Table des matières

1. Prétraitement

Traitement des valeurs manquantes

Parmi les 6000 observations dans la base train, on n'en retrouve qu'environ 1000 avec toutes les valeurs des covariables complètes. Pour faire une analyse correcte et inclusive des données, on propose de trouver des remplacements par différentes méthodes. Dans cette section on discutera les méthodes utilisées ainsi que la justification de leur utilisation lors de l'entraînement de notre outil de prédiction.

Méthode 1 : remplacement par les moyennes

Sachant que les valeurs manquantes dans la base ne sont que de caractère numérique, on pourrait penser à calculer une moyenne par variable (utilisant l'information disponible) et remplacer les valeurs manquantes par la moyenne correspondante. Cette méthode étant simpliste, on pourrait l'améliorer en utilisant l'information fournie par le reste des variables pour essayer de prédire la valeur manquante.

Méthode 2 : prédiction via un modèle linéaire

Une autre manière de traiter les valeurs manquantes est d'ajuster un modèle linéaire pour chacune des variables dont on sait qu'il y a des valeurs manquantes (bien entendu, par rapport aux autres variables sauf charge). Cette méthode généralise l'approche expliquée précédemment (celle de remplacement par la moyenne) car au cas où il n'existe pas de relation linéaire entre les covariables et la variable à expliquer, le modèle linéaire prédira la moyenne des variables de sorties. Cette méthode est donc une meilleure approche que celle proposée ci-dessus car on profite d'une éventuelle relation linéaire entre les covariables pour mieux prédire les valeurs manquantes.

La méthode implémentée dans le programme (fonction `predict_NA`) est comme suit : étant donnée une covariable à prédire, Y, et les variables autres que charges, on effectue une sélection des variables à utiliser via la fonction `stepAIC` en utilisant les données de la base dont toutes les variables (y compris Y) sont complètes. Une fois les features sélectionnées, on prédit toutes les valeurs de la base dont la valeur de Y est NA et dont le reste des covariables sélectionnées sont complètes via un modèle linéaire (`lm`). On remplace finalement la base d'origine par la base avec les nouvelles prédictions et on répète cette procédure avec les autres covariables.

Méthode 3 : remplacement par les valeurs données

Les valeurs données dans les instructions sont considérées des valeurs jugées adéquates par les médecins experts. Ces valeurs sont utilisées dans notre programme dans le cas où l'on ne disposera pas de méthode fiable de prédiction.

Dans notre programme, on utilise d'abord la méthode de prédiction via un modèle linéaire (cf. méthode 2 ci-dessus) pour essayer de remplir le plus possible de valeurs manquantes avec une fiabilité certaine. On parvient avec cette méthode à prédire 3000 observations (complètes) additionnelles, soit un total de 4000 observations sur 6300 dans la base d'origine. Ensuite, on remplace le reste des valeurs manquantes en utilisant la méthode 3, i.e. en utilisant les données jugées correctes par les experts.

Transformation des variables

Dans le but d'améliorer le pouvoir prédictif de notre modèle, on se propose d'évaluer la pertinence linéaire des covariables par rapport à la variable à expliquer, charges. En effectuant des transformations non-linéaires sur les variables, on trouvera des relations plus pertinentes (au sens du R2 pour la régression simple) entre

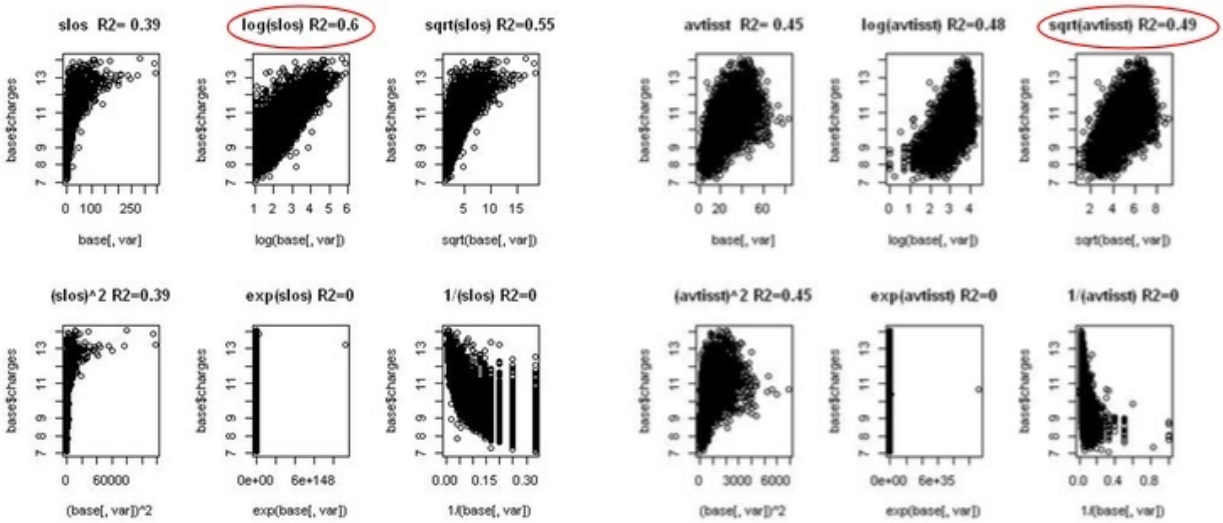
charges et le reste des covariables. Pour ce faire, on a développé une fonction qui permet de visualiser sur un même écran l'ajustement de la variable charges par rapport à des différentes transformations des variables à expliquer. On effectuera ensuite une transformation sur les covariables si leur mesure R2 est bonne par rapport aux autres transformations proposées (log, racine carrée, carré, fonction inverse, et l'exponentielle).

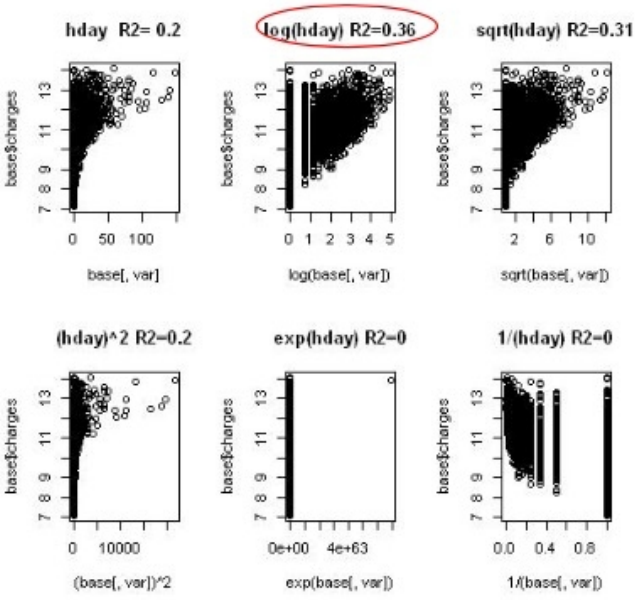
Transformation de charges

Une étude des différentes combinaisons de transformations sur charges et ses covariables nous amène à considérer une transformation de type logarithmique sur cette variable. Ceci est dû au fait que les R2 retrouvées (voir graphiques ci-dessous) seront bien meilleurs lorsqu'on effectue une transformation logarithmique de charges. Toute prédiction de charges nécessitera donc une retransformation par l'exponentielle.

Covariables à transformer

En utilisant l'outil de visualisation par les différentes transformations, on se retrouve avec les graphiques suivants :





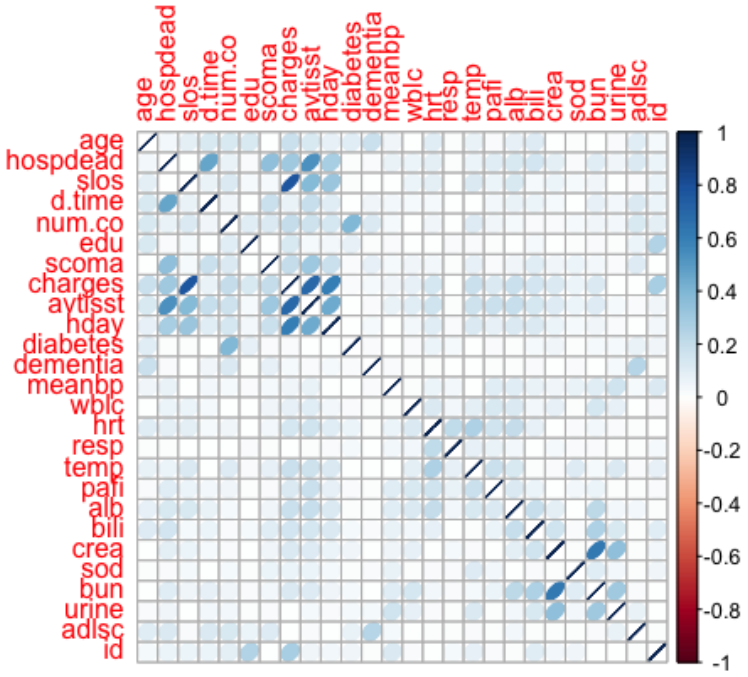
On observe que l'ajustement du modèle est meilleur lorsqu'on considère les transformations suivantes :

- `log(hday)`
- `sqrt(avtisst)`
- `log(slos)`

Aucune relation importante n'a été trouvée sur les autres variables. De plus, leur coefficient R2 est très faible (<<0.1) comparé avec celle des variables `hday, avtisst` et `slos`. Ceci nous amènera à ne garder que ces trois variables dans certaines étapes de notre étude (voir section 3 pour plus de détails).

2. Conditionnement de la matrice des observations

Après avoir effectué ces transformations, on étudie la présence d'éventuelles colinéarités dans la matrice des observations. On commence donc par calculer la matrice de corrélation des variables continues. Ci- dessous sa représentation (en valeur absolue) sous forme de heatmap :



Les couleurs plus foncées correspondant aux corrélations les plus élevées, on constate qu'on a peu de corrélations élevées, sauf pour le groupe de variables `hday`, `slos`, `charges`, `avtisst`.

Si on s'intéresse plus particulièrement aux corrélations avec la variable à expliquer `charges`, on obtient :

```
> sort(abs(mat_cor["charges",]))
      resp      adlsc      urine    diabetes    dementia      meanbp
0.001457060 0.002469398 0.005260377 0.022795789 0.033101756 0.054324990
       sod     d.time       wblc       crea       pafi        bun
0.067220538 0.075107713 0.085931844 0.120902567 0.124279617 0.128761975
       edu        hrt       bili        alb        age       temp
0.130732963 0.146834496 0.156819988 0.181995631 0.182632600 0.188281723
    num.co      scoma    hospdead       hday    avtisst       slos
0.204454802 0.209925553 0.302954670 0.600456317 0.697287087 0.776021689
   charges
1.000000000
```

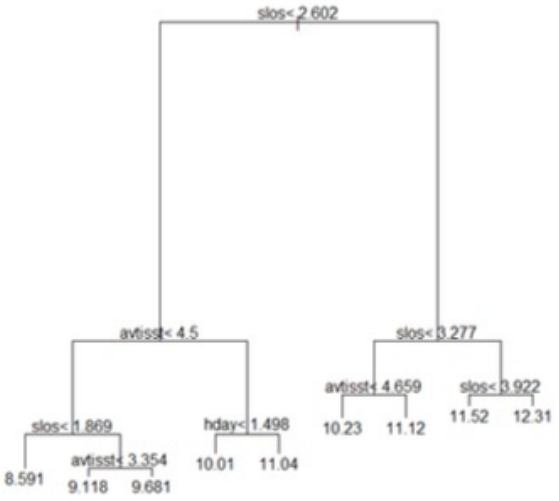
Seules 3 variables ont une forte corrélation linéaire avec la variable charges : `slos`, `avtisst`, `hday`.

De plus, le rapport de la plus grande valeur propre et de la plus petite valeur propre de la matrice de corrélations des variables explicatives est d'environ 53.28. Enfin, les coefficients de VIF sont tous < 2.04. On peut donc conclure qu'on n'a pas de problème de colinéarité entre les variables explicatives.

3. Choix des variables

On vient de voir que parmi les variables explicatives, 3 semblent avoir plus d'importance que les autres en se basant sur les corrélations : `slos`, `avtisst`, `hday` (après transformation). Plusieurs autres arguments vont dans ce sens :

1. Lorsque l'on regarde la signification de ces variables, il semble logique qu'elles expliquent la variable charges :
    - `hday` : nombre de jours à l'hôpital
    - `slos` : nombre de jours dans l'étude
    - `avtisst` (Therapeutic Intervention Scoring System) : un score qui est un indicateur des coûts des soins à l'hôpital.
2. Ce sont les seules variables pour lesquelles on observe une relation particulière avec la variable à expliquer lorsqu'on trace les nuages de points.
3. La procédure de sélection automatique de R (`stepAIC`) choisit ces variables. De plus, lorsqu'on analyse les résultats de cette procédure, on constate qu'ajouter ou retirer ces variables fait fortement varier l'AIC, ce qui n'est pas le cas pour les autres variables.
4. Un modèle linéaire avec toutes les variables donne un R2 de 0.89, contre 0.84 si on ne choisit que les 3 variables `slos`, `avtisst`, `hday`. L'ajout de toutes les autres variables dans le modèle ne fait donc pas beaucoup augmenter le R2, la variance est donc essentiellement expliquée par ces 3 variables. De plus les p-values associées sont < 2.10^-16.
5. La régression pénalisée LASSO met à 0 les coefficients d'un grand nombre de variables (ou modalités) mais conserve ces 3 variables.
6. On utilise un algorithme CART pour construire un arbre de régression pour prédire charges en fonction de toutes les autres variables. Cet algorithme est capable de détecter des liens non linéaires. On obtient l'arbre suivant :

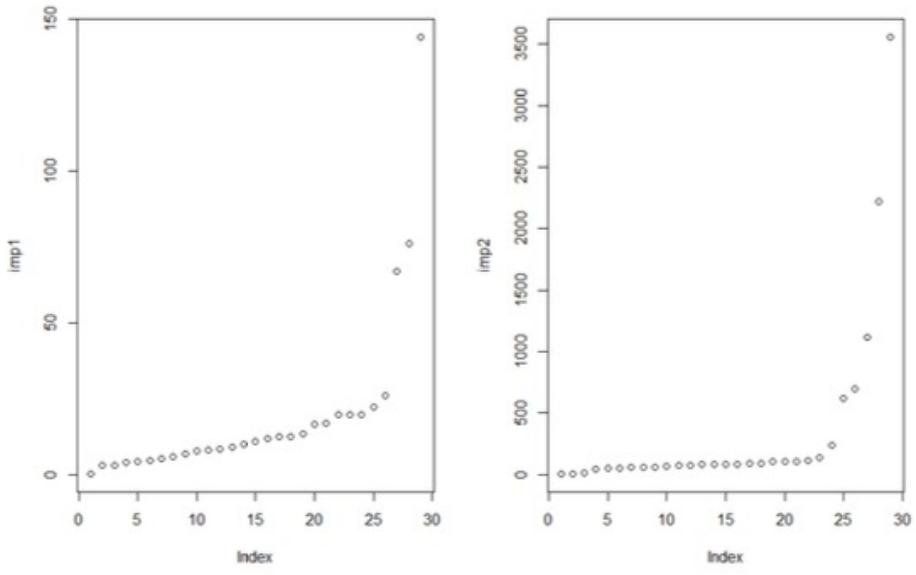


On constate qu'après élagage de l'arbre, seules 3 variables interviennent : `slos`, `avtisst`, et dans une moindre mesure `hday`.

Les forêts aléatoires combinent le principe de l'arbre de régression avec celui du bootstrap pour construire un grand nombre d'arbres. Chacun de ces arbres est construit avec un échantillon bootstrap des données d'origine (aléatoire avec remise) et à chaque nœud, un sous-ensemble aléatoire de variables est choisi. Le package R randomForest contient une implémentation de cet algorithme qui produit deux mesures différentes de l'importance des variables :

```
> importance(foret)
               %IncMSE IncNodePurity
age        19.64721680    133.598173
sex         0.08283853     10.951120
hospdead    9.09601669    116.336085
slos      144.31303634   3559.277886
d.time     19.58633029    239.650599
dzgroup    22.14134143    697.981213
dzclass    16.76058094    619.194156
num.co     13.33322324     52.803388
edu        26.10992807    102.281570
scoma      10.80604031     87.162461
avtisst    76.22549964   2218.001084
race       11.80240902     44.431658
hday       66.90927855   1116.508175
diabetes    3.97499541      8.514189
dementia    3.04863291      2.908874
ca         12.43269571     56.660728
meanbp      9.78597114     87.243509
wblc        5.90539882     83.777996
hrt         3.04895426     81.830633
resp        5.19909644     61.385839
temp        6.73333014     79.331379
pafi        4.61920863     77.685619
alb         7.88677917     63.059947
bili       16.43080235    105.378410
crea        8.40329315     63.767629
sod         4.34905952     73.143786
bun        12.46088362     81.298871
urine      19.59005092    102.687698
adlsc       8.12660628     54.776379
```

Ces deux mesures d'importance classent les variables `slos`, `avtisst`, `hday` (dans cet ordre) devant les autres variables, avec une nette avance, comme le montrent les graphes suivants :



Cette forêt aléatoire explique 87.21% de la variance. Lorsqu'on n'inclut dans l'algorithme que 3 les variables les plus importantes (au sens précédent), on obtient une forêt qui explique 81.97% de la variance.

Pour les différentes raisons que nous venons d'énoncer, il semble que seules 3 variables ont un impact significatif sur la variable à expliquer. Inclure davantage de variables dans le modèle améliorerait peu l'ajustement, et risquerait d'induire un risque de sur-apprentissage. On considèrera donc dans la suite un modèle linéaire de la forme `charges~slos+avtisst+hday`.

4. Analyse des résultats de la régression

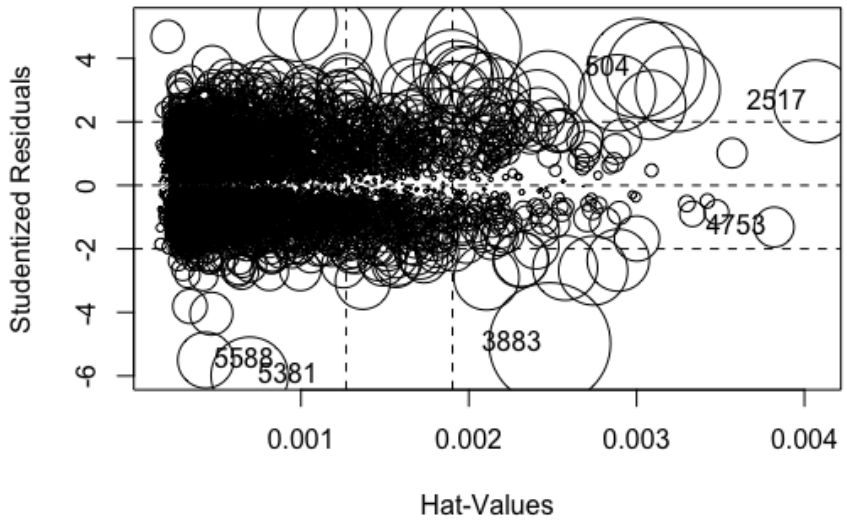
Le modèle linéaire `charges~slos+avtisst+hday` (sur les variables transformées) donne R2 = 0.8378 et toutes les variables sont significatives au sens des test de Student (p-values < 2.10^-16). Les coefficients sont les suivants :

```
> coef(modele)
(Intercept)        slos     avtisst        hday
  6.2751928   0.8567880   0.3407187   0.3121579
```

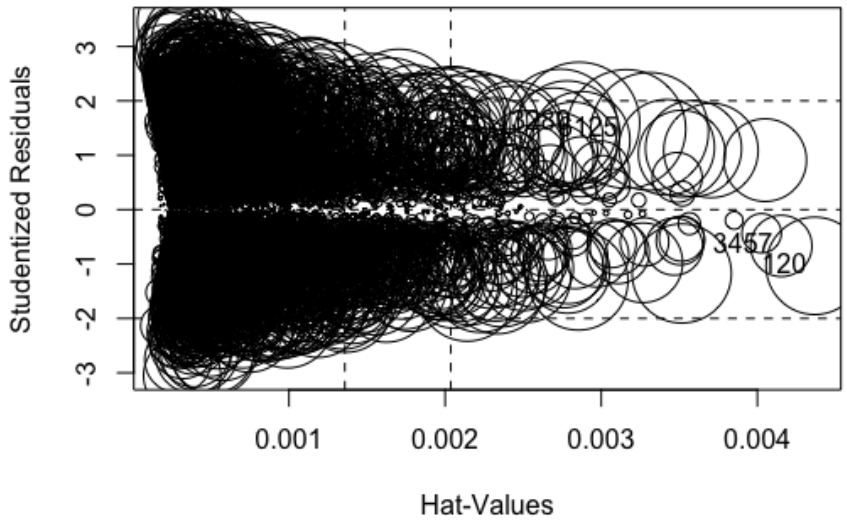
Analyse des résidus

Avant d'effectuer le choix du modèle à utiliser, il est pertinent de réaliser une analyse des possibles valeurs aberrantes qui pourraient rendre la prédiction de charges mauvaise. On cherche donc à trouver les observations influentes et les enlever pour avoir une estimation robuste des paramètres du modèle. Pour ce faire, on propose d'effectuer une régression linéaire par rapport aux variables les plus influentes (`avtisst`, `slos` et `hday`), pour ensuite étudier le comportement des résidus et des leviers.

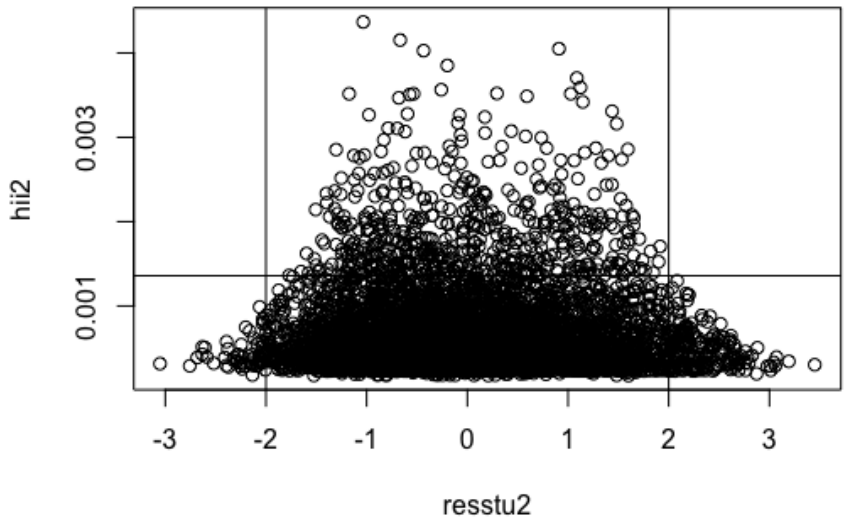
Une première graphique d'influence en utilisant influencePlot, qui montre résidus (y) contre leviers (x) contre distance de Cook (z, taille de la bulle), est donné par le graphique suivant.



On peut remarquer qu'il y a des observations avec des résidus studentisés très élevés, ainsi que des valeurs avec des leviers très hauts par rapport à la tolérance de 2p/n. La taille des bulles étant très significative pour ces observations, on peut se douter que leur influence sur l'estimation des paramètres sera importante. On enlève donc les observations avec une distance de Cook élevée (supérieure à 6/n, une limite moins stricte que le 4/n proposé faute de normalité des résidus) dans un premier temps. Une fois ces observations enlevées, on se retrouve avec une influence Plot donnée par le graphique suivante :

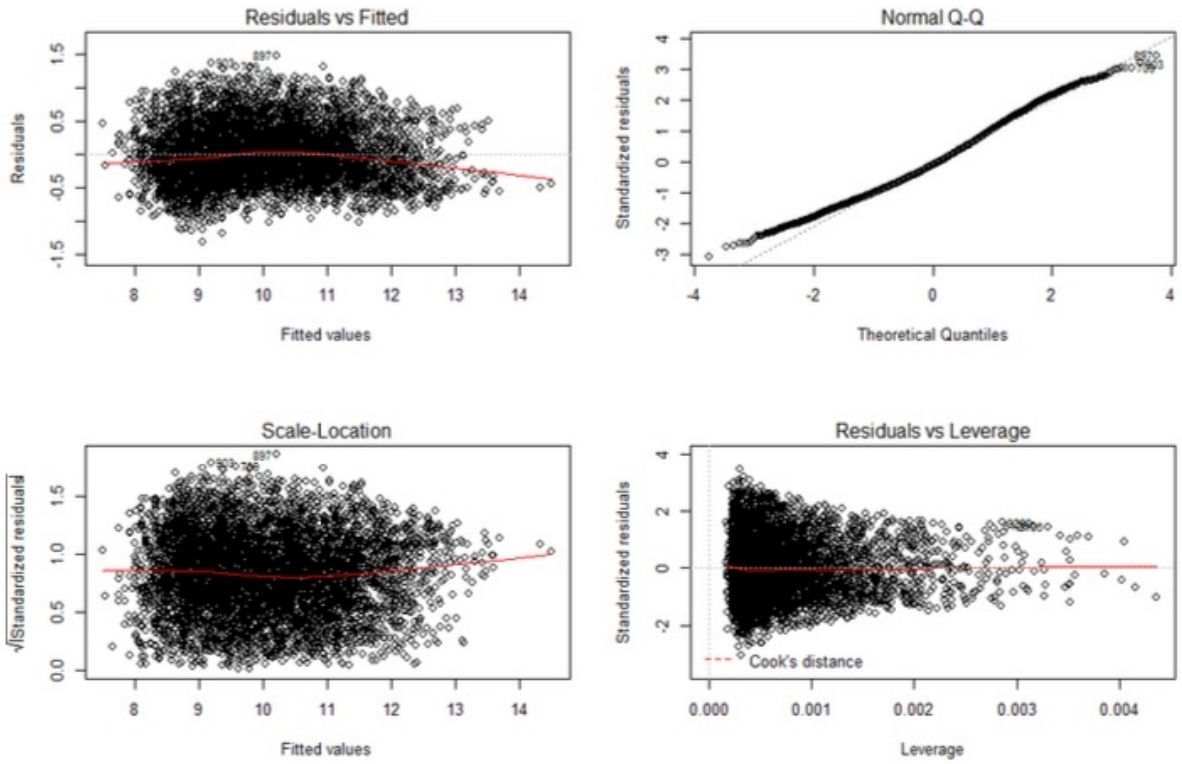


On retrouve donc un graphique leviers (y) contre résidus studentisés (x) donnée par le graphique suivant :



On observe donc qu'il n'y a plus d'observations qui se retrouve dans les zones d'influence « extrême » (i.e. celle d'en haut à gauche et en haut à droite). Même si l'on se retrouve avec des leviers et des résidus hors des bornes de tolérance, enlever ces observations aurait un impact radical sur le modèle étant donné le nombre élevé d'observations qui seraient enlevées par cette procédure.

Après avoir retiré les observations trop influentes, le modèle linéaire précédent donne un R2 légèrement amélioré à 0.8708. Les différents graphiques associés à ce modèle sont les suivants :



L'analyse des résidus donne des résultats satisfaisants (indépendance, homoscédasticité), même si la normalité des résidus n'est pas assurée (les queues de distribution sont trop fines, ce qui peut s'expliquer par les observations influentes qui ont été éliminées).

5. Choix du modèle

Modèles linéaires

Tout au long des dernières sections, on s'est servi de la forte relation linéaire qu'il existe entre `log(charges)` et les covariables. Grâce à ce phénomène, on considère qu'une simple régression linéaire de charges (sans transformation par log) contre les autres variables ne fournira pas de bons résultats. De plus, on ne peut pas effectuer une prédiction de `log(charges)` puis élever à l'exponentielle car on perdrait toutes les propriétés du modèle linéaire (et on n'aurait plus un estimateur non-biaisé). Au même temps, on ne peut pas assurer un contrôle de la variance de notre estimateur. On est donc obligés d'adopter une méthode qui nous permette de capturer les relations non-linéaires qu'il existe entre charges et les covariables.

Méthodes non-linéaires

Le traitement des valeurs aberrantes via le modèle linéaire nous permet d'assurer que les modèles non-linéaires seront plus stables lors de l'entraînement. On dispose de deux méthodes pour effectuer la prédiction avec des relations non-linéaires : les arbres de régression et les réseaux de neurones.

Dans cette section on discutera les différentes méthodes disponibles et on justifiera le choix de notre modèle de prédiction.

Arbres de régression

Dans la section 3, nous avons utilisé les arbres de régression et les forêts aléatoires pour sélectionner les variables en entrée d'un modèle linéaire. Bien entendu, ces algorithmes peuvent être utilisés directement pour construire un modèle et réaliser des prédictions. Ils sont cependant plus instables qu'une régression linéaire, au sens où une petite variation d'une variable explicative peut faire passer un individu d'une feuille à une autre, et donc faire varier largement la valeur prédite (le modèle n'est pas continu mais constant par morceaux).

Réseaux de neurones

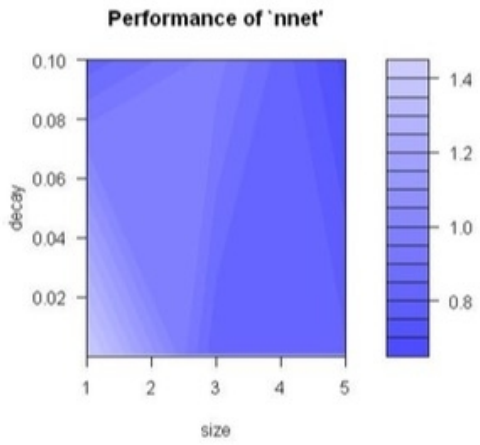
Le package `nnet` nous permet d'implémenter une méthode non-linéaire très utilisée dans l'industrie : le perceptron multi-couches. Rappelons qu'un perceptron multi-couches est un réseau neuronal conformé de trois couches (typiquement) et avec des fonctions d'activation (i.e. les valeurs propagées vers les neurones de la couche cachée et la couche de sortie) non-linéaires telles que la fonction sigmoïde. Un avantage important de l'utilisation des réseaux de neurones consiste à pouvoir identifier des relations non-linéaires entre les variables d'entrée. En utilisant la fonction `nnet` du package éponyme, on peut apprendre notre réseau avec la base `data_train` de manière analogue à l'appel des fonctions du type `lm`.

En fixant taille de la couche cachée à 5, et la valeur du paramètre de régularisation à 0.001 (tous les deux choisis par une méthode de cross-validation via la fonction `tune.nnet`, qui demande beaucoup de temps de calcul machine), on a prédit charges en fonction de toutes les covariables espérant capturer les possibles relations non-linéaires avec les covariables autres que `slos`, `hday` et `avtisst`. La commande est la suivante :

(*)
```
regrnett <- nnet(charges~., data=data_train2, size = 5, decay = 0.001, linout=TRUE, maxit=3000,na.action = na.omit)
```

Remarque : l'option `linout=TRUE` est utilisée pour rendre la couche de sortie linéaire et non pas de type sigmoïde (qui donnerait comme sortie des valeurs entre 0 et 1).

La graphique qui nous a permis d'obtenir la valeur des méta-paramètres (fonction `tune.nnet`) est la suivante :




```
Detailed performance results:
  size decay     error dispersion
1    1 1e-01 0.8170145 0.33657774
2    3 1e-01 0.9733499 0.28420457
3    5 1e-01 0.6817920 0.13065675
4    1 1e-03 1.4204185 0.05567440
5    3 1e-03 0.8092671 0.33074283
6    5 1e-03 0.8071836 0.36099394
7    1 1e-06 1.4391697 0.05893708
8    3 1e-06 1.3429458 0.24746473
9    5 1e-06 1.2441747 0.34669902
```

Plus la valeur est foncée, meilleure est l'erreur cross-validée (à échelle logarithmique de `charges`) retrouvée dans la zone de la grille correspondante. La valeur d'erreur cross-validée via la fonction `tune.nnet` appliquée au modèle utilisé dans les sections précédentes est de l'ordre de 0.807.

Les prévisions envoyées proviennent de ce modèle en utilisant la commande libellée par (*).

Annexe

```
library('MASS')
library(glmnet)
library(corrplot)

data_train<-read.csv("data_train.csv",sep=",")
data_test<-read.csv("data_test.csv",sep=",")

##############################################
#            PRETRAITEMENT                   #
##############################################

####### Train ##########

#Liste des variables qui ne sont pas factor
factor<-sapply(data_train,is.factor)
namesnotfactor<-names(factor[factor==FALSE])
namesnotfactor

var_avecNA<-c()
for (i in namesnotfactor) {
  if (sum(is.na(data_train[,i]))>0) {
    var_avecNA<-c(var_avecNA,i)
  }
}

####REMPLIR LES DONNEES MANQUANTES

#On trouve d'abord quelles sont les observations avec des valeurs
manquantes
obs_NA<-c()
NA_parvariable<-list()
for (i in var_avecNA) {
  obs_NA<-union(which(is.na(data_train[,i])),obs_NA)
  NA_parvariable[[i]]<-which(is.na(data_train[,i]))
}    #obs_NA contient donc les numeros d'observations qui ont des
valeurs manquantes, et NA_parvariable est une liste contenant un
vecteur des observations qui ont des valeurs manquantes

#On obtient combien de chaque variable on a des NA
combien_NA<-c()
for (k in var_avecNA) {
  combien_NA[k]<-length(NA_parvariable[[k]])
}
combien_NA

###METHODE 1: regression lineaire pour predire les valeurs

#Fonctions auxiliaires
```

```
writemodel<-function(apredire,variables) {

return(formula(paste(apredire,"~",paste(variables,collapse="+"))))
}
#La fonction suivante remplira les valeurs NA de la base 'data' en
faisant une regression par rapport aux autres variables, utilisant
les observations qui n'ont pas de NAs dans les variables
significatives
predict_NA<-function(data,var_predire) {
  model<-
writemodel(var_predire,setdiff(colnames(data),c("charges","id")))
  forwardmodel<-
stepAIC(lm(model,data=na.omit(data)),direction="backward",na.actio
n="omit")

  #Predictions
  data<-cbind(data,IDOBS=1:dim(data)[1])
  var_gardees<-attr(forwardmodel$terms,"term.labels")

  #Enlever les observations que l'on ne peut pas predire
  data_pour_predire<-na.omit(data[is.na(data[,var_predire]),
(colnames(data) %in% c(var_gardees,"IDOBS") )]) #Les donnees avec
NA dans la variable mais sans NA dans les variables qui nous
servent pour predire

  #Obtenir variables qui sont factor
  types<-sapply(data_pour_predire,class)
  factors<-names(types[types=="factor"])

  #On va verifier si les observations a predire ont les 'levels'
que l'on peut predire. (Il a fallu faire ca car parfois on ne
pouvait pas predire a cause d'un nouvel 'level' dans une
observation nouvelle)
  if (length(factors)>0) {
    predictable<-c()
    for (i in 1:nrow(data_pour_predire)) {
      predictable[i]<-
prod((as.matrix(data_pour_predire[i,factors]) %in%
unlist(forwardmodel$xlevels)))
    }
    data_pour_predire<-data_pour_predire[as.logical(predictable),]
  }
  #On predit les valeurs manquantes
  predictions<-predict(forwardmodel,newdata=data_pour_predire)
  print("On a predit"); print(length(predictions));
print("nouvelles observations")
  data[data$IDOBS %in% data_pour_predire$IDOBS,var_predire]<-
predictions
  data<-data[,!(colnames(data) %in% c("IDOBS"))]
  return(data)
}
```

```
#Effectuer la prediction pour toutes les variables manquantes
data_NApred<-data_train
for (variable in var_avecNA) {
  data_NApred<-predict_NA(data_NApred,variable)
}

#Attention: avec la methode d'au dessus on ne peut predire que les
observations qui n'ont qu'une seule valeur NA dans les variables
representatives. Il nous en resteront quelques unes sans pouvoir
les predire
#Regarder combien d'observations nous restent
for (i in names(data_NApred)) {
  if (sum(is.na(data_NApred[,i]))>0) {
    print(i)
    print(sum(is.na(data_NApred[,i])))
  }
}
dim(na.omit(data_NApred))

#Finalement: remplacer les obs que l'on ne peut pas predire par
les valeurs qu'elle a donne
fillNA<-
data.frame(alb=3.5,pafi=333.3,bili=1.01,crea=1.01,bun=6.51,wblc=9,
urine=2502)
for (i in names(fillNA)) {
  data_NApred[is.na(data_NApred[,i]),i]<-fillNA[,i]
}

#Enlever donnees avec des valeurs manquantes
data_NApred<-na.omit(data_NApred)

#On choisit data_NApred comme l'ensemble d'apprentissage (outil
pour predire)
#(ne pas executer cette ligne si on ne veut pas choisir cette base
la)
data_train<-data_NApred



###METHODE 2: remplacer directement (sans predire) par les valeurs
qu'elle a donne
#On remplit le reste des donnees avec cette methode
fillNA<-
data.frame(alb=3.5,pafi=333.3,bili=1.01,crea=1.01,bun=6.51,wblc=9,
urine=2502)
for (i in names(fillNA)) {
  data_train[is.na(data_train[,i]),i]<-fillNA[,i]
}

#Enlever donnees avec des valeurs manquantes (s'il en reste)
data_train<-na.omit(data_train)
```

```
###METHODE 3: remplacer par les moyennes de chaque variable

fillNA=colMeans(data_train[,var_avecNA],na.rm=T)
for(var in var_avecNA){
     data_train[is.na(data_train[,var]),var]<-fillNA[var]
}

data_train[data_train$urine<0,"urine"]<-0



######## Test ############

#Liste des variables qui ne sont pas factor
factor_test<-sapply(data_test,is.factor)
namesnotfactor_test<-names(factor_test[factor_test==FALSE])
namesnotfactor_test

var_avecNA_test<-c()
for (i in namesnotfactor_test) {
  if (sum(is.na(data_test[,i]))>0) {
    var_avecNA_test<-c(var_avecNA_test,i)
  }
}

#On trouve d'abord quelles sont les observations avec des valeurs
manquantes
obs_NA_test<-c()
NA_parvariable_test<-list()
for (i in var_avecNA_test) {
  obs_NA_test<-union(which(is.na(data_test[,i])),obs_NA_test)
  NA_parvariable_test[[i]]<-which(is.na(data_test[,i]))
}    #obs_NA contient donc les numeros d'observations qui ont des
valeurs manquantes, et NA_parvariable est une liste contenant un
vecteur des observations qui ont des valeurs manquantes

#On obtient combien de chaque variable on a des NA
combien_NA_test<-c()
for (k in var_avecNA_test) {
  combien_NA_test[k]<-length(NA_parvariable_test[[k]])
}
combien_NA_test

##METHODE 1: Effectuer la prediction pour toutes les variables
manquantes
test_NApred<-data_test
for (variable in var_avecNA) {
  test_NApred<-predict_NA(test_NApred,variable)
}
```

```
#Attention: avec la methode d'au dessus on ne peut predire que les
observations qui n'ont qu'une seule valeur NA dans les variables
representatives. Il nous en resteront quelques unes sans pouvoir
les predire
#Regarder combien d'observations nous restent
for (i in names(test_NApred)) {
  if (sum(is.na(test_NApred[,i]))>0) {
    print(i)
    print(sum(is.na(test_NApred[,i])))
  }
}
dim(na.omit(test_NApred))

#Choisir train_NApred comme la base a utiliser (ne pas executer
cette ligne si on ne veut pas choisir cette base la)
data_test<-test_NApred

##METHODE 2:Remplir donnees manquantes
fillNA<-
data.frame(alb=3.5,pafi=333.3,bili=1.01,crea=1.01,bun=6.51,wblc=9,
urine=2502)
for (i in names(fillNA)) {
  data_test[is.na(data_test[,i]),i]<-fillNA[,i]
}

#Regarder les variables encore avec des valeurs manquantes
for (i in names(data_test)) {
  print(i)
  print(sum(is.na(data_test[,i])))
}

#Predire avtisst (on n'a pas de remplacement et il ne nous reste
qu'une observation)
data_test<-predict_NA(data_test,"avtisst")

###METHODE 3: remplacer par les moyennes de chaque variable

#Regarder les variables avec des valeurs manquantes
fillNA_test=colMeans(data_test[,var_avecNA_test],na.rm=T)
for(var in var_avecNA_test){
  data_test[is.na(data_test[,var]),var]<-fillNA_test[var]
}


sum(is.na(data_test))

### Avant de continuer, corriger les valeurs sans aucun sens:

data_test[data_test$urine<0,"urine"]<-0


######## Transformation des variables ############
```

```
# fonction pour visualiser charges contre les r?gresseurs
transform?s par diff?rentes fonctions

data_train$charges <- log(data_train$charges)


scatter<-function(var,base) {
  par(mfrow=c(2,3))
  R2<-round(summary(lm(base$charges~base[,var]))
$adj.r.squared,2) ;
plot(y=base$charges,x=base[,var],main=paste(var,"
R2=",eval(R2),collapse=""))
  R2<-round(summary(lm(base$charges~log(base[,var])))
$adj.r.squared,2) ;
plot(y=base$charges,x=log(base[,var]),main=paste(c("log(",var,")
R2=",eval(R2)),collapse=""))
  R2<-round(summary(lm(base$charges~sqrt(base[,var])))
$adj.r.squared,2) ;
plot(y=base$charges,x=sqrt(base[,var]),main=paste(c("sqrt(",var,")
R2=",eval(R2)),collapse=""))
  R2<-round(summary(lm(base$charges~(base[,var])^2))
$adj.r.squared,2) ;
plot(y=base$charges,x=(base[,var])^2,main=paste(c("(",var,")^2
R2=",eval(R2)),collapse=""))
  R2<-round(summary(lm(base$charges~exp(base[,var])))
$adj.r.squared,2) ;
plot(y=base$charges,x=exp(base[,var]),main=paste(c("exp(",var,")
R2=",eval(R2)),collapse=""))
  R2<-round(summary(lm(base$charges~1/base[,var]))
$adj.r.squared,2) ;
plot(y=base$charges,x=1/(base[,var]),main=paste(c("1/(",var,")
R2=",eval(R2)),collapse=""))
}

scatter("age",data_train) #Peu significative comme variable..
scatter("slos",data_train)  #La transformer en log semble bien vu
le R2
scatter("avtisst",data_train) #Sqrt semble bien
scatter("temp",data_train) #Tres peu explicative
scatter("bili",data_train) #Tres peu explicative
scatter("bun",data_train) #Tres peu explicative
scatter("hday",data_train) #log semble bien

par(mfrow=c(1,1))

data_train$avtisst <- sqrt(data_train$avtisst)
data_train$slos <- log(data_train$slos)
data_train$hday <- log(data_train$hday)

data_test$avtisst <- sqrt(data_test$avtisst)
data_test$slos <- log(data_test$slos)
```

```
data_test$hday <- log(data_test$hday)

##############################################
#        CONDITIONNEMENT DE LA MATRICE       #
##############################################

# Calcul de la matrice de corr?lation

mat_cor<-cor(data_train[,namesnotfactor])
heatmap(abs(mat_cor))
# avec corrplot
corrplot(mat_cor, method = "ellipse")
corrplot(abs(mat_cor), method = "ellipse")

# Corr?lations avec la variable ? expliquer

mat_cor["charges",]

# valeurs propres de la matrice de corr?lation

vp=eigen(mat_cor)$values
max(vp)/min(vp)   # << 1000

library(car)

# Calcul du VIF

modlin=lm(charges~.,data=data_train[,namesnotfactor])
vif=vif(modlin)
max(vif)  # < 10

##############################################
#        SELECTION DES VARIABLES             #
##############################################

# Selection backward

back_AIC=stepAIC(lm(charges~.,data=data_train),direction="both")
summary(back_AIC)

# Selection par LASSO

Y<-data_train[,"charges"]
# TOUTES LES VARIABLES
X<-model.matrix(~.,data_train[,!(colnames(data_train) %in%
c("charges"))])
cvLasso<-
cv.glmnet(as.matrix(X),as.matrix(Y),family="gaussian",alpha=1)
coef_lasso=coef(cvLasso)
coef_lasso[(coef_lasso[,1]==0),]

# p-values et R?
```

```
modlin=lm(charges~.,data=data_train)
# adjR? = 0.89
modlin2=lm(charges~hday+slos+avtisst,data=data_train)
# adjR? = 0.84
modlin3=lm(charges~edu+scoma+avtisst+hday+meanbp+temp+bili+bun+adl
sc+age+hospdead+slos+d.time+dzgroup,data=data_train)
#adjR? = 0.86

# Arbre de r?gression

library(rpart)
library(randomForest)
arbre=rpart(charges~.,data=data_train[,!(colnames(data_train) %in%
c("id"))])
plot(arbre)
text(arbre)

#foret=randomForest(charges~.,data=data_train,importance=T)
#plot(foret)
#imp1=importance(foret)[,1]
#plot(names(imp1),imp1)
#imp2=importance(foret)[,2]
#imp1=imp1[order(imp1)]
#plot(imp1)
#imp2=imp2[order(imp2)]
#plot(imp2)


##############################################
#           ANALYSE DU MODELE                #
##############################################

# on retient le modele qui a le R adj le plus petit
modele=lm(charges~slos+avtisst+hday,data=data_train)
summary(modele)

p=length(modele$coef)
n=nrow(data_train)

##### Diagnostics sur les r?sidus #####

par(mfrow=c(2,2))
plot(modele)

par(mfrow=c(1,1))
influencePlot(modele)

resstu=rstudent(modele)
hii=hatvalues(modele)
plot(data_train[,'slos'],resstu)
plot(data_train[,'hday'],resstu)
```

```
plot(data_train[,'avtisst'],resstu)

plot(resstu,hii)
abline(v=-2)
abline(v=2)
abline(h=2*p/n)


qqnorm(resstu)
qqline(resstu)
length(resstu[abs(resstu)>2])

CD=cooks.distance(modele)
plot(CD,type='h')
length(CD[CD>4/n])
length(CD[CD>6/n])  #Seuil propose, faute de normalite des donnees
infl=which(CD>4/n)
#infl=which(CD>6/n)
id_infl=data_train[infl,"id"]

##### Elimination des observations trop influentes

data_train2=data_train[-infl,]
modele2=lm(charges~slos+avtisst+hday,data=data_train2)
summary(modele2)

par(mfrow=c(2,2))
plot(modele2)

par(mfrow=c(1,1))
influencePlot(modele2)   # La c'est mieux

resstu2=rstudent(modele2)
hii2=hatvalues(modele2)
plot(data_train2[,'slos'],resstu2)
plot(data_train2[,'hday'],resstu2)
plot(data_train2[,'avtisst'],resstu2)

plot(resstu2,hii2)
abline(v=-2)
abline(v=2)
abline(h=2*p/nrow(data_train2))

qqnorm(resstu2)
qqline(resstu2)
length(resstu2[abs(resstu2)>2])

CD2=cooks.distance(modele2)
plot(CD2,type='h')
length(CD2[CD2>4/nrow(data_train2)])
length(CD2[CD2>6/nrow(data_train2)])  #Seuil propose, faute de
normalite des donnees
```

```
#infl2=which(CD2>4/nrow(data_train2))
infl2=which(CD2>6/nrow(data_train2))
id_infl2=data_train2[infl2,"id"]

##############################################
#           PREDICTION                       #
##############################################

### Methode a) Modele lineaire lm

y_pred_log=predict(modele2,newdata=data_test)
y_pred=exp(y_pred_log)

#Estimation de l'erreur commise:

#Avec data_train
Y<-data_train2[,"charges"]
X<-model.matrix(~slos+avtisst+hday,data_train2[,!
(colnames(data_train) %in% c("charges"))])

cvLasso<-
cv.glmnet(as.matrix(X),as.matrix(Y),family="gaussian",alpha=1)
cvRidge<-
cv.glmnet(as.matrix(X),as.matrix(Y),family="gaussian",alpha=0)
cvEN<-
cv.glmnet(as.matrix(X),as.matrix(Y),family="gaussian",alpha=0.5)

#L'erreur estimee (en echelle logarithmique)
min(cvLasso$cvm)
min(cvRidge$cvm)
min(cvEN$cvm)


### Methode b) (methode choisie): NEURAL NETWORK

library(nnet)
library(MASS)

#On utilise d'abord TUNE pour determiner la taille de la couche
cachee et la valeur decay

library(rpart)
library(e1071)

tim<-proc.time()
#Attention: ca prend trop de temps de calcul. Il faut faire la
commande (commentee) suivante:
#tune.mod = tune.nnet(charges~., data = data_train2, size = c(1,
3, 5),decay = c(0.1, 0.001, 0.000001), linout=TRUE, na.action =
na.omit)
tim<-proc.time()-tim
plot(tune.mod)
```

```
summary(tune.mod)

#Prediction
set.seed(123456)
regrnett <- nnet(charges~., data=data_train2, size = 5, decay =
0.001, linout=TRUE, maxit=3000,na.action = na.omit)
prednn<-exp(predict(regrnett, newdata = data_test,
type="raw",na.action = na.omit))
# car on a pris le log de la variable Y  donc exp(log(Y)) = Y

#Data_test avec prediction
test_prednn<-data_test
test_prednn$Y_pred<-prednn

#Save as file

test_prednn$id<-as.character(test_prednn$id)
write.table(test_prednn[,c("id","Y_pred")],"prediction_data.csv",r
ow.names=FALSE)
```