Master 2 MMS
Apprentissage automatique

TP9 : Forêts aléatoires

Exercice 1. Récupérer le jeu de données d'apprentissage habituel synth_train.txt. On a $Y \in \{1, 2\}$ et $X \in \mathbb{R}^2$. On dispose de 100 données d'apprentissage.

1. Charger le jeu de données dans R. Transformer la variable de sortie `y` en facteur.

```
data <- read.table(file="../data/synth_train.txt", header=TRUE)
dim(data)

## [1] 100   3

data$y <- as.factor(data$y)
```

2. Charger le package `randomForest`. Construire une forêts aléatoire à l'aide de la fonction `randomForest` en gardant les paramètres par défaut. Consulter l'aide de la fonction `randomForest` notez bien tous les paramètres par défaut afin de savoir exactement quel algorithme est appliqué.

```
# install.packages("randomForest")
library(randomForest)

## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.

help(randomForest)
# http://gradientdescending.com/unsupervised-random-forest-example/
```

3. Afficher les résultats et vérifiez que vous comprenez les sorties associée au `print` de la forêt. Vérifiez ensuite que vous comprenez les éléments suivants de la forêt : `predicted`, `confusion`, `importance`, `importanceSD`. Testez la fonction `varImpPlot`. Regardez ensuite les éléments `votes`, `oob.times` et leur lien avec l'argument `norm.votes`.

```
rf <- randomForest(x=data[,-1], y=data$y)
rf

##
## Call:
##  randomForest(x = data[, -1], y = data$y)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 7%
## Confusion matrix:
##    1  2 class.error
## 1 18  4  0.18181818
## 2  3 75  0.03846154
```

```
rf$predicted
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
##   1   2   2   2   1   2   2   1   2   2   2   2   2   2   2
##  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
##  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
##   2   1   1   2   1   2   2   2   2   2   2   2   2   2   2
##  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   2   1   2   2   2   2   2   1   2   2   1   2   2   2   2
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75
##   2   2   2   2   2   1   2   1   1   1   2   2   2   2   2
##  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   2   2   2   2   2   1   1   2   1   2   2   2   2   2   2
##  91  92  93  94  95  96  97  98  99 100
##   2   1   2   2   1   2   1   1   2   1
## Levels: 1 2
```

```
# predicted : prediction OOB des données d'apprentissage
rf$confusion # sur les prediction OOB.
```

```
##    1  2 class.error
## 1 18  4  0.18181818
## 2  3 75  0.03846154
```

```
table(data[,1],rf$predicted)
```

```
##
##      1  2
##   1 18  4
##   2  3 75
```

```
# colonne class.error : 1-TVP et 1-TVN
#ajouter importance=TRUE
rf <- randomForest(x=data[,-1], y=data$y,importance=TRUE)
rf
```

```
##
## Call:
##  randomForest(x = data[, -1], y = data$y, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 5%
## Confusion matrix:
##    1  2 class.error
## 1 19  3  0.13636364
## 2  2 76  0.02564103
```

```
rf$importance
```

```
##            1          2 MeanDecreaseAccuracy
## x1 0.3894053 0.12163868           0.17807076
```

```
## x2 0.1258372 0.04508233           0.06164878
##    MeanDecreaseGini
## x1         21.58176
## x2         12.28228
```

```
# Augmentation moyenne (sur les ntree=500 arbres) du taux d'erreur OOB causée par la pertur
# deux premières colonnes dans les classes donc avec class.err.
# Ecart-type des 500 augmentation des taux d'erreurs OOB
rf$importanceSD
```

```
##              1           2 MeanDecreaseAccuracy
## x1 0.008741581 0.002725322          0.003565377
## x2 0.007367176 0.002257882          0.002643844
```

```
# Si on veut avoir les mesures d'importance "normalisées", on utilise la fonction imporranc
?importance
importance(rf, scale=FALSE) # on retrouve la sortie précédence
```

```
##            1          2 MeanDecreaseAccuracy
## x1 0.3894053 0.12163868           0.17807076
## x2 0.1258372 0.04508233           0.06164878
##    MeanDecreaseGini
## x1         21.58176
## x2         12.28228
```

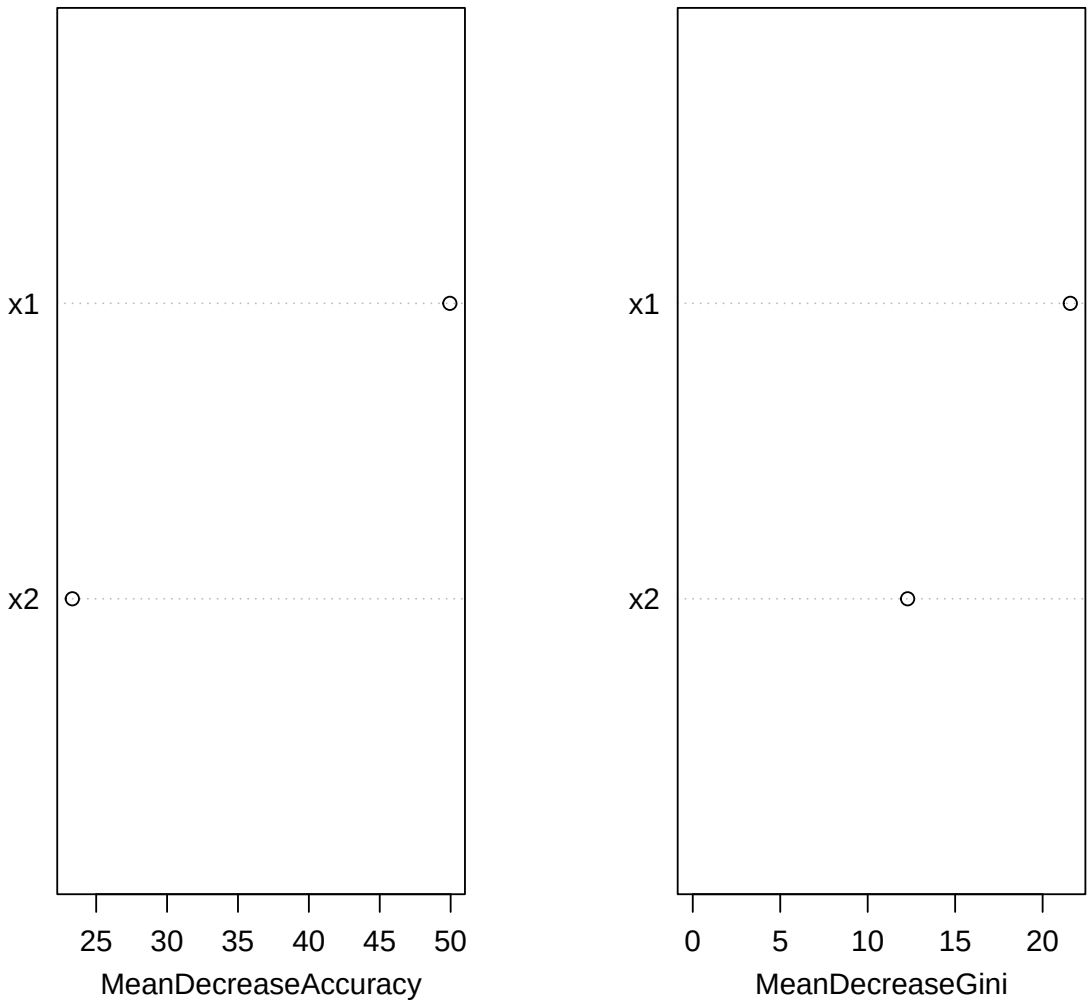
```
importance(rf) # scale=TRUE par defaut
```

```
##           1        2 MeanDecreaseAccuracy MeanDecreaseGini
## x1 44.54632 44.63278             49.94444         21.58176
## x2 17.08079 19.96665             23.31786         12.28228
```

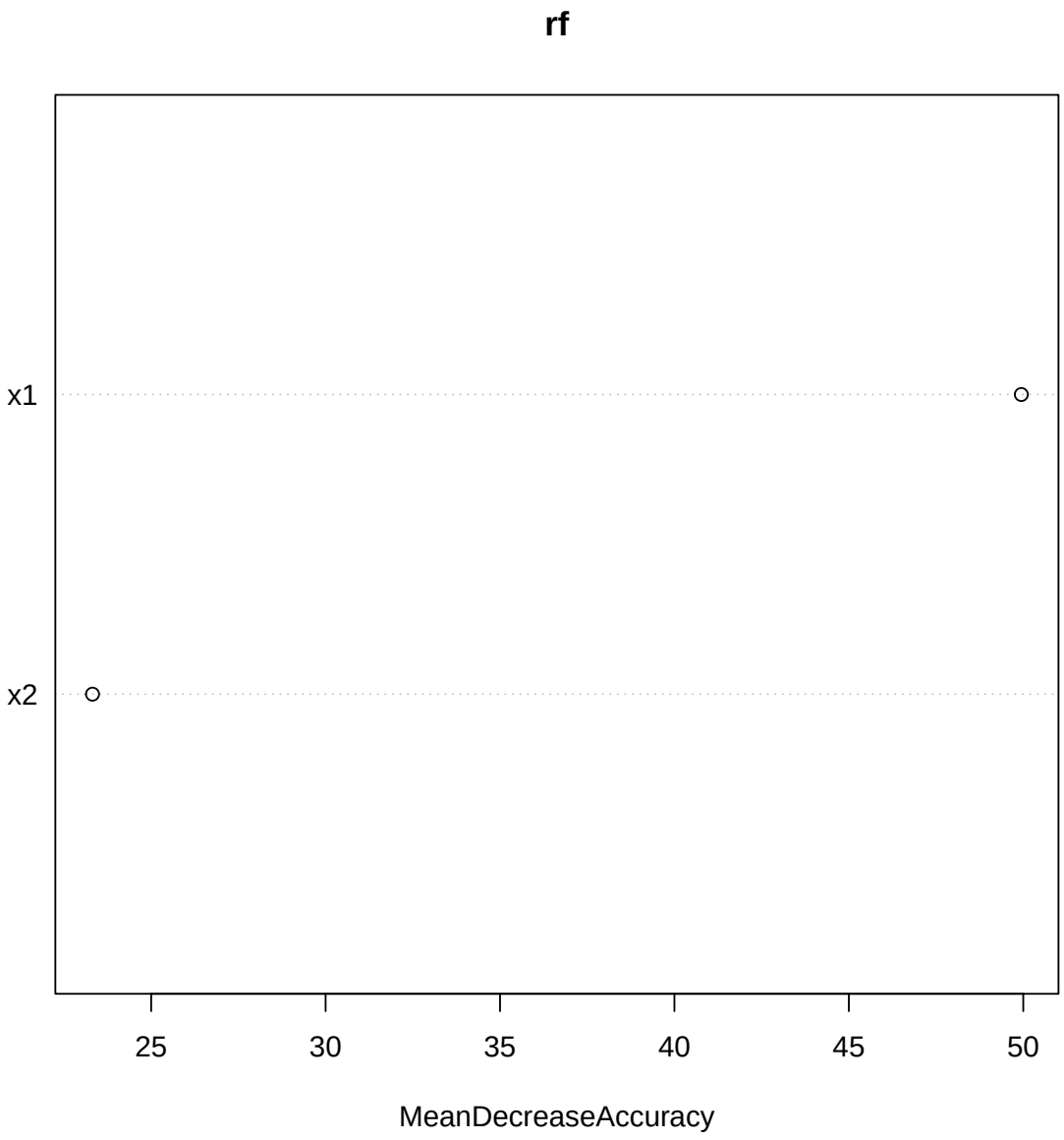
```
importance(rf, scale=FALSE, type=1)
```

```
##    MeanDecreaseAccuracy
## x1           0.17807076
## x2           0.06164878
```

```
?varImpPlot
varImpPlot(rf)
```




```r
varImpPlot(rf, type=1) # par defaut, scale=TRUE dans varImpPlot
```




```
#votes
dim(rf$votes)
```

```
## [1] 100   2
```

```
head(rf$votes) #ligne i : proportion d'arbres pour lequel i est OOB qui classent i en 1.
```

```
##           1         2
## 1 0.9600000 0.0400000
## 2 0.1098266 0.8901734
## 3 0.1564246 0.8435754
## 4 0.0000000 1.0000000
## 5 0.8770950 0.1229050
## 6 0.1494845 0.8505155
```

```
head(rf$oob.times) #nombre d'arbre OOB par donnée.
```

```
## [1] 200 173 179 179 179 194
```

```r
head(sweep(rf$votes,1,rf$oob.times,'*')) #nombre d'abre qui classent en 1 et 2
```

```
##     1   2
## 1 192   8
## 2  19 154
## 3  28 151
## 4   0 179
## 5 157  22
## 6  29 165
```

```r
# pour avoir des effectifs ajouter norm.votes=TRUE
rf2 <- randomForest(x=data[,-1], y=data$y,importance=TRUE,
                    norm.votes=FALSE)
head(rf2$votes) #somme colonne=nb d'arbre OOB de chaque entree.
```

```
##     1   2
## 1 194   5
## 2  28 173
## 3  26 134
## 4   0 204
## 5 153  32
## 6  16 172
```

4. Vérifiez ensuite que vous comprenez l'élément **err.rate** de la forêt. Retrouver dans **err.rate** le taux d'erreur OOB global et par classe obtenus avec la fonction **print**. Utiliser ensuite les résultats présents dans **err.rate** pour faire un graphique donnant une idée du calibrage du paramètre **ntree**.

```r
#err.rate: une ligne par arbre (500 ici). Taux d'erreurs OOB avec les $i$ premiers arbres.
dim(rf$err.rate)
```

```
## [1] 500   3
```

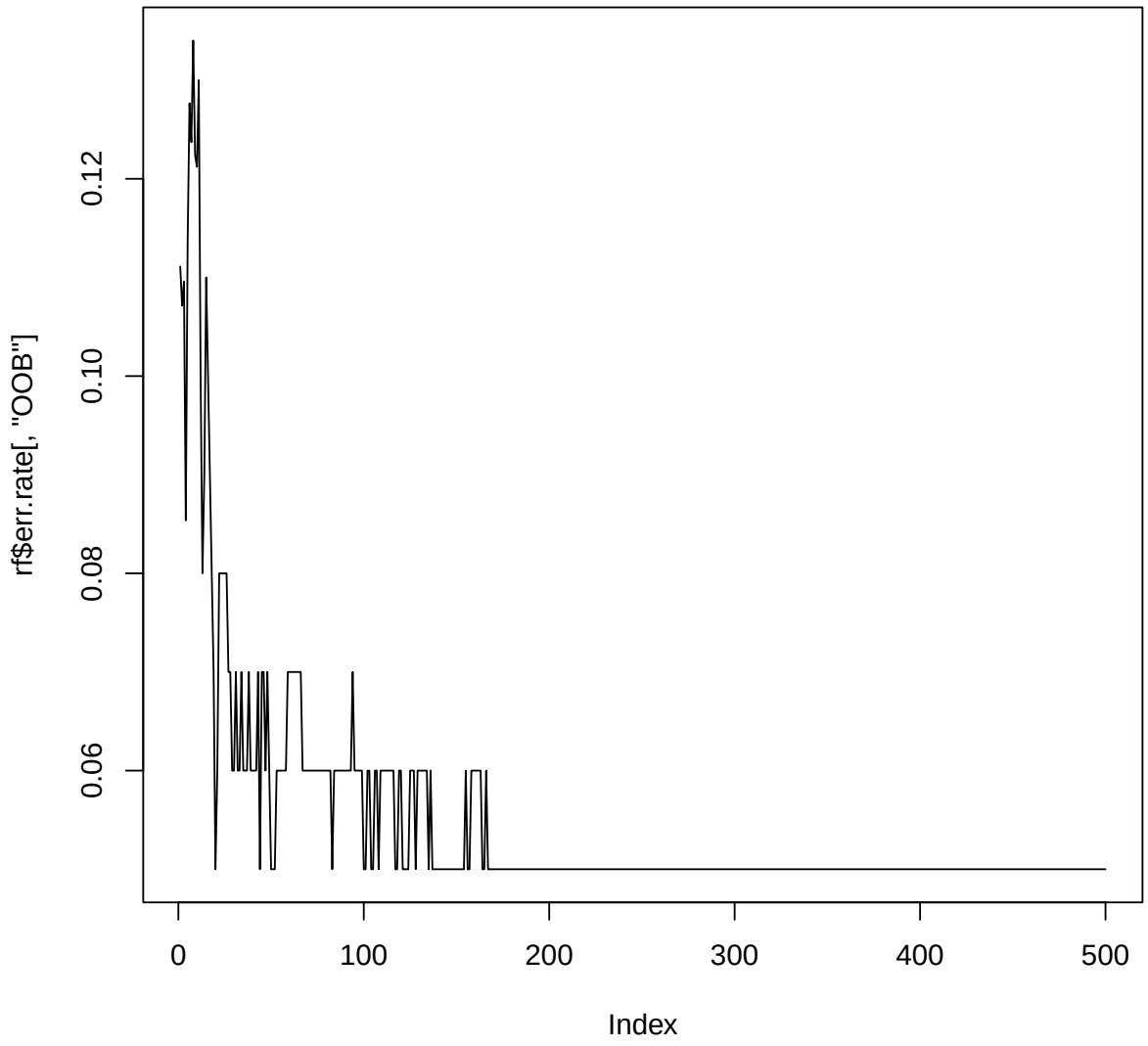
```r
head(rf$err.rate)
```

```
##             OOB          1          2
## [1,] 0.11111111 0.22222222 0.07407407
## [2,] 0.10714286 0.08333333 0.11363636
## [3,] 0.10958904 0.14285714 0.10169492
## [4,] 0.08536585 0.12500000 0.07575758
## [5,] 0.11363636 0.26315789 0.07246377
## [6,] 0.12765957 0.33333333 0.06849315
```

```r
rf$err.rate[500,] # on retrouve les résultats de print(rf)
```

```
##        OOB          1          2
## 0.05000000 0.13636364 0.02564103
```

```r
plot(rf$err.rate[,"OOB"], type="l")#ntree=500 largement suffisant.
```



5. Calculer le taux d'erreur d'apprentissage.

```
pred <- predict(rf, newdata=data, type="class")
sum(pred!=data$y)/length(pred)

## [1] 0
```

6. Charger le jeu de données test synth_test.txt puis calculer le taux d'erreur test de la forêt paramétrée par défaut.

```
data.test <- read.table(file="../data/synth_test.txt", header=TRUE)
dim(data.test)

## [1] 200   3

pred.test <- predict(rf, newdata=data.test, type="class")
sum(pred.test!=data.test$y)/length(pred.test)
```

```
## [1] 0.05
```

7. Modifiez le paramètrage de la fôret pour que la méthode d'ensemble utilisée soit le `bagging`. Calculer alors le taux d'erreur des données test.

```
#ntree=2000
#mtry=p donc pas d'aléa et par CART-RI donc bagging.
bag <- randomForest(x=data[,-1], y=data$y, mtry=2, ntree=2000)
bag
```

```
##
## Call:
##  randomForest(x = data[, -1], y = data$y, ntree = 2000, mtry = 2)
##                Type of random forest: classification
##                      Number of trees: 2000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 8%
## Confusion matrix:
##    1  2 class.error
## 1 18  4  0.18181818
## 2  4 74  0.05128205
```

```
pred.test <- predict(bag, newdata=data.test, type="class")
sum(pred.test!=data.test$y)/length(pred.test)
```

```
## [1] 0.075
```

Exercice 2. On reprend les données concernant $n = 1260$ exploitations agricoles. Les variables explicatives sont $p = 30$ critères économiques et financiers et la variable qualitative à expliquer est la variable difficulté de paiement (0=saine t 1=défaillant).

1. Charger le jeu de données Desbois_complet.rda dans R.

```
load("../data/Desbois_complet.rda")
table(data$DIFF)
```

```
##
##   0   1
## 653 607
```

```
data$DIFF <- as.factor(data$DIFF)
```

2. Créez un découpage aléatoire des données en 945 observations d'apprentissage et 315 observations test.

```
set.seed(10)
tr <- sample(1:nrow(data),945)
train <- data[tr,]
test <- data[-tr,]
```

3. Quelle est l'erreur OOB de la forêt constuite sur les données d'apprentissage avec les paramètres `mtry` et `ntree` par défaut ?

```
rf <- randomForest(DIFF~., data=train)
rf <- randomForest(DIFF~., data=data)
```

```
#OOB error rate
print(rf)
```

```
##
## Call:
##  randomForest(formula = DIFF ~ ., data = data)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 9.92%
## Confusion matrix:
##     0   1 class.error
## 0 590  63  0.09647779
## 1  62 545  0.10214168
```

```
rf$err.rate[500,1]
```

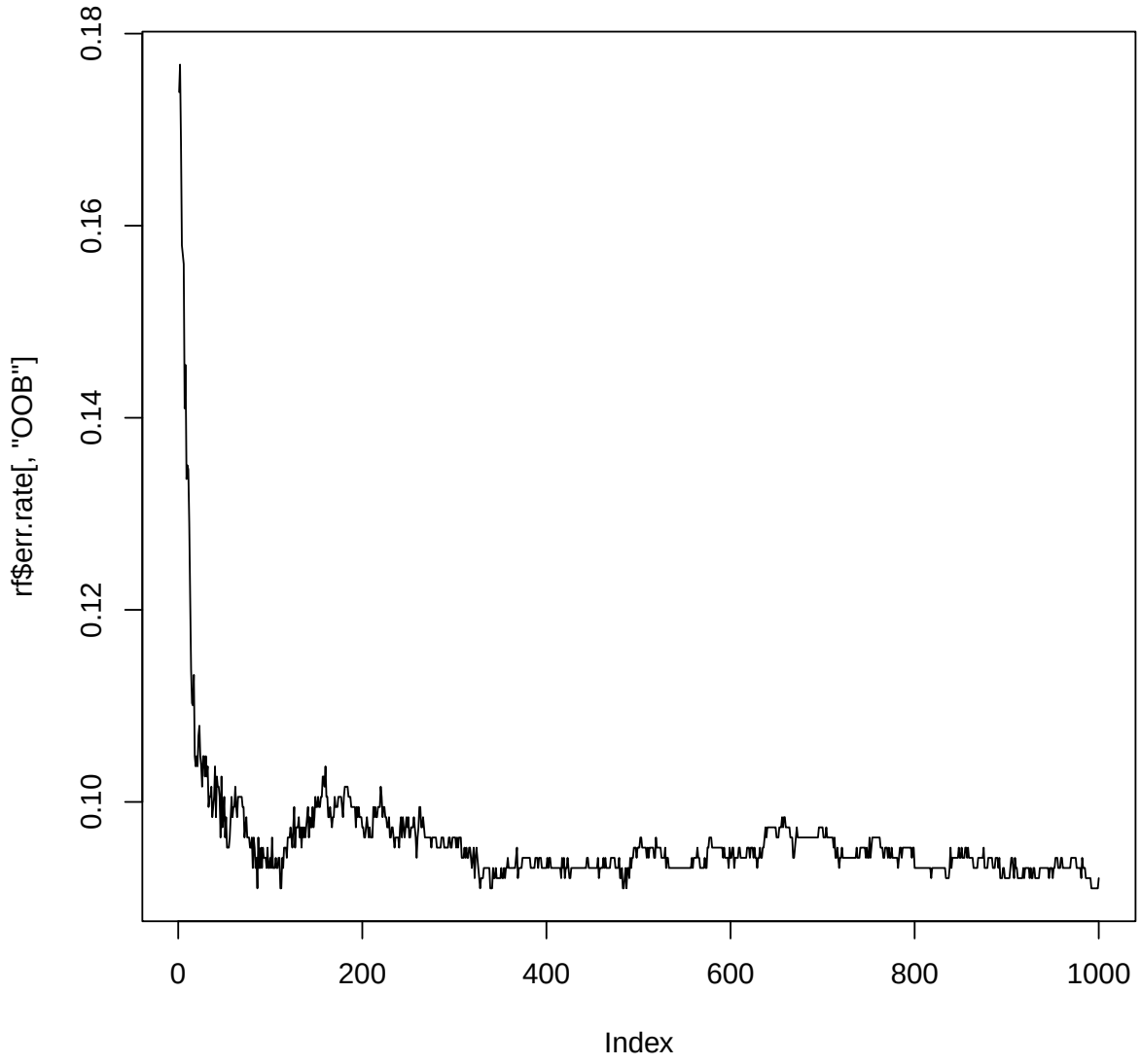
```
##        OOB
## 0.09920635
```

```
pred.test <- predict(rf, newdata=test[,-1], type="class")
sum(pred.test!=test$DIFF)/length(pred.test) # taux erreur test
```

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

4. Le nombre d'arbre par défaut vous semble-il suffisant ?

```
rf <- randomForest(DIFF~., data=train,ntree=1000)
plot(rf$err.rate[,"OOB"], type="l")#ntree=500  suffisant.
```



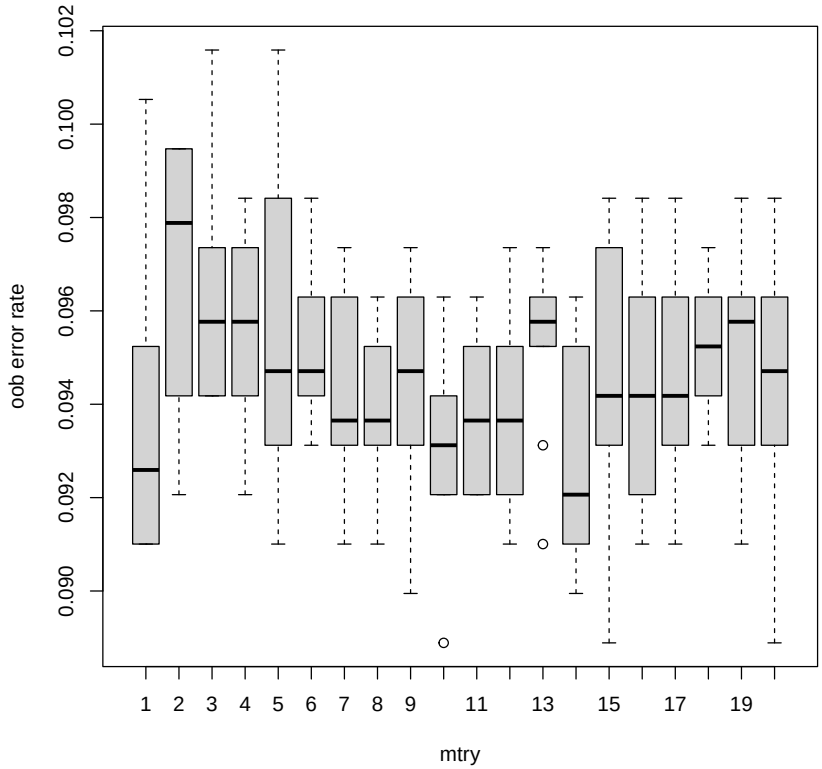
5. Afin d'avoir une première idée du choix du paramètre `mtry` reproduire le graphique ci-dessous.

```
sqrt(22)
rf$mtry
# boucle sur une grille de valeur de mtry
grille <- 1:20
B <- 10
err_oob <- matrix(NA,B,length(grille))
for (j in 1:B)
  for (i in seq(grille))
  {
    rf <- randomForest(DIFF~., data=train,
                       mtry=grille[i])
    err_oob[j,i] <- rf$err.rate[500,1]
```

```
    }
save(err_oob, file="../data/choix_mtry.rda")
```

```
load("../data/choix_mtry.rda")
```

```
boxplot(err_oob,ylab="oob error rate",xlab="mtry")
```



6. Proposez une procédure de choix automatique du paramètre `mtry` (utilisant l'erreur OOB).

```
# choix automatique pour mtry de 1 à 7
# boucle sur une grille de valeur de mtry
grille <- 1:20
oob <- rep(NA, length(grille))
for (i in seq(grille))
{
  rf <- randomForest(DIFF~.,
                     data=train, mtry=grille[i])
  oob[i] <- rf$err.rate[500,1]
}
plot(oob*100, type="l", ylim=c(6,12))
m <- grille[which.min(oob)] #mtry opti
```

7. Prédire les données test avec la valeur optimale de `mtry` obtenue à la question précédente et calculer le taux d'erreur test.

```
pred.test <- predict(rf, newdata=test[,-1],
                     type="class", mtry=m)
# taux erreur test
sum(pred.test!=test$DIFF)/length(pred.test)
```

```
## [1] 0.1206349
```

8. Comparez ensuite les performances des forêts aléatoires avec celles de la régression logistique pour ces données. Attention de bien inclure le calibrage automatique du paramètre `mtry` dans la procédure de comparaison.

Exercice 3. Refaire le traitement proposé dans ces articles de blog concernant les *imbalanced data* :

https://shiring.github.io/machine_learning/2017/04/02/unbalanced
https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18

```
data <- read.table("../data/numbers_train.txt", header=TRUE)
Xtrain <- as.matrix(data[,-1])
Ytrain <- as.factor(data[,1])
rf <- randomForest(x=Xtrain, y=Ytrain)
rf
```