Exercice

Malek BEN NEYA

chargement des données

```
getwd()
```

```
## [1] "C:/Users/asus/Downloads/technic-test-master/technic-test-
master/Code+Predictions"
```

```
trainset<- read.csv('C:/Users/asus/Downloads/technic-test-master/technic-
test-master/data/train/trainset.CSV')
validset1<- read.csv('C:/Users/asus/Downloads/technic-test-master/technic-
test-master/data/valid/test_2017-07-12.CSV')
validset2<- read.csv('C:/Users/asus/Downloads/technic-test-master/technic-
test-master/data/valid/test_2017-07-13.CSV')
testset1<- read.csv('C:/Users/asus/Downloads/technic-test-master/technic-
test-master/data/test/testset_2017-07-12.CSV')
testset2<- read.csv('C:/Users/asus/Downloads/technic-test-master/technic-
test-master/data/test/testset_2017-07-13.CSV')
```

On vérifie les valeurs manquantes dans chaque base

```
CountNa<- sapply(trainset, function(x) length(which(is.na(x))))
print("Pas de données  manquantes pour trainset")
```

```
## [1] "Pas de données  manquantes pour trainset"
```

```
CountNa<- sapply(validset1, function(x) length(which(is.na(x))))
print("32 valeurs manquantes pour la variable consommation dans validset1" )
```

```
## [1] "32 valeurs manquantes pour la variable consommation dans validset1"
```

```
CountNa<- sapply(validset2, function(x) length(which(is.na(x))))
print("28 valeurs manquantes pour la variable consommation dans validset2")
```

```
## [1] "28 valeurs manquantes pour la variable consommation dans validset2"
```

prétraitement des données

netoyage est une fonction qui prend en paramétre une base de donnée -elle effectue des transformations sur la colonne Date et la sépare en jour, mois et année -elle effectue des transformations sur la colonne full et extraie les Min et Heures -elle transforme la variable sun en facteur car c'est variable binaire

```r
netoyage=function(data){
  data$annee=format(as.Date(data$Date), "%Y")
  data$mois=format(as.Date(data$Date), "%m")
  data$jour=format(as.Date(data$Date), "%d")
  data$min=as.POSIXlt(data$full)$min
  data$heure=as.POSIXlt(data$full)$hour
  data$sun<-as.factor(data$sun)

  #suppression des variables

  data$Date<-NULL
  data$full.date<-NULL
  data$Heures<-NULL
  data$X<-NULL

  #supression des variable avec une seule valeure
  v_oneLevels=c()
  for (i in names(data)){
    if(length(unique(data[[i]]))==1){
      data[[i]]<-NULL
      v_oneLevels=c(v_oneLevels,i)
    }
  }

  return(data)
}

#je combine le trainset et les 2 validset pour faire le netoyage
alldata=rbind(trainset,validset1,validset2)
alldata=netoyage(alldata)



trainset=alldata[1:nrow(trainset),]
alldata=alldata[-c(1:nrow(trainset)),]
validset1=alldata[1:nrow(validset1),]
alldata=alldata[-c(1:nrow(validset1)),]
validset2=alldata
dim(validset2)
```

```
## [1] 96 11
```

Modéle linéaire

```r
#On commence par normaliser la consommation, cela pourra augmenter légérement
les performance des modéles
m=mean(trainset$Consommation)
s=sd(trainset$Consommation)
trainset$Consommation=(trainset$Consommation-m)/s
m1 <- lm(Consommation~., trainset)
```

Prédiction et Erreur Mape

```
#fonction pour calculer l'erreur Mape
mape<-function(Y,D)
  return(mean(abs((D-Y)/D))*100)

#Prédiction en utilisant la 1ere validset
validTemp1<-validset1[which(!is.na(validset1$Consommation)),]
p1=(predict(m1, validTemp1)*s)+m
erreur1= mape(p1,validTemp1$Consommation)

#Prédiction en utilisant la 2eme validset
validTemp2<-validset2[which(!is.na(validset2$Consommation)),]
p2=(predict(m1, validTemp2)*s)+m
erreur2= mean(abs((validTemp2$Consommation-p2)/validTemp2$Consommation))*100

#affichage des erreurs
print(paste('MAPE pour valdiation 1 = ', erreur1))
```

```
## [1] "MAPE pour valdiation 1 =  3.40468168719256"
```

```
print(paste('MAPE pour valdiation 2 = ', erreur2))
```

```
## [1] "MAPE pour valdiation 2 =  3.10238707008323"
```

selection des variables par stepwise

On peut selectionner les variables les plus pertinantes avec les stepwise, dans ce cas le stepwise. Mais le resultat obtenu ne permet pas d'eliminer des variables.

```
step.AIC <- step(m1,  direction='both', trace=FALSE)
step.BIC <- step(m1, direction='both', k=log(nrow(trainset)), trace=FALSE)
summary(step.AIC)
```

```
##
## Call:
## lm(formula = Consommation ~ temperature + nebulosite + prev_conso +
##     workday + workday_2 + mois + jour + min + heure, data = trainset)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.78194 -0.17546  0.00179  0.17109  1.98251
##
## Coefficients:
##               Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -6.219e+00  5.461e-02 -113.878  < 2e-16 ***
## temperature  1.950e-03  1.294e-04   15.064  < 2e-16 ***
## nebulosite   1.858e-02  3.749e-03    4.956 7.37e-07 ***
## prev_conso   1.307e-04  1.080e-06  121.055  < 2e-16 ***
## workday     -6.066e-02  1.316e-02   -4.610 4.10e-06 ***
```

```
## workday_2WE -1.007e-01  2.058e-02   -4.893 1.02e-06 ***
## mois06      -3.617e-02  1.117e-02   -3.237 0.001215 **
## mois07      -1.130e-01  1.781e-02   -6.341 2.44e-10 ***
## jour02      -1.809e-01  3.441e-02   -5.258 1.51e-07 ***
## jour03      -2.373e-01  3.615e-02   -6.564 5.63e-11 ***
## jour04      -3.351e-01  3.441e-02   -9.737  < 2e-16 ***
## jour05      -4.881e-03  3.282e-02   -0.149 0.881759
## jour06      -1.636e-01  3.175e-02   -5.153 2.64e-07 ***
## jour07      -2.278e-01  3.182e-02   -7.158 9.09e-13 ***
## jour08       5.964e-02  3.262e-02    1.828 0.067559 .
## jour09      -2.521e-01  3.173e-02   -7.945 2.27e-15 ***
## jour10      -1.323e-01  3.176e-02   -4.165 3.16e-05 ***
## jour11      -1.882e-01  3.185e-02   -5.909 3.61e-09 ***
## jour12      -2.235e-01  3.585e-02   -6.236 4.77e-10 ***
## jour13      -6.929e-02  3.542e-02   -1.956 0.050471 .
## jour14      -2.862e-01  3.559e-02   -8.041 1.05e-15 ***
## jour15      -1.136e-01  3.611e-02   -3.146 0.001664 **
## jour16      -1.373e-01  3.629e-02   -3.784 0.000156 ***
## jour17      -7.999e-02  3.609e-02   -2.217 0.026690 *
## jour18      -2.671e-01  3.569e-02   -7.484 8.14e-14 ***
## jour19      -3.188e-01  3.614e-02   -8.822  < 2e-16 ***
## jour20      -1.046e-01  3.591e-02   -2.914 0.003584 **
## jour21      -1.907e-01  3.628e-02   -5.255 1.53e-07 ***
## jour22      -2.432e-01  3.656e-02   -6.651 3.15e-11 ***
## jour23      -1.043e-01  3.643e-02   -2.864 0.004202 **
## jour24      -9.191e-02  3.604e-02   -2.550 0.010788 *
## jour25      -2.963e-01  3.784e-02   -7.828 5.75e-15 ***
## jour26      -1.582e-01  3.648e-02   -4.337 1.46e-05 ***
## jour27      -2.027e-01  3.620e-02   -5.600 2.23e-08 ***
## jour28      -2.868e-01  3.620e-02   -7.924 2.68e-15 ***
## jour29      -3.598e-01  3.632e-02   -9.908  < 2e-16 ***
## jour30      -2.726e-01  3.617e-02   -7.538 5.42e-14 ***
## jour31      -2.087e-01  4.460e-02   -4.679 2.94e-06 ***
## min          5.913e-04  2.492e-04    2.373 0.017680 *
## heure        9.799e-03  6.964e-04   14.070  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3369 on 6458 degrees of freedom
## Multiple R-squared:  0.8872, Adjusted R-squared:  0.8865
## F-statistic:  1302 on 39 and 6458 DF,  p-value: < 2.2e-16
```

```
summary(step.BIC)
```

```
##
## Call:
## lm(formula = Consommation ~ temperature + nebulosite + prev_conso +
##     workday + workday_2 + mois + jour + heure, data = trainset)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -1.79501 -0.17557  0.00019  0.17144  1.99568
##
## Coefficients:
##                Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -6.204e+00  5.428e-02 -114.297  < 2e-16 ***
## temperature  1.952e-03  1.295e-04   15.080  < 2e-16 ***
## nebulosite   1.861e-02  3.751e-03    4.962 7.17e-07 ***
## prev_conso   1.306e-04  1.080e-06  120.989  < 2e-16 ***
## workday     -6.064e-02  1.316e-02   -4.607 4.16e-06 ***
## workday_2WE -1.010e-01  2.058e-02   -4.905 9.56e-07 ***
## mois06      -3.628e-02  1.118e-02   -3.246 0.001175 **
## mois07      -1.132e-01  1.782e-02   -6.350 2.29e-10 ***
## jour02      -1.810e-01  3.443e-02   -5.257 1.51e-07 ***
## jour03      -2.374e-01  3.617e-02   -6.565 5.62e-11 ***
## jour04      -3.351e-01  3.442e-02   -9.733  < 2e-16 ***
## jour05      -4.867e-03  3.283e-02   -0.148 0.882152
## jour06      -1.635e-01  3.176e-02   -5.149 2.70e-07 ***
## jour07      -2.277e-01  3.183e-02   -7.153 9.41e-13 ***
## jour08       5.966e-02  3.263e-02    1.828 0.067548 .
## jour09      -2.520e-01  3.174e-02   -7.941 2.35e-15 ***
## jour10      -1.322e-01  3.177e-02   -4.162 3.19e-05 ***
## jour11      -1.883e-01  3.186e-02   -5.909 3.63e-09 ***
## jour12      -2.236e-01  3.586e-02   -6.235 4.81e-10 ***
## jour13      -6.931e-02  3.543e-02   -1.956 0.050511 .
## jour14      -2.862e-01  3.560e-02   -8.039 1.07e-15 ***
## jour15      -1.137e-01  3.613e-02   -3.147 0.001655 **
## jour16      -1.374e-01  3.631e-02   -3.785 0.000155 ***
## jour17      -8.009e-02  3.610e-02   -2.219 0.026552 *
## jour18      -2.672e-01  3.571e-02   -7.484 8.15e-14 ***
## jour19      -3.188e-01  3.616e-02   -8.818  < 2e-16 ***
## jour20      -1.046e-01  3.592e-02   -2.912 0.003603 **
## jour21      -1.907e-01  3.630e-02   -5.254 1.54e-07 ***
## jour22      -2.433e-01  3.658e-02   -6.651 3.15e-11 ***
## jour23      -1.044e-01  3.644e-02   -2.865 0.004178 **
## jour24      -9.201e-02  3.605e-02   -2.552 0.010729 *
## jour25      -2.964e-01  3.786e-02   -7.828 5.75e-15 ***
## jour26      -1.583e-01  3.649e-02   -4.339 1.46e-05 ***
## jour27      -2.029e-01  3.622e-02   -5.602 2.20e-08 ***
## jour28      -2.870e-01  3.621e-02   -7.927 2.63e-15 ***
## jour29      -3.599e-01  3.633e-02   -9.908  < 2e-16 ***
## jour30      -2.727e-01  3.618e-02   -7.538 5.42e-14 ***
## jour31      -2.088e-01  4.461e-02   -4.681 2.92e-06 ***
## heure        9.807e-03  6.967e-04   14.077  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.337 on 6459 degrees of freedom
## Multiple R-squared:  0.8871, Adjusted R-squared:  0.8864
## F-statistic:  1336 on 38 and 6459 DF,  p-value: < 2.2e-16
```

Régularisation Ridge

```
library(glmnet)
```

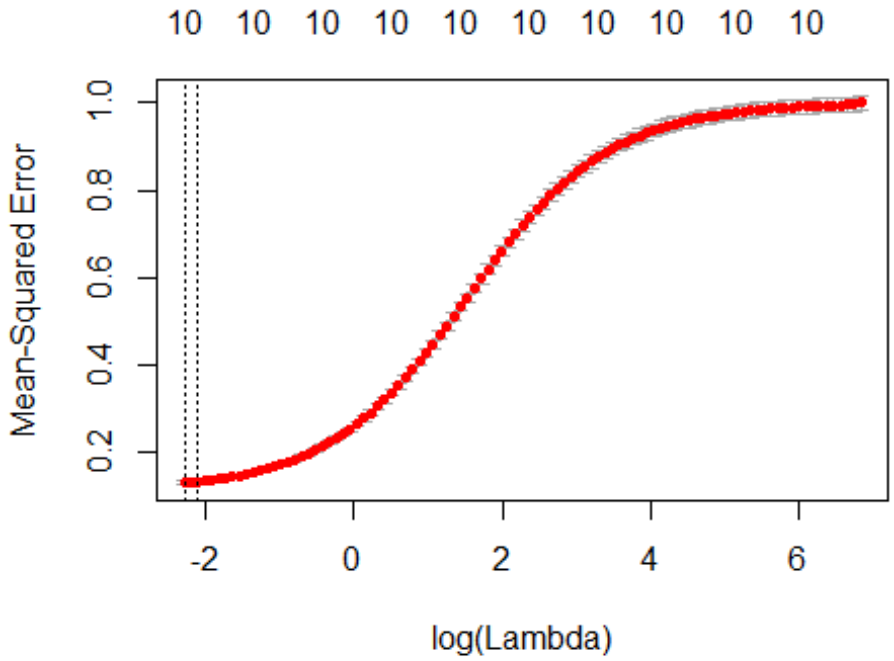
```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-10
```

```
#il faut transformer les dataframe en matrice pour utiliser glmnet
x <- data.matrix(trainset[,-which(names(trainset)=='Consommation')])
colnames(x) <- NULL
y <- as.vector(trainset$Consommation)

#validation pour obtenir le meilleur lambda
cv.ridge <- cv.glmnet(x,y,alpha=0)
plot(cv.ridge)
```




```
cv.ridge$lambda.min
```

```
## [1] 0.1022136
```

```
ridge <- glmnet(x,y,alpha=0, lambda=cv.ridge$lambda.min)


#Prédiction pour la set 1
v1=data.matrix(validTemp1[,-which(names(validTemp1)=='Consommation')])
```

```r
colnames(v1) <- NULL
pred1=(predict(ridge, newx=v1)*s)+m
erreur3=mape(pred1,validTemp1$Consommation)
#Prédiction pour la set 2
v2=data.matrix(validTemp2[,-which(names(validTemp2)=='Consommation')])
colnames(v2) <- NULL
pred1=(predict(ridge, newx=v2)*s)+m
erreur4=mape(pred1,validTemp2$Consommation)

#affichage des erreurs
print(paste('MAPE pour valdiation 1 = ', erreur3))
```

```
## [1] "MAPE pour valdiation 1 =  3.61813856385813"
```

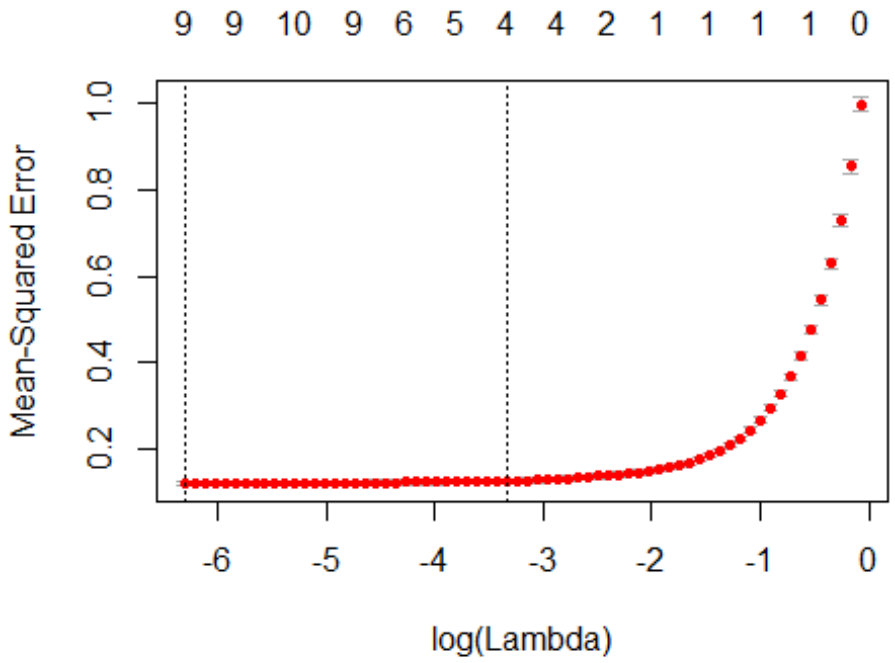
```r
print(paste('MAPE pour valdiation 2 = ', erreur4))
```

```
## [1] "MAPE pour valdiation 2 =  3.96112287772708"
```

Régularisation Lasso

```r
#transformation du train en matrice pour appliquer le modele
x <- data.matrix(trainset[,-which(names(trainset)=='Consommation')])
colnames(x) <- NULL
y <- as.vector(trainset$Consommation)

#validation croisée sur le lambda
cv.lasso <- cv.glmnet(x,y,alpha=1)
plot(cv.lasso)
```




```
cv.lasso$lambda.min
```

```
## [1] 0.001828243
```

```
lasso <- glmnet(x,y,alpha=1, lambda=cv.lasso$lambda.min)

#Prédiction pour la set 1
v1=data.matrix(validTemp1[,-which(names(validTemp1)=='Consommation')])
colnames(v1) <- NULL
pred1=(predict(lasso, newx=v1)*s)+m
erreur5=mape(pred1,validTemp1$Consommation)
#Prédiction pour la set 2
v2=data.matrix(validTemp2[,-which(names(validTemp2)=='Consommation')])
colnames(v2) <- NULL
pred1=(predict(lasso, newx=v2)*s)+m
erreur6=mape(pred1,validTemp2$Consommation)

#affichage des erreurs
print(paste('MAPE pour valdiation 1 = ', erreur5))
```

```
## [1] "MAPE pour valdiation 1 =  3.05085469509191"
```

```
print(paste('MAPE pour valdiation 2 = ', erreur6))
```

```
## [1] "MAPE pour valdiation 2 =  3.57216754859909"
```

Prédiction en utilisant les tests

netoyage_test est une fonction qui prend en argument deux jeux de donées, elle effectue des transformations sur les colonnes dates et hour du test elle supprime les variables de test dont le nom n'apparait pas dans data

```
#sauvegarde des variables qui seront utilisés dans l'affichage finales des resultats
resultat1=testset1[c('region.code','Date','Heures')]
resultat2=testset2[c('region.code','Date','Heures')]

netoyage_test=function(test,data){
#modification des types
  test$annee=format(as.Date(test$Date), "%Y")
  test$mois=format(as.Date(test$Date), "%m")
  test$jour=format(as.Date(test$Date), "%d")
  test$min=as.POSIXlt(test$full)$min
  test$heure=as.POSIXlt(test$full)$hour
  test$sun<-as.factor(test$sun)
  #suppression des variables
  test=test[names(data[,-which(names(data)=='Consommation')])]
  return(test)
}

testset1=netoyage_test(testset1,trainset)
testset2=netoyage_test(testset2,trainset)
```

prédiction pour le testset1 en utilisant la régularisation lasso

La régularisation Lasso a la meilleure performance pour la validation de la validset1 donc on l'utilise pour la prédiction sur le testset1

```
t1=data.matrix(testset1)
colnames(t1) <- NULL
ptest1=(predict(lasso, newx=t1)*s)+m

resultat1=cbind(resultat1,ptest1)
names(resultat1)[4]='pred'
write.csv2(resultat1,'pred1.csv',row.names=FALSE)
```

prédiction pour le testset2 en utilisant le modéle lineaire

Le modéle linéire a donné la meilleure performance pour la validation de la validset2 donc on l'utilise pour la prédiction sur le testset2

```
ptest2=(predict(m1, testset2)*s)+m
resultat2=cbind(resultat2,ptest2)
```

```
names(resultat2)[4]='pred'
write.csv2(resultat2,'pred2.csv',row.names=FALSE)
```