TP 1 Apprentissage supervisé

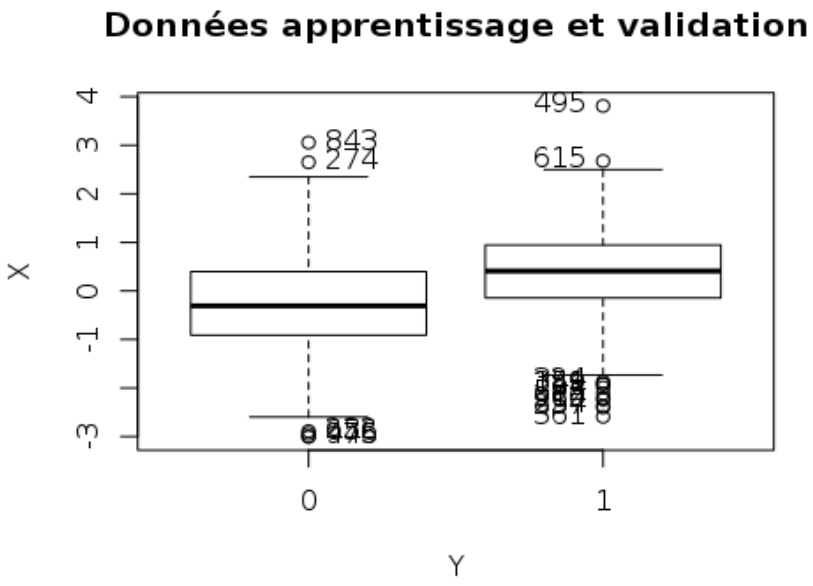
B Gelein

12/09/2019

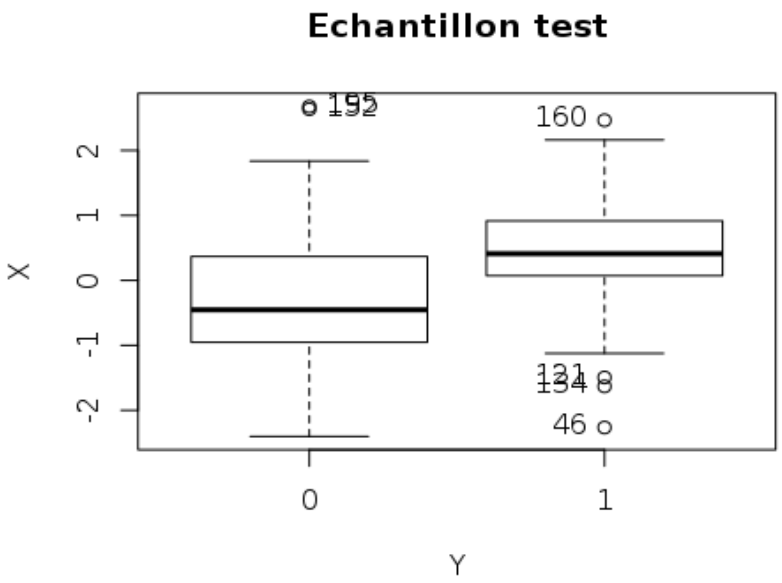
Exercice 1 - question 3

Générons 1000 observations qui serviront en apprentissage et/ou validation :

```
n= 1000
set.seed(1) # pour avoir tous les mêmes données
X=rnorm(n)
set.seed(2) # pour avoir tous les mêmes données
U=runif(n)
Y1=rep(0,n)
Y1[X<=0 & U <=0.2]=1
Y1[U>0.4 & X>0]=1
Y=as.factor(Y1) ### !!!!
donnees=data.frame(X,Y)

library(car)

## Loading required package: carData

Boxplot(data=donnees,X~Y,main='Données apprentissage et validation')
```



```
##  [1] "232" "446" "656" "975" "274" "843" "14"  "24"  "141" "257" "324"
## [12] "359" "361" "595" "854" "960" "495" "615"

# Simulation échantillon test

n= 200
set.seed(3) # pour avoir tous les mêmes données
X=rnorm(n)
set.seed(4) # pour avoir tous les mêmes données
U=runif(n)
Y1=rep(0,n)
Y1[X<=0 & U <=0.2]=1
Y1[U>0.4 & X>0]=1
Y=as.factor(Y1) ### !!!!
test=data.frame(X,Y)

Boxplot(data=test,X~Y,main='Echantillon test')
```




```
## [1] "152" "195" "46"  "121" "134" "160"
```

Exercice 1 - question 4 - K plus proches voisins

Règle de décision associée au K plus proches voisins cf cours diapos 57-58 Rappel : les Knn sont de complexité décroissante en fonction de K le nombre de voisins (ou croissante en fonction de 1/K) cf Hastie et al. à relier avec la notion de sur-apprentissage.

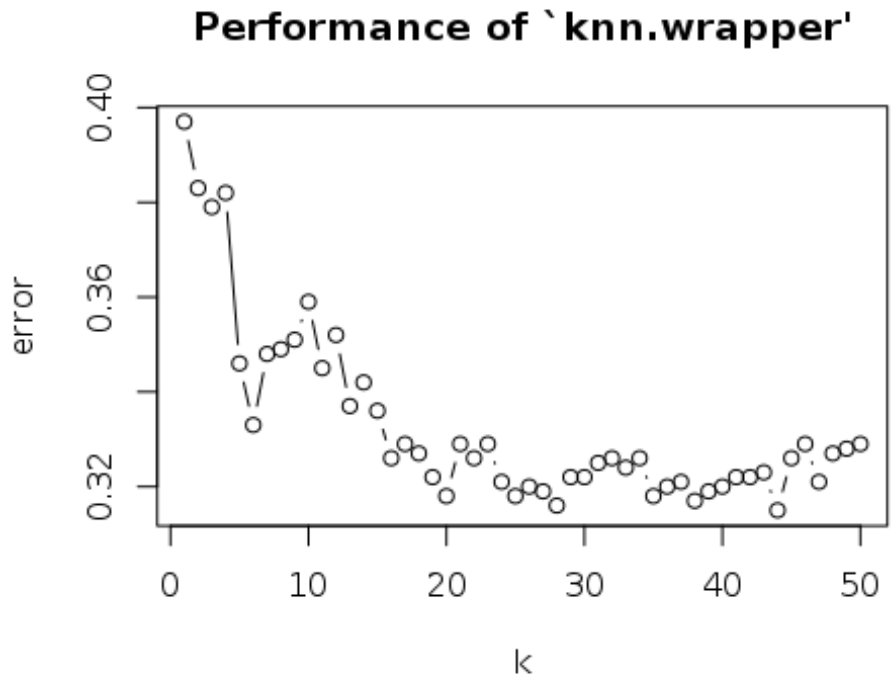
Recherche du meilleur K par validation croisée 10-blocs/10-folds (attention pas le même K que pour Knn !) :

```r
library(e1071)

knn.cross <- tune.knn(x = donnees$X,
y = donnees$Y, k = 1:50,
tunecontrol=tune.control(sampling = "cross"),
cross=10)

summary(knn.cross)
```

```
##
## Parameter tuning of 'knn.wrapper':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   k
##  44
##
## - best performance: 0.315
##
## - Detailed performance results:
##     k error dispersion
## 1   1 0.397 0.07242621
## 2   2 0.383 0.07196450
## 3   3 0.379 0.06226288
## 4   4 0.382 0.06142746
## 5   5 0.346 0.04351245
## 6   6 0.333 0.03973523
## 7   7 0.348 0.04366539
## 8   8 0.349 0.03928528
## 9   9 0.351 0.03142893
## 10 10 0.359 0.04433459
## 11 11 0.345 0.03922867
## 12 12 0.352 0.04104198
## 13 13 0.337 0.05618422
## 14 14 0.342 0.05138093
## 15 15 0.336 0.04742245
## 16 16 0.326 0.04812022
## 17 17 0.329 0.05043147
## 18 18 0.327 0.05121849
## 19 19 0.322 0.05884065
## 20 20 0.318 0.04779586
## 21 21 0.329 0.05087021
## 22 22 0.326 0.04835057
## 23 23 0.329 0.04175324
## 24 24 0.321 0.04306326
## 25 25 0.318 0.04732864
## 26 26 0.320 0.04320494
## 27 27 0.319 0.04653553
## 28 28 0.316 0.04168666
```

```
## 29 29 0.322 0.03705851
## 30 30 0.322 0.03852849
## 31 31 0.325 0.04503085
## 32 32 0.326 0.04575296
## 33 33 0.324 0.04325634
## 34 34 0.326 0.04005552
## 35 35 0.318 0.04467164
## 36 36 0.320 0.04690416
## 37 37 0.321 0.03842742
## 38 38 0.317 0.03653005
## 39 39 0.319 0.04201851
## 40 40 0.320 0.03651484
## 41 41 0.322 0.04341019
## 42 42 0.322 0.03765339
## 43 43 0.323 0.04398232
## 44 44 0.315 0.04169999
## 45 45 0.326 0.04221637
## 46 46 0.329 0.04280446
## 47 47 0.321 0.04175324
## 48 48 0.327 0.04715224
## 49 49 0.328 0.04962078
## 50 50 0.329 0.04306326
```

```
plot(knn.cross)
```



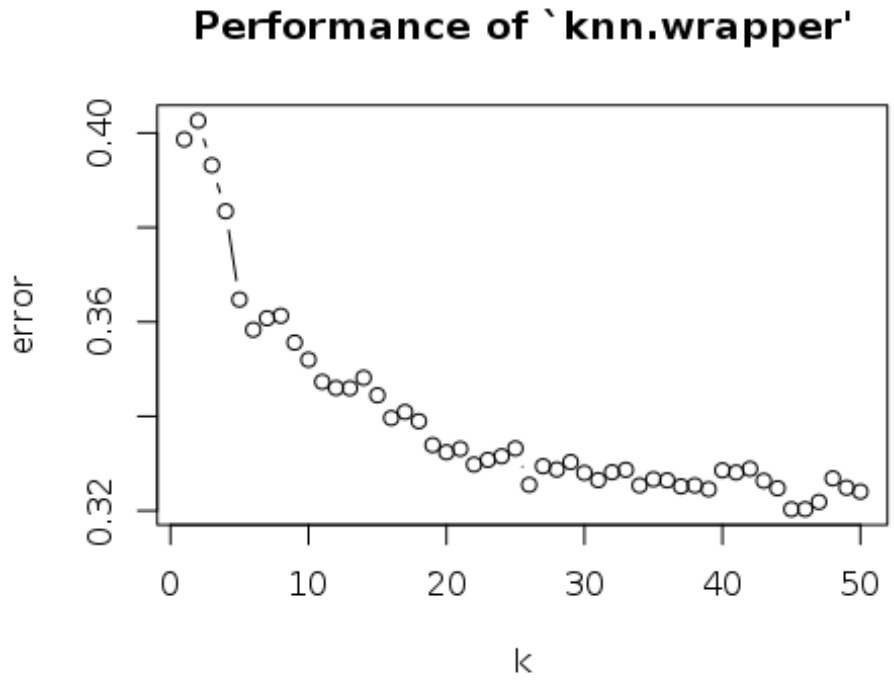
Recherche du meilleur K par validation croisée bootstrap :

```r
library(e1071)

knn.boot <- tune.knn(x = donnees$X,
y =donnees$Y, k = 1:50,
tunecontrol=tune.control(sampling = "boot"))

summary(knn.boot)
```

```
##
## Parameter tuning of 'knn.wrapper':
##
## - sampling method: bootstrapping
##
## - best parameters:
##   k
##  45
##
## - best performance: 0.3202992
##
## - Detailed performance results:
##     k     error dispersion
## 1   1 0.3986532 0.01936331
## 2   2 0.4025928 0.02711233
## 3   3 0.3931594 0.03615440
## 4   4 0.3834336 0.03173655
## 5   5 0.3647217 0.02460558
## 6   6 0.3583314 0.02170132
## 7   7 0.3607730 0.01700034
## 8   8 0.3612899 0.01800846
## 9   9 0.3556186 0.01842427
## 10 10 0.3519798 0.01652626
## 11 11 0.3473010 0.01686121
## 12 12 0.3459966 0.02742980
## 13 13 0.3459364 0.02324159
## 14 14 0.3481605 0.01975081
## 15 15 0.3444483 0.02253879
## 16 16 0.3396339 0.02712678
## 17 17 0.3409154 0.02347881
## 18 18 0.3389609 0.02203423
## 19 19 0.3338993 0.01818912
## 20 20 0.3323792 0.02119648
## 21 21 0.3330960 0.02098998
## 22 22 0.3297836 0.02024625
## 23 23 0.3307628 0.02405952
## 24 24 0.3315373 0.02304775
## 25 25 0.3331907 0.02522552
## 26 26 0.3254869 0.02235373
## 27 27 0.3294245 0.01973738
## 28 28 0.3287197 0.02264960
## 29 29 0.3303153 0.02096160
```

```
## 30 30 0.3280113 0.02033097
## 31 31 0.3264846 0.02131212
## 32 32 0.3281526 0.01871406
## 33 33 0.3286593 0.01860402
## 34 34 0.3253539 0.01867474
## 35 35 0.3266657 0.01978250
## 36 36 0.3264327 0.02160261
## 37 37 0.3251573 0.01847291
## 38 38 0.3253512 0.01918700
## 39 39 0.3245506 0.02079627
## 40 40 0.3285531 0.01908402
## 41 41 0.3281023 0.01729120
## 42 42 0.3288775 0.01759010
## 43 43 0.3263369 0.01974112
## 44 44 0.3247471 0.02000671
## 45 45 0.3202992 0.01915397
## 46 46 0.3203181 0.01775182
## 47 47 0.3218535 0.01822580
## 48 48 0.3268479 0.01714252
## 49 49 0.3248683 0.01630750
## 50 50 0.3240452 0.01540216
```

```
plot(knn.boot)
```



Après avoir fait tourné plusieurs fois les 2 méthodes et en tenant compte de l'écart-type, c'est souvent autour de k=30 que l'on obtient le "best parameter".

Prédiction et calcul de l'erreur sur échantillon test avec les 30 plus proches voisins:

```
library(class)
knn.pred <- knn(as.matrix(donnees$X),as.matrix(test$X),donnees$Y,k = 30)
# accuracy = taux de succes
sum(knn.pred==test$Y)/200
```

```
## [1] 0.725
```

```
# accuracy = taux d erreur
1-sum(knn.pred==test$Y)/200
```

```
## [1] 0.275
```

Si on compare le taux d'erreur en test et l'erreur de Bayes : on est tout proche ce qui est très satisfaisant !

Exercice 1 - question 5 - Classifieur Bayésien naïf

Règle de décision associée au Classifieur Bayésien naïf cf cours diapos 78

```
library(klaR)
```

```
## Loading required package: MASS
```

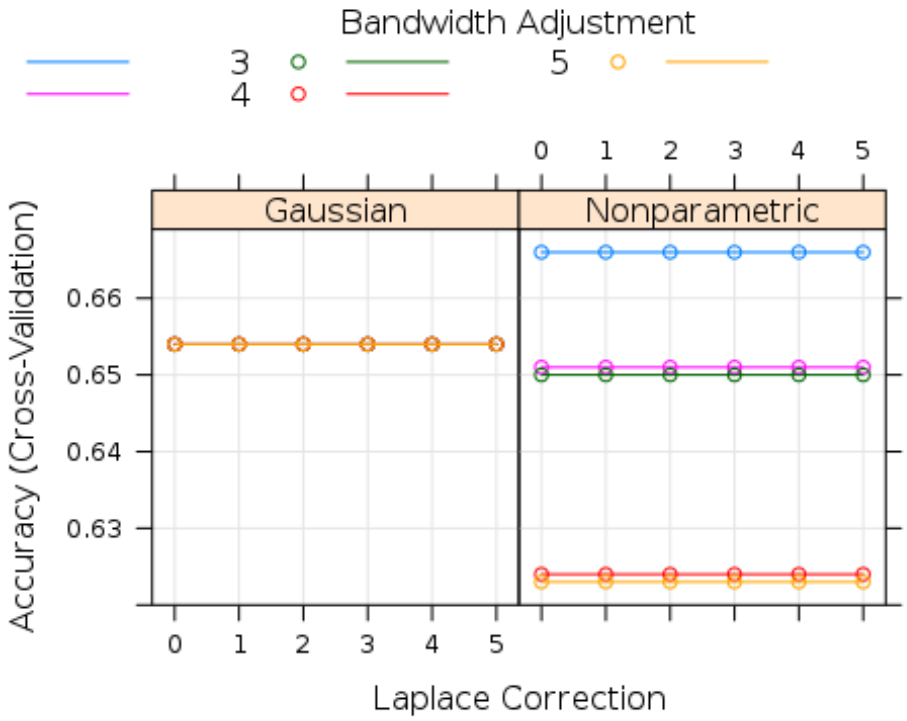
```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
grid<- expand.grid(usekernel=c(TRUE,FALSE),fL=0:5,adjust=seq(1,5,by=1))
control<-trainControl(method="cv",number=10)

names(donnees)<-c("X","Y")

x <- as.data.frame(donnees[,1])
names(x)=c("X") #contrainte de caret, il faut que x ait un nom ici "X"

y <-donnees[,2]
# train model
NB1<-train(x=x,y=y, method="nb",trControl=control,tuneGrid=grid)

#plot search grid results avec library(ggplot2)

plot(NB1)
```



Ici le meilleur modèle en validation croisée 10-blocs : estimation non paramétrique de densité à noyau gaussien avec BW=1

```
NB1.pred<-predict(NB1,test)

# accuracy = taux de succes sur echantillon test
sum(NB1.pred==test$Y)/200

## [1] 0.68

# accuracy = taux d erreur sur echantillon test
1-sum(NB1.pred==test$Y)/200

## [1] 0.32
```

A noter : inutile de faire un prétraitement de X avec la transformation de BOX COX car X suit une loi normale.

Conclusion : meilleur modèle Knn avec K=30 et estimateur de densité non paramétrique bw=1.

Exercice 2

```
library(mlbench)
data(Vehicle)
summary(Vehicle)
```

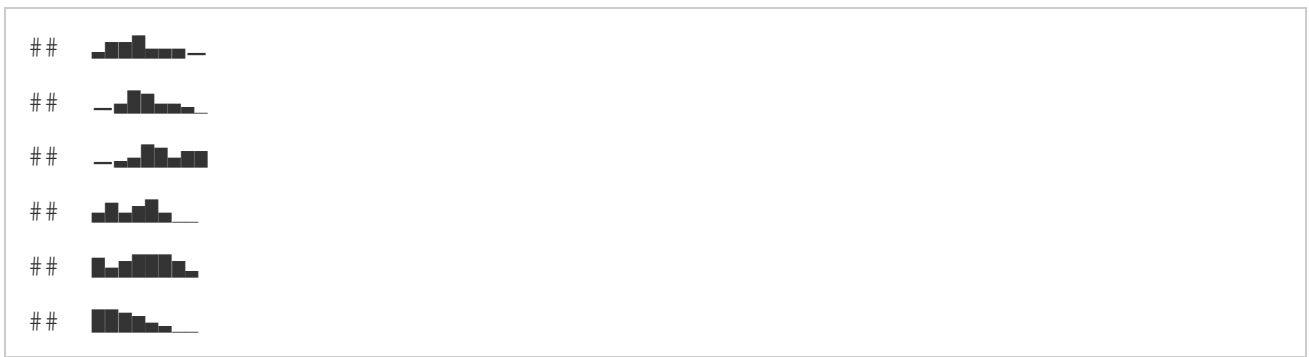
```
##       Comp            Circ           D.Circ           Rad.Ra     
##  Min.   : 73.00   Min.   :33.00   Min.   : 40.00   Min.   :104.0  
##  1st Qu.: 87.00   1st Qu.:40.00   1st Qu.: 70.00   1st Qu.:141.0  
##  Median : 93.00   Median :44.00   Median : 80.00   Median :167.0  
##  Mean   : 93.68   Mean   :44.86   Mean   : 82.09   Mean   :168.9  
##  3rd Qu.:100.00   3rd Qu.:49.00   3rd Qu.: 98.00   3rd Qu.:195.0  
##  Max.   :119.00   Max.   :59.00   Max.   :112.00   Max.   :333.0  
##    Pr.Axis.Ra       Max.L.Ra         Scat.Ra          Elong       
##  Min.   : 47.00   Min.   : 2.000   Min.   :112.0   Min.   :26.00  
##  1st Qu.: 57.00   1st Qu.: 7.000   1st Qu.:146.2   1st Qu.:33.00  
##  Median : 61.00   Median : 8.000   Median :157.0   Median :43.00  
##  Mean   : 61.69   Mean   : 8.567   Mean   :168.8   Mean   :40.93  
##  3rd Qu.: 65.00   3rd Qu.:10.000   3rd Qu.:198.0   3rd Qu.:46.00  
##  Max.   :138.00   Max.   :55.000   Max.   :265.0   Max.   :61.00  
##   Pr.Axis.Rect     Max.L.Rect   Sc.Var.Maxis    Sc.Var.maxis   
##  Min.   :17.00   Min.   :118   Min.   :130.0   Min.   : 184.0  
##  1st Qu.:19.00   1st Qu.:137   1st Qu.:167.0   1st Qu.: 318.2  
##  Median :20.00   Median :146   Median :178.5   Median : 364.0  
##  Mean   :20.58   Mean   :148   Mean   :188.6   Mean   : 439.9  
##  3rd Qu.:23.00   3rd Qu.:159   3rd Qu.:217.0   3rd Qu.: 587.0  
##  Max.   :29.00   Max.   :188   Max.   :320.0   Max.   :1018.0  
##      Ra.Gyr        Skew.Maxis       Skew.maxis       Kurt.maxis   
##  Min.   :109.0   Min.   : 59.00   Min.   : 0.000   Min.   : 0.0  
##  1st Qu.:149.0   1st Qu.: 67.00   1st Qu.: 2.000   1st Qu.: 5.0  
##  Median :173.0   Median : 71.50   Median : 6.000   Median :11.0  
##  Mean   :174.7   Mean   : 72.46   Mean   : 6.377   Mean   :12.6  
##  3rd Qu.:198.0   3rd Qu.: 75.00   3rd Qu.: 9.000   3rd Qu.:19.0  
##  Max.   :268.0   Max.   :135.00   Max.   :22.000   Max.   :41.0  
##    Kurt.Maxis       Holl.Ra       Class     
##  Min.   :176.0   Min.   :181.0   bus :218  
##  1st Qu.:184.0   1st Qu.:190.2   opel:212  
##  Median :188.0   Median :197.0   saab:217  
##  Mean   :188.9   Mean   :195.6   van :199  
##  3rd Qu.:193.0   3rd Qu.:201.0             
##  Max.   :206.0   Max.   :211.0             
```

Pour connaître plus précisément la description de Vehicle, on peut taper “Vehicle” dans l’onglet Help du panneau de droite dans Rstudio. Le libellé des variables apparaît en clair. Il y a 18 régresseurs. La variable à expliquer est Class, variable qualitative à 4 modalités.

```
library(skimr)
```

```
## Registered S3 method overwritten by 'skimr':
##   method      from
##   print.spark pillar

##
## Attaching package: 'skimr'
```

```
## The following object is masked from 'package:stats':
##
##     filter

skim(Vehicle)

## Skim summary statistics
##  n obs: 846
##  n variables: 19
##
## ── Variable type:factor
──────────────────────────────────────────────────────────────────────────────
##  variable missing complete   n n_unique
##     Class       0      846 846        4
##                            top_counts ordered
##  bus: 218, saa: 217, ope: 212, van: 199   FALSE
##
## ── Variable type:numeric
──────────────────────────────────────────────────────────────────────────────
##      variable missing complete   n   mean     sd  p0    p25   p50 p75 p100
##          Circ       0      846 846  44.86   6.17  33  40     44    49   59
##          Comp       0      846 846  93.68   8.23  73  87     93   100  119
##        D.Circ       0      846 846  82.09  15.77  40  70     80    98  112
##         Elong       0      846 846  40.93   7.81  26  33     43    46   61
##       Holl.Ra       0      846 846 195.63   7.44 181 190.25 197   201  211
##    Kurt.maxis       0      846 846  12.6    8.93   0   5     11    19   41
##    Kurt.Maxis       0      846 846 188.93   6.16 176 184    188   193  206
##      Max.L.Ra       0      846 846   8.57   4.6    2   7      8    10   55
##    Max.L.Rect       0      846 846 148     14.52 118 137    146   159  188
##    Pr.Axis.Ra       0      846 846  61.69   7.89  47  57     61    65  138
##  Pr.Axis.Rect       0      846 846  20.58   2.59  17  19     20    23   29
##        Ra.Gyr       0      846 846 174.7   32.55 109 149    173   198  268
##        Rad.Ra       0      846 846 168.94  33.47 104 141    167   195  333
##  Sc.Var.maxis       0      846 846 439.91 176.69 184 318.25 364   587 1018
##  Sc.Var.Maxis       0      846 846 188.63  31.39 130 167    178.5 217  320
##       Scat.Ra       0      846 846 168.84  33.24 112 146.25 157   198  265
##    Skew.maxis       0      846 846   6.38   4.92   0   2      6     9   22
##    Skew.Maxis       0      846 846  72.46   7.49  59  67     71.5  75  135
##      hist
```



```
##  ▂▆▇▅▂▁
##  ▆▂▁▁▁▁
##  ▂▆▇▅▂▁
##  ▃▇▁▁▁▁
##  ▃▇▁▂▂▁
##  ▂▃▅▇▅▁
##  ▃▇▇▅▁▁
##  ▃▇▁▂▁▁
##  ▃▇▂▃▁▁
##  ▃▇▅▂▃▁
##  ▆▇▅▂▂▁
##  ▃▇▁▁▁▂
```

2. La fonction de coût est la même que pour l'exercice 1.
3. La règle de décision est un vote majoritaire dans le voisinage local de chaque observation, avec cette fois 4 modalités possibles.

$\widehat{f(X)} = \underset{y \in Y}{argmax}(\sum Wi\ 1(Y = y))$ ...adaptation de la formule diapo 57, avec les Wi définis diapo 58

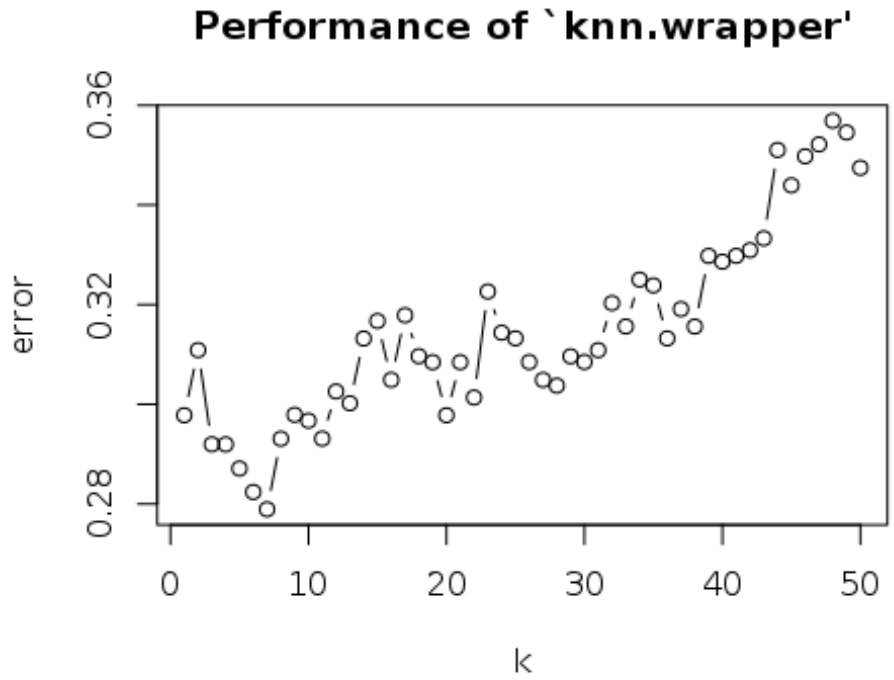
On pourrait tester avec les 18 régresseurs ou moins en raison du fléau de la dimension auquel l'algorithme des plus proches voisins est sensible (voir diapo 67). Sélection des régresseurs par statistique bivariée mais en étant pas trop sélectif sur du bivarié (prendre un régresseur même si le lien n'est pas trop fort avec la variable Class).

Ici possibilité d'approfondir avec de la sélection de variables. Recherche du meilleur K par validation croisée 10-blocs/10-folds (attention pas le même K que pour Knn !) : Ici il faudra du scale car plusieurs variables de différentes variances.

```
knn.cross <- tune.knn(x = scale(Vehicle[,1:18]),
y = Vehicle[,19], k = 1:50,
tunecontrol=tune.control(sampling = "cross"),
cross=10)

summary(knn.cross)
```

```
##
## Parameter tuning of 'knn.wrapper':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  k
##  7
##
## - best performance: 0.2789216
##
## - Detailed performance results:
##     k     error dispersion
## 1   1 0.2978011 0.05995507
## 2   2 0.3108683 0.04735828
## 3   3 0.2919608 0.04202989
## 4   4 0.2919468 0.06836514
## 5   5 0.2871008 0.05542151
## 6   6 0.2823950 0.05707019
## 7   7 0.2789216 0.05005310
## 8   8 0.2930812 0.04517854
## 9   9 0.2978852 0.05479155
## 10 10 0.2967087 0.06601476
## 11 11 0.2931092 0.06936338
## 12 12 0.3025770 0.06912746
## 13 13 0.3002101 0.07228258
## 14 14 0.3131933 0.06999006
## 15 15 0.3167087 0.07096421
## 16 16 0.3049020 0.07572938
## 17 17 0.3178571 0.07357373
## 18 18 0.3096499 0.06654076
## 19 19 0.3084454 0.08178496
## 20 20 0.2978011 0.07512613
## 21 21 0.3084454 0.07741424
## 22 22 0.3013585 0.07616290
## 23 23 0.3226331 0.08543438
## 24 24 0.3143697 0.07126958
## 25 25 0.3132353 0.06253432
## 26 26 0.3084734 0.06783819
## 27 27 0.3048880 0.06493179
## 28 28 0.3037395 0.06632011
## 29 29 0.3096218 0.06693493
## 30 30 0.3084874 0.06345138
## 31 31 0.3108543 0.06406250
## 32 32 0.3203081 0.06682002
## 33 33 0.3156162 0.06493876
## 34 34 0.3250140 0.07155348
## 35 35 0.3238515 0.06778506
## 36 36 0.3132213 0.06061089
## 37 37 0.3191036 0.05333208
```

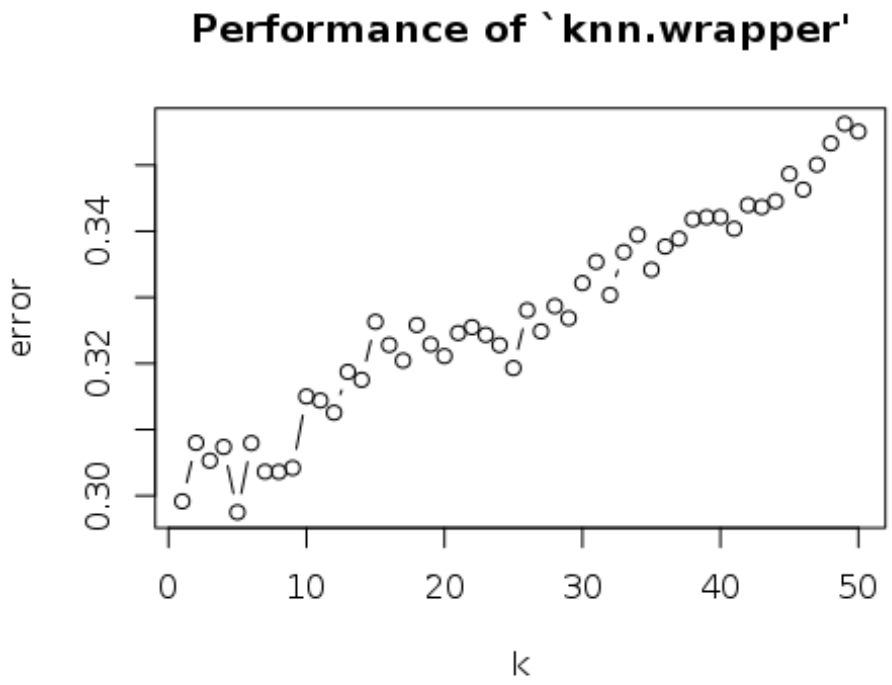
```
## 38 38 0.3156022 0.04378698
## 39 39 0.3297759 0.05277266
## 40 40 0.3286134 0.05627844
## 41 41 0.3297899 0.05179027
## 42 42 0.3309244 0.06632250
## 43 43 0.3333053 0.06581096
## 44 44 0.3510364 0.06841769
## 45 45 0.3439356 0.06606703
## 46 46 0.3497759 0.07282614
## 47 47 0.3521569 0.06730531
## 48 48 0.3568908 0.07032600
## 49 49 0.3545518 0.06870027
## 50 50 0.3474510 0.06103536

plot(knn.cross)
```




```
library(e1071)

knn.boot <- tune.knn(x =scale( Vehicle[,1:18]),
y =Vehicle[,19], k = 1:50,
tunecontrol=tune.control(sampling = "boot"))

summary(knn.boot)

##
## Parameter tuning of 'knn.wrapper':
##
```

```
## - sampling method: bootstrapping
##
## - best parameters:
##  k
##  5
##
## - best performance: 0.2974757
##
## - Detailed performance results:
##     k     error dispersion
## 1   1 0.2991389 0.02445726
## 2   2 0.3079881 0.03248799
## 3   3 0.3052867 0.02588796
## 4   4 0.3073626 0.01513020
## 5   5 0.2974757 0.01491942
## 6   6 0.3079482 0.01226655
## 7   7 0.3036189 0.01350257
## 8   8 0.3035910 0.01242989
## 9   9 0.3041797 0.01595243
## 10 10 0.3150015 0.02336514
## 11 11 0.3143873 0.02044466
## 12 12 0.3125477 0.01734655
## 13 13 0.3187170 0.02621171
## 14 14 0.3175134 0.02538507
## 15 15 0.3262998 0.02689979
## 16 16 0.3227980 0.02626607
## 17 17 0.3204269 0.02387950
## 18 18 0.3258065 0.02381407
## 19 19 0.3228616 0.02535571
## 20 20 0.3211055 0.02317985
## 21 21 0.3246009 0.03023085
## 22 22 0.3254797 0.02449033
## 23 23 0.3243163 0.02591252
## 24 24 0.3227733 0.02440027
## 25 25 0.3192968 0.02491439
## 26 26 0.3280465 0.02379741
## 27 27 0.3248552 0.02206792
## 28 28 0.3286682 0.02387417
## 29 29 0.3268313 0.02304623
## 30 30 0.3321571 0.01968519
## 31 31 0.3353551 0.01829203
## 32 32 0.3303599 0.01936890
## 33 33 0.3368531 0.02092598
## 34 34 0.3394414 0.02040601
## 35 35 0.3341791 0.02058210
## 36 36 0.3376787 0.02099048
## 37 37 0.3388531 0.02361523
## 38 38 0.3417949 0.02696245
## 39 39 0.3421185 0.02541945
## 40 40 0.3421388 0.01980033
```

```
## 41 41 0.3403722 0.01963342
## 42 42 0.3439491 0.02306329
## 43 43 0.3436483 0.02438761
## 44 44 0.3445189 0.02508200
## 45 45 0.3486602 0.02407796
## 46 46 0.3462905 0.02155498
## 47 47 0.3500566 0.02081695
## 48 48 0.3532954 0.01964215
## 49 49 0.3562508 0.02311442
## 50 50 0.3550709 0.02575686

plot(knn.boot)
```

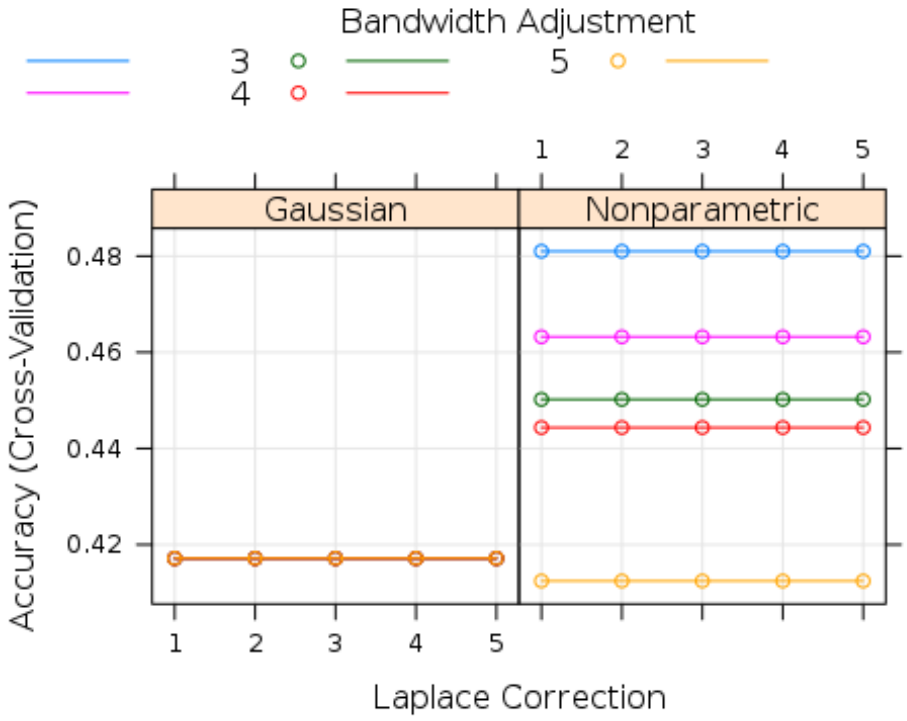


Essayer avec moins de régresseurs, voir si c'est ok en termes d'erreur par validation, puis vérifier avec prédict l'erreur sur l'échantillon test. Même type de programme que pour l'exo 1.

4. Règle de décision de la diapo 78 (mais là il y aura 4 modalités possibles pour Class) Là il faudra utiliser une sélection de régresseurs (renseigner selreg) car ça plante sinon…et tester avec des prétraitements (diapo 89) qui peuvent aider puisque les régresseurs sont loin d'être normaux.

```
grid<- expand.grid(usekernel=c(TRUE,FALSE),fL=1:5,adjust=seq(1,5,by=1))
control<-trainControl(method="cv",number=10)

selreg<- as.data.frame(Vehicle[,1:2]) #mettre ici votre selection de colonnes
regresseurs à modifier !
names(selreg)<-c(names(Vehicle)[1:2]) #à modifier
# train model
```

```
NB1exo2<-train(x=selreg,y=Vehicle$Class,
               preProc=c("BoxCox","center","scale","pca"),
               method="nb",trControl=control,tuneGrid=grid)

#plot search grid results avec library(ggplot2)

plot(NB1exo2)
```



Ici le meilleur modèle en validation croisée 10-blocs : estimation non paramétrique de densité à noyau gaussien avec… en fonction des résultats

Exo 3

1. Ici je pense que les seuls régresseurs utilisables pour ce particulier qui possède déjà son logement sont les variables de localisation (peut-être une sélection parmi les 4), et le room_type listing

2. Ici la variable à expliquer est quatntitative => on utilise la perte quadratique (diapo 17)

3. Cette fois la règle de décision sera basée sur un calcul de moyenne locale de Y (et non assimilable à un vote à la majorité entre des modalités) cf diapo 57 et 58.

4. et 5. reprendre le même canevas de programmes que précédemment