TP R : PCR, PLS, Ridge et Lasso

C. Preda / V. Vandewalle

13 Mars 2018

Objectif du TP

L'objectif de ce TP est de présenter des solutions pour réaliser la régression linéaire en condition de multicolinéarité. On utilisera les packages **pls, glmnet** et **glmnet** et notamment les fonctions *pcr, plsr, glmnet*. Nous allons illustrer les principales fonctionnalités de ces fonctions à l'aide de la base de données disponible à l'adresse :

http://math.univ-lille1.fr/~preda/GIS4/car.txt

Avant tout

Installer les packages **pls, glmnet** et jetez un coup d'oeil sur les fonctions *pcr, plsr, glmnet*

```
install.packages("pls") # installe le package pls
library(pls)            # charge la librairie pls
help(package = "pls")   # donnee des informations sur pls

install.packages("glmnet")
library(glmnet)
help(package ="glmnet")
```

Les données

Chargement de la base de données dans R:

```
d = read.table("http://math.univ-lille1.fr/~preda/GIS4/car.txt", header=TRUE, sep="\t", row.names=1)
str(d)
```

```
'data.frame':   18 obs. of  9 variables:
 $ CYL     : int  1350 1588 1294 1222 1585 1297 1796 1565 2664 1166 ...
 $ PUIS    : int  79 85 68 59 98 82 79 55 128 55 ...
 $ LON     : int  393 468 424 412 439 429 449 424 452 399 ...
 $ LAR     : int  161 177 168 161 164 169 169 163 173 157 ...
 $ POIDS   : int  870 1110 1050 930 1105 1080 1160 1010 1320 815 ...
 $ VITESSE : int  165 160 152 151 165 160 154 140 180 140 ...
 $ NAT     : Factor w/ 6 levels "D","F","GB","I",..: 4 1 2 2 4 4 2 2 2 5 ...
 $ FINITION: Factor w/ 3 levels "B","M","TB": 1 3 2 2 1 3 1 1 3 2 ...
 $ PRIX    : int  30570 39990 29600 28250 34900 35480 32300 32000 47700 26540 ...
```

```
summary(d)
```

```
      CYL            PUIS            LON            LAR
 Min.   :1166   Min.   : 55.00   Min.   :393.0   Min.   :157.0
 1st Qu.:1310   1st Qu.: 70.75   1st Qu.:424.0   1st Qu.:162.2
 Median :1578   Median : 82.00   Median :434.5   Median :167.0
 Mean   :1632   Mean   : 84.61   Mean   :433.5   Mean   :166.7
 3rd Qu.:1798   3rd Qu.: 98.00   3rd Qu.:448.0   3rd Qu.:169.8
```

```
 Max.   :2664   Max.   :128.00   Max.   :469.0   Max.   :177.0
    POIDS         VITESSE      NAT    FINITION      PRIX
 Min.   : 815   Min.   :140.0   D :3   B :7     Min.   :22100
 1st Qu.:1020   1st Qu.:151.2   F :6   M :5     1st Qu.:29842
 Median :1088   Median :160.0   GB:1   TB:6     Median :33345
 Mean   :1079   Mean   :158.3   I :4            Mean   :34159
 3rd Qu.:1127   3rd Qu.:165.0   J :3            3rd Qu.:38458
 Max.   :1370   Max.   :180.0   U :1            Max.   :47700
```

Expliquer/prédire le PRIX des voitures à partir leurs caractéristiques

Note : Les variables qualitatives (ici NAT et FINITION) ne peuvent pas faire partie d'un modèle de régression basé sur les composantes principales ou PLS car ces techniques ne sont pas adaptées à ce type de données. Par contre Ridge et Lasso traitent aussi bien les variables qualitatives que les variables quantitatives. On ne va donc considérer dans la liste des prédicteurs **que les variables quantitatives**.

```
d = d[, -c(7,8)]
```

Régression linéaire "oridinaire" :

```
m0 = lm(PRIX~., data = d)  # le . remplace toutes les autres variables dans la base d
summary(m0)
```

```
Call:
lm(formula = PRIX ~ ., data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-8289.8 -1720.6  -166.6  2913.3  5420.0

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -8239.363  42718.423  -0.193    0.851
CYL            -3.505      5.551  -0.631    0.541
PUIS          282.169    174.883   1.613    0.135
LON           -15.038    129.747  -0.116    0.910
LAR           208.694    412.048   0.506    0.623
POIDS          12.575     24.622   0.511    0.620
VITESSE      -111.114    222.257  -0.500    0.627

Residual standard error: 4406 on 11 degrees of freedom
Multiple R-squared:  0.7091,    Adjusted R-squared:  0.5504
F-statistic: 4.469 on 6 and 11 DF,  p-value: 0.0156
```

Malgré un bon R^2 et un test global de Fisher nous indiquant que le modèle explique le PRIX de manière significative (p = 0.0156) mieux que le modèle null (sans variables explicatives), aucune variable n'est significative ! L'explication peut venir d'une inflation de la variance des estimateurs due soit au fait que le nombre d'observations est plus petit que le nombre de variables (ce n'est pas le cas ici), soit au fait que la matrice $(X^T X)$ est mal-conditionnée (son inverse n'existe pas ou a des valeurs trop grandes) et dans ce dernier cas on parle de *multicolinéarité* : une ou plusieurs variables prédicteurs sont rédondantes dans l'analyse.

Diagnostiquer la multicolinéarité.

On réalise une ACP (normée en général) afin d'identifier des valeurs propres proches de 0.

```
library(FactoMineR)
acp = PCA(d[,-7], scale=TRUE, graph = FALSE)  #-7 : sans la variable PRIX
print(acp$eig)
```

```
       eigenvalue percentage of variance cumulative percentage of variance
comp 1 4.42085806             73.6809677                           73.68097
comp 2 0.85606229             14.2677048                           87.94867
comp 3 0.37306608              6.2177680                           94.16644
comp 4 0.21392209              3.5653682                           97.73181
comp 5 0.09280121              1.5466869                           99.27850
comp 6 0.04329027              0.7215045                          100.00000
```

Effectivement, les deux dernières valeurs propres sont particulierement proches de 0 et elles expliquent quantités négligeables d'information (1.54% et 0.72%). Multicolinéarité en vue!

Une autre manière de diagnostiquer la multicolinéarité et d'investiguer la statistique VIF (Variance Inflation Factor) qui pour chaque variable prédicteur $X_i \in \{X_1, \ldots, X_p\}$ calcule la quantité

$$VIF(X_i) = \frac{1}{1 - R_i^2}$$

où R_i^2 est le R^2 (le pouvoir explicatif donc) de la variable X_i par les autres variables $\{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_p\}$.

Une valeur de VIF supérieure à 10 évoque une quasi-certitude sur la présence de multicolinéarité. Voir aussi : https://en.wikipedia.org/wiki/Variance_inflation_factor

```
library(car)
vif(m0)
```

```
      CYL      PUIS       LON       LAR     POIDS   VITESSE
 3.772014 11.118820  7.204195  4.197605  9.957283  6.375112
```

On observe ici la variable PUIS avec un VIF égal à 11.11 mais aussi la variable POIDS avec un VIF = 9.95. Il n'y a donc pas de doutes que la multicolinéarité est présente.

Solutions à la multicolinéarité: stepAIC, PCR, PLS, Ridge et Lasso

stepAIC

C'est la solution la plus simple : choisir un modèle de taille plus petite qui inclut juste les variables qui ajustent le mieux les données en termes de vraissemblance pénalisée (critère AIC : voir cours régression linéaire) :

```
library(MASS)
m_aic=stepAIC(m0, trace=0)  # pas d'affichage du processus de selection
summary(m_aic)
```

```
Call:
lm(formula = PRIX ~ PUIS + POIDS, data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-7148.2 -1874.6   296.3  2012.7  5263.7
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1775.60    8030.95   0.221   0.8280
PUIS          172.97      72.42   2.388   0.0305 *
POIDS          16.45      10.77   1.527   0.1476
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3916 on 15 degrees of freedom
Multiple R-squared:  0.6866,    Adjusted R-squared:  0.6448
F-statistic: 16.43 on 2 and 15 DF,  p-value: 0.0001661
```

Ce sont les variables PUIS et POIDS qui sont sélectionnées par le stepAIC (stratégie backward), avec seul la variable PUIS significative. C'est un modèle assez propre car on peut vérifier que les conditions d'application sont vérifiées (normalité, homoscédasticité et indépendance des résidus).

En termes de pouvoir prédictif, ce modèle fournit une erreur de prédiction par validation croisée (leave-one-out) égale à MSEP (Mean Squares Error of Prediction) :

$$MSEP = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i^{(-i)})^2$$

ou encore le Root Mean Squares Error of Prediction

$$RMSEP = \sqrt{MSEP}$$

```
rmsep=function(fit){
  h=lm.influence(fit)$h
  return(sqrt(mean((residuals(fit)/(1-h))^2)))
}

print(rmsep(m_aic))
```

```
[1] 4139.89
```

Donc, le RMSEP pour le modèle m_aic vaut 4140. A titre d'information, vérifiez combien vaut-il pour le modèle m0 ?

Régression sur les composantes principales (PCR)

Utile lorsqu'on veut construire un modèle avec toutes les variables !

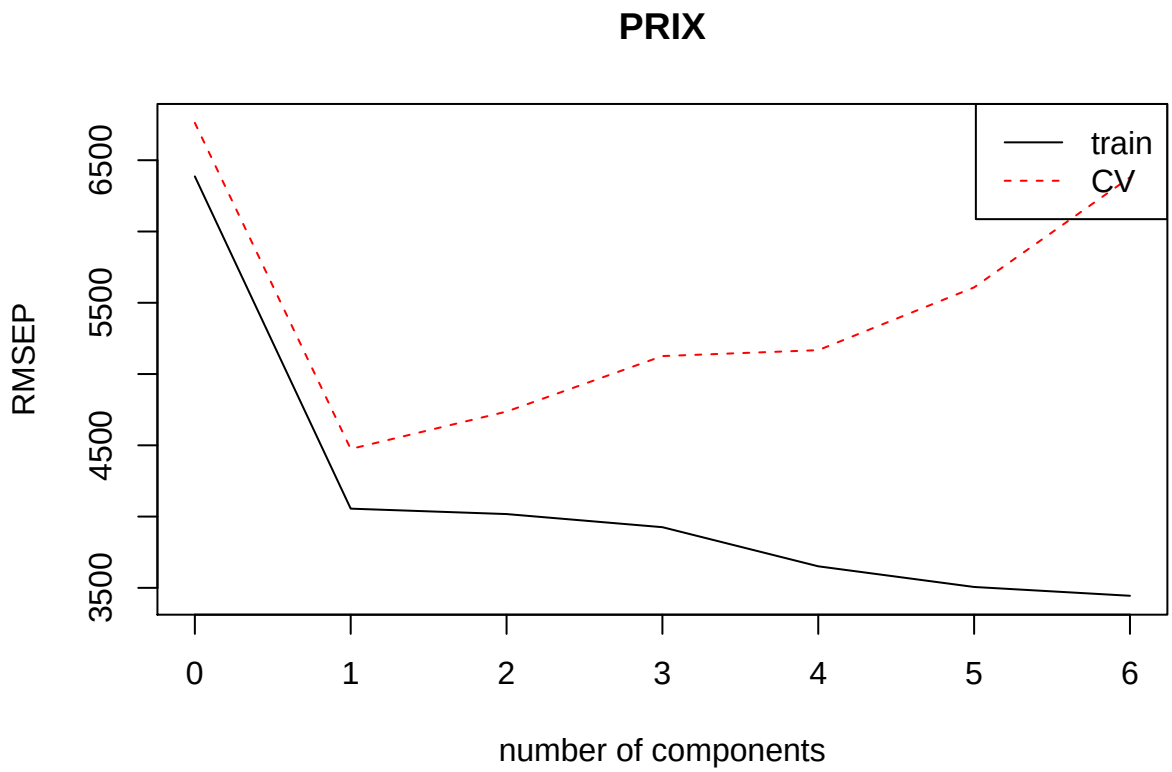
```
library(pls)
```

```
Attaching package: 'pls'

The following object is masked from 'package:stats':

    loadings
```

```
m_pcr=pcr(PRIX~., scale=TRUE, validation="LOO", jackknife = TRUE, data=d)
```

Pour le choix du nombre de composantes principales on regardera :

```r
plot(m_pcr,"validation", estimate = c("train", "CV"), legendpos = "topright")
```



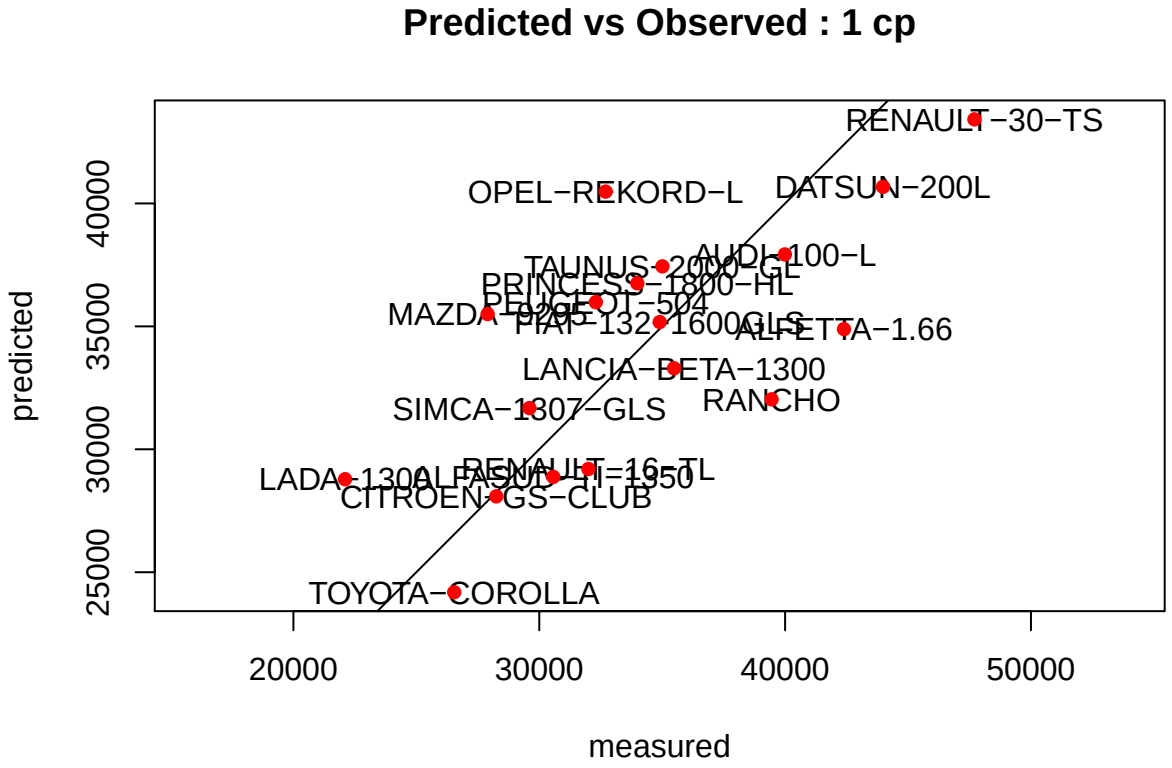

```r
summary(m_pcr)
```

```
Data:   X dimension: 18 6
    Y dimension: 18 1
Fit method: svdpc
Number of components considered: 6

VALIDATION: RMSEP
Cross-validated using 18 leave-one-out segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV            6762     4475     4737     5126     5167     5608     6380
adjCV         6762     4461     4716     5097     5123     5547     6295

TRAINING: % variance explained
      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
X       73.68    87.95    94.17    97.73    99.28   100.00
PRIX    59.67    60.42    62.22    67.32    69.86    70.91
```

On choisira donc une seule composante (la première) qui nous assure un RMSEP = 4475. Moins bien que le stepAIC en termes de prédiction !

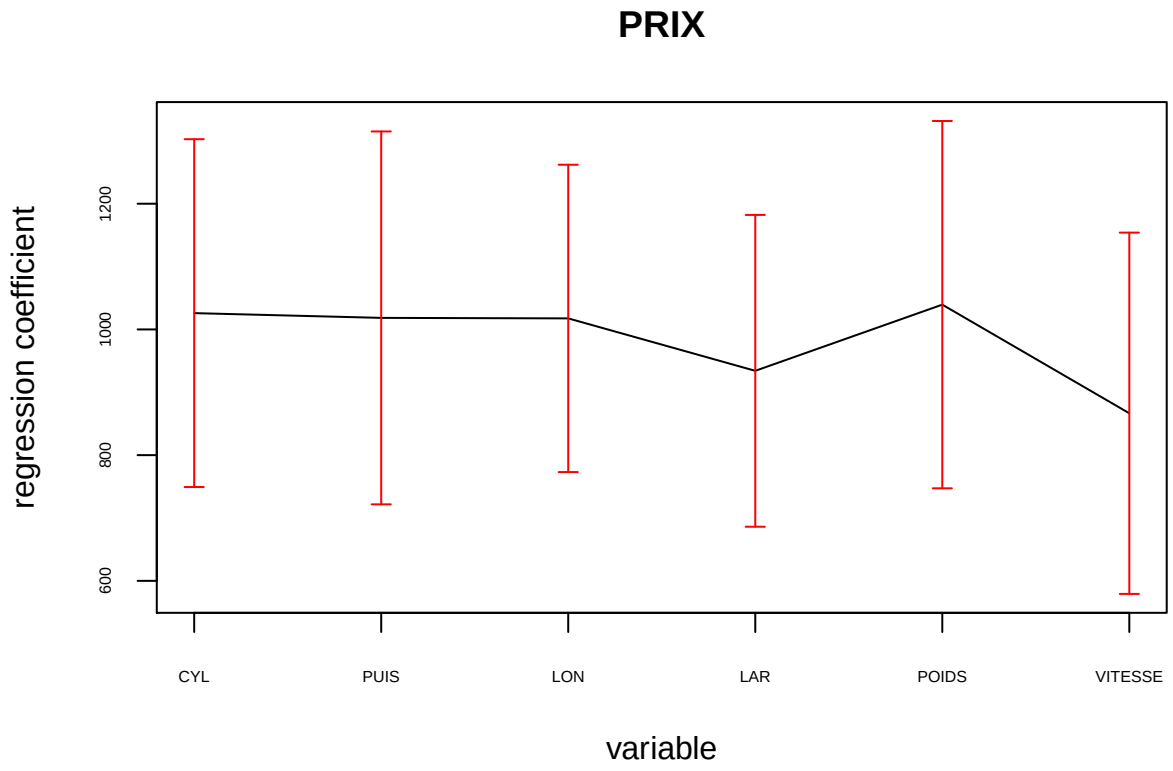
Voici les prédictions graphiquement (obtenues par validation croisée) :

```r
obsfit = predplot(m_pcr, ncomp=1, which = "validation",  asp=1,line=TRUE, main="Predicted vs Observed :
points(obsfit, pch=16, col="red")
```



Les coefficients et leur significativité est obtenue avec :

```
coefplot(m_pcr, ncomp=1, se.whiskers = TRUE, labels = prednames(m_pcr), cex.axis = 0.5)
```



```
jack.test(m_pcr, ncomp=1)
```

```
Response PRIX (1 comps):
        Estimate Std. Error Df t value  Pr(>|t|)
CYL      1025.95     276.71 17  3.7077 0.0017483 **
PUIS     1018.36     296.64 17  3.4329 0.0031732 **
LON      1017.55     244.49 17  4.1619 0.0006531 ***
LAR       934.17     248.02 17  3.7665 0.0015390 **
POIDS    1039.41     292.20 17  3.5572 0.0024238 **
VITESSE   866.62     287.39 17  3.0154 0.0077937 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

L'intercept est obtenu avec :

```
coef(m_pcr, ncomp=1, intercept=TRUE)
```

```
, , 1 comps

                   PRIX
(Intercept) -43286.4592
CYL           1025.9460
PUIS          1018.3610
LON           1017.5534
LAR            934.1672
POIDS         1039.4083
VITESSE        866.6185
```

Ces coefficents sont cependant ceux obtenus avec les variables réduites (mais pas centrées), car ACP normée au départ. Les coefficients des variables d'origine sont donc :

```
sds = apply(d,2, "sd" ) # calcul des ecart-types de chaque variable
coef(m_pcr, ncomp=1, intercept=TRUE)[2:7]/sds[-7] #pour avoir les coefs des variables d'origine et non
```

```
       CYL       PUIS        LON        LAR      POIDS    VITESSE
  2.743686  49.977766  46.027817 175.803886   7.589259  71.383131
```

Le modèle de régression obtenu est donc :

PRIX = -43286.46 +2.75CYL +49.98PUIS + 46.03LON +175.80LAR + 7.59POIDS + 71.38VITESSE + erreur

Tous les coefficients sont positifs, ce qui va dans le bon sens de la vie courante ! Plus la voiture a des caractéristiques élévées, plus elle est chère ! Il faut comprendre cette chose ainsi : par exemple, a puissance égale, cylindrée égale, une voiture plus longue est logiquement plus chere qu'une moins longue !

Ce modèle a donc un RMSEP de 4475. On peut l'utiliser pour la prédiction grace a la fontion `predict`. Par exemple, que peut-on dire d'une voiture avec les caracteristiques :

CYL = 1600, PUIS = 100, LON=450, LAR=180, POIDS=1400, VITESSE = 160 ?

Pour prédire, on construit un data frame test ainsi :

```
dtest = data.frame(CYL=c(1600), PUIS = c(100), LON=c(450), LAR=c(180), POIDS=c(1400), VITESSE = c(160))
print(dtest)
```

```
   CYL PUIS LON LAR POIDS VITESSE
1 1600  100 450 180  1400     160
```

Appliquons la fonction `predict` et on obtient la prédiction :

```
predict(m_pcr, comps=1, newdata=dtest)
```

```
     PRIX
1 40504.7
```

Régression PLS

Le même parcours comme pour la PCR peut être réalisé avec la fonction `plsr` pour réaliser la régression PLS. Réalisez le !

```
m_pls=plsr(PRIX~., scale=TRUE, validation="LOO", jackknife = TRUE, data=d)
summary(m_pls)
```

```
Data:   X dimension: 18 6
    Y dimension: 18 1
Fit method: kernelpls
Number of components considered: 6

VALIDATION: RMSEP
Cross-validated using 18 leave-one-out segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV            6762     4515     5495     5601     5987     6270     6380
adjCV         6762     4502     5440     5545     5919     6189     6295

TRAINING: % variance explained
      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
X       73.62    82.66    91.92    96.59    99.07   100.00
PRIX    60.84    67.08    69.45    70.36    70.81    70.91
```

On trouve que le modèle avec une composante PLS nous fournit la meilleure prédiction en termes de RMSEP avec une valeur de : 4502 ! Moins bien que PCR en termes de RMSEP mais mieux en termes de R^2 : 0.5957 pour la PCR et 0.6064 pour la PLS.

Régression Ridge, Lasso et elastic-net

Comme vu en cours, PCR et PLS sont des techniques de régularisation basées sur la réduction de la dimension des prédicteurs. Les techniques Ridge et Lasso sont elles basées sur la pénalisation des moindres carrés par des pénalités de type L_2 (Ridge) et L_1 (Lasso).

Pour réaliser la régression Ridge, on peut utiliser soit la fonction *lm.ridge* du package MASS soit utiliser la fonction *glmnet* du package avec le même nom.

Nous illustrons ici la fonction *glmnet* qui permet de faire à la fois ridge, lasso et elastic-net.

On se souvient (voir cours) que le critère à minimiser dans un modèle elastic-net est :

$$\min_{\beta}\{\|Y - X\beta\|^2 + \lambda\left[(1-\alpha)\|\beta\|_2^2 + \alpha\|\beta\|_1\right]\}$$

avec $\|\beta\|_2^2 = \sum_1^p \beta_i^2$ et $\|\beta\|_1 = \sum_1^p |\beta_i|$

On a ainsi pour $\alpha = 0$ la régression Ridge, pour $\alpha = 1$ la régression Lasso et pour $\alpha \in]0, 1[$, la regression elastic-net.

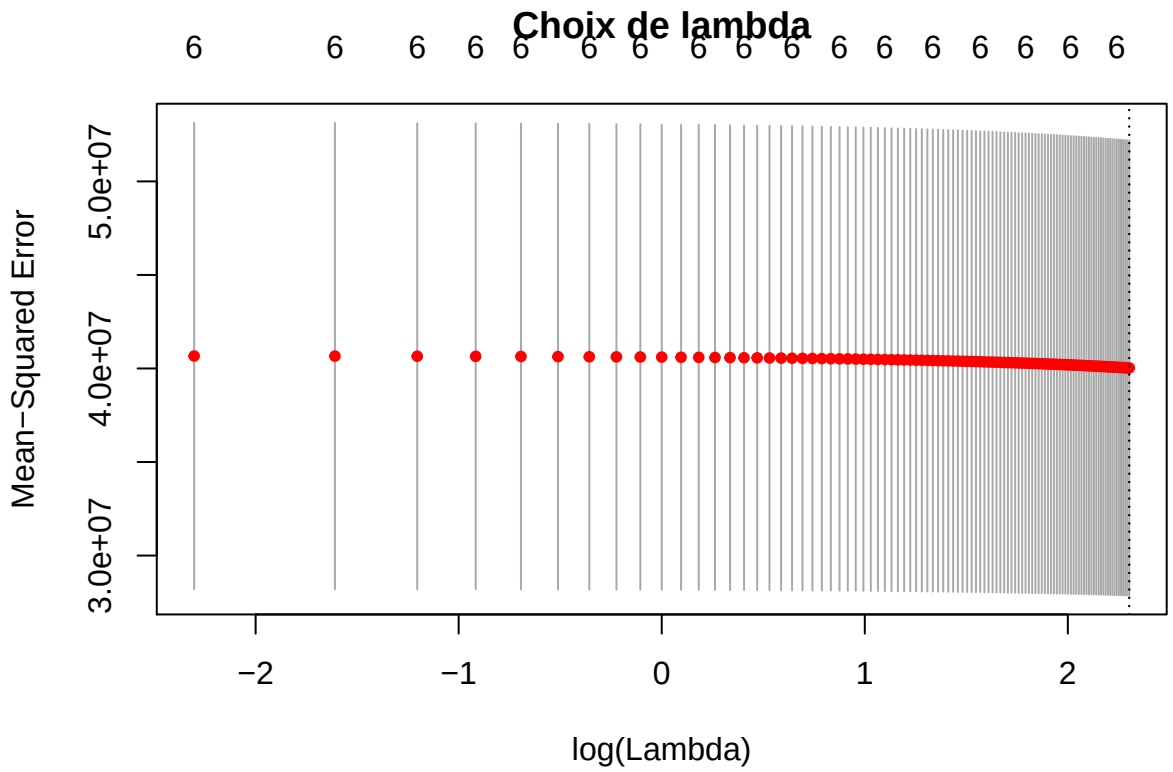
Ridge regression

Préparation de données pour `glmnet` :

```
X = model.matrix(PRIX~., d)[,-1]     # on met les predicteurs sous la forme d'une matrice. Si des variab
Y =d$PRIX  # la variable réponse
```

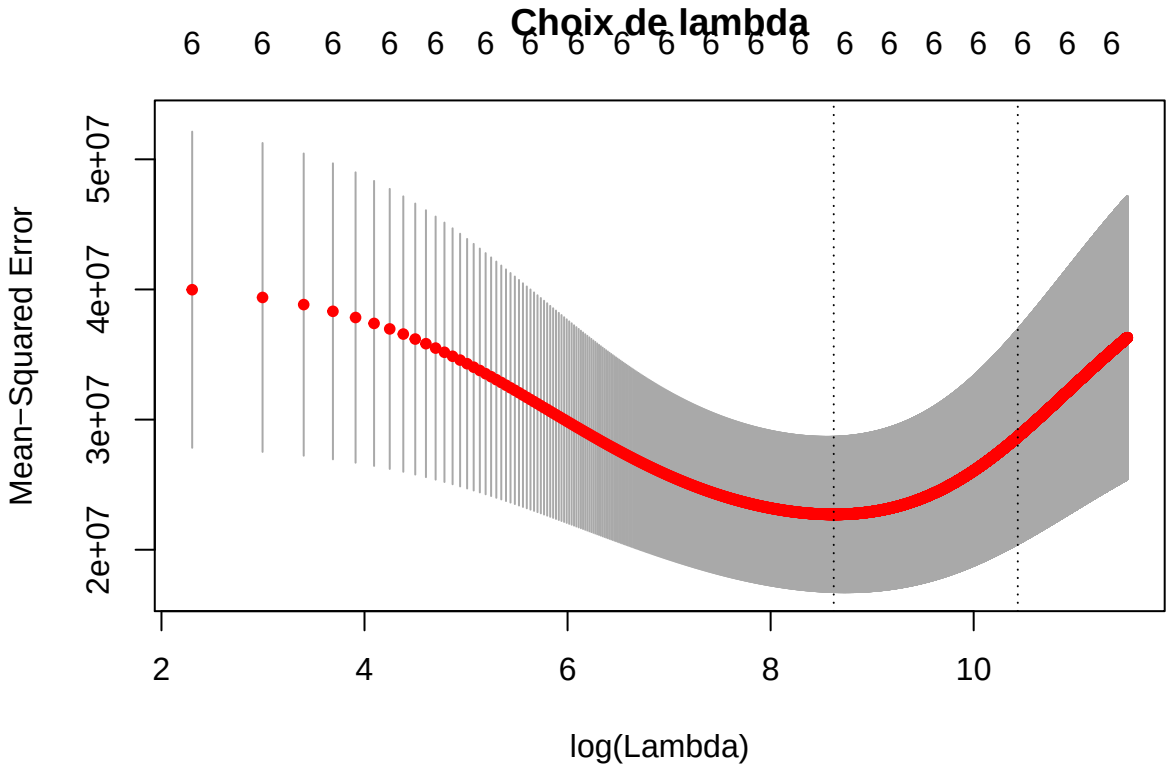
Pour le choix de lambda, on se donne un ensemble initial de valeurs possibles, soit par exemple, lambda = seq(0,10,0.1).

```
library(glmnet)
#choisisons le lambda qui minimise le RMSEP (ou equivalent, la cross-validation = MSEP):

cv_fit <- cv.glmnet(X,Y, alpha = 0, lambda = seq(0,10, 0.1), grouped = FALSE, nfolds =nrow(d))

plot(cv_fit, main = "Choix de lambda")
```



On observe que la décroissance de l'erreur de prédiction n'est pas assez prononcée, du coup, on élargi l'intervalle de lambda ! Posons lambda = seq(0,100000, 10)

```
cv_fit <- cv.glmnet(X,Y, alpha = 0, lambda = seq(0,100000, 10), grouped = FALSE, nfolds =nrow(d))

plot(cv_fit, main = "Choix de lambda")
```



On observe sur ce graph la valeur de lambda qui minimise l'erreur de prediction. On l'obtient avec :

```
lambda_optimal = cv_fit$lambda.min
print(lambda_optimal)
```

```
[1] 5550
```

On trouve la valeur optimale : lambda_optimal=5550 pour laquelle on obtient une MSEP égale à 22714930 ce qui correspond à un RMSEP = sqrt(MSEP) = 4766.

Observons donc que Ridge fait moins bien que PLS et PCR!

```
print(min(cv_fit$cvm))
```

```
[1] 22714930
```

```
rmsep_ridge =sqrt(min(cv_fit$cvm))
print(rmsep_ridge)
```

```
[1] 4766.018
```

Les coefficients du modèle Ridge optimal sont données par :

```
m_ridge <- glmnet(X,Y, alpha = 0, lambda = lambda_optimal)
coef(m_ridge) # pour voir les coefficients
```

```
7 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept) -17937.407121
CYL              1.324353
PUIS            72.631483
LON             29.839976
LAR             79.819401
POIDS            8.812178
```

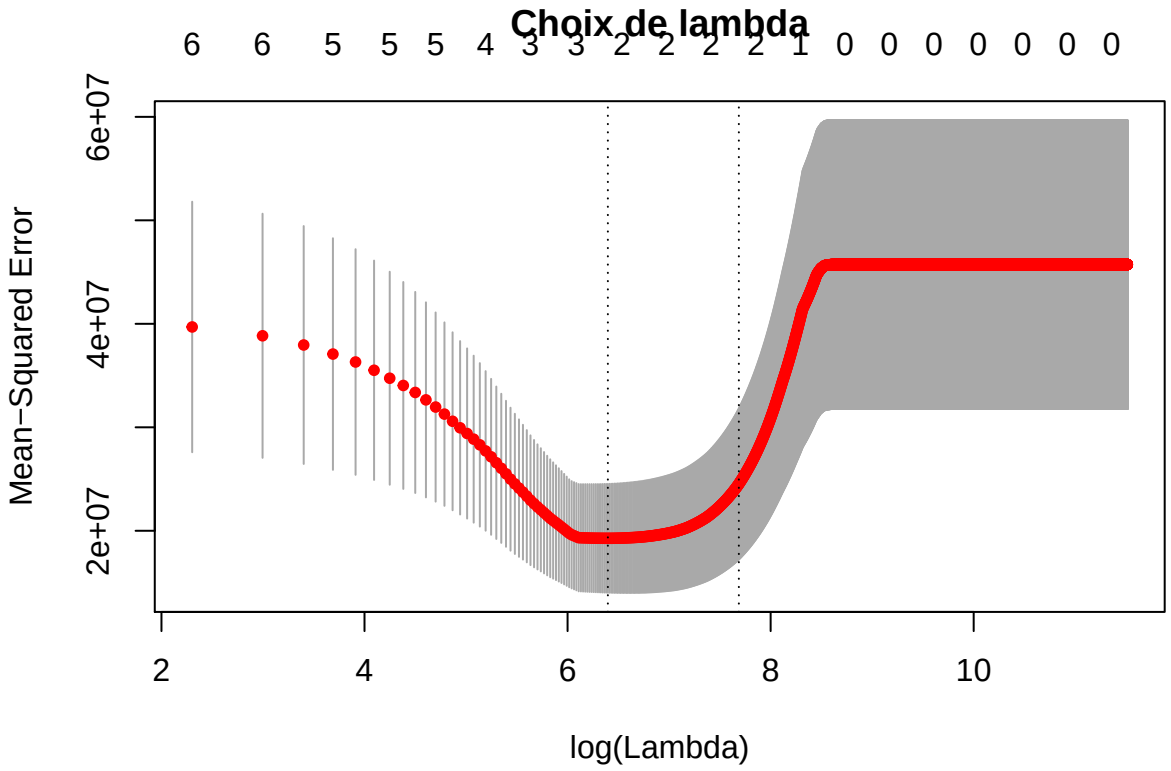
```
VITESSE          50.821733
```

Note: On n'aura pas leur significativité avec glmnet !

Régression Lasso

Pareil que pour la Ridge sauf que maintenant le paramètre `alpha` vaut 1.

```
cv_fit <- cv.glmnet(X,Y, alpha = 1, lambda = seq(0,100000, 10), grouped = FALSE, nfolds =nrow(d))
plot(cv_fit, main = "Choix de lambda")
```



Vous vous souvenez que la régression LASSO met certains coefficients à zero grace à la pénalité L_1. Le nombre de variables non-nulles dans le modèle (pour chaque valeur de lambda) est présenté de manière graphique sur la ligne supérieure (haut) du graphique. On voit ici que l'erreur de prédiction minimale est obtenue pour un modèle avec 2 variables ! (comme en stepAIC). Voyons lesquelles.

```
lambda_optimal = cv_fit$lambda.min
print(lambda_optimal)
```

```
[1] 600
```

```
print(min(cv_fit$cvm))
```

```
[1] 19273445
```

```
rmsep_lasso =sqrt(min(cv_fit$cvm))
print(rmsep_lasso)
```

```
[1] 4390.153
```

```
m_lasso <- glmnet(X,Y, alpha = 1, lambda = lambda_optimal)
coef(m_lasso)
```

```
7 x 1 sparse Matrix of class "dgCMatrix"
```

```
                    s0
(Intercept) 5985.12074
CYL            .
PUIS         155.86314
LON            .
LAR            .
POIDS         13.89069
VITESSE        .
```

Le RMSEP de Lasso égal à 4390.153 est le meilleur parmi les moèles PCR, PLS et Ridge ! Il selectionne les mêmes variables que stepAIC mais ne le bat pas en termes de prédiction (pour rappel, stepAIC donnait un RMSEP de 4139 !) Tout ce travail pour en arriver là !

Notons enfin que la fonction glmnet peut tout aussi bien s'appliquer dans le contexte de la régression logistique, de Poisson ou encore survie (voir le paramètre "family").

A vous maintenant!

On souhaite modéliser l'indice d'octane moteur de l'essence en fonction de son processus de fabrication impliquant plusieurs composantes. On dispose des variables suivantes :

- y = indice octane moteur
- x1 = distilation directe (valeur comprise entre 0 et 0.21)
- x2 = reformat (valeurs entre 0 et 0.62)
- x3 = naphta de craquage termique (entre 0 et 0.12)
- x4 = naphta de craquage cataitique (entre 0 et 0.62)
- x5 = polymere (entre 0 et 0.12)
- x6 = alkylat (entre 0 et 0.74)
- x7 = essence naturelle (entre 0 et 0.08)

Les données des 12 melanges d'essence sont présentées ci-dessous.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	y
0	0,23	0	0	0	0,74	0,03	98,7
0	0,1	0	0	0,12	0,74	0,04	97,8
0	0	0	0,1	0,12	0,74	0, 04	96,6
0	0,49	0	0	0,12	0,37	0,02	92
0	0	0	0,62	0,12	0,18	0,08	86,6
0	0,62	0	0	0	0,37	0,01	91,2
0,17	0,27	0,1	0,38	0	0	0,08	81,9
0,17	0,19	0,1	0,38	0,02	0,06	0,08	83,1
0,17	0,21	0,1	0,38	0	0,06	0,08	82,4
0,17	0,15	0,1	0,38	0,02	0,1	0,08	83,2
0,21	0,36	0,12	0,25	0	0	0,06	81,4
0	0	0	0,55	0	0,37	0,08	88,1

Faites une analyse de régression linéaire et choisisez le meilleur modèle prédictif. TP à rendre sous la forme d'un rapport écrit au format pdf et à déposer sur Moodle, cours : Modélisation avancée, Séance : TP Régression.