rd Qu.:0.1183   3rd Qu.:0.0000   3rd Qu.:0.3000   3rd Qu.:3.000
##  Max.   :0.4020   Max.   :0.2527   Max.   :0.4400   Max.   :4.000
```

```r
corrplot(cor((df_cor)))
```



Question 2

```r
library(rpart)
library(rpart.plot)
library(caret)

#CART sans élagages
cart.0 <- rpart(Purchase~.,
                data = train,
                control = rpart.control(minsplit = 1, cp = 0, xval = 5))


pred.0 <- predict(cart.0, test, type = "class")
mean(test$Purchase!=pred.0) #[1] 0.256
```

```
## [1] 0.2555556
```

```r
#matrice de confusion
confusion_M.0 <- confusionMatrix(data = pred.0, reference = test$Purchase, dnn = c("Predicion.0", "test
```

```r
#matrice de confusion en proportions
proportion_confusion_M.0= confusion_M.0$table/(confusion_M.0$table[,1]+ confusion_M.0$table[,2])


## CART avec élagage sur le paramètre de complexité optimal
cart.pruned <- prune(cart.0, cp = cart.0$cptable[which.min(cart.0$cptable[,"xerror"]),"CP"])

#prediction
pred.pruned <- predict(cart.pruned, test,type="class")
#erreur
mean(pred.pruned!=test$Purchase) #0.17
```

```
## [1] 0.1740741
```

```r
#matrice de confusion
confusion_M.pruned <- confusionMatrix(data = pred.pruned, reference = test$Purchase, dnn = c("Predicion

#matrice de confusion en proportion
proportion_confusion_M.pruned=confusion_M.pruned$table/(confusion_M.pruned$table[,1]+confusion_M.pruned$
```

```r
confusion_M.0
```

```
## $positive
## [1] "CH"
##
## $table
##            test ref.
## Predicion.0  CH  MM
##          CH 133  36
##          MM  33  68
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.444444e-01   4.575156e-01   6.880527e-01   7.953827e-01   6.148148e-01
## AccuracyPValue  McnemarPValue
##   4.681851e-06   8.097321e-01
##
## $byClass
##          Sensitivity          Specificity       Pos Pred Value
##            0.8012048            0.6538462            0.7869822
##       Neg Pred Value            Precision               Recall
##            0.6732673            0.7869822            0.8012048
##                   F1           Prevalence       Detection Rate
##            0.7940299            0.6148148            0.4925926
## Detection Prevalence    Balanced Accuracy
##            0.6259259            0.7275255
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

```r
confusion_M.pruned
```

```
## $positive
## [1] "CH"
##
## $table
##                 test ref.
## Predicion.prune  CH  MM
##               CH 147  28
##               MM  19  76
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   8.259259e-01   6.264351e-01   7.753299e-01   8.692041e-01   6.148148e-01
## AccuracyPValue  McnemarPValue
##   4.018798e-14   2.432427e-01
##
## $byClass
##          Sensitivity          Specificity       Pos Pred Value
##            0.8855422            0.7307692            0.8400000
##       Neg Pred Value            Precision               Recall
##            0.8000000            0.8400000            0.8855422
##                   F1           Prevalence       Detection Rate
##            0.8621701            0.6148148            0.5444444
## Detection Prevalence    Balanced Accuracy
##            0.6481481            0.8081557
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
##
## attr(,"class")
## [1] "confusionMatrix"
```

```r
proportion_confusion_M.0 #meilleur classement de CH par le premier
```

```
##             test ref.
## Predicion.0        CH        MM
##          CH 0.7869822 0.2130178
##          MM 0.3267327 0.6732673
```

```r
proportion_confusion_M.pruned #meilleur classement de MM pour le deuxième
```

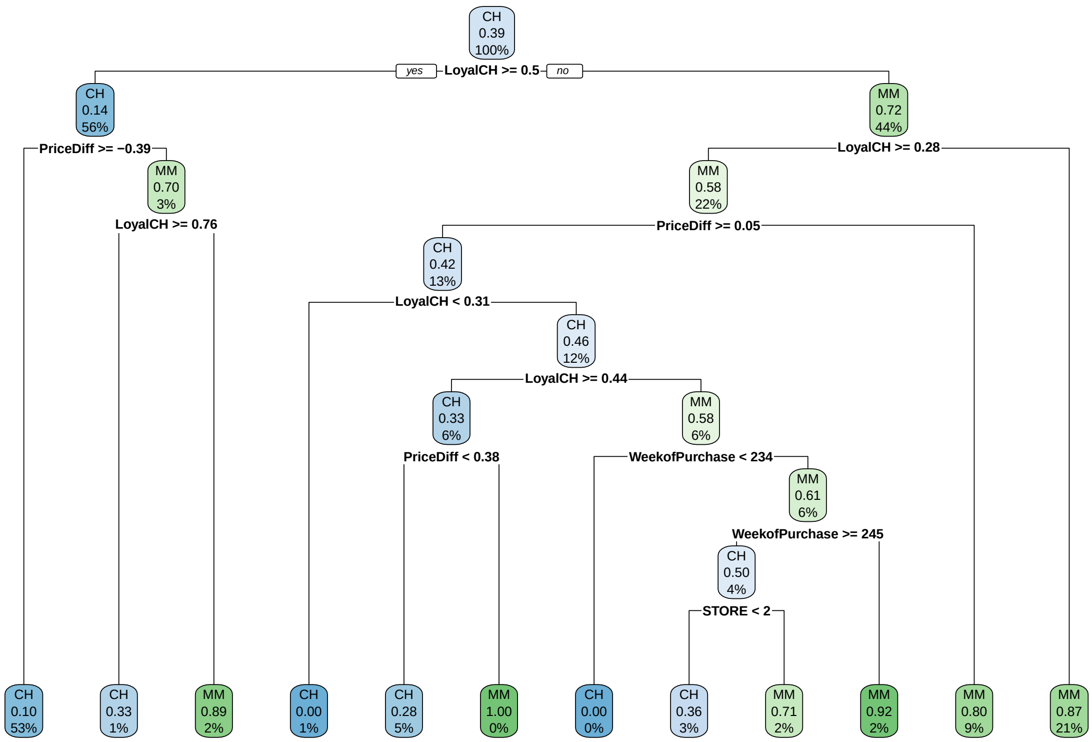
```
##                 test ref.
## Predicion.prune   CH   MM
##               CH 0.84 0.16
##               MM 0.20 0.80
```

On obtient une erreur de 0.26 sur l'arbre sans élagage vs 0.17 sur l'arbre avec elagage, donc une nette amélioration de la précision.

Question 3

Tracons donc l'arbre élagué

```
rpart.plot(cart.pruned)
```



L'arbre élagué est beaucoup plus court, on passe d'un large nombre de noeuds à moins de 10 noeuds. La diminution de l'erreur est du au fait que l'élagage a permis d'augmenter le biais mais de réduire la variance de l'arbre, permettant ainsi d'éviter le surapprentissage que l'on peut rencontrer lorsqu'on a un arbre trop profond.

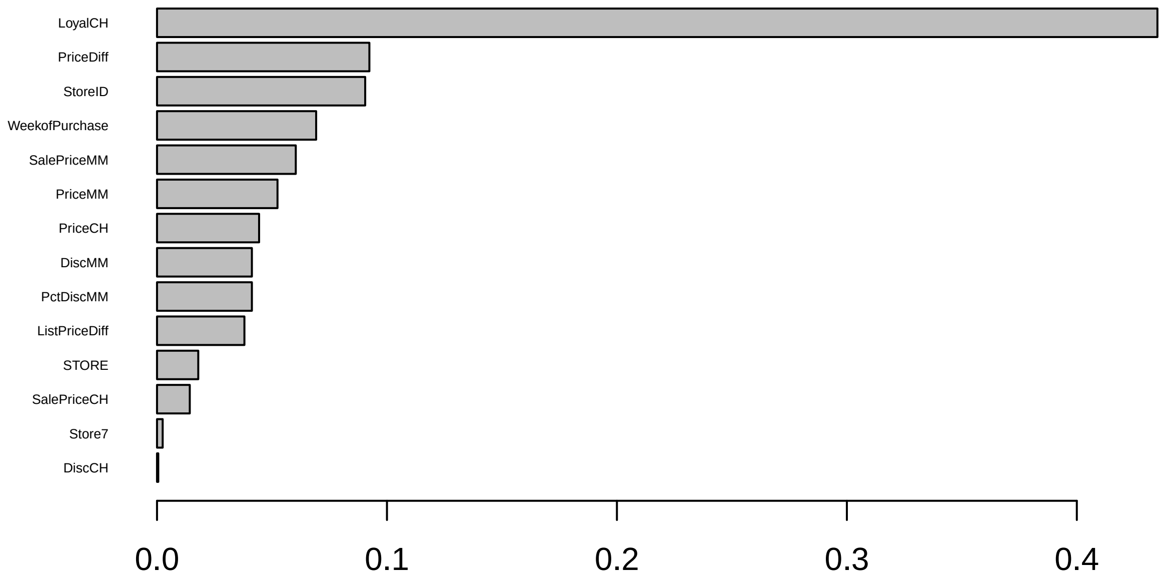
Les informations décrites sur la feuille (MM,0.80,9%) nous disent que la classe prédite pour les observations dans la région où 0.28<=LoyalCH<0.5. MM et PriceDiff>0.05 sont classées Minute Maid car c'est la classe majoritaire dans cette région. 0.80 nous donne la proportion d'observations du jeu d'apprentissage de classe MM dans la région. Enfin 9% signifie que 9% des observations du jeu de données d'apprentissage se trouve dans cette région.

Calculons l'importance relative des variables et traçons un barplot de l'importance des variables:

```
#importance relative
cart.pruned$variable.importance/sum(cart.pruned$variable.importance)
```

```
##        LoyalCH      PriceDiff        StoreID WeekofPurchase    SalePriceMM
##   0.4350504861   0.0923493758   0.0905354471   0.0692040921   0.0603407225
##        PriceMM        PriceCH         DiscMM      PctDiscMM  ListPriceDiff
##   0.0523999679   0.0443924690   0.0412588431   0.0412588431   0.0380141815
##          STORE    SalePriceCH         Store7         DiscCH
##   0.0179418457   0.0142457322   0.0024612160   0.0005467778
```

```
#plot de l'importance des variables
par(mfrow=c(1,1))
barplot(rev(cart.pruned$variable.importance/sum(cart.pruned$variable.importance)),cex.names=0.4,horiz =
```



La variable la plus importantes dans la détermination de la classe est LoyalCH. Mais PriceDiff, StoreID et WeekofPurchase ont aussi une importance élevée. Il est normal que le prix de CH et MM soient proche derrière sachant qu'ils sont fortement corrélé à PriceDiff. Ainsi c'est la loyauté du client à une marque qui va être le plus déterminant pour savoir si un client va acheter un des deux jus. De plus, l'endroit où le jus est vendu ainsi que la différence de prix entre les jus sera aussi déterminante. Enfin la semaine du mois où le client fait son achat aussi, en effet peut être qu'en fin de mois les clients acheterons le moins cher!

Partie 2: Forets Aléatoires

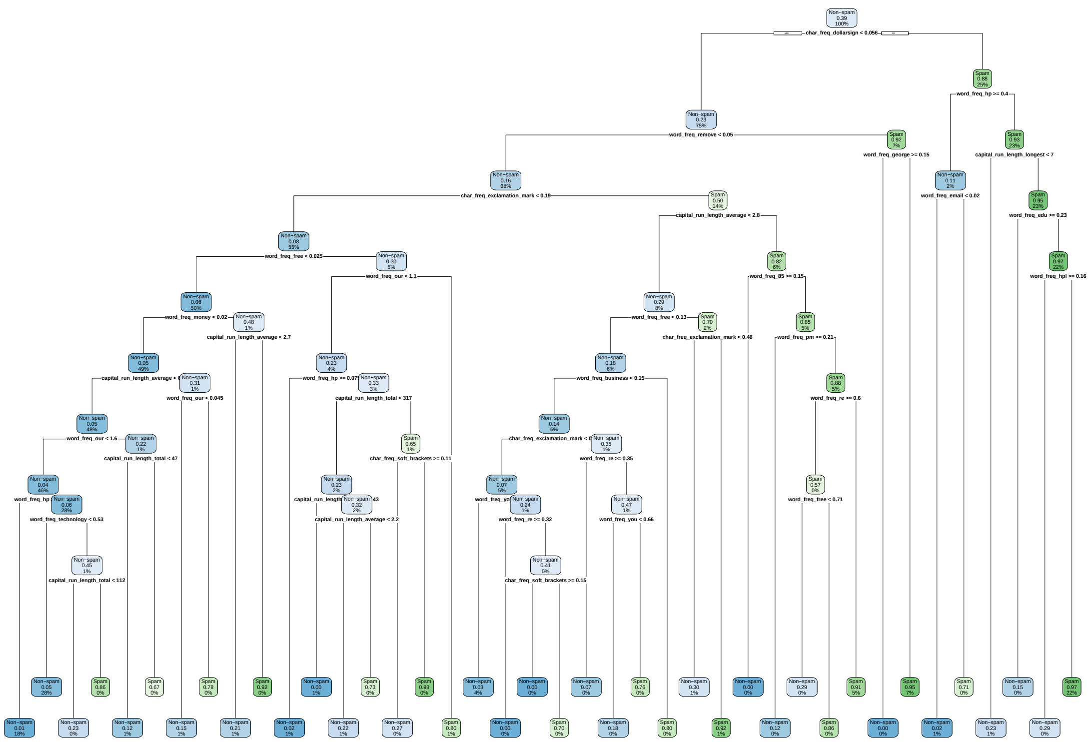
Questions 4,5,6,7

```
#chargement du jeu email
email <- read.csv("email.csv")
n<- length(email$word_freq_make)
sample_id <- sample(1:nrow(email), size = 0.75*n, replace = FALSE)

#séparation en test et train
email_train <- email[sample_id,]
email_test <- email[-sample_id,]

## arbre sans élagage
email.tree= rpart(Class~.,data=email_train,control=c(minsplit=1,cp=0,xval=5))

#paramètre de complexité optimal
cpopt= which.min(email.tree$cptable[,"xerror"])

#prediction et erreur
pred.email.tree= predict(email.tree,email_test,type='class')
mean(pred.email.tree!=email_test$Class) #[1] 0.08861859
```

```
## [1] 0.0894874
```

```
## arbre élagué
email.pruned=prune(email.tree,cp=email.tree$cptable[which.min(email.tree$cptable[,"xerror"]),"CP"])
```

```r
rpart.plot(email.pruned)
```



```r
#prediction et erreur
pred.email.pruned= predict(email.pruned,email_test,type='class')
mean(pred.email.pruned!=email_test$Class) #[1] 0.07732407
```

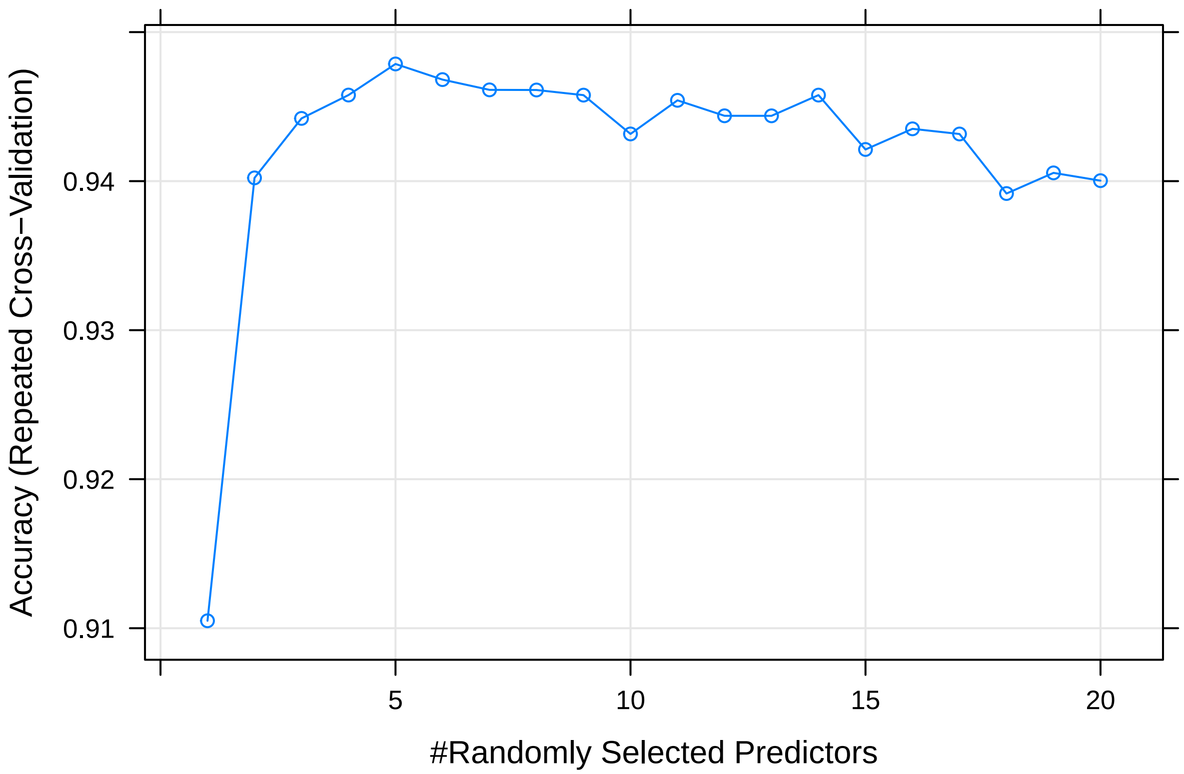
```
## [1] 0.08774978
```

```r
#On obtient un erreur de test de 7,8% pour l'arbre élagué contre 8.9% pour l'arbre sans élagage.


## Bagging 100 arbres:
B=100
predictions_bag = matrix(NA,nrow=nrow(email_test),ncol= B)
predictions_bag= as.data.frame(predictions_bag)
err_bootstrap = list()
n_train= nrow(email_train)
for(i in 1:100)
  {
    sample_index = sample(1:n_train,size=n_train,replace=TRUE)
    bootstrap= email_train[sample_index,]
    rpart_boot= rpart(Class~.,data = bootstrap,control=c(minsplit=1,cp=0,xval=5))
    pred_bootstrap= predict(rpart_boot,email_test,type='class')
    predictions_bag[,i]= (pred_bootstrap)
    err_bootstrap[[i]]= mean(pred_bootstrap!=email_test$Class)
}
train_bootstrapped = predictions_bag

#prediction bagging via seuillage sur la classe majoritaire
pred.bootstrapped= train_bootstrapped==c("Spam")
probas =apply(pred.bootstrapped,1,mean)
```

```
probas[probas>0.5]= "Spam"
probas[probas<=0.5]= "Non-spam"
#erreur bagging
mean((probas)!=email_test$Class) #[1] 0.06863597
```

```
## [1] 0.06516073
```

```
##On a réussi à diminuer l'erreur de prédiction grace au bootstrap


## Random forest
library(doParallel)
set.seed(123)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)
control <- trainControl(method="repeatedcv", number=5, repeats=5)
rfGrid <-  expand.grid(mtry = 1:20)
RFmodel <- train(Class~., data=email_test, method="rf",
                 trControl=control,
                 ntree=100,tuneGrid=rfGrid,
                 verbose=FALSE)
stopCluster(cl)

#plot de la précision en fonction du nombre de variables
#considérées pour les splits
plot(RFmodel)
```




```
#meilleur modèle obtenu par validation croisée sur mtry
RFmodel$bestTune #mtry=5
```

```
##   mtry
## 5    5
```

```r
#prediction
pred.rf.caret <- predict(RFmodel, email_test)

#erreur
mean(pred.rf.caret!=email_test$Class)
```

```
## [1] 0.006950478
```

Pour le bagging nous avons tiré 100 jeux de données de la même taille que le jeu d'apprentissage à partir du jeu d'apprentissage en tirant les observations par tirage avec remise. Ensuite pour chaque jeu nous ajustons un arbre sans élagage puis calculons la prediction Nous obtenons donc 100 prédictions pour le jeu test. Chaque observation test est classée en fonction de la classe majoritaire déterminée dans les 100 echantillons. On calcule ensuite l'erreur.

Pour la forêt aléatoire on obtient par validation croisée le nombre optimal de variables à sélectionner pour effectuer les branchments sur les arbres de la forêt qui est égal à mtry=5.

Voici les résultats des méthodes :

	Arbre simple	Arbre élagué	Bootstrap	Random Forest
Erreur	0.089	0.087	0.065	0.007

Moins de 1% d'erreur avec la forêt aléatoire. Ainsi le plus mauvais modèle est celui par arbre non élagué puis celui par arbre élagagué, puis le bagging sur 100 arbres puis la random forest qui est bien plus précise que toutes les autres méthodes.

Partie 3: Bootstrap

Les $\varepsilon \sim \mathcal{N}(0,1)$ on a que $Y_i \sim \mathcal{N}(\theta,1)$. Ainsi la vraisemblance vaut:

$$L(\theta;Y) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(y_i-\theta)^2}{2}\right) \implies \log(L(\theta,Y)) = -n\log(\sqrt{2\pi}) - \frac{1}{2}\sum_{i=1}^{n}(y_i-\theta)^2$$

En dérivant la log vraisemblance on a:

$$\frac{\delta}{\delta\theta}\log(L(\theta,Y)) = \sum_{i=1}^{n}(y_i-\theta) \text{ qui s'annule en } \hat{\theta} = \bar{Y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

La dérivée seconde étant négative, $\hat{\theta}$ est l'EMV. De plus on a alors que $\hat{\theta} \sim \mathcal{N}(4, \frac{1}{n})$

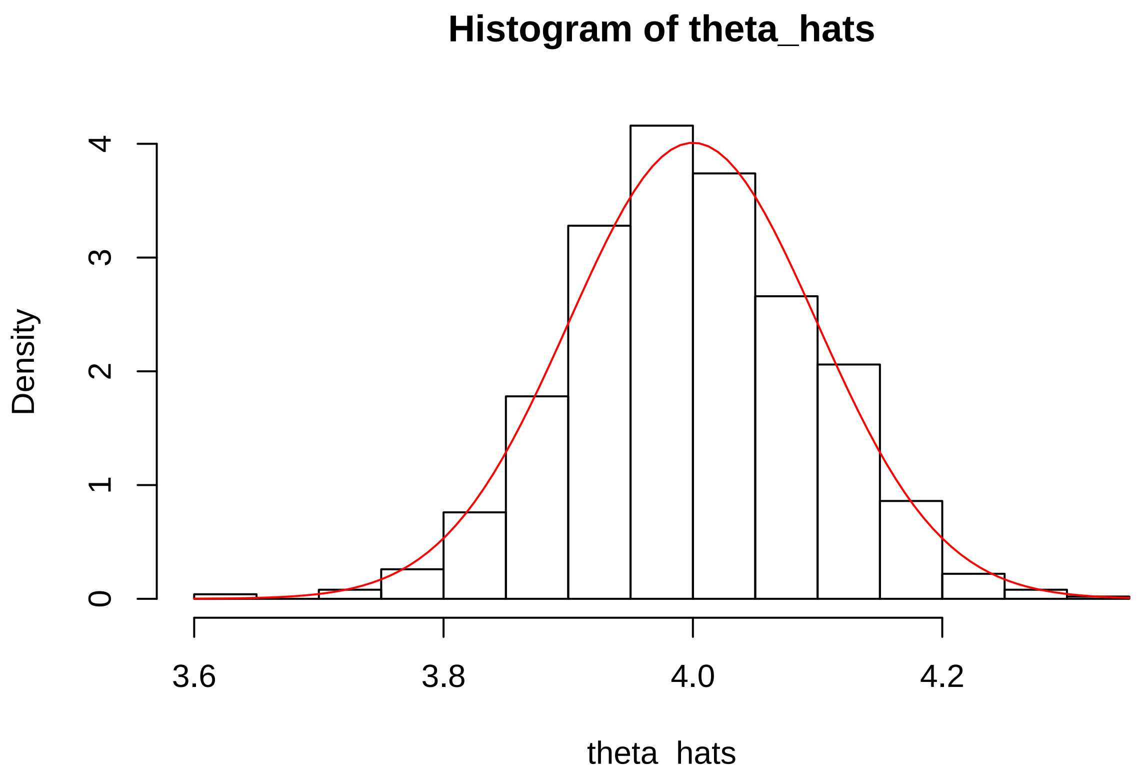
```r
B=1000
n = 100

#on génère un echantillon iid de taille 100
#N(4,1)
Y = rnorm(100,4,1)

#on génère B échantillons iid
echants = replicate(B,rnorm(n,4,1))
theta_hats=apply(echants,2,mean)

#Calcul de la moyenne et variance empirique
mean(theta_hats) # on retrouve bien environ 4 comme moyenne
```

```
## [1] 3.998864
```

```r
var(theta_hats)*((n-1)/n) # 1/100 pour la variance donc ok
```

```
## [1] 0.009558473
```

```r
#histogramme des B theta_hats
hist(theta_hats,proba=T)
curve(dnorm(x,4,sqrt(0.01*((n-1)/n))),add=T,col='red') #notre estimateur suit en effet bien la loi N(4,
```




```r
shapiro.test(theta_hats)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  theta_hats
## W = 0.99748, p-value = 0.1258
```

```r
#pvalue > 0.05 on conserve l'hypothèse que les données
#sont réparties normalement.
```

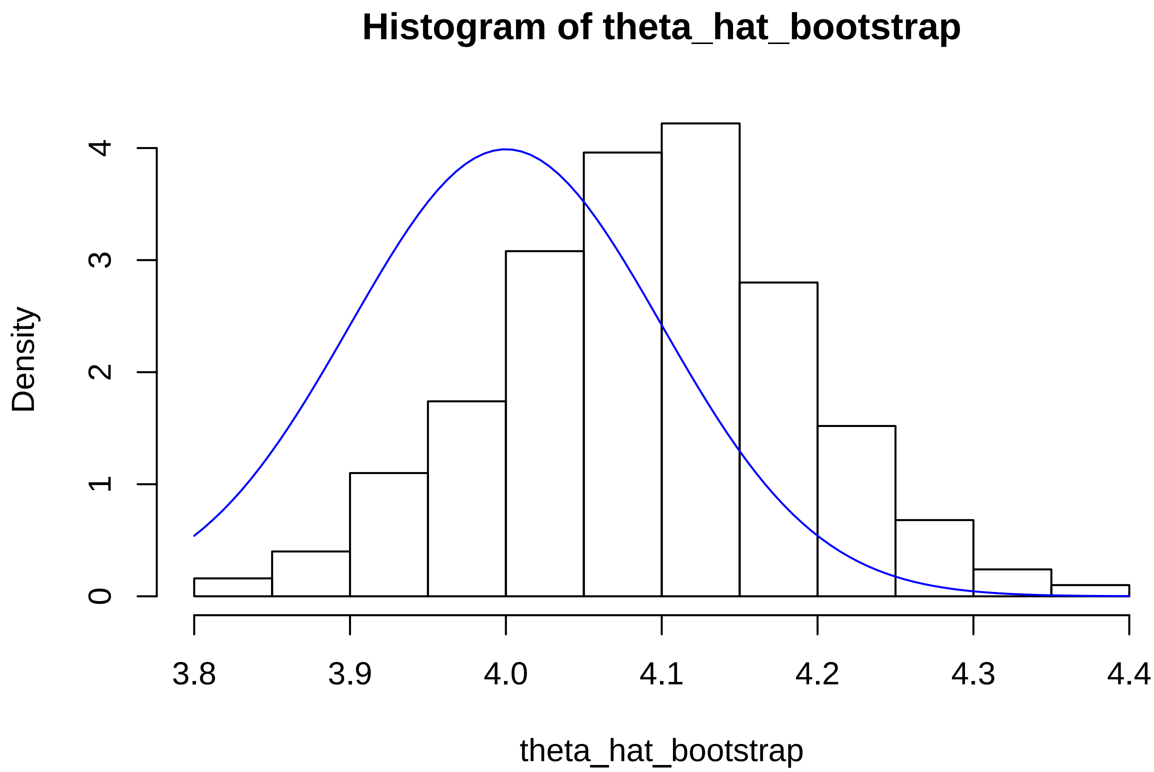
```r
set.seed(123)
#1000 echantillons de taille 100
#tirés avec remise sur le jeu d'apprentissage
echants_bootstrap= replicate(B,sample(Y,size = n,replace = TRUE))

#moyenne de chaque echantillon= vecteur des estimateurs
theta_hat_bootstrap= apply(echants_bootstrap,2,mean)

#moyenne sur les estimateurs
mean(theta_hat_bootstrap)
```

```
## [1] 4.091661
```

```r
#on est moins proche de la valeur
#théorique de la moyenne de theta =4 qu'en ayant 1000 echantillons
#iid simulés selon une n(4,1) mais on reste tout de même proche de 4

 #De même pour la variance
var(theta_hat_bootstrap)*((n-1)/n)
```

```
## [1] 0.009419671
```

```r
hist(theta_hat_bootstrap,proba=T)
curve(dnorm(x,4,sqrt(0.01)),add=T,col='blue')
```



On observe un plus grand biais par rapport à la moyenne théorique de l'estimateur de theta à travers cette méthode, en effet l'histogramme est centré autour de 4.091 vs 3.9989 précedemment. On observe que la méthode permet de bien estimer la variance de l'estimateur qui reste très proche de sa valeur théorique de 0.01 et ne varie pas beaucoup par rapport à la première simulation.

Partie 4: Boosting et Gradient Boosting

Question 10

```r
#On complète le tableau donné dans le sujet dans le dataframe:
df=data.frame("c_hat"=c(0.3,-0.2,1.5,-4.3),"y_hat"=rep(NA,4),
              "perte_exp"=rep(NA,4),"perte_binaire"=rep(NA,4)
              ,"y_star" = c(-1,-1,1,1))
```

```
df$y_hat= as.numeric(df$c_hat>=0) - as.numeric(df$c_hat<0)
df$perte_exp = exp(-df$y_star*df$c_hat)
df$perte_binaire = as.numeric(df$y_hat!=df$y_star)
df
```

```
##   c_hat y_hat  perte_exp perte_binaire y_star
## 1   0.3     1  1.3498588             1     -1
## 2  -0.2    -1  0.8187308             0     -1
## 3   1.5     1  0.2231302             0      1
## 4  -4.3    -1 73.6997937             1      1
```

La perte exponentielle est plus informative sur l'écart entre c_hat et 0. En effet la classification -1,1 est établie par rapport au signe de $\hat{c}$, ainsi si la classification est correcte, c'est à dire $\hat{c}y^* >= 0$, plus $\hat{c}$ sera très positif dans le cas où $y^* = 1$ plus la perte exponentielle prendra des valeurs faibles et de manière similaire dans les cas où $y^* = -1$, plus $\hat{c}$ prendra des valeurs très négatives, plus la quantité $\hat{c}y^*$ sera grande et donc la perte sera plus faible. Dans le cas où la classification n'est pas correcte, c'est à dire $\hat{c}y^* < 0$ plus $\hat{c}(x)$ est très éloigné de 0, c'est à dire $-\hat{c}(x)y^* >> 0$, et donc plus grande sera la perte exponentielle. Ainsi la perte exponentielle peut être plus informative sur la significativité de la classification. En effet, si on est proche de zéro pour $\hat{c}(x)$ on est moins certain que la classification est correcte à partir des données vu que la classification binaire se fait sur le signe.

De plus l'avantage de cette fonction de perte est sa convexité en $\hat{c}(x)$.

Cette information peut donc être utilisée pour affecter les nouveaux poids aux observations mal prédites. Celles qui ont été mal prédites avec une grande certitude se verront donner un plus grand poids.

On le voit bien sur le tableau ci dessus. Pour $\hat{c}(x) = -4.3$ alors que $y^* = 1$ on a une perte de 73 tandis que pour $\hat{c}(x) = 0.3$ alors que $y^* = -1$ la perte est de 1.34.

Question 11

```
###Adaboost
library(ada)

?ada

##algorithme original adaboost
ada.0 <- ada(Class~., data = email_train, loss = "exponential",
             iter = 50,type="discrete",bag.frac=0,nu=1)

print(ada.0)
```

```
## Call:
## ada(Class ~ ., data = email_train, loss = "exponential", iter = 50,
##     type = "discrete", bag.frac = 0, nu = 1)
##
## Loss: exponential Method: discrete   Iteration: 50
##
## Final Confusion Matrix for Data:
##           Final Prediction
## True value Non-spam Spam
##   Non-spam     2090    1
##   Spam            1 1358
##
```

```
## Train Error: 0.001
##
## Out-Of-Bag Error:  0  iteration= 6
##
## Additional Estimates of number of iterations:
##
## train.err1 train.kap1
##         23         25
```

```r
#prediction
pred.ada.0 <- predict(ada.0, email_test)

#erreur
mean(email_test$Class!=pred.ada.0)
```

```
## [1] 0.04344049
```

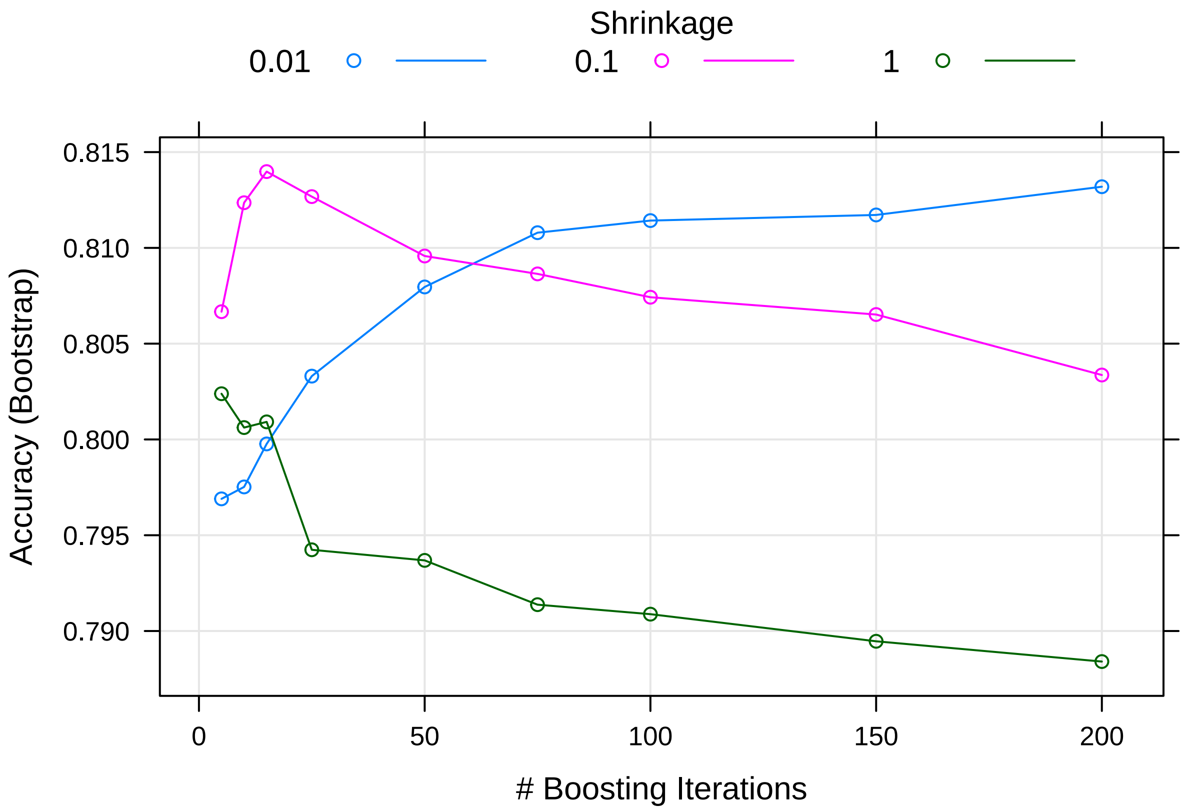
Ici loss= exponential correspond à la fonction de perte $\ell(Y, g) = \exp(-yg)$ pour laquelle on souhaite minimiser la somme sur les différents classifieurs.$nu = 1$ correspond à l'initialisation du pas de la descente de gradient et pour 1, cela correspond quasiment à Adaboost.M1. Bag.frac est un paramètre de bootstrap qui améliore la performance de l'algorithme et que nous n'utiliserons pas car pas utilisé dans l'algorithme original.

On obtient une erreur de l'ordre de 4% avec Adaboost.

Question 12

Prenons le jeu de données oj qui a moins d'observations. Si on fixe λ et qu'on essaye d'ajuster une règle de classification par xgboost pour différents nombres d'itérations:

```r
library(caret)
library(xgboost)

#validation croisée répetée sur les hyperparamètres M,lambda
control=trainControl(method = "repeatedcv", number =5,repeats=3)
grid = expand.grid(nrounds=c(5,10,15,25,50,75,100,150,200),eta=c(0.01,0.1,1),gamma=0,max_depth = 4,subsa

#apprentissage par xgboost
xgboost.oj <- train(Purchase~., data = train,method="xgbTree",tuneGrid=grid)

#illustration de la relation entre M et lambda
par(mfrow=c(1,1))
plot(xgboost.oj)
```



Dans le plot ci dessus nous avons entrainé un modèle pour le jeu de données oj qui contient moins de données afin d'avoir un temps de calcul rapide avec xgboost sur plusieurs valeurs du nombre d'itérations et plusieurs valeurs de lambda 0.01, 0.1et 1. donc des valeurs extrêmes et une valeur plus raisonnable.

On a vu théoriquement que dans le cadre du boosting, un trop grand nombre d'itérations crée du surapprentissage et un trop petit nombre du sous apprentissage à cause d'un biais élevé:

$$L(\hat{g}_M) \leq L_n(\hat{g}_M) + \mathcal{O}\left(\sqrt{\frac{MV}{n}}\right)$$

De plus lors de l'utilisation d'une descente de gradient pour minimiser L, on a vu que plus lambda est grand moins le nombre d'itération est nécessaire pour obtenir la meilleur précision et vice versa. Voici une illustration de ce résultat sur le jeu de données oj.

En effet plus λ est petit, plus la précision maximale est atteinte en un nombre d'itération plus grand. On observe aussi sur les courbes associées aux λ atteignant un maximum, qu'après le nombre d'itération optimal, que pour les M superieurs à celui en lequel la meilleure précision est atteinte, la précision décroit . On ne le voit pas sur la courbe associée à $\lambda = 0.01$ car par souci de lisibilité nous avons limité M jusqu'à $M_max = 200$ mais en testant pour $M = 300 : 1000$ on a bien une décroissance de la précision à partir du M_opt.